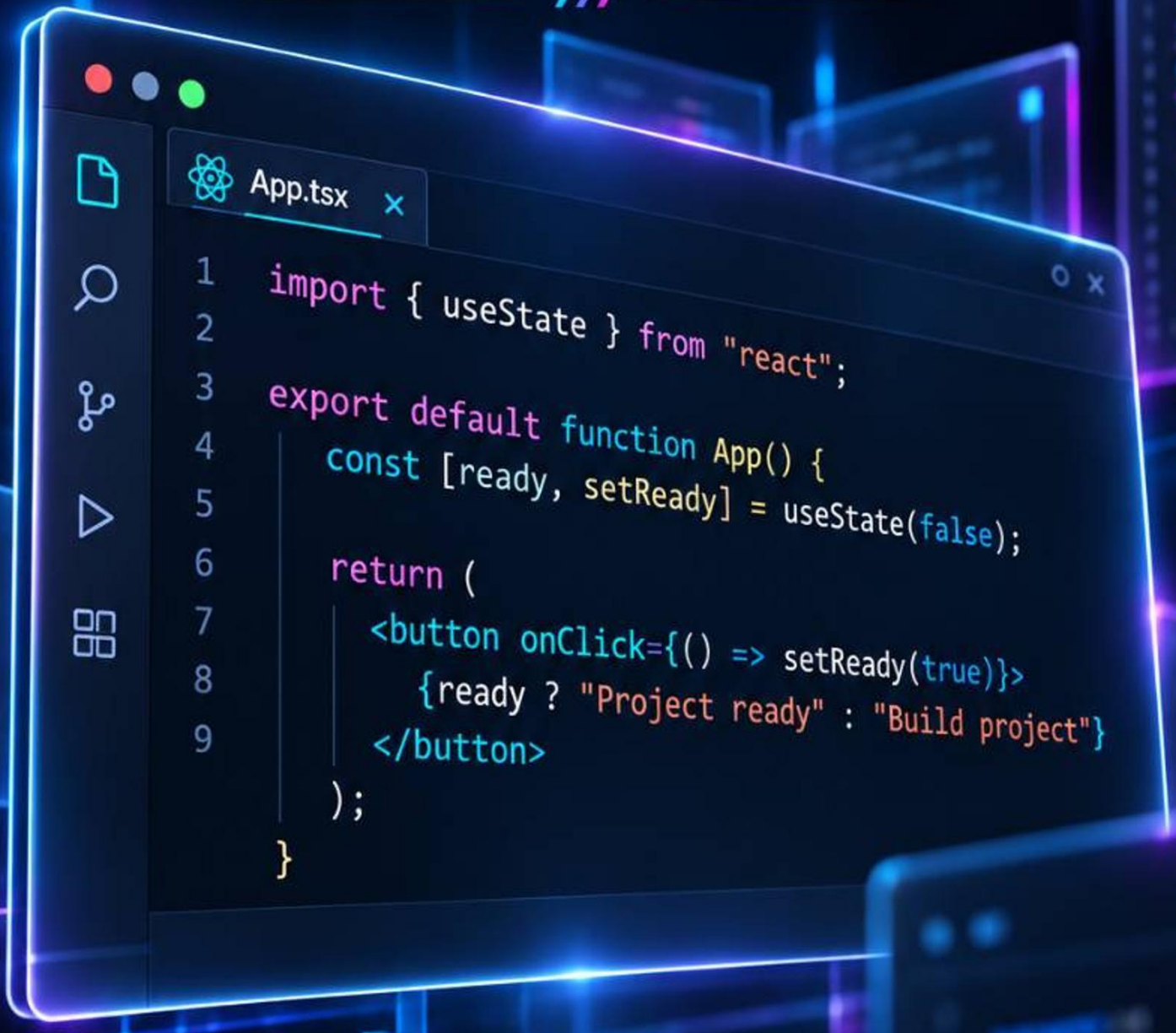
КОМФОРТНЫЙ ВАЙБ-КОДИНГ ДЛЯ НОВИЧКОВ



18+

САБИР АЛМАСОВ

Сабир Алмасов

Комфортный вайб-кодинг для новичков

«Автор»
2026

Алмасов С.

Комфортный вайб-кодинг для новичков / С. Алмасов — «Автор», 2026

Эта книга поможет начать создавать профессиональные сайты, телеграм ботов, мобильные приложения и SaaS сервисы без необходимости месяцами изучать языки программирования.Шаг за шагом вы освоите Cursor AI и соберёте пять настоящих проектов: современный сайт, полезного Telegram-бота, веб-приложение с личным кабинетом, мобильное приложение и собственный микро-SaaS. Внутри — 70 готовых промптов, которые можно копировать и использовать в своих проектах.Вы научитесь ставить задачи ИИ, проверять результат, находить ошибки, безопасно отменять неудачные изменения, работать с Git, базами данных, авторизацией и публикацией. Отдельная часть посвящена заработку: какие услуги можно предлагать клиентам, где искать первые заказы и за какие проекты новичку лучше не браться.Без обещаний быстрых и лёгких денег. Только понятный рабочий процесс, который превращает идею в реальный цифровой продукт, который решает проблемы людей и приносит доход своему владельцу.

Содержание

Сабир Алмасов
Комфортный вайб-кодинг для новичков

Комфортный вайб-кодинг для новичков

Практическая книга о создании сайтов, ботов, приложений и собственных цифровых продуктов в Cursor AI

Введение

Ещё недавно путь от идеи до работающего приложения начинался с нескольких месяцев изучения синтаксиса. Человек осваивал переменные, циклы, функции, затем выбирал фреймворк, настраивал среду и только после этого собирал что-то, что можно показать другому человеку. Сейчас этот порядок изменился. Cursor Ai способен прочитать проект, предложить план, создать несколько связанных файлов, запустить команды, найти ошибку и проверить результат. Новичок может начать не с заучивания конструкций языка, а с понятной задачи.

Но это не значит, что освоив вайб-кодинг вы сможете моментально разбогатеть создав приложение, которое вам кажется необходимым рынку. ИИ пишет код быстро, но так же быстро может построить неудобный интерфейс, подключить лишнюю библиотеку, забыть проверку доступа или исправить одну ошибку ценой трёх новых. Если пользователь не понимает, что происходит в проекте, скорость только ускоряет движение в неверную сторону.

