



## 3.1.1 Основы работы веб приложений

Модуль 3. Интеграция приложений и prompt engineering

# Оглавление

<b>1. Введение .....</b>	<b>3</b>
<b>2. Как устроено веб-приложение .....</b>	<b>4</b>
<b>3. Сервер – Бизнес логика и База данных.....</b>	<b>5</b>
<b>4. Frontend – как работает браузер .....</b>	<b>6</b>
<b>5. Запросы и ответы (HTTP).....</b>	<b>7</b>
<b>6. HTML .....</b>	<b>9</b>
<b>7. CSS .....</b>	<b>10</b>
<b>8. JavaScript.....</b>	<b>11</b>
<b>9. Заключение .....</b>	<b>12</b>

## 1. Введение

Здравствуйте. Сегодня мы с вами поговорим об интеграции языковых моделей с веб-приложениями. Это важная и, можно сказать, фундаментальная тема для каждого, кто хочет работать в области Prompt Engineering.

Веб-приложения сегодня составляют львиную долю цифрового пространства. Большинство сервисов, которыми мы пользуемся ежедневно — от поисковых систем до интернет-магазинов и корпоративных платформ — построены как веб-приложения. Это гибкий и доступный способ доставки функциональности пользователям через браузер, без необходимости устанавливать дополнительные программы. С технической стороны, это мощная архитектура, в которой соединяются интерфейс, логика приложения и данные, зачастую работающие на разных уровнях — от клиентского JavaScript до серверных API и баз данных.

Теперь представьте, что в эту структуру вы добавляете языковую модель. Как только мы начинаем применять LLM в реальных продуктах — будь то чат-бот, генератор текста, помощник в интерфейсе или внутренний аналитический инструмент — мы неизбежно сталкиваемся с вопросом: как соединить промпт, который мы разработали, с остальной системой? Как он дойдет от пользователя до модели и обратно? Где мы его формируем — на клиенте, на сервере, или внутри промежуточного слоя? Как учитываются параметры модели, безопасность, лимиты и ошибки?

Для того чтобы ответить на эти вопросы и влиять на итоговый результат, Prompt Engineer должен понимать основы работы веб-приложений. Не обязательно становиться full-stack разработчиком, но нужно хорошо разбираться в том, как устроен маршрут запроса от пользователя до модели, как обрабатываются данные, как оформляется ответ, и где именно ваш промпт становится частью этой цепочки.

Сегодня мы разберёмся, как всё это работает.

## 2. Как устроено веб-приложение

Веб-приложение — это программа, которая работает через интернет и запускается прямо в вашем браузере. Когда вы, например, заходите на сайт и что-то делаете — вводите текст, нажимаете кнопку, отправляете форму — вы взаимодействуете с веб-приложением. Оно похоже на обычное приложение, только не нужно ничего скачивать: всё работает через сайт.

Внутри веб-приложения есть две основные части. Первая — это то, что вы видите на экране. Это кнопки, поля, текст, картинки. Эта часть называется фронтеном. Она отвечает за то, чтобы вам было удобно и понятно взаимодействовать с программой. Она работает в вашем браузере. Вторая часть — это сервер, или бэкенд. Он находится где-то в интернете, на компьютере, который обрабатывает ваши действия. Если вы нажимаете «отправить» — эта информация уходит на сервер. Сервер думает, обрабатывает, может обращаться к базе данных или к языковой модели, и отправляет обратно результат. Вы видите этот результат на экране.

Между вашим браузером и сервером идет постоянный обмен сообщениями. Они обмениваются данными по сети. Обычно эти данные отправляются в виде простого текста в специальном формате, например, JSON. Браузер говорит: «Вот данные от пользователя». Сервер отвечает: «Вот результат, покажи его». Всё это происходит за доли секунды.

Когда вы как prompt engineer работаете с LLM, то вы встраиваете модель в эту цепочку. Модель становится частью логики, которая работает на сервере. Промпт, который вы придумали, отправляется модели через специальный запрос. Модель возвращает ответ, и этот ответ появляется у пользователя в интерфейсе. Чтобы эта схема работала правильно, вам нужно понимать, где именно ваша часть — где живёт промпт, как он попадает в модель, как результат возвращается. Чем лучше вы это понимаете, тем точнее вы можете влиять на поведение всей системы, даже не будучи программистом.

### **3. Сервер – Бизнес логика и База данных**

Веб-сервер — это компьютер, который всё время подключён к интернету и ждёт запросов от пользователей. Когда вы нажимаете кнопку на сайте или отправляете форму, ваш браузер отправляет запрос на этот сервер. Сервер получает данные, обрабатывает их и возвращает ответ. По сути, он как официант в ресторане — принимает заказ, относит его на кухню и приносит результат.

