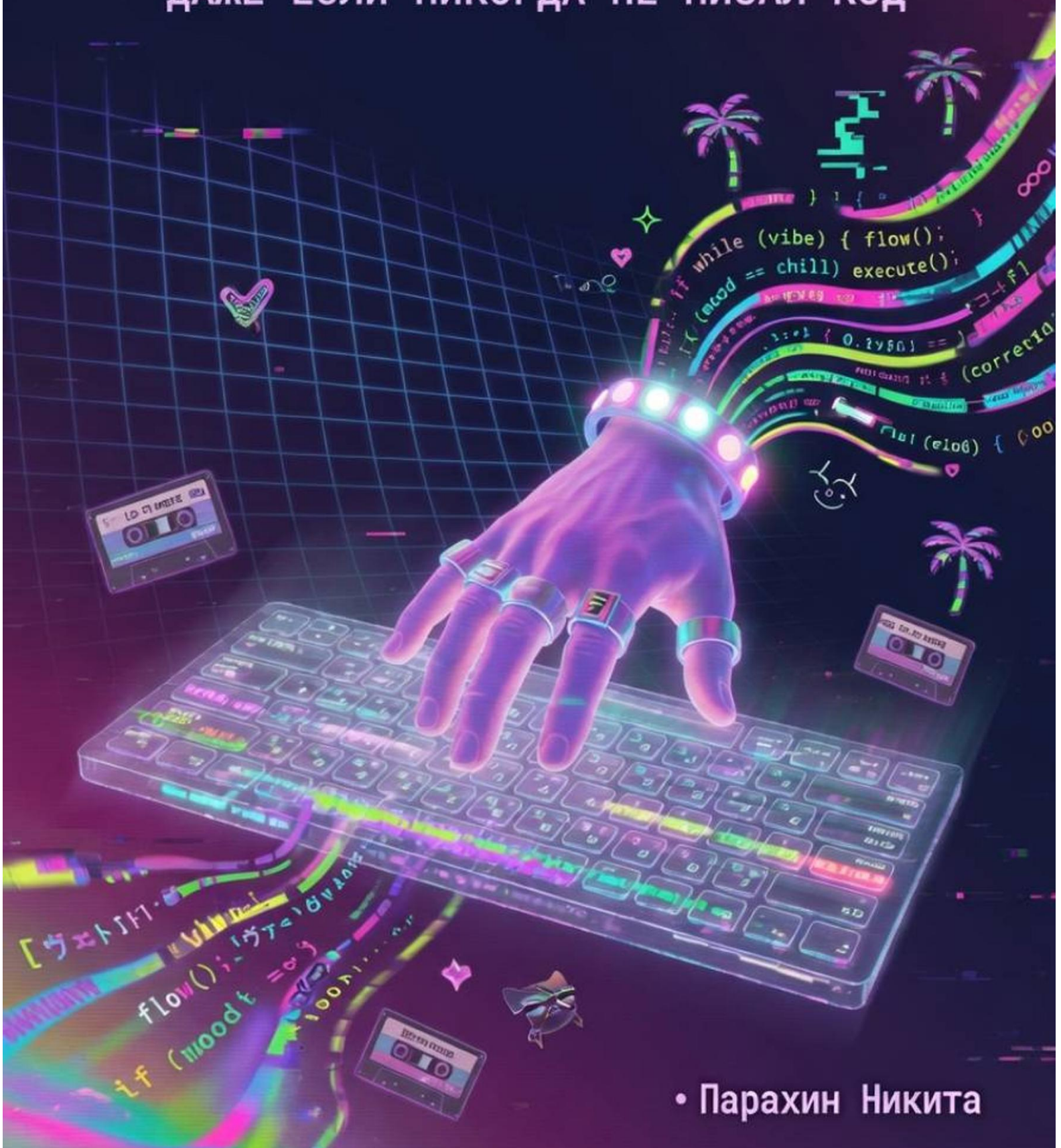
ВАЙБКОДИНГ

СОЗДАВАЙ ПРОДУКТЫ С ПОМОЩЬЮ AI, ДАЖЕ ЕСЛИ НИКОГДА НЕ ПИСАЛ КОД



• Парахин Никита

Никита Парахин

Вайб-кодинг: создавай продукты с помощью AI, даже если никогда не писал код

«Автор»
2026

Парахин Н. Л.

Вайб-кодинг: создавай продукты с помощью AI, даже если никогда не писал код / Н. Л. Парахин — «Автор», 2026

Книга по вайбкодингу — революционному подходу к созданию цифровых продуктов через общение с искусственным интеллектом. Вы научитесь строить сайты, веб-приложения, Telegram-боты и SaaS-сервисы с помощью Cursor и Claude, не написав ни строчки кода вручную. Пошаговые инструкции, готовые промпты, четыре реальных проекта — от лендинга до продукта с платежами. Книга для предпринимателей, дизайнеров, маркетологов и всех, кто хочет создавать и зарабатывать в эпоху AI.

Содержание

Никита Парахин
Вайб-кодинг: создавай продукты с помощью AI, даже если никогда не писал код

Вайбкодинг
Создавай продукты с помощью AI, даже если никогда не писал код
Парахин Никита
2026

Введение

Эта книга не должна была существовать.

Ещё три года назад сама идея о том, что человек без единого дня опыта в программировании сможет создать работающий веб-сайт, мобильное приложение или Telegram-бота, звучала как фантастика. Программирование было закрытым клубом со своим языком, своими ритуалами и высоким порогом входа. Чтобы написать простейшую программу, нужно было потратить месяцы на изучение синтаксиса, годы — на понимание архитектуры, и целую карьеру — на то, чтобы стать по-настоящему продуктивным.

А потом всё изменилось.

В феврале 2025 года Андрей Карпати — бывший директор по AI в Tesla, один из основателей OpenAI — написал пост, который разлетелся по всему интернету. Он рассказал, как провёл выходные, создавая программу, практически не написав ни строчки кода вручную. Он просто разговаривал с AI, описывая словами, что хочет получить. Он назвал это **vibe coding** — «кодинг по вайбу», программирование на ощущениях.

«Ты просто видишь вещи, говоришь вещи, запускаешь вещи, и оно работает», — написал он.

Это было не просто новое модное слово. Это было начало новой эпохи.

Сегодня, в 2026 году, вайбкодинг — это не экзотика для энтузиастов. Это мейнстрим. Тысячи людей по всему миру — предприниматели, дизайнеры, маркетологи, учителя, студенты — создают цифровые продукты, общаясь с искусственным интеллектом на обычном человеческом языке. Они не учились программированию годами. Они просто научились правильно объяснять AI, чего хотят.

И вот что важно: **вам не нужно становиться программистом, чтобы создавать программы**. Вам нужно стать вайбкодером.

Для кого эта книга

Эта книга написана для вас, если:

— Вы никогда не писали код и не знаете, что такое переменная — но у вас есть идея продукта, который хотите создать.

— Вы предприниматель, которому надоело платить разработчикам за каждую мелкую правку.

— Вы дизайнер, маркетолог или менеджер, который хочет прототипировать идеи самостоятельно.

— Вы студент, который ищет новый способ зарабатывать.

— Вы просто любопытный человек, который слышал про «вайбкодинг» и хочет разобраться.

Если хотя бы один пункт про вас — вы в нужном месте.

Чего ожидать от книги

Эта книга — не академический учебник. Здесь нет длинных теоретических рассуждений о том, как устроены компиляторы, и нет упражнений на запоминание синтаксиса. Вместо этого здесь есть:

Практика с первой страницы. Каждая глава содержит конкретные задания, которые вы выполняете в реальном редакторе кода. К концу книги у вас будет портфолио из четырёх полноценных проектов.

Фокус на двух инструментах. Мы не будем распылять внимание на десяток разных AI-помощников. Мы глубоко освоим два лучших инструмента: **Cursor** — AI-редактор кода, и **Claude** — языковую модель от Anthropic, которая станет вашим напарником в программировании.

Язык, понятный человеку. Каждый технический термин объясняется простыми словами и аналогиями. Если я пишу «API», я тут же объясню, что это как меню в ресторане — список того, что можно заказать.

Путь от нуля до заработка. Последняя часть книги посвящена тому, как превратить навыки вайбкодинга в источник дохода — через фриланс, создание SaaS-продуктов или работу в компании.

Как устроена книга

Книга разделена на шесть частей, которые последовательно проведут вас от полного нуля до уверенного создания цифровых продуктов:

Часть I. Новая эра — вы поймёте, что такое вайбкодинг, как работает AI, и настроите своё рабочее место.

Часть II. Искусство промптинга — вы научитесь разговаривать с AI так, чтобы он делал именно то, что вам нужно. Это ключевой навык вайбкодера.

Часть III. Основы веб-разработки — минимальная теория, которую полезно понимать, чтобы эффективно общаться с AI о коде.

Часть IV. Реальные проекты — четыре проекта нарастающей сложности, от лендинга до SaaS-продукта с платежами.

Часть V. Мастерство — Git, тестирование, безопасность — то, что отличает любителя от профессионала.

Часть VI. Бизнес и карьера — как зарабатывать на вайбкодинге и что ждёт нас в будущем.

Вы можете читать книгу последовательно от начала до конца — это рекомендуемый путь. Но если вам не терпится, можете после Части II сразу перейти к проектам в Части IV, возвращаясь к теории по мере необходимости.

Одно важное предупреждение

Вайбкодинг — это мощный инструмент, но не волшебная палочка. AI не заменяет мышление. Он заменяет рутину. Вам по-прежнему нужно будет думать: что именно вы хотите построить, для кого, и зачем. AI берёт на себя «как» — синтаксис, структуру, реализацию. Вы отвечаете за «что» и «зачем».