Вайб-кодинг в этой книге означает практический способ разработки, при котором вы описываете желаемый результат обычным языком, а Cursor помогает превратить описание в строки кода, рабочий интерфейс, красивый современный дизайн и т.д. Нельзя всё целиком и полностью поручить ИИ. Вы всё-равно выбираете задачу, проверяете работу, принимаете решения и сохраняете рабочие версии. Cursor берёт на себя значительную часть рутинной технической работы.

Мы начнём с пустой папки и небольшого сайта. Затем намеренно столкнёмся с ошибками, научимся пользоваться Git, создадим Telegram-бота, веб-приложение, мобильное приложение и микро-SaaS. Технические понятия будут появляться тогда, когда без них нельзя сделать следующий шаг, но все непонятные новичку слова и действия мы будем разбирать простым языком. Вам не придётся заранее учить отдельный словарь.

Основная ценность книги — рабочий процесс. Конкретная кнопка может переехать после обновления Cursor, название режима может измениться, а популярная модель уступит место новой. Но основа вайб-кодинга ближайшие несколько лет останется неизменной: сначала уточнить задачу, затем изучить проект, составить короткий план, начать разработку, запустить проверку и сохранить рабочее состояние.

По состоянию на сентябрь 2026 года Cursor развивает две связанные среды: классический редактор с файлами, терминалом и панелью Agent, а также ориентированное на агентов рабочее пространство, где задачи могут выполняться локально, в изолированных рабочих копиях и в облаке. В книге основной путь построен вокруг локального проекта в редакторе. Он проще для первого знакомства и позволяет видеть каждый файл. Возможности облачных агентов, браузерного тестирования, правил, навыков и внешних подключений рассматриваются позже, когда они действительно приносят пользу.

Не пытайтесь запомнить всё при первом чтении. Откройте Cursor и повторяйте действия. После каждой крупной части у вас будет работающий результат. Сломанный проект тоже считается полезным результатом, если вы научились находить причину и возвращать рабочую версию.

Что вам понадобится?

Нужен обычный компьютер на Windows, macOS или Linux, стабильный интернет и возможность устанавливать программы. Для публикации проектов потребуются учётные записи в нескольких сервисах. Их тарифы и условия меняются, поэтому книга не привязана к конкретным ценам. Начинайте с бесплатных возможностей, а платные подключайте только тогда, когда понимаете, зачем они нужны.

Создайте отдельную папку для проектов книги. Не храните её в случайной папке загрузок. Подойдут, например, C:\projects\vibe-book в Windows или ~/projects/vibe-book в macOS и Linux. Внутри каждый проект получит собственную папку.

Ещё одно правило действует с первой страницы: не вставляйте в чат пароли, платёжные данные и действующие секретные ключи. Позже мы настроим файлы .env, но запомнить это надо сейчас.

Часть первая. Первые часы с Cursor

Установка

Скачивайте Cursor с официального сайта. Выберите версию для своей операционной системы, установите программу и войдите в учётную запись. Если установщик предлагает импортировать настройки из другого редактора, а вы никогда редакторами кода не пользовались, пропустите импорт. Чужие сочетания клавиш и расширения на старте только создадут дополнительные неудобства.

После запуска вы увидите знакомые элементы любого редактора: дерево файлов, центральную область, нижнюю панель с терминалом и боковую панель Agent. Расположение может меняться между версиями, поэтому ищите не конкретную иконку, а назначение элемента.

Дерево файлов показывает содержимое открытой папки. Центральная область открывает выбранный файл. Терминал выполняет команды в папке проекта. Agent получает задачу обычным языком, изучает нужные файлы, редактирует их и при необходимости запускает команды. В актуальных версиях Agent также умеет искать по проекту и в интернете, использовать браузер для проверки интерфейса и работать с подключёнными инструментами. Набор доступных действий зависит от версии, настроек и тарифа.

Создайте папку first-session, откройте её через команду открытия папки и убедитесь, что в дереве файлов пока пусто. Cursor считает открытую папку границей проекта. Если открыть слишком широкую папку, например весь каталог документов, Agent увидит много лишнего. Если открыть только вложенную папку с картинками, он не увидит код рядом. Привычка открывать корень конкретного проекта избавляет от множества странных ответов.

Первое безопасное поручение

Не начинайте с команды «создай приложение». Сначала попросите Agent объяснить рабочую область и предложить минимальное действие.

Промпт 1. Знакомство с пустым проектом

Я впервые работаю в Cursor и раньше не программировал. Сейчас открыта пустая папка проекта.

Сначала коротко объясни, какие файлы понадобятся для простой локальной веб-страницы.

Затем предложи план из 3–5 шагов. Пока ничего не создавай и не выполняй команды.

Все технические термины объясняй простыми словами при первом упоминании.

Последняя фраза «пока ничего не создавай» важнее, чем кажется. Agent может действовать, а не только отвечать. Когда вы хотите сначала разобраться, ограничьте действие явно.

Теперь разрешите создать минимальную страницу.

Промпт 2. Первые файлы

Создай в этой папке минимальную веб-страницу из трёх файлов: index.html, styles.css и script.js.

На странице должны быть заголовок «Мой первый проект», короткий текст и кнопка.

При нажатии кнопка меняет текст под ней.

Не используй библиотеки и сборщики. После создания перечисли файлы и объясни роль каждого одним абзацем.

HTML описывает содержание страницы, CSS отвечает за внешний вид, JavaScript — за поведение. Сейчас этого объяснения достаточно. Откройте index.html в браузере. В некоторых версиях Cursor Agent может сам запустить локальный просмотр и проверить страницу встроенным браузером. Если такой инструмент доступен, попросите его об этом. Если нет, откройте файл через проводник.

Посмотрите изменения до того, как продолжить. Agent обычно показывает, какие файлы созданы или изменены. Не нажимайте бездумно «принять всё», если не понимаете масштаба. Для трёх коротких файлов допустимо просмотреть каждый целиком. В большом проекте вы будете изучать сводку, список файлов и ключевые участки изменений.

Как
Cursor
видит проект

Для вас проект может быть «сайтом для записи клиентов». Для Cursor это папка, файлы, зависимости, настройки и история изменений. Чем точнее эти материалы описывают продукт, тем меньше Agent вынужден угадывать.

Контекст — это информация, доступная модели в текущей работе: ваш запрос, выбранные файлы, найденные фрагменты кода, правила проекта, вывод терминала и предыдущая переписка. Контекст не бесконечен. Если отправить длинную беседу, десятки файлов и несколько разных задач, важная деталь может потеряться среди второстепенных.

