

ОСНОВЫ WEB ПРОГРАММИРОВАНИЯ

02.03.03 -Математическое обеспечение и администрирование информационных систем,
направленность (профиль) -разработка и администрирование информационных систем

<http://vikchas.ru>

Лекция 8-9. ««Особенности WEB-программирования при использовании больших фундаментальных моделей искусственного интеллекта»

»

Часовских Виктор Петрович
доктор технических наук, профессор кафедры
ШИиКМ, ФГБОУ ВО «Уральский государственный
экономический университет

Екатеринбург 2026

1. Цель лекции

Цель лекции — рассмотреть, какие особенности появляются при разработке WEB-приложений, использующих большие фундаментальные модели искусственного интеллекта: языковые, мультимодальные, генеративные и специализированные модели.

Такие приложения отличаются от классических WEB-сервисов тем, что их логика частично переносится на AI-модель, а результат работы становится вероятностным, контекстно-зависимым и требующим дополнительных механизмов контроля.

2. Классическое WEB-приложение и AI-WEB-приложение

В классическом WEB-приложении поведение системы в основном детерминировано:

Запрос пользователя → бизнес-логика → база данных → ответ

Например:

- пользователь нажал кнопку;
- сервер выполнил запрос к базе данных;
- вернул список товаров;
- frontend отобразил результат.

В AI-WEB-приложении появляется дополнительный интеллектуальный компонент:

Запрос пользователя → backend → AI-модель → обработка ответа → frontend

Или более сложный вариант:

Запрос пользователя



Backend



Поиск данных / RAG / база знаний



Большая AI-модель



Постобработка и проверка



Ответ пользователю

Главная особенность: ответ модели не всегда полностью предсказуем, поэтому разработчик должен учитывать вероятность ошибок, неоднозначность и необходимость валидации результата.

3. Что такое большие фундаментальные модели ИИ

Большие фундаментальные модели искусственного интеллекта — это модели, обученные на огромных объёмах данных и способные решать широкий круг задач.

К ним относятся:

- большие языковые модели, LLM;
- мультимодальные модели;
- модели генерации изображений;
- модели распознавания речи;
- модели синтеза речи;
- модели генерации программного кода;
- модели анализа документов.

Примеры возможностей:

- генерация текста;
- ответы на вопросы;
- резюмирование;
- классификация;

- перевод;
- анализ изображений;
- извлечение данных из документов;
- генерация кода;
- диалоговые интерфейсы.

Для WEB-разработчика важно не столько внутреннее устройство модели, сколько способы её подключения, ограничения, стоимость, задержки, безопасность и контроль качества.

4. Новая роль backend в AI-WEB-приложениях

При использовании больших моделей backend выполняет не только классические функции:

- авторизация;
- работа с базой данных;
- бизнес-логика;
- обработка HTTP-запросов.

Он становится промежуточным интеллектуальным слоем между пользователем и моделью.

Backend отвечает за:

1. Подготовку запроса к модели.
2. Добавление системных инструкций.
3. Подстановку контекста.
4. Проверку прав доступа.
5. Ограничение длины запроса.
6. Логирование.
7. Обработку ошибок API модели.
8. Постобработку ответа.
9. Фильтрацию опасного или некорректного результата.
10. Контроль расходов на использование модели.

То есть AI-модель нельзя просто «напрямую подключить к форме на сайте». Между пользователем и моделью обязательно должен быть контролируемый серверный слой.

5. Почему нельзя вызывать AI API напрямую из frontend

Одна из распространённых ошибок — вызывать API большой модели прямо из браузера.

Например, так делать нельзя:

JAVASCRIPT

```
fetch("https://api.ai-provider.com/v1/chat", {  
  headers: {  
    "Authorization": "Bearer SECRET_API_KEY"  
  }  
});
```

Проблемы такого подхода:

- API-ключ окажется доступен пользователю;
- любой сможет использовать ключ за счёт владельца;
- нельзя нормально ограничить запросы;
- сложно контролировать безопасность;
- невозможно надёжно скрыть системные инструкции;
- нельзя проверить права доступа к данным;
- нельзя централизованно логировать обращения.

Правильная архитектура:

Frontend → Backend → AI API

Frontend отправляет запрос на собственный backend, а backend уже обращается к модели.

6. Особенности проектирования API для AI-сервисов

Обычный API часто возвращает заранее определённые структуры данных.