Лучшие вайбкодеры — это не те, кто знает больше всех о коде. Это те, кто яснее всех мыслит и точнее всех формулирует.

Готовы? Тогда начнём.

Часть I. НОВАЯ ЭРА: ПРОГРАММИРОВАНИЕ БЕЗ КОДА

Глава 1. Что такое вайбкодинг и почему это меняет всё

«Есть новый вид программирования, который я называю "vibe coding", где ты полностью отдаёшься вайбу, принимаешь экспоненциальные возможности и забываешь о том, что код вообще существует.»

— Андрей Карпати, февраль 2025

1.1. История: от перфокарт до промптов

Чтобы по-настоящему оценить революцию, которую мы переживаем, стоит оглянуться назад и увидеть, какой путь прошло программирование.

В 1950-х годах, чтобы «поговорить» с компьютером, нужно было перемещать физические переключатели или пробивать отверстия в картонных карточках — перфокартах. Одна ошибка — и стопку карточек приходилось пересобирать заново. Программист того времени был больше похож на оператора сложного станка, чем на творца.

В 1960-х появились первые языки программирования высокого уровня — FORTRAN, COBOL, BASIC. Вместо бинарных кодов можно было писать нечто похожее на английские предложения: PRINT "HELLO WORLD". Это был огромный шаг, но программирование всё ещё оставалось уделом специалистов.

В 1980-х и 1990-х произошла персональная компьютерная революция. Появились Visual Basic, Delphi — среды, где можно было «рисовать» интерфейс мышкой и дописывать логику. Порог входа снизился, но всё равно требовал серьёзного обучения.

В 2000–2010-х расцвёл веб. PHP, JavaScript, Python — языки, которые привлекли миллионы новых программистов. Появились фреймворки, библиотеки, Stack Overflow, где можно было найти ответ на любой вопрос. Порог входа стал ещё ниже, но всё равно измерялся месяцами обучения.

В 2020-х началась эра AI-ассистентов. GitHub Copilot (2021) впервые показал, что AI может дописывать код за программиста. ChatGPT (2022) научил миллионы людей общаться с AI. А в 2025 году инструменты вроде Cursor с Claude довели эту идею до логического завершения: теперь можно описать словами, что ты хочешь, и получить работающий код.

Каждое поколение технологий снижало барьер входа. Перфокарты → языки → визуальные среды → фреймворки → копилоты → **вайбкодинг**. Мы живём в момент, когда барьер практически исчез.

1.2. Андрей Карпати и рождение термина

Историю стоит рассказать подробнее, потому что она прекрасно иллюстрирует суть явления.

Андрей Карпати — не случайный блогер. Это один из самых влиятельных людей в мире AI. Он был сооснователем OpenAI, затем возглавлял подразделение искусственного интеллекта в Tesla, где создавал систему автопилота. Он — человек, который умеет программировать лучше, чем 99.9% людей на планете.

И вот этот человек в феврале 2025 года пишет, что ему больше нравится не программировать вручную, а просто объяснять AI, что нужно сделать.

Его пост в соцсетях стал вирусным. Он описал, как создал небольшой проект, общаясь с Cursor и Claude:

— Он описывал, что хочет, обычными словами.

— AI генерировал код.

— Карпати смотрел на результат — если работает, оставлял.

— Если не работает — копировал ошибку и просил AI исправить.

— Он признался, что даже не читал большую часть сгенерированного кода.

«Я просто вижу вещи, говорю вещи, запускаю вещи — и оно по большей части работает», — написал он.

Термин **vibe coding** мгновенно разлетелся. Он попал в заголовки The Verge, Wired, TechCrunch. К лету 2025 года его использовали повсюду — от конференций до вакансий. К 2026 году слово вошло в повседневный лексикон технологической индустрии, а в русскоязычном пространстве прижилась калька — «вайбкодинг».

Но самое интересное в этой истории — не терминология. Самое интересное — это подтекст. Если даже один из лучших программистов мира предпочитает объяснять задачи AI вместо того, чтобы писать код вручную, это значит, что **навык формулирования стал важнее навыка написания кода**.

1.3. Почему 2025–2026 стал переломным моментом

Идея генерации кода с помощью AI существовала давно. Почему именно сейчас всё сошлось?

Три фактора совпали одновременно.

Фактор первый: качество моделей. Claude 3.5 Sonnet, вышедший в середине 2024 года, стал первой моделью, которая могла генерировать сложный, продуктовый код без постоянного «надзора». Claude Opus 4, появившийся в 2025-м, и последующие версии подняли планку ещё выше. Код стал не просто работающим — он стал хорошим. Модели научились понимать архитектуру проектов, следовать best practices и самостоятельно отлавливать ошибки.

Фактор второй: инструменты. Cursor совершил прорыв, интегрировав AI не как внешний чатбот, а как первоклассную часть редактора кода. Composer позволил редактировать несколько файлов одновременно через текстовый промпт. Background Agents научились выполнять сложные задачи автономно. AI перестал быть «помощником, который подсказывает» — он стал «напарником, который делает».

Фактор третий: экосистема. Vercel, Supabase, Stripe, Telegram Bot API — появилась целая экосистема сервисов, которые позволяют деплоить продукты за минуты. Раньше, даже написав код, нужно было разбираться с серверами, DNS, сертификатами. Сейчас деплой сайта — это одна команда в терминале.

Когда качественные модели встретились с удобными инструментами и готовой инфраструктурой, порог входа обрушился. Впервые в истории технологий **идея стала дороже реализации**.

1.4. Вайбкодинг — это не «игрушка»: реальные продукты

Скептики часто спрашивают: «Ну хорошо, можно сделать красивый демо. А реальные продукты?»

Давайте посмотрим на факты.

К началу 2026 года сотни стартапов были построены людьми без традиционного технического образования, используя подход вайбкодинга. Вот несколько характерных примеров:

Предприниматель из Санкт-Петербурга за три недели создал SaaS-платформу для автоматизации ресторанного бизнеса. Он никогда не учился программированию — до этого он был управляющим ресторана. Он использовал Cursor и Claude, чтобы построить систему приёма заказов, управления складом и аналитики. Через полгода у него было 40 платящих клиентов.

Дизайнер из Москвы создала маркетплейс для продажи цифровых шаблонов. Весь фронтенд и бэкенд были написаны через промпты в Cursor. Она зарабатывает на подписках и комиссии с продаж.

Школьный учитель из Казани сделал Telegram-бота для проверки домашних заданий по математике, подключив Claude API. Бот проверяет решения, даёт подсказки и объясняет ошибки. Им пользуются несколько сотен учеников.

Это не единичные случаи. Это тенденция. На Хабре, Product Hunt и в Telegram-каналах каждую неделю появляются новые истории людей, которые запустили работающий продукт, не зная программирования в традиционном смысле.

Вайбкодинг — это не «игрушка для блогеров». Это инструмент создания реальной ценности.

1.5. Кому подходит эта книга и чего от неё ожидать

Давайте будем честны друг с другом с самого начала.

Вайбкодинг подходит, если:

— Вы готовы учиться формулировать мысли чётко и конкретно.

— Вы не боитесь ошибок и готовы экспериментировать.

— У вас есть конкретная идея или хотя бы направление — что хотите создать.

— Вы готовы тратить время на практику, а не только на чтение.

Вайбкодинг может не подойти, если:

— Вы ожидаете, что AI сделает всё за вас без какого-либо вашего участия.

— Вам нужно создать критически важное программное обеспечение (медицинское оборудование, банковские системы) — здесь по-прежнему нужны профессиональные инженеры.

— Вы не готовы вкладывать время в понимание того, что делает AI, хотя бы на поверхностном уровне.

К концу этой книги вы будете:

— Уверенно работать в Cursor и общаться с Claude для создания кода.

— Понимать базовую структуру веб-приложений.

— Уметь строить промпты, которые дают предсказуемый результат.

— Иметь портфолио из четырёх работающих проектов.

— Знать, как зарабатывать на своих навыках.

1.6. Мифы о вайбкодинге: что правда, а что нет

За год существования термина вокруг вайбкодинга накопилось немало мифов. Давайте разберём основные.

Миф 1: «Вайбкодинг убьёт профессию программиста»

Нет. Вайбкодинг не убивает программирование — он его демократизирует. Так же, как Instagram (принадлежит Meta Platforms Inc., признанной экстремистской организацией и запрещённой в РФ) не убил профессиональную фотографию, а Canva не убила дизайн. Появился новый слой людей, которые могут создавать «достаточно хорошие» решения. Про-

фессиональные разработчики по-прежнему нужны для сложных систем, но простые продукты теперь могут делать все.

Миф 2: «Код, написанный AI, плохой и ненадёжный»

Это было отчасти верно для ранних моделей в 2023 году. Но Claude образца 2026 года генерирует код, который проходит тесты, следует современным паттернам и зачастую чище, чем код среднего джуниор-разработчика. Да, бывают ошибки. Но они есть и у людей. Разница в том, что AI-ошибки обычно типовые и хорошо ловятся.

Миф 3: «Вайбкодинг — это просто копирование чужого кода»

Нет. LLM не копирует код из базы данных. Модель генерирует новый код на основе паттернов, которые она выучила при обучении — примерно так же, как человек пишет текст, используя правила языка, которые он освоил, а не копируя чужие предложения.

Миф 4: «Достаточно сказать AI — и всё само заработает»

Самый опасный миф. В реальности вайбкодинг — это итеративный процесс. Вы объясняете → AI делает → вы проверяете → уточняете → AI доделывает. Иногда нужно три-четыре итерации, чтобы получить то, что нужно. Умение вести этот диалог — и есть главный навык.

Миф 5: «Вайбкодеры не понимают, что создают»

Хороший вайбкодер понимает. Не на уровне каждой строчки кода, но на уровне архитектуры: что с чем связано, какие данные куда идут, почему этот подход лучше другого. Эта книга научит вас именно такому уровню понимания.