Хороший контекст не обязательно большой. Для изменения формы достаточно показать компонент формы, обработчик отправки и описание желаемого поведения. Весь проект нужен тогда, когда изменение затрагивает архитектуру, общие стили, базу или авторизацию.

В интерфейсе можно прикреплять файлы и другие источники через упоминания и элементы добавления контекста. Agent также умеет искать нужные места сам. Новичку полезно сочетать оба подхода: назвать известный файл и попросить найти остальные связанные части.

Промпт 3. Изучение проекта перед изменением

Изучи текущий проект, но пока не меняй файлы.

Определи:

1. с какого файла запускается страница;

2. где находятся стили;

3. какой код обрабатывает нажатие кнопки;

4. какие файлы связаны с этим поведением.

После анализа предложи одно небольшое улучшение, которое не усложнит проект.

Этот запрос формирует полезную привычку: сначала карта, затем движение. Если Agent неверно описал простой проект, не стоит поручать ему сложное изменение. Уточните, что он пропустил.

Поиск по кодовой базе

Кодовая база — все файлы, из которых собирается продукт. В небольшом сайте вы найдёте нужную строку обычным поиском. В приложении с сотнями файлов важнее смысловой поиск: «где проверяется право пользователя удалить запись» или «какой компонент показывает карточку заказа». Agent способен искать имена файлов, точные слова и смысловые связи.

Промпт 4. Поиск нужного места

Найди все места в проекте, связанные с [ФУНКЦИЯ ИЛИ ЭЛЕМЕНТ].

Не вноси изменения.

Покажи путь каждого важного файла и объясни его роль.

Отдельно укажи, какие файлы точно придётся менять, а какие нужно только учитывать.

Когда вы не знаете название функции, описывайте наблюдаемое поведение: «кнопка, которая открывает окно входа», «запрос, загружающий список клиентов», «проверка, запрещающая пустое имя».

Agent
и режим планирования

Agent подходит для задач, где нужно читать и изменять файлы, запускать команды и проверять результат. Режим планирования нужен перед заметным изменением. В нём Cursor исследует кодовую базу, может задать уточняющие вопросы и формирует план, который вы проверяете до редактирования.

Название и расположение переключателя могут меняться. Ориентируйтесь на принцип: если задача затрагивает несколько частей проекта или вы сами ещё не понимаете решение, сначала план. Для замены текста на кнопке отдельный план не нужен. Для добавления регистрации, новой таблицы базы и личного кабинета — нужен.

Промпт 5. План без реализации

Нужно добавить в проект [ФУНКЦИЯ].

Сначала изучи существующую структуру и подготовь план реализации.

В плане укажи:

- какие файлы изменятся;

- какие новые файлы появятся;

- какие данные будут передаваться;

- как проверить результат;

- основные риски для уже работающих функций.

Не пиши код и не выполняй команды до моего подтверждения плана.

План проверяют не по красоте формулировок. Спросите себя: Agent понял задачу? Не предлагает ли переписать половину проекта? Есть ли способ проверить результат? Не хранится ли секрет в коде? Если ответ вызывает сомнения, попросите упростить план.

Терминал без страха и без слепого доверия

Терминал — строка, через которую программы получают команды. В разработке через него устанавливают зависимости, запускают локальный сервер, выполняют тесты и собирают проект. Вам не нужно запоминать сотни команд. Нужно понимать назначение команды до запуска.

Команда npm install обычно устанавливает зависимости JavaScript-проекта, перечисленные в его настройках. npm run dev запускает режим разработки. npm run build проверяет, способен ли проект собраться для публикации. Значение конкретной команды хранится в файле package.json, поэтому одинаковое имя может выполнять разные действия в разных проектах.

Agent умеет запускать команды сам. Это экономит время, но требует границ. Установка известной библиотеки в учебном проекте и удаление папки — действия разного риска.

Промпт 6. Объяснение команды до выполнения

Ты предлагаешь выполнить команду:

[КОМАНДА]

Пока не запускай её. Объясни простыми словами:

1. что она сделает;

2. какие файлы или данные может изменить;

3. можно ли безопасно отменить результат;

4. почему команда нужна именно сейчас.

Не разрешайте запуск команды, если объяснение остаётся туманным. Особенно внимательно относитесь к массовому удалению, принудительному перезаписыванию Git-истории, изменению прав доступа и командам, скопированным с неизвестного сайта.

Зависимости и менеджер пакетов

Зависимость — готовый программный модуль, который проект использует вместо повторного написания стандартной функции. Менеджер пакетов скачивает такие модули и фиксирует их версии. В JavaScript-проектах часто применяется npm, pnpm или yarn. На старте выберите один инструмент и не смешивайте файлы блокировки разных менеджеров без причины.

Больше зависимостей не означает лучше. Каждая библиотека добавляет обновления, возможные уязвимости и несовместимости. Просите Agent объяснить, зачем нужен новый пакет.

Промпт 7. Проверка необходимости зависимости

Перед установкой пакета [НАЗВАНИЕ ПАКЕТА] проверь текущий проект.

Объясни, какую конкретную задачу решает пакет, есть ли уже установленное средство с такой возможностью и можно ли обойтись без новой зависимости.

Если пакет нужен, предложи команду установки, но не выполняй её.

Изменение, проверка, следующий шаг

Комфортная работа с Cursor строится короткими циклами. Вы ставите одну проверяемую задачу. Agent изучает нужные файлы и вносит изменение. Затем запускаются проверки. Вы смотрите результат в браузере или приложении. Только после этого начинается следующий шаг.

Плохой запрос выглядит так:

Сделай современный сайт, добавь регистрацию, оплату, чат, админку и мобильное приложение.

Он не задаёт аудиторию, сценарий, ограничения и критерии готовности. Agent вынужден выбрать всё сам и создаёт много кода, который трудно проверить.

Улучшенный запрос:

Промпт 8. Одно ограниченное изменение

В существующей странице добавь форму заявки с полями «Имя» и «Телефон».

Требования:

- оба поля обязательны;

- под пустым полем показывается понятная ошибка;

- после корректной отправки появляется сообщение «Заявка принята»;

- данные пока никуда не отправляются и хранятся только до обновления страницы;

- существующий дизайн нужно сохранить.

Сначала назови файлы, которые изменишь. После изменения проверь сценарии с пустыми и заполненными полями.

Здесь известны границы и результат. Позже форму можно подключить к серверу, но текущий этап не маскируется под готовую систему.