Например:

JSON

```
{  
  "id": 15,  
  "name": "Иван",  
  "role": "student"  
}
```

AI-API может возвращать текст, JSON, код, таблицу, список, объяснение или смешанный формат.

Поэтому важно заранее определить формат ответа.

Например, backend может требовать от модели вернуть строго JSON:

JSON

```
{  
  "summary": "Краткое резюме текста",  
  "keywords": ["AI", "WEB", "API"],  
  "risk_level": "low"  
}
```

Особенности API с ИИ:

- ответы могут быть длинными;
- ответы могут генерироваться постепенно;
- возможен streaming;
- нужно контролировать формат;
- нужно обрабатывать неполные ответы;
- требуется повтор запроса при ошибке;
- желательно сохранять историю диалога;
- необходимо учитывать стоимость каждого вызова.

7. Streaming-ответы

Многие AI-сервисы используют потоковую генерацию ответа.

Обычный ответ:

Пользователь отправил запрос → ждёт → получил полный ответ

Streaming:

Пользователь отправил запрос → модель постепенно отдаёт части ответа

Для пользователя это выглядит как печать текста в реальном времени.

Преимущества:

- быстрее появляется первый результат;
- улучшается пользовательский опыт;
- длинные ответы воспринимаются легче;
- можно прервать генерацию.

Технологии для streaming:

- Server-Sent Events, SSE;
- WebSocket;
- HTTP streaming.

Пример сценария:

Frontend открывает потоковое соединение

Backend получает части ответа от AI API

Backend пересылает их пользователю

Frontend постепенно отображает текст

8. Prompt как часть программной логики

В AI-WEB-приложениях prompt (запрос нейронной сети) становится частью бизнес-логики.

Prompt — это не просто текстовый запрос, а управляющая инструкция для модели.

Он может включать:

- роль модели;
- задачу;
- стиль ответа;
- ограничения;
- формат;
- контекст;
- примеры;
- правила безопасности.

Пример системной инструкции:

Ты — помощник образовательной платформы.

Отвечай только по материалам курса.

Если информации недостаточно, скажи: "В материалах курса нет данных для ответа".

Не придумывай факты.

Ответ должен быть кратким и понятным студенту.

Особенность WEB-разработки с ИИ состоит в том, что prompt нужно проектировать, версионировать, тестировать и защищать так же, как обычный код.

9. Prompt templates

В реальных системах prompt обычно собирается из шаблонов.

Пример:

Системная роль:

{system_instruction}

Контекст пользователя:

{user_profile}

Фрагменты документов:

{retrieved_documents}

Вопрос пользователя:

{user_question}

Формат ответа:

{output_format}

Преимущества шаблонов:

- единообразие запросов;
- проще обновлять инструкции;
- легче тестировать;
- можно адаптировать prompt под разные роли пользователей;
- можно подставлять данные из базы.

Но шаблоны требуют осторожности: пользовательский ввод нельзя смешивать с управляющими инструкциями без разграничения.

10. Контекст и ограничение окна контекста

Большие модели не имеют бесконечной памяти. Они работают в рамках **окна контекста**.

Контекст может включать:

- системную инструкцию;
- историю диалога;
- документы;
- данные пользователя;
- текущий вопрос.

Проблема: если контекста слишком много, он не помещается в запрос или становится дорогим.

Разработчику нужно решать:

- какие данные передавать модели;

- что исключить;
- как сжать историю;
- как выбрать релевантные документы;
- как ограничить размер пользовательского ввода;
- как не передать лишние персональные данные.

11. RAG как типичная архитектура WEB-приложений с LLM

RAG, Retrieval-Augmented Generation, — это архитектура, при которой модель отвечает не только на основе своих знаний, но и на основе найденных документов.

Схема:

Пользовательский вопрос



Backend



Поиск релевантных фрагментов



Формирование prompt



LLM



Ответ с учётом документов

RAG особенно важен для WEB-сервисов, где ответы должны основываться на конкретной базе знаний:

- учебные платформы;
- корпоративные порталы;
- юридические системы;
- медицинские справочники;
- техническая поддержка;

- электронные библиотеки.

Особенность WEB-разработки: кроме обычной базы данных появляется векторное хранилище и pipeline обработки документов.

12. Векторизация данных

Для работы RAG документы преобразуются в embedding-векторы.