Итоги главы 1

В этой главе мы разобрались с основами:

— Вайбкодинг — это подход к созданию программ через общение с AI на естественном языке, без ручного написания кода.

— Термин придумал Андрей Карпати в феврале 2025 года.

— 2025–2026 годы стали переломными благодаря совпадению трёх факторов: качество моделей, удобство инструментов, готовая инфраструктура.

— Реальные люди уже создают реальные продукты и зарабатывают с помощью вайбкодинга.

— Это не магия — это навык, который требует практики, особенно в области формулирования мыслей.

В следующей главе мы заглянем «под капот» и разберёмся, как на самом деле работает AI, с которым вы будете общаться каждый день. Не волнуйтесь — это будет объяснение для людей, а не для машин.

Задание к главе: Зайдите на сайт cursor.com и claude.ai. Создайте бесплатные аккаунты, если у вас их ещё нет. Попробуйте задать Claude любой вопрос о программировании — например, «Объясни мне, как работает веб-сайт, простыми словами». Обратите внимание на то, насколько подробным и понятным будет ответ. Это ваш будущий напарник.

Глава 2. Как думает искусственный интеллект

«AI не понимает мир. Он понимает язык. И этого оказалось достаточно, чтобы перевернуть программирование.»

2.1. Что такое LLM: объяснение для нетехнарей

Вы открываете Claude, пишете: «Создай форму регистрации на React с валидацией email» — и через десять секунд получаете готовый, работающий код. Как это возможно? Как машина «понимает» вашу просьбу?

Давайте разберёмся, но без зубрёжки формул и математики. Нам нужно понимание на уровне водителя, а не автомеханика. Вы ведь не разбираете двигатель каждый раз, когда садитесь за руль — но знаете, что бензин нужно заливать, а не молоко.

LLM расшифровывается как Large Language Model — «большая языковая модель». Это программа, которая обучена на огромном массиве текстов: книги, статьи, форумы, документация, код. Представьте человека, который прочитал всю Википедию, весь Stack Overflow, все учебники по программированию и миллионы строк кода на GitHub. Причём прочитал не один раз, а проанализировал паттерны — какие слова обычно идут за какими, какие конструкции кода решают какие задачи, как структурированы ответы на вопросы.

Вот ключевая аналогия. Представьте, что вы читали тысячи кулинарных рецептов. Вы никогда не готовили, но вы прочитали столько рецептов, что если вас попросят написать рецепт тыквенного супа, вы сможете составить вполне правдоподобный. Вы знаете, что сначала идут ингредиенты, потом шаги, что тыкву обычно запекают или варят, что к ней добавляют сливки и мускатный орех. Вы не «понимаете» вкус тыквенного супа — вы понимаете *паттерн* рецепта.

LLM работает похоже, только масштаб другой. Модель проанализировала триллионы слов и научилась предсказывать: «Если мне дан этот контекст, какое продолжение будет наиболее уместным?» Когда вы просите создать форму регистрации, модель не «придумывает» её с нуля. Она генерирует код на основе тысяч похожих форм, которые видела при обучении, адаптируя под ваш конкретный запрос.

2.2. Как модели «понимают» запросы: токены, контекст, внимание

Три концепции, которые полезно знать вайбкодеру, чтобы работать эффективнее.

Токены — это «слова» для машины.

Когда вы пишете промпт, модель не видит слова так, как видите их вы. Она разбивает текст на маленькие кусочки — токены. Слово «программирование» может быть разбито на «програм», «мир», «ование». Английские слова обычно компактнее: «code» — один токен, «function» — один-два токена.

Почему это важно для вас? Потому что у каждой модели есть лимит контекста — максимальное количество токенов, которое она может обработать за раз. Для Claude в 2026 году это около 200 000 токенов — примерно 150 000 слов, или целая книга. Это значит, что вы можете загрузить в контекст весь свой проект и обсуждать его целиком. Но если проект очень большой, придётся выбирать, какие файлы показать модели.

Контекст — это «память» разговора.

Всё, что вы написали в текущем разговоре с AI, плюс всё, что он ответил, плюс системные инструкции — всё это контекст. Модель «помнит» весь разговор и учитывает его при каждом ответе.

Вот почему важно вести чистый диалог. Если вы в середине разговора вдруг начнёте обсуждать совершенно другой проект, модель может запутаться — старый контекст будет мешать. Лучше начать новый чат.

Это как разговор с человеком. Если вы полчаса обсуждали рецепт борща, а потом вдруг спросили «а что с тем дизайном?» — собеседник растеряется. Модель тоже.

Механизм внимания — это «фокус» модели.

Когда вы задаёте вопрос, модель не просто просматривает весь контекст слева направо. Она научилась обращать внимание на самые важные части. Если вы спросили про кнопку в форме, модель «сфокусируется» на той части контекста, где описана форма, а не на обсуждении базы данных, которое было раньше.

Это работает хорошо, но не идеально. Иногда модель теряет фокус, особенно в очень длинных разговорах. Вот почему периодически полезно повторить ключевую информацию или начать новый чат.

2.3. Claude vs ChatGPT vs Gemini: ключевые различия

На рынке есть три основных семейства моделей для программирования. Давайте разберёмся, чем они отличаются и почему мы в этой книге используем Claude.

Claude (Anthropic)

Claude разработан компанией Anthropic, основанной бывшими сотрудниками OpenAI. На момент написания книги актуальные модели — Claude Opus 4 (самая мощная) и Claude Sonnet 4 (быстрая и качественная).

Сильные стороны Claude для кодинга:

— Отличное следование инструкциям. Если вы подробно описали, что хотите, Claude сделает именно это, а не что-то «похожее».

— Длинный контекст. 200 000 токенов — это значит, что Claude может работать с большими проектами целиком.

— Аккуратный, чистый код. Claude склонен генерировать код, который хорошо структурирован, с понятными названиями переменных и комментариями.

— Честность. Если Claude не уверен — он скажет об этом, а не придумает ответ.

ChatGPT (OpenAI)

ChatGPT — самая известная модель, с которой начался AI-бум. Модели GPT-4o и GPT-5 хороши для программирования, но имеют свои особенности:

— Более «разговорный» стиль — иногда добавляет лишние объяснения, когда вы просто хотите код.

— Хорош для генерации идей и мозгового штурма.

— Иногда «галлюцинирует» — уверенно утверждает вещи, которые неверны.

Gemini (Google)

Gemini — модель от Google, интегрированная в экосистему Google.

— Хороша для работы с данными и аналитикой.

— Глубоко интегрирована в Google-сервисы.

— Для кодинга используется реже, чем Claude и ChatGPT.

Для вайбкодинга мы выбираем Claude по трём причинам: он лучше всех следует детальным инструкциям (а это основа промптинга), он генерирует более чистый и безопасный код, и он идеально интегрирован с Cursor — лучшим AI-редактором кода.

Это не значит, что вы должны забыть про другие модели. Cursor позволяет переключаться между ними. Иногда для конкретной задачи другая модель может подойти лучше. Но как основной рабочий инструмент — Claude.

2.4. Почему Claude особенно хорош для кода

Давайте копнём глубже и разберёмся, что конкретно делает Claude таким подходящим для вайбкодинга.

Архитектурное мышление. Claude не просто генерирует код по строчке — он способен мыслить на уровне архитектуры проекта. Если вы попросите его создать приложение, он предложит структуру папок, разделение на компоненты, выбор подходящих библиотек. Это как иметь архитектора, а не просто каменщика.

Понимание контекста проекта. Через Cursor вы можете показать Claude все файлы вашего проекта. Он проанализирует их и будет генерировать код, который вписывается в существующую структуру. Новый компонент будет написан в том же стиле, что и остальные. Новая функция будет использовать уже существующие утилиты, а не создавать дубликаты.

Итеративная доработка. Claude отлично работает в режиме диалога. Вы можете сказать: «Сделай, но поменяй цвета на более тёмные и добавь анимацию при наведении» — и он точечно изменит нужное, не ломая остальное. Это критически важно для вайбкодинга, где вы строите продукт шаг за шагом.

Объяснение кода. Даже если вы новичок, Claude может объяснить любой фрагмент кода простыми словами. Вы можете выделить непонятный кусок в Cursor и спросить: «Что это делает?» Вы будете постепенно учиться, даже не ставя такой цели.

Осторожность. Claude скорее предупредит вас о потенциальных проблемах, чем промолчит. «Этот подход может создать проблемы с безопасностью, давайте сделаем по-другому» — типичная фраза Claude. Для новичка это бесценная страховочная сетка.

2.5. Ограничения AI: что он не может (пока)

Было бы нечестно рассказывать только о достоинствах. У AI-моделей есть реальные ограничения, которые вайбкодер должен знать, чтобы не попасть в ловушку.

Галлюцинации. Иногда модель уверенно генерирует код, который использует несуществующие функции или библиотеки. Например, она может придумать метод array.removeDuplicatesInPlace(), которого нет в JavaScript. Она не врёт намеренно — она генерирует «правдоподобное продолжение», и иногда это продолжение не соответствует реальности.

Как с этим бороться? Всегда запускайте код. Если он не работает — покажите ошибку Claude, и он исправит.

Отсутствие памяти между сессиями. Каждый новый чат — это чистый лист. Claude не помнит, что вы обсуждали вчера, если только вы не используете специальные инструменты для сохранения контекста (мы поговорим о них позже). Это значит, что в начале каждого рабочего дня нужно заново «ввести в курс дела» — показать файлы проекта, объяснить, над чем работаете.