Внутри сервера работает программа — она называется серверной логикой или бизнес-логикой. Это набор правил и инструкций, которые говорят серверу, что делать с полученными данными. Например, если пользователь пишет запрос в чат, бизнес-логика решает: нужно взять

этот текст, сформировать правильный промпт, отправить его языковой модели, дождаться ответа, возможно, обработать этот ответ и только потом отправить его обратно пользователю. Бизнес-логика может быть простой, а может быть очень сложной — с проверками, фильтрами, условиями и вызовами других сервисов.

База данных — это место, где сервер хранит информацию. Она похожа на таблицу или электронную таблицу Excel, но гораздо мощнее. Если, например, вы создаёте аккаунт на сайте, ваши данные сохраняются в базе данных. Если вы пишете историю запросов в чат-боте, то они тоже хранятся там. Когда серверу нужно что-то узнать — например, какой у пользователя был предыдущий запрос — он делает запрос к базе данных, получает нужную информацию и использует её в работе.

Все три части — веб-сервер, бизнес-логика и база данных — работают вместе. Сервер принимает запрос, бизнес-логика решает, что с ним делать, и если нужно, обращается к базе данных. А затем всё это вместе формирует ответ и отправляет его обратно пользователю. Именно в этой цепочке и живёт ваша языковая модель, и ваша задача как prompt engineer — понимать, где и как она встраивается в этот процесс.

## 4. Frontend – как работает браузер

Фронтенд — это всё, что вы видите и с чем взаимодействуете на сайте. Это кнопки, поля ввода, тексты, изображения, анимации — всё, что открывается у вас в браузере. Он отвечает за внешний вид и за то, чтобы вы могли удобно общаться с программой. Фронтенд работает прямо на вашем компьютере, в вашем браузере. Он не просто

показывает информацию, но и реагирует на ваши действия — клики, ввод текста, прокрутку страницы. Когда вы нажимаете кнопку или отправляете форму, фронтенд может собрать данные и отправить их на сервер.

Браузер — это программа, которая загружает и отображает веб-страницы. Он получает код страницы — обычно это HTML, CSS и JavaScript — и превращает его в то, что вы видите на экране. HTML описывает структуру, CSS задаёт внешний вид, а JavaScript добавляет интерактивность. Когда вы открываете сайт, браузер сначала запрашивает данные с сервера, потом по частям загружает и собирает из них страницу. JavaScript позволяет делать страницу «живой» — менять элементы без перезагрузки, отправлять запросы на сервер, получать ответы и показывать их вам.

Фронтенд может сам обращаться к языковой модели, если она доступна через API, или он может передавать промпт на сервер, где уже работает логика взаимодействия с моделью. В обоих случаях важно, чтобы данные были правильно собраны, переданы и результат был удобно показан пользователю. Поэтому знание, как работает фронтенд и браузер, помогает prompt engineer понимать, где именно пользователь сталкивается с моделью и как сделать это взаимодействие более понятным и эффективным.

## 5. Запросы и ответы HTTP

Когда вы открываете сайт или нажимаете на кнопку в веб-приложении, ваш браузер отправляет запрос на сервер. Этот запрос передаётся по специальным правилам, которые называются HTTP (Hypertext Transfer Protocol) — это как язык, на котором общаются

браузер и сервер. Запрос содержит информацию о том, что именно вы хотите получить или отправить. Например, вы хотите увидеть страницу, отправить сообщение, загрузить данные из базы или получить результат от языковой модели.

Запрос всегда содержит адрес — это как указание, куда именно обратиться. В нём может быть также метод — чаще всего это GET, если вы просто что-то запрашиваете, или POST, если вы отправляете данные. Кроме того, внутри запроса может быть текст, который вы хотите отправить на сервер, например, текст промпта для модели. Этот текст передаётся в теле запроса в определённом формате, обычно JSON.

Сервер получает запрос, обрабатывает его и формирует ответ. Ответ тоже передаётся по правилам HTTP. Он содержит статус — например, успешно выполнено или произошла ошибка — и сам результат. Это может быть текст, HTML-страница, данные в формате JSON или что-то ещё. Если вы, например, отправляете промпт модели, то в ответ вы получите сгенерированный текст. Ваш браузер или ваше приложение принимает этот ответ и показывает его вам.

Вся эта схема — от запроса до ответа — происходит за доли секунды, и именно через неё фронтенд, бэкенд и языковая модель обмениваются информацией. Понимание того, как устроены HTTP-запросы и ответы, помогает точно управлять тем, какие данные вы отправляете модели и как вы получаете результат, а значит — влияет на качество и надёжность вашей работы как prompt engineer.

## 6. HTML

HTML (HyperText Markup Language) — это язык, с помощью которого создаются страницы в интернете. Его задача — описывать структуру содержимого, то есть объяснять браузеру, где заголовок, где абзац текста, где картинка, где кнопка. Когда вы заходите на любой сайт, в основе его отображения лежит HTML. Он не делает страницу красивой и не заставляет её работать — он просто говорит, что на странице есть и как это расположено.