Процесс:

1. Документы загружаются в систему.
2. Текст очищается.
3. Делится на фрагменты.
4. Каждый фрагмент превращается в embedding.
5. Embedding сохраняется в векторной базе.
6. При вопросе пользователя выполняется поиск похожих фрагментов.

Пример архитектуры:

PDF / DOCX / HTML



Извлечение текста



Разбиение на chunks



Embedding model



Vector database

Затем при запросе:

Вопрос пользователя



Embedding вопроса



Поиск похожих chunks



Передача chunks в prompt

13. Проблема галлюцинаций

Галлюцинация модели — это ситуация, когда модель уверенно выдаёт неверную или вымышленную информацию.

Для WEB-приложений это критично, потому что пользователь воспринимает ответ как ответ сервиса.

Причины галлюцинаций:

- недостаточно контекста;
- неясный вопрос;
- устаревшие знания модели;
- противоречивые данные;
- слишком свободная инструкция;
- отсутствие проверки ответа.

Способы уменьшения риска:

- использовать RAG;
- просить модель ссылаться на источники;
- ограничивать ответ только предоставленным контекстом;
- проверять результат правилами;
- использовать классификаторы риска;
- показывать пользователю предупреждение;
- добавлять human-in-the-loop для критичных задач.

14. Валидация и постобработка ответов

Ответ модели часто нужно проверять перед отправкой пользователю.

Примеры проверок:

- является ли ответ валидным JSON;
- не содержит ли запрещённый контент;
- не раскрывает ли персональные данные;
- соответствует ли формат требованиям;
- есть ли ссылки на источники;
- не выходит ли ответ за рамки разрешённой темы.

Пример:

PYTHON

```
import json
```

```
def parse_ai_response( ):
    try:
        data = json.loads( )
        return data
    except json.JSONDecodeError:
        return {
            "error": "Модель вернула некорректный формат"
        }
```

Для критичных систем нельзя просто брать текст модели и сразу показывать пользователю без контроля.

15. Управление состоянием диалога

В классическом чат-приложении история сообщений хранится и передаётся пользователю.

В AI-чате история нужна ещё и модели, чтобы сохранять контекст.

Проблемы:

- история быстро растёт;
- её дорого передавать в модель;
- в истории могут быть персональные данные;

- старые сообщения могут противоречить новым;
- пользователь может внедрить вредные инструкции.

Подходы:

- хранить полную историю в базе;
- передавать модели только последние сообщения;
- делать краткое резюме диалога;
- удалять чувствительные данные;
- отделять системные инструкции от пользовательского текста;
- задавать срок хранения данных.

16. Асинхронность и долгие операции

AI-запросы могут выполняться дольше обычных API-операций.

Причины:

- большая длина prompt;
- длинная генерация;
- нагрузка на провайдера;
- обработка документов;
- генерация изображений или аудио.

Поэтому WEB-приложение должно поддерживать:

- асинхронные задачи;
- очереди;
- статусы выполнения;
- отмену операции;
- повтор при временной ошибке;
- таймауты.

Пример архитектуры:

Frontend



Backend API



Очередь задач



Worker



AI API



Сохранение результата



Frontend получает статус/ответ

Технологии:

- Celery;
- Redis Queue;
- RabbitMQ;
- Kafka;
- BullMQ;
- background tasks;
- WebSocket для уведомлений.

17. Производительность и задержки

Интеграция с большими моделями увеличивает latency.

На задержку влияют:

- размер входного prompt;
- количество документов в контексте;
- выбранная модель;
- режим streaming;
- сетевые задержки;
- нагрузка API;

- постобработка;
- количество последовательных вызовов модели.

Методы оптимизации:

- уменьшать prompt;
- кешировать частые ответы;
- использовать более лёгкие модели для простых задач;
- применять routing между моделями;
- использовать streaming;
- выполнять независимые операции параллельно;
- заранее индексировать документы;
- ограничивать максимальный размер ответа.

18. Стоимость AI-запросов

В отличие от обычной бизнес-логики, вызовы больших моделей могут иметь заметную стоимость.

Стоимость зависит от:

- количества входных токенов;
- количества выходных токенов;
- выбранной модели;
- числа пользователей;
- частоты запросов;
- использования embedding;
- генерации изображений, аудио или видео.