Нет доступа к интернету в реальном времени. Claude при работе через Cursor не ходит в интернет, чтобы проверить, существует ли библиотека или какая у неё актуальная версия. Он полагается на знания, полученные при обучении. Иногда эти знания устаревают. Новая библиотека, вышедшая на прошлой неделе, может быть ему неизвестна.

Как с этим бороться? Если Claude предлагает использовать библиотеку, быстро проверьте на npmjs.com или pypi.org, что она существует и поддерживается. Это занимает 30 секунд и может сэкономить часы отладки.

Ограниченная «логическая цепочка». Модели хороши в генерации кода по паттернам, но могут ошибаться в сложной бизнес-логике. Если у вас система скидок с десятью усло-

виями — лучше разбить задачу на части, а не просить всё сразу. AI отлично справляется с каждым шагом, но может запутаться в сложной комбинации.

Не понимает ваш бизнес. AI знает, как написать код. Он не знает, кто ваши клиенты, что им нужно и как устроен ваш рынок. Продуктовое мышление — это ваша работа. AI — исполнитель, вы — продюсер.

2.6. Этика и ответственность: кто автор кода?

Этот вопрос звучит философски, но имеет вполне практические последствия.

Авторское право. На момент написания книги (2026) юридический статус кода, сгенерированного AI, остаётся в серой зоне. В большинстве юрисдикций AI не может быть автором — авторские права принадлежат человеку, который сформулировал задачу и направлял процесс. То есть вам. Но законодательство продолжает развиваться, и в разных странах ситуация может отличаться.

Что это значит на практике? Если вы создаёте коммерческий продукт с помощью AI — вы имеете на это право. Вы можете продавать код, созданный при участии Claude. Но стоит следить за обновлениями в законодательстве вашей страны.

Ответственность за качество. Вот здесь принципиально важный момент. Код написал AI, но ответственность за его работу несёте вы. Если ваш сайт взломали, потому что AI оставил уязвимость — виноваты вы, не Anthropic. Если бот отправил клиентам неправильную информацию — отвечаете вы.

Это не повод бояться. Это повод быть внимательным. В главе 10 мы подробно разберём практики безопасности и тестирования, которые помогут минимизировать риски.

Этичное использование. AI — мощный инструмент, и его можно использовать по-разному. Создавать полезные продукты, автоматизировать рутину, помогать людям — это прекрасно. Создавать спам-сайты, копировать чужие продукты, генерировать обманчивый контент — это не путь вайбкодера.

Мы все формируем культуру вайбкодинга прямо сейчас. От того, как мы используем эти инструменты, зависит, как к ним будут относиться завтра.

Прозрачность. Стоит ли говорить клиентам, что вы используете AI? Мнения расходятся. Моя позиция: это ваш инструмент, как Photoshop для дизайнера. Дизайнер не обязан говорить клиенту: «Я использовал Photoshop». Но если клиент спрашивает напрямую — лучше быть честным. Честность укрепляет доверие, а доверие — основа долгосрочного бизнеса.

Итоги главы 2

В этой главе мы заглянули под капот AI:

— LLM — это модель, обученная на огромном массиве текстов, которая предсказывает наиболее уместное продолжение на основе вашего запроса.

— Токены, контекст и механизм внимания — три концепции, которые объясняют, как модель обрабатывает ваши промпты.

— Claude выделяется среди конкурентов точностью следования инструкциям, чистотой кода и длинным контекстом.

— У AI есть реальные ограничения: галлюцинации, отсутствие памяти, незнание свежих данных. Зная о них, вы можете их компенсировать.

— Юридически вы — автор кода. Этически вы — ответственны за результат.

В следующей главе мы наконец перейдём от теории к практике. Установим Cursor, подключим Claude и напишем первый код — вернее, попросим AI написать его за нас.

Задание к главе: Откройте claude.ai и проведите небольшой эксперимент. Задайте один и тот же вопрос — например, «Напиши функцию на JavaScript, которая считает факториал числа» — и посмотрите на ответ. Затем спросите: «Объясни этот код построчно, как будто мне 10 лет». Обратите внимание, насколько по-разному модель может объяснять одно и то же в зависимости от вашего запроса. Это и есть сила промптинга, которую мы освоим в главе 4.

Глава 3. Ваше рабочее место: Cursor + Claude

«Дайте мне шесть часов, чтобы срубить дерево, и первые четыре я буду точить топор.»

— приписывается Аврааму Линкольну

Хороший инструмент — половина успеха. В этой главе мы установим и настроим всё, что нужно для продуктивной работы. К концу главы вы напишете — вернее, попросите AI написать — ваш первый работающий проект.

3.1. Что вам понадобится

Давайте начнём с хорошей новости: вам не нужен мощный компьютер. Cursor — это редактор кода, а не видеоигра. Весь «тяжёлый» AI-вычислительный процесс происходит на серверах Anthropic и Cursor, а не на вашем компьютере.

Минимальные требования:

Компьютер. Любой ноутбук или настольный ПК, выпущенный после 2018 года. Windows 10/11, macOS 11+, или Linux. Минимум 4 ГБ оперативной памяти (рекомендуется 8 ГБ). Минимум 2 ГБ свободного места на диске.

Интернет. Стабильное подключение обязательно — без него AI работать не будет. Скорость не критична: хватит даже мобильного интернета, хотя за Wi-Fi вы будете благодарны, когда начнёте загружать библиотеки.

Аккаунты. Вам понадобится создать аккаунт на Cursor (cursor.com) и, опционально, на Anthropic (claude.ai). Cursor предоставляет доступ к Claude через свою подписку, так что отдельный аккаунт Claude нужен скорее для экспериментов и чтения документации.

Бюджет. Cursor предлагает бесплатный тариф с ограниченным количеством запросов — этого хватит для обучения. Для серьёзной работы потребуется подписка Pro ($20/месяц на момент написания книги). Считайте это инвестицией: один оплаченный заказ на лендинг окупит подписку на год.

3.2. Установка и настройка Cursor: пошаговая инструкция

Cursor — это редактор кода, построенный на основе VS Code (популярный бесплатный редактор от Microsoft), но с глубоко встроенным AI. Если вы когда-нибудь видели VS Code — интерфейс покажется знакомым. Если нет — не волнуйтесь, мы всё разберём.

Шаг 1. Скачайте Cursor.

Зайдите на cursor.com и нажмите кнопку загрузки. Сайт автоматически определит вашу операционную систему и предложит нужную версию.

Шаг 2. Установите.

На Windows: запустите скачанный .exe файл и следуйте инструкциям установщика. На macOS: откройте .dmg файл и перетащите Cursor в папку Applications. На Linux: загрузите .AppImage или .deb файл, в зависимости от вашего дистрибутива.

Шаг 3. Первый запуск.

При первом запуске Cursor предложит:

— Импортировать настройки из VS Code (если он у вас был — соглашайтесь).

— Создать аккаунт или войти через Google/GitHub.

— Выбрать тему оформления (тёмную или светлую — это чисто эстетический выбор, но большинство разработчиков предпочитают тёмную, она меньше утомляет глаза).

Шаг 4. Активируйте подписку (опционально).

Бесплатный тариф даёт ограниченное количество «быстрых» запросов к AI в день. Для обучения этого хватит. Когда почувствуете, что упираетесь в лимит — время перейти на Pro.

3.3. Подключение Claude к Cursor

По умолчанию Cursor может использовать несколько моделей. Нам нужно убедиться, что Claude — наша основная модель.

Откройте настройки Cursor: нажмите Cmd + , на Mac или Ctrl + , на Windows/Linux. Перейдите в раздел «Models» или «AI Settings».

Здесь вы увидите список доступных моделей. Убедитесь, что Claude Sonnet выбран как модель по умолчанию для Composer и Chat. Для сложных задач можно переключаться на Claude Opus — он мощнее, но медленнее и дороже по квоте.

Рекомендация для начала: используйте Claude Sonnet для повседневной работы. Он быстрый и достаточно умный для 90% задач. Переключайтесь на Opus, когда сталкиваетесь с по-настоящему сложной архитектурной задачей или когда Sonnet не справляется.

3.4. Первое знакомство с интерфейсом

Интерфейс Cursor может показаться пугающим, если вы никогда не работали с редактором кода. Давайте разберём его по частям.

Боковая панель (слева). Здесь отображается структура вашего проекта — папки и файлы, как в проводнике Windows или Finder на Mac. Вы кликаете на файл — он открывается в центральной области.

Центральная область. Здесь вы видите и редактируете код. Каждый открытый файл — это вкладка наверху, как в браузере. Вы можете открыть несколько файлов одновременно.

Терминал (внизу). Это командная строка. Звучит страшно, но на практике вы будете использовать её для простых вещей: запуск проекта, установка библиотек, деплой. И даже эти команды вам обычно подскажет Claude.

Чтобы открыть терминал, нажмите ` Ctrl + `` (клавиша с обратной кавычкой, слева от цифры 1).

AI-панель (справа или снизу). Это чат с Claude. Здесь вы будете проводить большую часть времени. Открывается по Cmd + L (Mac) или Ctrl + L (Windows/Linux).

Не пытайтесь запомнить все горячие клавиши сразу. Запомните три:

— Cmd/Ctrl + L — открыть чат с AI

— Cmd/Ctrl + I — открыть Composer (мультифайловое редактирование)

— Ctrl + `` — открыть терминал

Остальное придёт с практикой.

3.5. Tab, Chat, Composer, Agent: четыре режима работы

Cursor предлагает четыре способа взаимодействия с AI. Каждый подходит для своих ситуаций.

Tab (автодополнение).

Самый простой режим. Вы просто пишете код, и Cursor подсказывает продолжение серым текстом. Нажимаете Tab — подсказка вставляется. Это работает как «предиктивный ввод» на телефоне, только для кода.