Представьте документ. В нём может быть заголовок, под ним текст, дальше картинка, потом список. HTML как раз описывает, что вот это — заголовок первого уровня, дальше идёт обычный текст, потом изображение с таким-то адресом, а потом маркированный список из трёх пунктов. Всё это записывается с помощью тегов — таких коротких команд в угловых скобках, которые говорят, где что начинается и где заканчивается. Например, тег `<h1>` используется для заголовка, `<p>` — для абзаца, `<img>` — для картинки, `<button>` — для кнопки.

Браузер читает HTML и показывает вам страницу на экране в том виде, как описано в коде. Если бы не было HTML, браузер бы просто не знал, как собрать страницу. Когда вы создаёте веб-приложение, именно с HTML всё начинается. Он задаёт основу, к которой потом добавляется стиль через CSS и поведение через JavaScript. И даже если вы работаете не как разработчик, а как prompt engineer, важно понимать, что HTML — это та самая оболочка, через которую пользователь видит и взаимодействует с вашей моделью. Если вы выводите ответ модели на экран, он тоже будет встроен в HTML-страницу.

## 7. CSS

CSS (Cascade Style Sheet) — это язык, который отвечает за внешний вид веб-страницы. Если HTML говорит, что на странице есть заголовок, текст или кнопка, то CSS задаёт, как всё это должно выглядеть. С его помощью можно настроить цвета, шрифты, отступы, размеры, расположение элементов на странице и даже анимации. Без CSS все сайты выглядели бы одинаково — просто чёрный текст на белом фоне, без оформления.

Например, у вас есть заголовок, написанный с помощью HTML. Он по умолчанию будет крупным и чёрным. Если вы хотите сделать его синим, поставить по центру и поменять шрифт — это делается с помощью CSS. Вы пишете правила, в которых указываете: заголовок должен быть синего цвета, выравниваться по центру и использовать такой-то шрифт. Эти правила браузер читает и применяет при отображении страницы.

CSS можно сравнить с одеждой для HTML. HTML задаёт структуру — что есть на странице, а CSS отвечает за стиль — как всё это будет выглядеть. Благодаря CSS веб-приложения становятся не только удобными, но и приятными визуально. Даже простой чат с языковой моделью можно оформить так, чтобы он выглядел аккуратно, понятно и приятно пользователю. И хотя вы как prompt engineer не обязаны писать CSS сами, понимание его роли помогает лучше представить, как будет выглядеть результат вашей работы в реальном интерфейсе.

## 8. JavaScript

JavaScript — это язык, который делает веб-страницу «живой». Если HTML задаёт, что именно есть на странице, а CSS отвечает за внешний вид, то JavaScript управляет поведением — он позволяет странице реагировать на действия пользователя. Благодаря ему сайт может менять содержимое без перезагрузки, показывать подсказки, анимировать элементы, отправлять и получать данные с сервера.

Например, когда вы нажимаете кнопку, и на экране появляется новый текст или результат от языковой модели — за это отвечает JavaScript. Он отслеживает нажатие, собирает нужные данные, отправляет запрос на сервер и получает ответ. После этого он может показать этот ответ прямо на странице, не перезагружая её. Именно так работают чаты, формы, фильтры и всё, что требует взаимодействия с пользователем.

JavaScript выполняется прямо в браузере. Это значит, что он может мгновенно реагировать на ваши действия, делать интерфейс удобнее и быстрее. В случае с интеграцией LLM JavaScript часто используется для того, чтобы отправить текст промпта на сервер, получить ответ модели и вставить его в нужное место на странице.

Вы как prompt engineer можете не писать JavaScript сами, но вы должны понимать, что именно он связывает интерфейс пользователя с моделью. Он управляет тем, когда и как отправляется ваш промпт, и как результат появляется перед пользователем. Без него взаимодействие с моделью не выглядело бы живым и удобным.

## 9. Заключение

Вы теперь знаете, из чего состоит веб-приложение и как работает его внутренняя структура. Вы поняли, как браузер отображает страницу с помощью HTML, как CSS придаёт ей внешний вид и как JavaScript делает её интерактивной. Вы увидели, как эти части взаимодействуют с сервером, где работает бизнес-логика, обрабатываются запросы и подключается языковая модель. Вы познакомились с тем, как данные проходят путь от пользователя до модели и обратно, используя HTTP-запросы и ответы.

Для prompt engineer это не просто техническая справка. Это каркас, внутри которого живёт ваш промпт. От того, как вы понимаете эту структуру, зависит, насколько точно вы можете проектировать взаимодействие с моделью. Это помогает вам не только придумывать хорошие запросы, но и эффективно внедрять их в реальные продукты. Поэтому ваша задача — не стать программистом, но научиться говорить на одном языке с теми, кто создаёт интерфейсы и серверную логику, чтобы ваша совместная работа давала максимум пользы.

Спасибо за внимание!