Правила проекта

Повторять в каждом запросе «не меняй стек», «объясняй новичку», «проверяй сборку» неудобно. Для постоянных инструкций Cursor поддерживает правила пользователя, команды и проекта, а также файл AGENTS.md. Проектные правила обычно хранятся рядом с кодом, поэтому их можно включать в Git и передавать вместе с проектом.

Правило должно быть коротким, проверяемым и связанным с проектом. Фраза «всегда пиши идеальный код» ничего не меняет. Инструкция «после изменения TypeScript-файлов запускай проверку типов и сборку» задаёт наблюдаемое действие.

Промпт 9. Подготовка правил проекта

Изучи текущий проект и предложи короткий набор проектных правил для Cursor.

Учти:

- используемый стек и менеджер пакетов;

- запрет менять технологии без отдельного согласования;

- сохранение существующего стиля кода;

- обязательную проверку после изменений;

- запрет помещать секреты в код;

- требование объяснять мне новые технические решения простыми словами.

Не создавай файл автоматически. Сначала покажи текст правил и объясни каждое правило.

После проверки сохраните правила в поддерживаемом вашей версией формате. Если используете AGENTS.md, располагайте его в корне проекта. Для больших репозиториев могут применяться правила с более узкой областью. Не копируйте огромный универсальный набор из интернета: лишние инструкции конкурируют с задачей.

Как выбирать модель

Список моделей и режимы маршрутизации меняются слишком быстро, чтобы строить вокруг них учебный процесс. Практическое правило проще. Для маленького понятного изменения подходит быстрый вариант. Для архитектуры, сложной ошибки и анализа нескольких файлов нужен режим, который лучше рассуждает и удерживает контекст. Автоматический выбор удобен, пока вы не видите систематической проблемы.

Если результат слабый, сначала улучшите задачу и контекст. Смена модели не исправит противоречивое техническое задание. Не запускайте одну и ту же расплывчатую фразу через пять моделей в надежде на чудо.

Часть вторая. Учимся ставить задачи

Из идеи в техническое задание

Фраза «хочу сервис для клиентов» не является задачей разработки. До кода нужно определить пользователя, его действие и результат. Техническое задание для небольшого проекта не обязано занимать пятьдесят страниц. Оно должно снять главные неопределённости.

Минимальный набор вопросов:

- Кто пользуется продуктом?
- Какую проблему он решает?
- Какое одно действие составляет основную ценность?
- Какие данные вводятся и что получается на выходе?
- Нужны ли учётные записи?
- Что входит в первую версию, а что откладывается?
- По каким сценариям вы поймёте, что всё работает?

Промпт 10. Интервью по идее

Помоги превратить идею в техническое задание.

Идея: [ОПИШИТЕ ИДЕЮ]

Предполагаемый пользователь: [КТО БУДЕТ ПОЛЬЗОВАТЬСЯ]

Сначала задай мне не больше 10 вопросов, без которых нельзя определить первую рабочую версию.

Не предлагай функции до моих ответов. Избегай профессионального жаргона или сразу объясняй его.

Промпт 11. Короткое техническое задание

На основе моих ответов составь техническое задание для первой версии проекта [НАЗВАНИЕ].

Структура:

1. цель продукта;

2. пользователи;

3. главный пользовательский сценарий;

4. функции первой версии;

5. что явно не входит в первую версию;

6. данные и правила их обработки;

7. критерии готовности;

8. основные риски и неизвестные вопросы.

Пиши понятным русским языком. Не выбирай технологии, пока требования не сформулированы.

Проверка идеи без выдуманной уверенности

ИИ не знает, заплатят ли люди за вашу идею. Он может найти аналоги, сформулировать предположения и показать слабые места. Решение подтверждается разговорами с потенциальными пользователями, заявками, предзаказами или реальным использованием.

Промпт 12. Критический анализ идеи

Проведи критический анализ идеи: [ИДЕЯ].

Целевая аудитория: [АУДИТОРИЯ].

Регион: [СТРАНА ИЛИ РЫНОК].

Раздели ответ на:

- какую конкретную проблему решает продукт;

- как люди решают её сейчас;

- почему они могут не перейти на новый продукт;

- какие предположения нужно проверить до разработки;

- самый дешёвый способ проверить интерес;

- признаки того, что идею стоит изменить или закрыть.

Не придумывай статистику и не изображай спрос как доказанный.

Если Cursor имеет доступ к веб-поиску, попросите его дать ссылки и даты. Для важных бизнес-решений открывайте первоисточники сами. Сводка агента ускоряет исследование, но не заменяет проверку.

Как сократить замысел до
MVP

MVP — первая версия, которая решает одну полезную задачу достаточно хорошо, чтобы её попробовал реальный пользователь. Это не сломанная демоверсия и не макет всех будущих функций.

Допустим, вы хотите сервис записи для мастеров. В голове быстро появляются календарь, приложение клиента, онлайн-оплата, чат, программа лояльности, аналитика, рассылки и реферальная система. Первая версия может состоять из страницы выбора времени, подтверждения записи и простого кабинета мастера. Если никто не хочет пользоваться этой основой, дополнительные модули не спасут продукт.

Промпт 13. Сокращение идеи

Сократи идею до MVP, который можно разработать одному человеку:

[ПОЛНОЕ ОПИСАНИЕ ИДЕИ]

Определи:

1. одну главную проблему;

2. одну основную группу пользователей;

3. один главный сценарий;

4. минимальный набор функций для завершения сценария;

5. функции, которые нужно отложить;

6. простую проверку ценности до сложной разработки.

Если идея всё ещё слишком велика, сократи её ещё раз.

Промпт 14. Главная функция продукта

Для продукта [ОПИСАНИЕ] найди функцию, ради которой пользователь вернётся второй раз.

Объясни, какой результат она даёт и что должно произойти от первого действия пользователя до этого результата.

Не добавляй регистрацию, оплату, рекомендации и социальные функции, если они не нужны для основного результата.

Архитектура человеческим языком

Архитектура — способ разделить продукт на части и определить их связи. Для сайта-визитки архитектура почти незаметна. В веб-приложении появляются интерфейс, серверная логика, база данных и внешние сервисы.

Frontend — часть, которую видит пользователь: страницы, кнопки, формы. Backend принимает запросы, проверяет правила, работает с базой и возвращает результат. API — согласованный способ обмена данными между частями. База данных хранит записи после закрытия приложения. Сервер — компьютер или облачная среда, где backend работает постоянно.