Когда использовать: когда вы уже в процессе написания кода и нужно быстро дополнить строку или блок. Tab особенно хорош для рутины — импорт библиотек, типовые конструкции, повторяющиеся паттерны.

Chat (чат).

Разговор с Claude в боковой панели. Вы задаёте вопрос или описываете задачу, Claude отвечает — объясняет, предлагает решение, генерирует код. Код из чата можно вставить в файл одним кликом.

Когда использовать: когда нужно обсудить подход, задать вопрос, разобраться в ошибке, попросить объяснение. Chat — это ваш «разговор с наставником».

Composer (композитор).

Это самый мощный инструмент для вайбкодера. Вы описываете, что хотите, и Composer вносит изменения сразу в несколько файлов одновременно. Он показывает diff — какие строки будут добавлены, удалены или изменены — и вы подтверждаете или отклоняете каждое изменение.

Когда использовать: когда нужно создать новый компонент, добавить фичу, исправить баг, который затрагивает несколько файлов. Composer — это ваш «главный рабочий инструмент».

Agent (агент).

Начиная с версии Cursor 3.0, появился режим Agent — AI, который работает более автономно. Он может сам читать файлы, выполнять команды в терминале, устанавливать библиотеки, запускать тесты и итеративно исправлять ошибки. Вы описываете задачу высокого уровня, а Agent сам решает, как её выполнить.

Когда использовать: для больших задач, где вы доверяете AI действовать самостоятельно. Например: «Добавь аутентификацию через Google в наш проект, используя Supabase Auth».

Background Agents идут ещё дальше — они работают в фоне, пока вы занимаетесь другими делами, и присылают результат, когда закончат.

Для новичка рекомендуемый путь освоения: Chat → Composer → Agent → Background Agents. Начните с простого, наращивайте доверие к AI постепенно.

3.6. Claude Code: работа через командную строку

Помимо Cursor, у Anthropic есть отдельный инструмент — Claude Code. Это способ работать с Claude прямо из терминала (командной строки), без графического интерфейса.

Зачем это нужно? Claude Code даёт максимальный контроль и гибкость. Он может читать файлы на вашем компьютере, выполнять команды, работать с Git — всё через текстовый диалог. Многие опытные вайбкодеры используют его для сложных задач, потому что он работает ближе к «сырому» Claude без ограничений интерфейса.

Для установки Claude Code на macOS или Linux откройте терминал и выполните:

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

Для этого вам понадобится Node.js — мы установим его чуть позже, когда начнём работать над проектами.

На начальном этапе Claude Code вам не обязателен — Cursor покрывает все потребности. Но знать о его существовании полезно: когда вы вырастете как вайбкодер, он станет мощным дополнением к вашему арсеналу.

3.7. Настройка для комфортной работы

Несколько настроек, которые сделают вашу жизнь приятнее.

Размер шрифта. В настройках Cursor (Cmd/Ctrl + ,) найдите «Font Size» и установите комфортный размер. Стандарт — 14px, но если у вас большой монитор, можно увеличить до 16–18px. Не стесняйтесь делать шрифт крупным — глаза скажут спасибо.

Тема оформления. Зайдите в Cmd/Ctrl + K, Cmd/Ctrl + T и выберите тему. Для работы вечером рекомендуется тёмная тема (Dark+, One Dark Pro, GitHub Dark). Для дневной работы при ярком свете — светлая (Light+, GitHub Light).

Автосохранение. Включите автоматическое сохранение файлов: Settings → «Auto Save» → выберите «afterDelay». Это избавит от классической ситуации, когда вы забыли сохранить файл и не понимаете, почему изменения не работают.

Расширения. Cursor поддерживает расширения от VS Code. Несколько полезных для начала:

— **Prettier** — автоматическое форматирование кода. Нажали сохранить — код стал красивым.

— **Error Lens** — показывает ошибки прямо в строке кода, а не только в панели проблем.

— **Auto Rename Tag** — при переименовании HTML-тега автоматически переименовывает закрывающий тег.

Установка расширений: нажмите Cmd/Ctrl + Shift + X, найдите нужное по названию, нажмите Install.

3.8. Ваш первый проект: «Hello, World» по-вайбкодерски

Теория закончилась. Давайте создадим что-то реальное.

Традиционно первая программа каждого разработчика выводит на экран «Hello, World». Мы пойдём дальше и создадим полноценную персональную страницу-визитку. С нуля. За 10 минут. Не написав ни строчки кода вручную.

Шаг 1. Создайте папку для проекта.

Откройте терминал в Cursor (` Ctrl + `) и выполните:

mkdir my-first-vibe-project

cd my-first-vibe-project

Это создаст новую папку и перейдёт в неё. Затем откройте эту папку в Cursor: File → Open Folder → выберите созданную папку.

Шаг 2. Откройте Composer.

Нажмите Cmd/Ctrl + I для открытия Composer. Это ваш главный инструмент.

Шаг 3. Напишите промпт.

Вот ваш первый промпт. Скопируйте его в Composer:

Создай одностраничный сайт-визитку на чистом HTML + CSS.

Требования:
- Тёмная цветовая схема с акцентным цветом #6366f1 (индиго)
- В шапке: моё имя (подставь "Имя Фамилия"), должность "Вайбкодер"
- Секция "Обо мне" с коротким текстом-заглушкой
- Секция "Навыки" с 4 карточками: Cursor, Claude, React, Tailwind
- Секция "Контакты" с иконками email и Telegram
- Адаптивный дизайн (хорошо выглядит на телефоне)
- Плавные анимации при прокрутке (CSS-only)
- Файл: index.html (всё в одном файле, включая CSS)

Нажмите Enter (или кнопку отправки).

Шаг 4. Наблюдайте магию.

Claude прочитает ваш промпт и создаст файл index.html. Вы увидите, как появляется код — HTML-разметка, CSS-стили, всё в одном файле. Composer покажет вам превью изменений. Нажмите «Accept» (принять).

Шаг 5. Посмотрите результат.

Откройте файл index.html в браузере. В Cursor кликните правой кнопкой на файл → «Open with Live Server» (если установлено расширение Live Server). Или просто перетащите файл из папки в окно браузера.

Вы увидите стильный, адаптивный сайт-визитку. С анимациями, карточками, правильной типографикой. Ни одной строчки вы не написали вручную.

Шаг 6. Доработайте.

Теперь самое интересное — итерация. Вернитесь в Composer и напишите:

Измени визитку:

- Имя: [ваше настоящее имя]
- Добавь градиентный фон в шапку
- В секцию навыков добавь прогресс-бары с процентами
- Добавь кнопку "Скачать резюме"

Claude внесёт точечные изменения. Обновите браузер — и увидите обновлённый сайт.

Поздравляю. Вы только что создали свой первый проект как вайбкодер.

Обратите внимание на процесс: вы ни разу не задумались о синтаксисе HTML или CSS. Вы думали о том, **что** хотите увидеть, а не о том, **как** это сделать. Вы были продюсером, а AI — исполнителем. Это и есть суть вайбкодинга.

Итоги главы 3

В этой главе мы перешли от теории к практике:

— Установили и настроили Cursor — наш основной инструмент для вайбкодинга.

— Подключили Claude как модель по умолчанию.

— Разобрали интерфейс: боковая панель, центральная область, терминал, AI-панель.

— Изучили четыре режима работы: Tab, Chat, Composer, Agent — и когда использовать каждый.

— Узнали о Claude Code как альтернативном инструменте для продвинутых.

— Настроили редактор для комфортной работы.

— Создали первый проект — персональный сайт-визитку — без единой строчки кода.

Вы теперь вайбкодер. Начинающий, но уже вайбкодер. В следующей главе мы освоим ваш главный навык — промпт-инженерию. Именно от качества ваших промптов зависит, будет ли AI создавать то, что вам нужно, или что-то «примерно похожее».

Задание к главе: Возьмите сайт-визитку, которую мы создали, и проведите три итерации доработки через Composer. Например: (1) поменяйте цветовую схему, (2) добавьте секцию с портфолио из трёх карточек, (3) добавьте анимацию появления элементов при прокрутке. Каждый раз формулируйте запрос чётко и конкретно. Замечайте, как качество результата зависит от качества вашего промпта.

Часть II. ИСКУССТВО ПРОМПТИНГА

Глава 4. Основы промпт-инженерии

«Качество вашего вывода прямо пропорционально качеству вашего ввода.»
— Старейшее правило программирования, переосмысленное для эпохи AI

Если бы мне нужно было выбрать одну главу из этой книги, которую обязательно нужно прочитать, — это была бы она. Промпт-инженерия — это ваша суперсила как вайбкодера. Два человека могут использовать один и тот же инструмент (Cursor + Claude), но один получит блестящий результат, а другой — посредственный. Разница — в промптах.

4.1. Что такое промпт и почему он решает всё

Промпт — это текст, который вы пишете AI. Буквально «подсказка», «затравка». Но за этим простым словом скрывается целое искусство.

Вот аналогия. Представьте, что у вас есть волшебный джинн, который исполняет желания. Но джинн буквален — он делает ровно то, что вы попросили, а не то, что вы имели в виду.

«Хочу дом» — и вы получаете картонную коробку. Формально — дом.

«Хочу двухэтажный дом с тремя спальнями, кухней-гостиной, гаражом, на участке 10 соток, в стиле скандинавского минимализма, с панорамными окнами, выходящими на юг» — и вы получаете дом мечты.

AI работает так же. Чем точнее вы описываете, что хотите, тем ближе результат к вашему видению.

Но промпт-инженерия — это не только про «писать длинно». Иногда короткий промпт работает лучше длинного. Искусство в том, чтобы дать AI правильное количество правильной информации.

4.2. Анатомия хорошего промпта

Хороший промпт для генерации кода обычно содержит несколько компонентов. Не все из них обязательны в каждом случае, но знать о них полезно.