Поэтому нужны:

- лимиты на пользователя;
- квоты;
- тарифные планы;
- мониторинг расходов;
- кеширование;

- выбор модели по сложности задачи;
- ограничение максимального размера prompt;
- защита от злоупотреблений.

Пример:

Простой вопрос → небольшая модель

Сложный анализ документа → большая модель

Критичный ответ → большая модель + проверка

19. Кеширование AI-ответов

Кеширование помогает снизить стоимость и задержку.

Можно кешировать:

- embedding документов;
- результаты поиска;
- ответы на типовые вопросы;
- промежуточные резюме;
- классификацию запросов.

Но есть ограничения:

- персональные ответы нельзя кешировать без контроля;
- устаревшие данные могут привести к неверным ответам;
- нужно учитывать версию prompt и модели;
- одинаковый вопрос в разном контексте может требовать разных ответов.

Ключ кеша может включать:

hash(user_question + prompt_version + model_name + con _version)

20. Безопасность AI-WEB-приложений

AI-WEB-приложение наследует все классические WEB-риски:

- XSS;
- CSRF;
- SQL injection;
- SSRF;
- утечки токенов;
- небезопасная авторизация;
- неправильные CORS-настройки.

И получает новые AI-специфические риски:

- prompt injection;
- jailbreak;
- data exfiltration;
- утечка системных инструкций;
- раскрытие документов из RAG;
- генерация вредных инструкций;
- обход ограничений;
- poisoning базы знаний;
- злоупотребление API-стоимостью.

21. Работа с персональными и конфиденциальными данными

AI-WEB-сервисы часто обрабатывают:

- ФИО;
- e-mail;
- номера телефонов;
- документы;
- медицинские данные;
- финансовые данные;
- корпоративную информацию.

Особенности:

- нельзя отправлять лишние данные во внешний AI-API;
- нужно учитывать законодательство о персональных данных;
- необходимо логировать только безопасную информацию;
- нужно маскировать чувствительные данные;
- требуется контроль доступа;
- важно устанавливать сроки хранения.

Пример маскировки:

Иван Петров, ivan@example.com

можно заменить на:

[USER_NAME], [EMAIL]

22. Авторизация и контроль доступа в RAG

В RAG-системе важно, чтобы пользователь получал ответы только по тем документам, к которым имеет доступ.

Неправильный вариант:

Все документы компании → единая векторная база → поиск для любого пользователя

Правильный вариант:

Документы + права доступа → фильтрация при поиске → только разрешённый контекст → LLM

Пример:

- студент видит только материалы своего курса;
- преподаватель видит материалы своих дисциплин;
- администратор видит всё;
- сотрудник компании видит только документы своего отдела.

Если не учитывать права доступа на этапе retrieval, модель может случайно раскрыть запрещённую информацию.

23. Логирование и мониторинг

Для AI-WEB-приложений нужно логировать (запись в журнал):

- время запроса;
- пользователя;
- выбранную модель;
- размер prompt;
- количество токенов;
- стоимость;
- ошибки;
- время ответа;
- версию prompt;
- версию базы знаний.

Но нельзя бездумно логировать всё содержимое запросов и ответов, потому что там могут быть персональные данные.

Метрики:

- средняя задержка;
- процент ошибок;
- стоимость на пользователя;
- количество отказов;
- качество ответов;
- доля некорректного JSON;
- частота срабатывания фильтров безопасности;
- частота жалоб пользователей.

24. Тестирование AI-WEB-приложений

Тестирование AI-сервисов сложнее, чем классических систем, потому что ответы модели могут немного отличаться.

Нужно тестировать:

- API;
- безопасность;
- prompt;
- RAG-поиск;
- формат ответа;
- устойчивость к prompt injection;
- качество ответов;
- стоимость;
- задержку;
- работу при ошибке модели.

Типы тестов:

1. Unit-тесты обычной логики.
2. Интеграционные тесты AI API.
3. Regression-тесты prompt.
4. Наборы эталонных вопросов.
5. Red-team тестирование.
6. Load-тестирование.
7. A/B-тестирование разных prompt и моделей.

25. Версионирование prompt, моделей и данных

В AI-WEB-приложении результат зависит не только от кода.

Он зависит от:

- версии модели;
- версии prompt;
- версии базы знаний;
- параметров генерации;
- embedding-модели;
- алгоритма retrieval;

- фильтров постобработки.