Эти части не обязаны быть отдельными программами. Современный веб-фреймворк способен объединять интерфейс и серверные функции в одном проекте. Для новичка это часто проще, чем поддерживать два репозитория.

Промпт 15. Выбор простой архитектуры

Предложи архитектуру первой версии проекта по этому техническому заданию:

[ТЕХНИЧЕСКОЕ ЗАДАНИЕ]

Условия:

- разработчик один и пока новичок;

- нужно минимизировать количество отдельных сервисов;

- технологии должны быть популярными и хорошо документированными;

- решение должно поддерживать дальнейшее развитие без преждевременного усложнения.

Объясни схему человеческим языком: интерфейс, серверная логика, база, внешние сервисы и путь данных.

Предложи один основной вариант и только один запасной, если у него есть реальное преимущество.

Не соглашайтесь на архитектуру из десяти сервисов, очередей сообщений и контейнерного оркестратора для приложения с двадцатью пользователями. Масштабирование решают после появления нагрузки, а не вместо запуска.

Разбивка работы на этапы

Большая задача опасна не потому, что Agent не способен написать много кода. Опасность в количестве непроверенных решений. Если после одного запроса изменились сорок файлов, найти причину ошибки сложнее.

Хороший этап заканчивается наблюдаемым результатом: «форма открывается и проверяет поля», «запись сохраняется в базе», «пользователь видит только свои проекты». Этап «сделать backend» слишком широк.

Промпт 16. План разработки по вертикальным срезам

Разбей разработку проекта на маленькие этапы.

Техническое задание: [ТЗ]

Архитектура: [АРХИТЕКТУРА]

Каждый этап должен:

- давать проверяемый пользовательский результат;

- затрагивать ограниченное количество файлов;

- содержать способ проверки;

- завершаться рабочим состоянием проекта.

Сначала сделай минимальный сквозной сценарий от интерфейса до сохранения данных, затем расширяй его. Не реализуй этапы, только составь план.

Вертикальный срез — маленькая законченная функция через все нужные слои. Например, пользователь вводит название проекта, backend проверяет значение, база сохраняет запись, интерфейс показывает её. После такого этапа продукт умеет мало, но путь работает целиком.

Критерии готовности

«Выглядит нормально» не позволяет проверить задачу. Критерий готовности описывает наблюдаемое поведение.

Для формы записи критерии могут быть такими:

• пустое имя вызывает сообщение рядом с полем;
• телефон с недопустимыми символами не отправляется;
• корректная заявка появляется в базе;
• повторное нажатие не создаёт две записи;
• на узком экране форма не выходит за границы;
• пользователь получает подтверждение.

Промпт 17. Критерии приёмки

Составь критерии готовности для функции [ФУНКЦИЯ].

Описание: [ТРЕБОВАНИЯ].

Включи:

- обычный успешный сценарий;

- пустые и неверные данные;

- повторные действия пользователя;

- поведение при недоступном сервере или внешнем API;

- мобильный экран, если есть интерфейс;

- проверку прав доступа, если функция работает с личными данными.

Каждый критерий должен быть проверяемым действием, а не общей фразой о качестве.

Запрос на реализацию

Когда план проверен, поручение должно содержать контекст, границы, требования и проверку. Не нужно писать роман. Достаточно убрать двусмысленность.

Промпт 18. Реализация этапа

Реализуй только этап [НОМЕР И НАЗВАНИЕ] из утверждённого плана.

Требования этапа:

[ТРЕБОВАНИЯ]

Перед изменениями:

1. изучи связанные файлы;

2. перечисли файлы, которые планируешь менять;

3. предупреди, если обнаружишь противоречие с текущей архитектурой.

После изменений:

- запусти доступные проверки;

- сообщи, что проверено автоматически;

- дай мне короткий ручной сценарий проверки;

- не переходи к следующему этапу.

Как проверять работу

Agent

Новичок часто смотрит только на красивую страницу. Проверка должна охватывать четыре уровня.

Первый — видимый сценарий. Нажмите кнопки, заполните форму неверно, обновите страницу. Второй — терминал. Нет ли красных ошибок при запуске и сборке? Третий — данные. Создалась ли нужная запись, принадлежит ли она правильному пользователю? Четвёртый — изменения. Не удалил ли Agent рабочую функцию и не добавил ли секрет в публичный файл?

Промпт 19. Проверка результата

Проверь реализацию функции [ФУНКЦИЯ] по требованиям:

[ТРЕБОВАНИЯ]

Сначала изучи фактические изменения. Затем:

1. запусти подходящие автоматические проверки;

2. проверь сборку;

3. перечисли непроверенные вручную сценарии;

4. найди изменения вне согласованного объёма;

5. укажи риски, которые ещё остались.

Ничего дополнительно не улучшай без отдельного запроса.

Практическое задание

Вернитесь к странице из первой части. Сначала составьте требования к переключателю светлой и тёмной темы. Попросите Cursor дать план без кода. Затем реализуйте изменение, проверьте обе темы и перезагрузку страницы. Решите сами, должна ли выбранная тема сохраняться после закрытия браузера, и явно внесите это решение в запрос.

После работы попросите Agent перечислить фактические изменения и объяснить одну новую конструкцию в JavaScript. Не пытайтесь выучить весь файл. Разберите только тот механизм, который появился из-за вашей функции.

Практическая грамотность без курса программирования

Читать код как карту действий

Вам не нужно садиться и писать приложение с чистого листа вручную. Но полностью игнорировать код нельзя. Если вы способны проследить путь действия, вы заметите многие ошибки ещё до запуска.

Начните с вопроса: какой файл отвечает за экран или команду? Затем найдите точку входа — место, где приложение начинает работу. В простом сайте это index.html. В проекте с фреймворком точка входа может быть страницей, маршрутом или конфигурацией. В Telegram-боте запуск обычно создаёт экземпляр бота, подключает обработчики и начинает получать обновления.

Дальше проследите одно действие. Пользователь нажал кнопку. Какая функция вызывается? Какие данные она получает? Отправляет ли запрос? Где ответ меняет экран? Такой маршрут полезнее чтения файлов подряд.

Попросите Agent нарисовать путь словами и подтвердить его ссылками на файлы. Затем откройте указанные места. Если объяснение расходится с кодом, доверяйте фактическому проекту и просите пересмотреть анализ.

Имена дают половину понимания

Хорошие имена снижают зависимость от комментариев. submitBooking понятнее, чем handle2, а isOwner точнее, чем check. Если Agent создаёт десятки переменных data, item и value, попросите уточнить названия в ограниченном участке.