Роль / контекст. Кто AI в этом диалоге? Что он должен знать о проекте?

Пример: «Ты — опытный фронтенд-разработчик, специализирующийся на React и Next.js. Мы строим SaaS-платформу для управления задачами.»

Задача. Что конкретно нужно сделать?

Пример: «Создай компонент модального окна для подтверждения удаления задачи.»

Требования. Какие есть ограничения и условия?

Пример: «Используй Tailwind CSS для стилей. Компонент должен принимать props: isOpen, onConfirm, onCancel, taskTitle. Добавь анимацию появления. Должен быть доступен с клавиатуры (Escape для закрытия).»

Формат вывода. Как должен выглядеть ответ?

Пример: «Код одним файлом, TypeScript, с комментариями на русском.»

Примеры (опционально). Показать, что вы хотите, на примере.

Пример: «Вот похожий компонент в нашем проекте — придерживайся такого же стиля: [вставка кода]»

Давайте соберём всё вместе:

Ты — опытный React-разработчик. Мы строим SaaS для управления задачами

на Next.js + Tailwind CSS + TypeScript.

Создай компонент ConfirmDeleteModal.

Требования:
- Props: isOpen (boolean), onConfirm (function), onCancel (function), taskTitle (string)
- Tailwind CSS для стилей
- Анимация появления (fade + scale)
- Закрытие по Escape и клику вне модалки
- Иконка предупреждения (используй Lucide React)
- Кнопки "Отмена" (серая) и "Удалить" (красная)
- Доступность: aria-labels, фокус-менеджмент

Стиль кода: TypeScript, комментарии на русском, именование переменных на английском.

Этот промпт даст предсказуемый, качественный результат с первой попытки. Сравните с «Сделай модалку для удаления» — результат тоже будет, но потребуются три-четыре итерации доработки.

4.3. Пять золотых правил промптинга для кода

За год практики вайбкодинга я вывела пять правил, которые работают всегда.

Правило 1: Будьте конкретны, а не абстрактны.

Плохо: «Сделай красивую форму.»

Хорошо: «Сделай форму регистрации с полями: имя, email, пароль, подтверждение пароля. Валидация в реальном времени. Кнопка неактивна, пока все поля не заполнены корректно. Стиль: rounded-xl, тени, градиентная кнопка.»

Конкретика не отнимает время — она экономит его. Вы потратите 2 минуты на подробный промпт вместо 20 минут на переделки.

Правило 2: Одна задача — один промпт.

Плохо: «Создай лендинг, подключи базу данных, настрой аутентификацию и добавь платежи.»

Хорошо: Разбейте на четыре отдельных промпта и выполняйте последовательно.

AI лучше справляется с фокусированными задачами. Когда вы просите всё и сразу, качество каждого элемента падает. Разбивайте большие задачи на шаги — это главный навык вайбкодера.

Правило 3: Давайте контекст.

Плохо: «Добавь кнопку.»

Хорошо: «В компонент Header.tsx добавь кнопку "Выйти" в правом верхнем углу. При клике вызывай функцию handleLogout из контекста AuthContext. Стиль как у остальных кнопок в проекте.»

AI не видит вашего экрана. Он не знает, какой файл вы сейчас смотрите, если вы ему не сказали. Через Cursor можно прикрепить файлы к промпту с помощью @ — используйте это.

Правило 4: Указывайте технологии и стиль.

Плохо: «Сделай таблицу с данными.»

Хорошо: «Сделай таблицу с данными на React + Tailwind. Колонки: имя, email, дата регистрации, статус. Статус показывай цветным бейджем (зелёный — активен, красный — заблокирован). Сортировка по клику на заголовок. Пагинация по 10 строк.»

Если не указать технологию, AI может выбрать что-то неподходящее для вашего проекта — например, написать на Vue, хотя у вас React.

Правило 5: Итерируйте, не переписывайте.

Получили результат, который «почти то, что нужно»? Не начинайте заново. Уточните:

«Отлично, но внеси изменения:

— Сделай шрифт заголовка крупнее (2xl → 4xl)

— Поменяй цвет фона на slate-900

— Добавь hover-эффект на карточки»

Итерация — естественный процесс. Даже профессиональные разработчики не пишут идеальный код с первой попытки. Разница в том, что итерация через промпты занимает секунды, а не часы.

4.4. Примеры: плохой → хороший → отличный

Давайте пройдёмся по нескольким реальным сценариям и посмотрим, как формулировка промпта влияет на результат.

Сценарий 1: Создание навигации

Плохо:

Сделай навигацию для сайта.

Результат: базовая навигация без стилей, не адаптивная, без мобильного меню.

Хорошо:

Создай адаптивную навигацию для сайта на React + Tailwind.

Ссылки: Главная, О нас, Услуги, Контакты.

Мобильное меню-бургер. Фиксированная при прокрутке.

Результат: работающая навигация, но без «изюминки».

Отлично:

Создай компонент Navbar для нашего Next.js сайта.

Десктоп (ширина > 768px):

- Логотип слева (текстовый, "BrandName", font-bold text-xl)

- Ссылки по центру: Главная, О нас, Услуги, Блог, Контакты

- Кнопка "Начать" справа (градиент indigo-500 → purple-500, rounded-full)

- При прокрутке вниз: фон становится blur + полупрозрачным (backdrop-blur-md bg-white/80)

Мобайл (ширина ≤ 768px):

- Логотип слева, бургер-иконка справа

- По клику: выезжающее меню сбоку (slide from right) с анимацией

- Оверлей с затемнением

- Закрытие по клику вне меню и по Escape

Используй: React, Tailwind CSS, Lucide React для иконок.

Файл: components/Navbar.tsx

Результат: профессиональная, анимированная, адаптивная навигация с первого промпта.

Сценарий 2: Работа с данными

Плохо:

Выведи данные из API.

Хорошо:

Загрузи список пользователей из API /api/users и выведи в таблицу.

Отлично:
Создай компонент UsersTable.tsx:

1. Загрузка данных:
- Используй SWR для запроса к /api/users
- Покажи скелетон во время загрузки (3 строки-заглушки с анимацией pulse)
- При ошибке покажи сообщение с кнопкой "Попробовать снова"

2. Таблица:
- Колонки: Аватар (круглое фото 32px), Имя, Email, Роль, Дата регистрации, Действия
- Роль показывай бейджем: admin = фиолетовый, user = серый, moderator = синий
- Сортировка по клику на заголовок (иконка стрелки вверх/вниз)
- Поиск по имени и email (инпут над таблицей с debounce 300ms)

3. Действия:
- Кнопка "Редактировать" (иконка карандаша)
- Кнопка "Удалить" (иконка корзины, красная при hover)

Стек: React, Tailwind, SWR, Lucide React.
Результат: полнофункциональный компонент продакшн-качества.
Видите разницу? Промпт «отлично» длиннее в 10 раз, но он экономит 10 итераций доработки. Итоговое время — меньше.

4.5. Системные промпты и .cursorrules

Есть способ не повторять одни и те же инструкции в каждом промпте — системные промпты и файл .cursorrules.

Системный промпт — это инструкция, которая действует на протяжении всего разговора. В Cursor вы можете задать его в настройках: Settings → Rules for AI. Всё, что вы напишете там, Claude будет «помнить» в каждом диалоге.

Пример системного промпта для вайбкодера:
Ты — опытный fullstack-разработчик, помогающий новичку-вайбкодеру.

Правила:
- Пиши код на TypeScript
- Используй React + Next.js App Router
- Стили: только Tailwind CSS
- Комментарии в коде: на русском
- Именование переменных и функций: на английском, camelCase
- Компоненты: функциональные, с хуками
- Всегда добавляй обработку ошибок
- Всегда добавляй loading states
- При создании нового файла указывай полный путь

Теперь вам не нужно каждый раз указывать стек технологий и стиль — Claude будет следовать этим правилам автоматически.

Файл .cursorrules — это файл в корне вашего проекта, который Cursor автоматически подгружает в контекст. Он работает как «конституция» проекта — набор правил, которым AI следует при любом взаимодействии.

Создайте файл .cursorrules в корне проекта:

Правила проекта

Стек
- Next.js 14+ (App Router)
- TypeScript (strict mode)
- Tailwind CSS
- Supabase (база данных и аутентификация)
- Lucide React (иконки)

Структура проекта
- app/ — страницы и API routes
- components/ — переиспользуемые компоненты
- components/ui/ — базовые UI-компоненты (Button, Input, Modal...)
- lib/ — утилиты и хелперы
- types/ — TypeScript типы

Правила кода
- Все компоненты: именованный экспорт (не default)
- Обязательно: loading state и error handling
- Стили: только Tailwind, никакого inline CSS
- Формы: React Hook Form + Zod для валидации
- Запросы данных: Server Components где возможно

Запрещено
- Не использовать any в TypeScript
- Не использовать CSS-модули или styled-components
- Не создавать файлы больше 200 строк — разбивать на компоненты

Этот файл — ваш лучший друг. Один раз настроили — и AI всегда работает по вашим правилам.

4.6. Chain of Thought, Few-shot и другие техники

Несколько продвинутых техник, которые поднимут качество ваших промптов на новый уровень.

Chain of Thought (цепочка рассуждений).

Попросите AI думать пошагово, прежде чем писать код.

Прежде чем писать код, давай спланируем:
1. Какие компоненты нам понадобятся?
2. Какие данные нужны и откуда?
3. Как они будут взаимодействовать?

Потом напиши код, следуя этому плану.

Эта техника особенно полезна для сложных задач. Когда AI «думает вслух», он реже ошибается — как человек, который записывает план перед тем, как действовать.

Few-shot (обучение на примерах).

Покажите AI пример того, что хотите, и попросите сделать аналогичное.