Поэтому важно фиксировать:

$\text{response} = f(\text{code_version}, \text{prompt_version}, \text{model_version}, \text{data_version})$

Если пользователь пожаловался на ответ, нужно понимать, при каких условиях этот ответ был получен.

26. Мультимодальные WEB-приложения

Большие фундаментальные модели могут работать не только с текстом.

WEB-приложения могут принимать:

- изображения;
- аудио;
- видео;
- PDF;
- таблицы;
- сканы документов.

Примеры функций:

- распознавание текста на изображении;
- анализ медицинских снимков;
- описание изображения;
- голосовые ассистенты;
- автоматическая расшифровка лекций;
- поиск по видео;
- анализ документов.

Особенности WEB-разработки:

- загрузка больших файлов;
- ограничение размера;
- предварительная обработка;
- хранение файлов;

- безопасность загрузок;
- очереди задач;
- асинхронная обработка;
- контроль форматов.

27. AI-агенты и выполнение действий

Следующий этап — AI-агенты, которые не только отвечают, но и выполняют действия:

- создают задачу;
- отправляют письмо;
- делают поиск;
- вызывают API;
- обновляют запись в базе;
- бронируют услугу;
- формируют отчёт.

Особенность: модели нельзя давать неограниченный доступ.

Нужны:

- список разрешённых действий;
- подтверждение пользователя;
- проверка параметров;
- журнал операций;
- откат изменений;
- ограничение прав;
- защита от prompt injection.

Пример безопасного подхода:

Модель предлагает действие → backend валидирует → пользователь подтверждает → backend выполняет

28. Типовая архитектура AI-WEB-сервиса

Frontend

|

| HTTP / WebSocket / SSE

↓

Backend API

|

| Авторизация, лимиты, логирование

↓

AI Orchestration Layer

|

| Prompt templates, routing, safety checks

↓

Retrieval Layer

|

| Поиск по БД / vector DB / документам

↓

AI Provider / LLM

|

| Ответ модели

↓

Postprocessing

|

| Валидация, фильтрация, форматирование

↓

Frontend

Компоненты:

- frontend-интерфейс;

- backend API;
- база пользователей;
- база истории;
- векторная база;
- слой prompt management;
- слой безопасности;
- очередь задач;
- мониторинг;
- AI-провайдер или собственная модель.

29. Основные ошибки при разработке AI-WEB-приложений

Частые ошибки:

1. Вызов AI API напрямую из frontend.
 2. Отсутствие контроля доступа к документам.
 3. Передача конфиденциальных данных в prompt.
 4. Отсутствие лимитов на пользователей.
 5. Игнорирование стоимости токенов.
 6. Слепое доверие ответам модели.
 7. Отсутствие проверки формата ответа.
 8. Нет защиты от prompt injection.
 9. Нет логирования версий prompt и модели.
 10. Нет стратегии обработки ошибок.
 11. Слишком длинный контекст.
 12. Отсутствие тестов качества.
 13. Использование одной большой модели для всех задач.
 14. Отсутствие мониторинга расходов.
 15. Хранение API-ключей в клиентском коде.
-

30. Практические рекомендации

При разработке AI-WEB-сервиса рекомендуется:

- использовать backend как обязательный защитный слой;
- хранить API-ключи только на сервере;
- проектировать prompt как код;
- использовать RAG для ответов по собственным данным;
- проверять права доступа до retrieval;
- ограничивать размер запросов;
- использовать streaming для длинных ответов;
- добавлять лимиты и квоты;
- логировать стоимость и задержки;
- валидировать ответы модели;
- применять кеширование;
- тестировать prompt injection;
- не использовать AI для критичных решений без проверки человеком.

Заключение

Использование больших фундаментальных моделей искусственного интеллекта существенно меняет WEB-программирование.

Главная особенность состоит в том, что в приложение добавляется вероятностный интеллектуальный компонент. Поэтому разработчик должен думать не только о маршрутах, базах данных и интерфейсе, но и о prompt, контексте, токенах, стоимости, безопасности, качестве ответов и ограничениях модели.

Ключевой вывод:

AI-WEB-приложение — это не просто сайт с подключённым чат-ботом. Это система, где WEB-архитектура, данные, безопасность и большие модели искусственного интеллекта должны быть спроектированы совместно.