Имя файла также должно отражать роль. Компонент формы, серверная операция и доступ к базе не обязаны жить в одном файле на тысячу строк. Но дробление каждой функции в отдельную папку тоже мешает. Разделяйте код по ответственности, когда файл действительно выполняет разные виды работы.

Комментарии должны объяснять причину или ограничение, а не повторять строку. Комментарий «увеличиваем счётчик на один» рядом с очевидной операцией не помогает. Комментарий «повторное событие платежа не должно продлевать доступ второй раз» фиксирует важное правило.

Значения, функции и условия

Переменная хранит значение: имя, список заказов, состояние загрузки. Функция получает данные, выполняет действие и иногда возвращает результат. Условие выбирает путь: если пользователь вошёл — показать кабинет, иначе — форму входа.

При проверке функции задайте четыре вопроса:

- Какие данные входят?
- Как они проверяются?
- Что меняется?
- Что возвращается при успехе и ошибке?

Если функция одновременно проверяет форму, отправляет письмо, пишет в базу и формирует HTML, её трудно тестировать и исправлять. Попросите разделить ответственности, но только после фиксации текущего поведения.

Состояние интерфейса

Состояние — данные, от которых зависит текущий вид приложения. Форма может быть пустой, заполняться, отправляться, завершиться успешно или показать ошибку. Если разработчик учитывает только «пусто» и «готово», пользователь получает двойные отправки и непонятные зависания.

Для сетевого действия обычно нужны состояния:

- исходное;
- загрузка;
- успех;
- ожидаемая ошибка пользователя;
- техническая ошибка сервера.

Кнопка во время загрузки блокируется или безопасно игнорирует повторное нажатие. Ошибка не стирает введённые данные без причины. После успеха интерфейс обновляется из подтверждённого результата, а не притворяется, что сервер всё сохранил.

Массивы и объекты

Объект объединяет свойства одной сущности. Заказ может содержать id, title, status и createdAt. Массив хранит несколько заказов. JSON передаёт похожую структуру в текстовом виде между браузером и сервером.

Порядок полей объекта обычно не важен, а порядок элементов массива важен. Если список сортируется по дате, проверьте, где выполняется сортировка и одинаково ли трактуется время.

Не храните разные значения в одном поле только ради скорости. Строка «Иван; +998...; завтра вечером» сложнее проверяется и ищется, чем отдельные поля. Но не дробите адрес на двадцать колонок, если приложение его только показывает. Структура следует реальным операциям.

Типы как ранняя проверка

TypeScript описывает форму данных. Если функция ожидает заказ, а ей передали пользователя, проверка типов может остановить сборку. Типы не доказывают правильность бизнеса: строка типа string всё ещё может быть пустой или содержать неверный телефон.

Избегайте распространённого обхода any, который отключает значительную часть проверки. Иногда он нужен на границе с неизвестными данными, но вход всё равно должен проверяться и превращаться в известный тип.

При ошибке типов прочитайте обе стороны: что ожидалось и что передано. Не просите Agent заглушить ошибку приведением типа, пока не установлена реальная форма данных.

Импорты и зависимости между файлами

Импорт позволяет файлу использовать функцию, компонент или значение из другого файла. Цепочка импортов показывает связи. Если низкоуровневый модуль базы начинает импортировать интерфейсную кнопку, границы смешались.

Циклическая зависимость возникает, когда файл A зависит от B, а B прямо или через цепочку зависит от A. Иногда среда её терпит, иногда появляются неопределённые значения и странные ошибки запуска. Исправление обычно требует вынести общую часть в третий модуль, а не добавить ещё один импорт.

Синхронная и асинхронная работа

Запрос к серверу, базе или файлу занимает время. Асинхронный код позволяет приложению ждать результат, не блокируя всё остальное. Ключевые слова async и await помогают описать последовательность, но ошибки ожидания остаются частой причиной проблем.

Если забыть дождаться результата, функция может получить обещание будущего значения вместо самого значения. Если не обработать отказ, ошибка уйдёт в общий журнал или завершит операцию без понятного ответа.

При анализе сетевого пути проверьте:

- установлен ли тайм-аут;
- что происходит при медленном ответе;
- обрабатывается ли неуспешный код;
- можно ли повторить действие;
- не создаст ли повтор дубль;
- отменяется ли устаревший запрос при уходе с экрана.

Клиент и сервер

Клиентский код доставляется на устройство пользователя. Его можно просмотреть и изменить. Поэтому клиентская проверка не является доказательством права или правильности данных.

Сервер получает запрос и применяет доверенные правила. Он проверяет сессию, формат, принадлежность записи и ограничения. После этого обращается к базе или внешнему API. Даже если кнопка удаления скрыта, злоумышленник может вызвать адрес операции вручную; сервер обязан отказать.

В полнофункциональном фреймворке клиентские и серверные файлы находятся рядом. Следите за границей. Импорт серверного секрета в компонент, который попадает в браузер, может раскрыть его при сборке.

HTTP

без таблицы кодов

HTTP-запрос содержит метод, адрес, заголовки и иногда тело. Метод сообщает намерение: получить данные, создать или изменить. Сервер отвечает кодом, заголовками и телом.

Различайте категории:

- успех;
- ошибка входных данных;
- пользователь не вошёл;
- действие запрещено;
- объект не найден;
- конфликт текущего состояния;
- ограничение частоты;
- внутренняя ошибка.

Если сервер возвращает один и тот же успешный ответ даже при провале, интерфейс не сможет правильно сообщить результат. Если возвращает внутренний текст базы, раскрывает устройство системы.

База как источник сохранённого состояния

Таблица содержит строки одного типа. Первичный ключ однозначно определяет строку. Внешний ключ связывает её с другой таблицей. Ограничения базы защищают правила даже при ошибке приложения.

Запрос на выборку должен ограничиваться текущим пользователем на сервере или политикой базы. Фильтр только после загрузки всех строк в браузер уже раскрыл данные.

Транзакция объединяет несколько изменений: либо выполняются все, либо ни одно. Она нужна, когда создание заказа и уменьшение остатка не должны расходиться. Не каждое действие требует транзакции, но Agent должен заметить связанную операцию.

Миграции вместо ручных изменений

Миграция — файл, который переводит схему базы из одного состояния в другое. Он позволяет повторить изменение на тестовой и рабочей базе. Название миграции должно описывать действие, а порядок фиксируется инструментом.