Вот как выглядит наш компонент карточки:

[вставка существующего кода ProductCard.tsx]

Создай аналогичный компонент BlogCard.tsx в таком же стиле.

Данные: title, excerpt, author, date, coverImage, slug.

AI увидит стиль, конвенции, паттерны вашего проекта и создаст компонент, который идеально впишется.

Разбиение на подзадачи.

Большую задачу всегда лучше разбить. Вместо «Создай интернет-магазин» используйте последовательность:

1. «Создай структуру проекта Next.js для интернет-магазина. Пока без кода — только файлы и папки.»
2. «Создай модель данных: какие таблицы нужны в базе данных? Опиши схему.»
3. «Создай компонент карточки товара.»
4. «Создай страницу каталога с фильтрами.»
5. «Создай корзину.»
6. ...и так далее.

Каждый шаг — управляемый, проверяемый, отлаживаемый.

Негативные инструкции.

Иногда важнее сказать, чего НЕ делать:

Создай форму обратной связи.

НЕ используй:
- Никаких внешних библиотек для формы (только нативный React)
- Никаких alert() — покажи сообщение в UI
- Никаких console.log в финальном коде

4.7. Контекст через @ в Cursor

Одна из самых мощных фич Cursor — возможность прикреплять контекст к промпту через символ @.

@файл — прикрепить конкретный файл. AI прочитает его и учтёт при генерации.

Посмотри на @components/Button.tsx и создай аналогичный компонент Link в таком же стиле.

@папка — прикрепить всю папку. AI увидит структуру и содержимое.

Посмотри на @components/ui/ и добавь новый компонент Tooltip, который вписывается в эту систему.

@web — поиск в интернете. Cursor может искать актуальную документацию.

@web Как настроить Supabase Auth с Next.js App Router в 2026?

@docs — ссылка на документацию. Cursor проиндексирует сайт документации и использует его как справочник.

@codebase — поиск по всему проекту. Claude проанализирует вашу кодовую базу.

@codebase Где в проекте мы обрабатываем аутентификацию?

Эти инструменты решают одну из главных проблем — дать AI достаточно контекста. Чем больше AI знает о вашем проекте, тем точнее его ответы.

4.8. Шаблоны промптов: ваша первая библиотека

Вот набор шаблонов, которые вы будете использовать постоянно. Сохраните их и адаптируйте под свои проекты.

Шаблон: Новый компонент

Создай компонент [Название].

Назначение: [что делает].
Props: [перечислить с типами].
Стили: [описать внешний вид].
Поведение: [описать интерактивность].
Файл: [путь].
Шаблон: Исправление бага
В компоненте @[файл] есть баг:
[описание проблемы].

Ожидаемое поведение: [что должно происходить].
Фактическое поведение: [что происходит].
Шаги воспроизведения: [как повторить].

Исправь баг, не ломая остальную логику.
Шаблон: Новая страница
Создай страницу [название] по адресу [путь].

Секции:
1. [описание секции 1]
2. [описание секции 2]
3. [описание секции 3]

Данные: [откуда берутся, статические или из API].
Адаптивность: [как выглядит на мобиле].
Шаблон: Рефакторинг
Отрефактори @[файл]:

Проблемы:
- [проблема 1]
- [проблема 2]

Требования к рефакторингу:
- Разбей на [количество] компонентов
- Вынеси логику в кастомный хук
- Улучши типизацию
- Сохрани текущую функциональность
Шаблон: Ревью кода
Проведи код-ревью @[файл].

Проверь:
- Безопасность (XSS, инъекции)
- Производительность (лишние ре-рендеры, тяжёлые вычисления)
- Доступность (a11y)
- Чистота кода (дублирование, нейминг)
- Обработка ошибок

Предложи конкретные улучшения с примерами кода.

4.9. Практикум: 10 упражнений на промптинг

Теория без практики мертва. Вот 10 упражнений, которые стоит выполнить в Cursor прямо сейчас.

Упражнение 1: Конкретика. Создайте кнопку. Сначала с промптом «Сделай кнопку», потом с подробным описанием цвета, размера, анимации, состояний (hover, active, disabled). Сравните результаты.

Упражнение 2: Контекст. Создайте карточку товара. Укажите контекст: это карточка для интернет-магазина косметики. Обратите внимание, как контекст влияет на дизайнерские решения AI.

Упражнение 3: Итерация. Создайте форму обратной связи за 5 итераций. Первый промпт — базовая форма. Каждый следующий — одно улучшение: валидация, анимация, дизайн, отправка данных, сообщение об успехе.

Упражнение 4: Chain of Thought. Попросите Claude спланировать структуру блога перед написанием кода. Пусть сначала опишет компоненты, маршруты, модели данных — и только потом начнёт кодить.

Упражнение 5: Few-shot. Покажите Claude один компонент из вашего проекта и попросите создать другой в таком же стиле.

Упражнение 6: Негативные инструкции. Создайте навигацию, перечислив, чего НЕ должно быть: без jQuery, без CSS-модулей, без анимаций при загрузке.

Упражнение 7: Описание ошибки. Специально создайте баг (удалите скобку из кода) и попросите Claude исправить, описав симптомы, а не причину.

Упражнение 8: .cursorrules. Создайте файл правил для проекта и проверьте, что Claude следует им без напоминаний.

Упражнение 9: @-контекст. Используйте @файл и @папка, чтобы Claude создал новый компонент, идеально вписывающийся в существующий проект.

Упражнение 10: Код-ревью. Попросите Claude провести ревью кода, который он сам написал. Посмотрите, найдёт ли он проблемы в своей же работе (спойлер: найдёт).

Итоги главы 4

В этой главе вы освоили фундамент промпт-инженерии:

— Промпт состоит из роли, задачи, требований, формата и примеров.

— Пять золотых правил: конкретика, одна задача, контекст, технологии, итерации.

— Разница между плохим и отличным промптом — это разница между часами отладки и результатом с первой попытки.

— Системные промпты и .cursorrules экономят время на повторяющихся инструкциях.

— Chain of Thought, Few-shot, разбиение на подзадачи — техники для сложных случаев.

— @ в Cursor — мощный инструмент для управления контекстом.

Вы теперь умеете разговаривать с AI на его языке. В следующей главе мы добавим к этому навыку понимание веб-технологий — минимальный набор знаний, который сделает ваши промпты ещё точнее.

Задание к главе: Выполните все 10 упражнений из практикума. Не пропускайте ни одного — каждое тренирует отдельный аспект промптинга. Сохраните лучшие промпты в отдельный файл — это начало вашей библиотеки.

Глава 5. Веб-разработка глазами вайбкодера

«Не нужно уметь строить двигатель, чтобы водить машину. Но полезно знать, что бензин заливают в бак, а не в багажник.»

Эта глава — самая необычная в книге. Мы не будем учиться программировать. Мы будем учиться **понимать** — ровно настолько, чтобы ваши промпты стали точнее, а общение с AI — продуктивнее.

Представьте, что вы режиссёр фильма. Вам не нужно самому держать камеру, выставлять свет и монтировать. Но вам нужно знать, что такое «крупный план», «контровой свет», «обратная точка» — чтобы объяснить оператору, чего вы хотите. Эта глава даст вам словарь и понимание, чтобы «режиссировать» AI.

5.1. HTML, CSS, JavaScript — минимум, который нужно понимать

Три кита веб-разработки. Три технологии, которые стоят за каждым сайтом, который вы когда-либо видели. Вам не нужно знать их синтаксис наизусть — вам нужно понимать, что каждая из них делает.

HTML (HyperText Markup Language) — это скелет. Структура. Содержание.

HTML определяет, **что** находится на странице: заголовок, абзац текста, картинка, кнопка, форма, ссылка. Если бы веб-страница была газетой, HTML — это текст и фотографии, расположенные на полосе.

Вот как выглядит простейший HTML:

<h1>Привет, мир!</h1>
Это мой первый сайт.
<button>Нажми меня</button>

Угловые скобки < > — это «теги». Каждый тег описывает тип элемента: h1 — заголовок первого уровня, p — параграф, button — кнопка. Большинство тегов парные: <h1> открывает, </h1> закрывает.

Зачем вайбкодеру это знать? Чтобы в промпте написать «заверни это в <section> с классом hero» вместо «сделай эту часть как-то отдельно». Конкретика экономит итерации.

CSS (Cascading Style Sheets) — это внешний вид. Одежда.

CSS определяет, **как выглядит** содержимое: цвета, шрифты, размеры, отступы, анимации, расположение элементов. Один и тот же HTML может выглядеть как строгий корпоративный сайт или как яркая игровая страница — зависит от CSS.

h1 {
color: #6366f1;
font-size: 48px;
font-weight: bold;
}

Это говорит: «Все заголовки первого уровня — цвета индиго, размером 48 пикселей, жирным шрифтом».

Зачем вайбкодеру это знать? Чтобы в промпте сказать «уменьши padding до 16px и поменяй border-radius на rounded-2xl» вместо «сделай посимпатичнее». AI — буквальный исполнитель, и конкретные CSS-термины дают конкретный результат.

JavaScript (JS) — это поведение. Мозг.

JavaScript определяет, **что происходит**, когда пользователь что-то делает: нажал кнопку — появилось окно, заполнил форму — данные отправились на сервер, прокрутил страницу — запустилась анимация.

```
button.addEventListener('click', () => {
alert('Привет!');
});
```

Это говорит: «Когда на кнопку кликнут — покажи всплывающее окно с текстом "Привет!"»

Зачем вайбкодеру это знать? Чтобы понимать разницу между «статическим сайтом» (только HTML + CSS) и «интерактивным приложением» (с JavaScript). И чтобы формулировать промпты с учётом поведения: «При клике на карточку — открой модальное окно с деталями товара».