Добавление необязательного поля обычно безопаснее удаления или изменения типа. Для опасного изменения применяют несколько выпусков: добавить новое поле, научить код писать оба, перенести данные, переключить чтение и только затем удалить старое. В маленьком проекте такой процесс кажется длинным, но он защищает рабочие данные.

Логи, ошибки и сообщения пользователю

Ошибка имеет три аудитории. Пользователю нужно понять, что делать. Разработчику нужен контекст диагностики. Системе мониторинга нужны тип и время.

Пользователю: «Не удалось сохранить заявку. Проверьте соединение и попробуйте ещё раз».

Журналу: операция, безопасный идентификатор запроса, категория ошибки, техническая причина и время.

Не показывайте пользователю стек и строку подключения. Не пишите в журнал пароль и полный текст приватного документа. Идентификатор помогает сопоставить обращение пользователя с конкретной записью.

Тест как исполняемое требование

Тест готовит данные, выполняет действие и проверяет результат. Хорошее имя описывает правило: «пользователь не может изменить чужой заказ». Если реализация меняется, правило остаётся.

Не просите Agent повысить процент покрытия любой ценой. Тест конструктора без бизнес-логики может увеличить цифру и ничего не защитить. Сначала покрывайте права, деньги, сохранность данных и главный сценарий.

Тест должен быть повторяемым и независимым. Он не обращается к рабочей базе, не зависит от текущего времени без контроля и очищает созданные данные.

Сборка и среда выполнения

Режим разработки оптимизирован для быстрых изменений и подробных ошибок. Production-сборка оптимизирует код и применяет другие настройки. Поэтому локально открытая страница ещё не означает готовность к публикации.

Команда сборки выявляет пропущенные импорты, ошибки типов и неподдерживаемые зависимости. После сборки полезно запустить production-версию локально, если стек это поддерживает.

Среда выполнения — версия Node.js, Python или другой платформы, где код работает. Локальная и облачная версии должны совпадать в поддерживаемом диапазоне. Зафиксируйте требование в конфигурации, чтобы хостинг не выбрал случайную старую версию.

Как разбирать незнакомый файл

Не просите пересказать каждую строку. Используйте порядок:

1. Назначение файла.
2. Что он импортирует и кому нужен.
3. Главные данные и функции.
4. Входы и выходы.
5. Ошибки и побочные действия.
6. Связь с пользовательским сценарием.
7. Риски изменения.

После объяснения закройте ответ и попытайтесь сами рассказать путь в двух абзацах. Если не получается, задайте один конкретный вопрос. Такое чтение постепенно создаёт техническую интуицию без зубрёжки синтаксиса.

Когда остановить

Agent

Остановитесь, если он:

• меняет стек без согласования;
• удаляет рабочий код вместо поиска причины;
• предлагает отключить проверку типов или безопасности;
• просит поместить секрет в публичный файл;
• повторяет одно исправление без новых данных;
• обновляет много зависимостей во время локальной ошибки;
• выполняет destructive-команду с широкой целью;
• заявляет об успешной проверке, не запустив её;
• добавляет функции за пределами ТЗ.

Остановка не означает провал. Вернитесь к последней контрольной точке, уменьшите задачу и соберите факты.

Часть третья. Первый настоящий сайт

Проект сайта для локального бизнеса

Первый полноценный проект — сайт небольшой студии детейлинга. Он показывает услуги, отвечает на частые вопросы и принимает заявку. Такой сайт можно адаптировать под автосервис, ремонт техники, клининг, фотографа или учебный центр. Мы не будем подключать оплату и сложный кабинет. Главная задача — привести посетителя к заявке и не потерять её.

Стек будет простым: HTML для структуры, CSS для оформления и JavaScript для интерактивных элементов. Заявки сначала будут работать в демонстрационном режиме, затем мы подключим простой серверный обработчик при публикации. Отсутствие фреймворка здесь полезно: вы увидите основу веб-страницы и не будете разбираться с лишней сборкой.

Создайте папку detailing-site, откройте её в Cursor и начните не с кода, а с описания продукта.

Промпт 20. Техническое задание сайта

Помоги подготовить короткое техническое задание для сайта студии детейлинга.

Цель сайта — получить заявку на консультацию.

Аудитория — владельцы автомобилей в [ГОРОД].

Услуги: [СПИСОК УСЛУГ].

Предложи структуру одностраничного сайта. Для каждого раздела укажи его задачу и главное действие посетителя.

Не придумывай отзывы, сертификаты, гарантии, цены или факты о компании.

Отметь, какие данные владелец должен предоставить до публикации.

Последняя строка защищает от распространённой проблемы: модель охотно заполняет пустоты убедительными, но вымышленными доказательствами. На коммерческом сайте нельзя придумывать стаж, количество клиентов или обещание результата.

Оставьте разделы: первый экран, услуги, процесс работы, ответы на вопросы, контакты и форма заявки. Портфолио добавляйте только с реальными фотографиями. Не перегружайте первую версию блогом, личным кабинетом и калькулятором стоимости.

Создание каркаса

Попросите Agent сначала создать файловую структуру и базовый макет. Зафиксируйте запрет на библиотеки, чтобы проект оставался прозрачным.

Промпт 21. Каркас сайта

Создай первую рабочую версию одностраничного сайта по этому техническому заданию:

[ВСТАВЬТЕ ТЗ]

Используй обычные HTML, CSS и JavaScript без фреймворков и внешних библиотек.

Создай понятную структуру файлов. Все тексты вынеси в HTML, стили — в отдельный CSS, поведение — в отдельный JavaScript.

Не добавляй вымышленные отзывы, цифры и изображения.

Сделай семантическую структуру страницы и понятные названия классов.

После создания объясни роль каждого файла и предложи способ локального просмотра.

Семантическая структура означает, что элементы названы по смыслу: заголовок, навигация, основное содержимое, секция, форма, подвал. Это помогает браузерам, поисковым системам и средствам доступности понять страницу.

Запустите сайт. Для простого HTML можно открыть файл напрямую, но локальный сервер ближе к реальным условиям. Cursor может предложить команду через установленный инструмент. Если для этого требуется новая зависимость, попросите объяснить её. Для первого просмотра допустим и встроенный предварительный просмотр.

Проверьте, что каждый пункт меню ведёт к нужному разделу, форма видна, тексты не выходят за границы, а консоль браузера не показывает ошибок. Консоль — журнал технических сообщений страницы. Встроенный браузер Cursor, если он доступен, может предоставить Agent доступ к консоли и сетевым запросам. Это полезно для диагностики, но не отменяет ваш собственный просмотр.

Дизайн через ограничения

Запрос «сделай красиво» почти гарантирует случайный результат. Дизайн-задача должна описывать настроение, иерархию, ограничения и устройства.

Промпт 22. Система оформления

Улучши визуальный стиль сайта студии детейлинга.

Характер: аккуратный, технологичный, без агрессивного неона и визуального шума.

Ограничения:

- один основной акцентный цвет;

- не больше двух семейств шрифтов;

- хорошо заметная основная кнопка;

- достаточный контраст текста;

- единые отступы, радиусы и размеры заголовков;

- никаких тяжёлых анимаций и декоративных элементов без функции.

Сначала предложи короткую систему оформления: цвета, типографика, отступы и кнопки. После согласования измени только CSS.

Полезно разделить систему и реализацию. Если палитра не подходит, вы меняете решение до переписывания всего файла.

Не оценивайте дизайн по количеству эффектов. Первый экран должен за несколько секунд ответить: что предлагает компания, в каком городе и что сделать дальше. Кнопка не должна спорить за внимание с пятью декоративными блоками.

Улучшение конкретного экрана

Промпт 23. Анализ визуальной проблемы

Проанализируй текущий вид раздела [НАЗВАНИЕ РАЗДЕЛА].

Проблема, которую я вижу: [ОПИСАНИЕ ИЛИ ПРИКРЕПЛЁННЫЙ СКРИНШОТ].

Сначала объясни вероятную причину на уровне структуры и CSS.

Предложи не больше трёх точечных изменений по приоритету.

Не меняй другие разделы и не вводи новую дизайн-систему.

Скриншот даёт Agent визуальный контекст, но добавьте текстом, что именно не устраивает. «Карточки слишком узкие на ноутбуке» полезнее, чем одно изображение.

Адаптивность

Адаптивный сайт подстраивается под ширину экрана. Это не уменьшенная настольная версия. На телефоне меню может свернуться, карточки выстроиться в одну колонку, а кнопки — стать удобнее для касания.

Начните с трёх проверок: широкий компьютерный экран, обычный ноутбук и узкий телефон. Между ними макет также не должен ломаться. В браузере есть режим имитации устройств, но окончательно проверьте сайт на настоящем телефоне.

Промпт 24. Адаптивная проверка

Проверь адаптивность текущей страницы на ширинах 360, 768, 1024 и 1440 пикселей.

Найди:

- горизонтальную прокрутку;

- обрезанный текст;

- слишком мелкие элементы управления;

- наложение блоков;

- неудачный порядок элементов;

- чрезмерные пустые области.

Сначала перечисли найденные проблемы с указанием селекторов или блоков. Затем исправь их минимальными изменениями CSS. Не меняй тексты и общую визуальную концепцию.

Если Agent может управлять браузером, попросите его сделать снимки на этих ширинах и проверить консоль. Если инструмента нет, изменяйте ширину окна вручную и сообщайте точное место дефекта.

Форма заявки

Форма состоит из видимой части и обработчика. На странице пользователь вводит данные. JavaScript проверяет очевидные ошибки и отправляет запрос. Сервер повторно проверяет данные и решает, что с ними делать. Проверка только в браузере недостаточна: злоумышленник может отправить запрос напрямую.

На первом этапе сделаем локальную форму, которая проверяет имя, телефон, согласие на обработку данных и показывает результат без реальной отправки.

Промпт 25. Клиентская проверка формы

Добавь в существующую форму поля «Имя», «Телефон» и обязательный флажок согласия на обработку данных.

Требования:

- ошибки показываются рядом с соответствующим полем;

- ошибочное поле получает заметное состояние;

- сообщение исчезает после исправления;

- повторное быстрое нажатие не отправляет форму несколько раз;

- при успешной локальной проверке показывается подтверждение;

- данные пока не передаются во внешний сервис.

Сохрани доступность формы с клавиатуры и не используй alert.

После реализации перечисли ручные сценарии проверки.

Доступность с клавиатуры означает, что человек может переходить между полями клавишей Tab, видеть фокус и отправить форму без мыши. Подписи должны быть связаны с полями, а ошибки понятны без одного лишь красного цвета.

Подключение реальной отправки

Для реальной заявки нужен серверный обработчик или проверенный сервис форм. Он принимает HTTP-запрос — сообщение от браузера серверу. Данные часто передаются в формате JSON: текстовой структуре из полей и значений. Например, имя и телефон становятся объектом, который обработчик может прочитать.

Не привязываемся к одному хостингу. Выберите платформу, способную обслуживать статический сайт и серверную функцию, либо отдельный сервис форм. Перед подключением прочитайте актуальную документацию выбранной платформы.

Промпт 26. Проектирование обработчика заявки

Нужно подключить реальную отправку формы на платформе [ПЛАТФОРМА].

Изучи текущий проект и актуальную официальную документацию платформы.

Сначала предложи план без кода.

Обработчик должен:

- принимать только ожидаемые поля;

- повторно проверять данные на сервере;

- ограничивать размер входных данных;

- не возвращать внутренние ошибки пользователю;

- не содержать секреты в клиентском коде;

- давать понятный ответ об успехе или ошибке.

Укажи, какие переменные окружения понадобятся и где их настроить.

Переменная окружения — настройка, которую платформа передаёт приложению отдельно от исходного кода. Секретный ключ можно хранить там, а не в публичном JavaScript.

Если заявки отправляются в Telegram, электронную почту или CRM, ключ интеграции должен оставаться на серверной стороне. Всё, что попало в код страницы, посетитель способен увидеть.

Производительность и доступность

На небольшом сайте главные проблемы создают тяжёлые изображения, лишние шрифты и скрипты. Сжимайте фотографии, задавайте им размеры и не загружайте огромный файл ради маленькой карточки. Проверка доступности должна охватывать заголовки, подписи полей, контраст, клавиатуру и альтернативный текст для содержательных изображений.

Промпт 27. Аудит перед публикацией

Проведи аудит сайта перед публикацией.

Проверь HTML, CSS и JavaScript по направлениям:

- ошибки и предупреждения;

- адаптивность;

- доступность;

- скорость загрузки;

- корректность формы;

- базовая поисковая разметка;

- отсутствие секретов и тестовых данных.

Составь список проблем по приоритету. Сначала исправь только критические и высокие проблемы, не меняя дизайн без необходимости. Затем запусти повторную проверку.

Конец ознакомительного фрагмента.