5.2. Как устроена веб-страница: аналогия с домом

Давайте соберём всё в одну картину. Веб-страница — это как дом.

HTML — это стены, комнаты и мебель. Планировка дома. Где кухня, где спальня, где входная дверь. Без HTML нет структуры — есть пустое поле.

CSS — это отделка, цвет стен, стиль мебели. Один и тот же дом может быть минималистичным лофтом или уютным кантри. CSS определяет атмосферу.

JavaScript — это электрика, водопровод и умный дом. Нажал выключатель — зажёгся свет. Открыл кран — потекла вода. Сказал «Алиса, включи музыку» — играет музыка. Без JavaScript сайт выглядит красиво, но ничего не делает.

Когда вы пишете промпт для AI, вы описываете дом: «Хочу гостиную (HTML) в скандинавском стиле (CSS) с умным освещением, которое включается по датчику движения (JavaScript)».

А теперь — подробнее о том, как современные сайты построены на практике.

5.3. React и Next.js: компоненты как кубики LEGO

Если бы HTML, CSS и JavaScript были кирпичами, досками и гвоздями, то **React** — это готовые модули: стеновые панели, оконные блоки, дверные проёмы. Собирать дом из модулей быстрее, чем из отдельных кирпичей.

React — это JavaScript-библиотека (инструмент) от Meta Platforms Inc. (Facebook (принадлежит Meta Platforms Inc., признанной экстремистской организацией и запрещённой в РФ)), которая позволяет строить интерфейсы из **компонентов**. Компонент — это самостоятельный кусок интерфейса: кнопка, карточка товара, навигация, модальное окно.

Вот компонент кнопки на React:

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

Этот компонент можно использовать в любом месте проекта сколько угодно раз:

```
<Button text="Купить" onClick={handleBuy} />
<Button text="Отменить" onClick={handleCancel} />
<Button text="Подписаться" onClick={handleSubscribe} />
```

Почему это важно для вайбкодера? Потому что компонентное мышление — это язык, на котором вы будете разговаривать с AI. Вместо «Сделай страницу» вы говорите: «Страница состоит из компонентов: Header, HeroSection, FeatureCards, Testimonials, Footer. Создай каждый отдельно.»

Next.js — это фреймворк поверх React. Если React — это движок, то Next.js — это готовый автомобиль. Он добавляет маршрутизацию (переход между страницами), серверный рендеринг (быстрая загрузка), API-маршруты (серверная логика) и множество оптимизаций. Большинство серьёзных проектов на React используют Next.js.

Для вайбкодера Next.js удобен тем, что у него чёткая структура: папка app/ — это страницы, components/ — компоненты, public/ — статические файлы. AI отлично знает эту структуру и генерирует код, который сразу ложится в нужные папки.

5.4. Tailwind CSS: стилизация через классы

Tailwind CSS — это подход к стилизации, который идеально подходит для вайбкодинга. Вместо того чтобы писать отдельные CSS-файлы, вы добавляете стили прямо в HTML/React через маленькие классы.

Традиционный CSS:

.button {
background-color: blue;
color: white;
padding: 8px 16px;
border-radius: 8px;
font-weight: bold;
}

Tailwind CSS:

<button class="bg-blue-500 text-white px-4 py-2 rounded-lg font-bold">

Одна строка вместо шести. И вот почему это важно для вайбкодера:

Во-первых, Tailwind-классы — это универсальный язык. Когда вы говорите AI «используй bg-gradient-to-r from-purple-500 to-pink-500», он точно знает, что вы имеете в виду. Никакой двусмысленности.

Во-вторых, классы легко менять в промпте. «Замени rounded-lg на rounded-full» — и кнопка из прямоугольной становится круглой. Это мгновенная итерация.

В-третьих, Tailwind — это система. Все цвета, размеры и отступы стандартизированы. p-4 — это всегда 16 пикселей отступа. text-xl — это всегда определённый размер текста. Нет хаоса «а тут отступ 17 пикселей, а тут 19».

Вот небольшой словарь Tailwind, который покроет 80% ваших потребностей:

Цвета: bg-blue-500 (фон), text-white (цвет текста), border-gray-200 (рамка). Цифра — интенсивность: 50 (почти белый) → 950 (почти чёрный).

Отступы: p-4 (padding, внутренний отступ), m-4 (margin, внешний отступ). px- только горизонтально, py- только вертикально. Цифра × 4 = пиксели: p-4 = 16px.

Размеры: w-full (ширина 100%), h-screen (высота экрана), max-w-4xl (максимальная ширина).

Текст: text-sm, text-lg, text-2xl (размер), font-bold (жирный), text-center (по центру).

Скругления: rounded (маленькое), rounded-lg (среднее), rounded-full (круг).

Flexbox: flex (включить), flex-col (вертикально), items-center (по центру по вертикали), justify-between (распределить по горизонтали).

Вам не нужно это заучивать. Но когда вы видите flex items-center gap-4 в коде, полезно понимать, что это значит: «элементы в ряд, выровнены по центру, с расстоянием между ними».

5.5. Базы данных и API: откуда берутся данные

До сих пор мы говорили о том, что видит пользователь — фронтенде. Но большинство приложений работают с данными: пользователи, товары, заказы, посты. Эти данные где-то хранятся. Давайте разберёмся где.

База данных — это организованное хранилище данных. Представьте таблицу в Excel: строки — это записи (пользователи), столбцы — это поля (имя, email, дата регистрации). Только база данных может хранить миллионы записей и мгновенно находить нужную.

Для вайбкодера самый простой способ работать с базой данных — **Supabase**. Это сервис, который даёт вам готовую базу данных, аутентификацию (логин/регистрация) и хранилище файлов — бесплатно для небольших проектов. Вы создаёте таблицу через визуальный интерфейс (как в Excel), а Supabase автоматически генерирует API для работы с данными.

API (Application Programming Interface) — это «меню ресторана». Вы (фронтенд) не идёте на кухню (сервер/база данных) сами. Вы смотрите в меню (API), выбираете блюдо (запрос) и получаете заказ (данные).

Пример: ваше приложение хочет показать список товаров. Фронтенд отправляет запрос на API: «Дай мне товары из категории "Электроника"». Сервер идёт в базу данных, находит нужные товары и отправляет их обратно в формате JSON — структурированных данных, которые фронтенд может красиво отобразить.

Зачем вайбкодеру это знать? Чтобы в промпте написать: «Создай страницу каталога. Данные о товарах загружай из Supabase, таблица products. Показывай: название, цену, изображение. Добавь фильтр по категории через API-запрос.» Без понимания этих концепций вы бы написали: «Сделай каталог товаров» — и получили бы страницу с захардкоженными (вшитыми в код) данными, которые невозможно менять.

5.6. Не нужно запоминать — нужно понимать, что спросить у AI

Вот ключевой сдвиг мышления, который отличает вайбкодера от студента курсов программирования.

Традиционное обучение: запомнить синтаксис → написать код → отладить → запомнить ещё.

Вайбкодинг: понять концепцию → сформулировать задачу → попросить AI → проверить результат.

Вам не нужно помнить, как написать цикл for на JavaScript. Вам нужно знать, что цикл существует и зачем он нужен — «повторить действие N раз». Остальное AI напишет за вас.

Вам не нужно помнить, как подключить Supabase к Next.js. Вам нужно знать, что Supabase — это база данных, которая хорошо работает с Next.js. Остальное — в промпте.

Вам не нужно помнить синтаксис Tailwind-классов. Вам нужно знать, что flex — это способ расположить элементы в ряд, а grid — в сетку. Какой именно класс — подскажет AI.

Это как знание иностранного языка. Вам не нужен словарный запас в 50 000 слов. Вам нужно знать 500 ключевых слов и уметь пользоваться переводчиком. AI — это ваш переводчик с «языка идей» на «язык кода».

Вот список концепций, которые полезно понимать на уровне «что это и зачем» (без синтаксиса):

— **Переменная** — ячейка для хранения данных. Как коробка с подписью.

— **Функция** — набор действий, который можно вызвать по имени. Как рецепт.

— **Условие** — «если это, то сделай то». Как развилка на дороге.

— **Цикл** — «повтори N раз» или «для каждого элемента списка».

— **Массив** — список чего-то: список пользователей, список товаров.
— **Объект** — набор свойств: у пользователя есть имя, email, возраст.
— **Компонент** — самостоятельный кусок интерфейса.
— **Пропсы (props)** — данные, которые передаются в компонент снаружи.
— **Состояние (state)** — данные, которые могут меняться внутри компонента.
— **API-запрос** — обращение к серверу за данными.
— **Асинхронность** — операция, которая занимает время (загрузка данных), и код не ждёт её окончания, а продолжает работать.

Этих 11 концепций хватит для 80% ваших промптов. Остальное подхватите по ходу работы.

5.7. Практикум: лендинг за 20 минут через промпты

Давайте соединим всё, что мы знаем, и создадим реальный проект — лендинг для вымышленной кофейни. Это будет репетиция перед большими проектами в следующих главах.

Шаг 1. Инициализация проекта (2 минуты)

Откройте Composer в Cursor и напишите:

Создай новый проект Next.js с Tailwind CSS.
Команды для терминала — я выполню их сам.
Имя проекта: coffee-landing.

Claude даст вам команды. Выполните их в терминале. Через минуту у вас готовый проект.

Шаг 2. Главный экран — Hero (5 минут)

Создай компонент HeroSection для лендинга кофейни "Бодрый Боб".

- Полноэкранная секция (h-screen)
- Фоновое изображение (используй placeholder от Unsplash: кофейня, тёплые тона)
- Затемнение поверх фото (overlay bg-black/50)
- По центру: заголовок "Бодрый Боб" (text-6xl, font-serif, белый)

Конец ознакомительного фрагмента.