# Введение

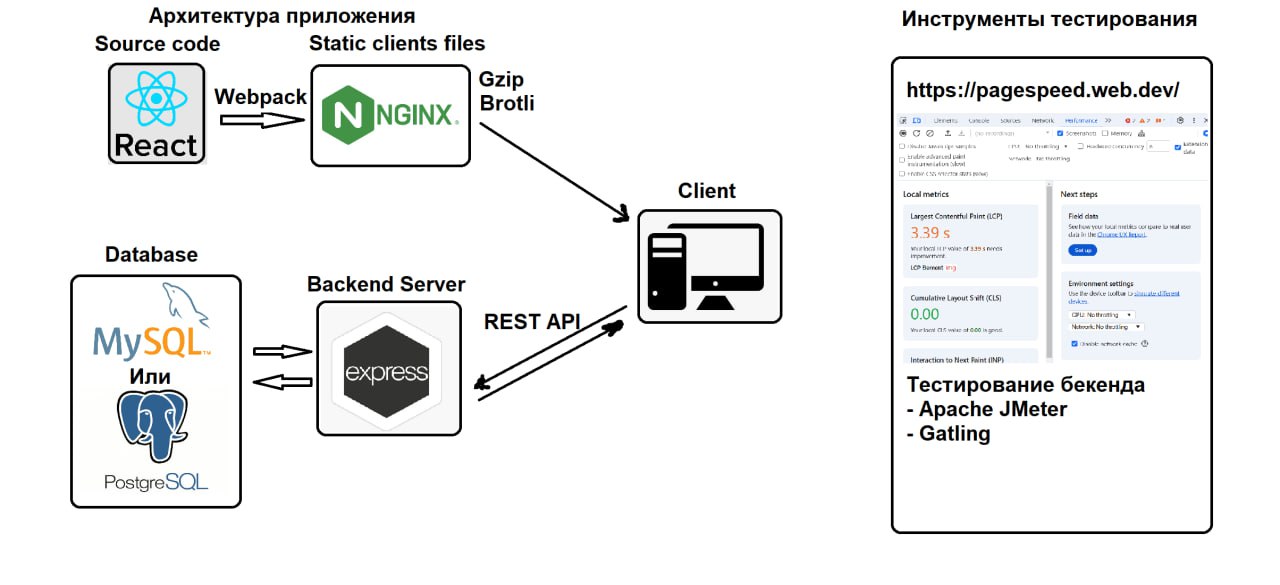
В современном мире скорость загрузки веб-сайта играет критически важную роль для обеспечения положительного пользовательского опыта (User Experience, UX) и достижения высоких позиций в результатах поиска. С увеличением требований пользователей к быстродействию и доступности информации оптимизация скорости разгрузки сайтов стала неотъемлемой частью процесса разработки веб-приложений. Медленная загрузка страниц может привести к ухудшению показателей вовлеченности, увеличению показателя отказов и, как следствие, снижению рейтинга сайта в поисковых системах.

Понимание процесса создания типичного веб-приложения позволяет выявить ключевые этапы, влияющие на скорость загрузки и общую эффективность работы сайта. Обычно разработка начинается с поступления от заказчика функциональных требований и требований к дизайну. Эти требования определяют цели проекта и ожидаемые результаты, служа основой для последующей работы команды разработки.

Далее процесс разработки делится на две основные части: фронтэнд и бекэнд. Фронтэнд-разработчики отвечают за визуальную и интерактивную составляющие приложения. Они настраивают роутинг, разрабатывают пользовательский интерфейс, "красят кнопки", и тесно сотрудничают с дизайнерами для обеспечения эстетической привлекательности и удобства использования сайта. Используя современные фреймворки, такие как React, Vue или Angular, фронтэнд-разработчики пишут код на TypeScript, HTML, Less или SASS. Эти исходные файлы затем компилируются в понятные для браузеров форматы: JavaScript, HTML и CSS, которые являются статическими файлами.

Бекэнд-разработчики, в свою очередь, занимаются реализацией функциональной части приложения. Они отвечают за авторизацию пользователей, обработку данных, интеграцию с базами данных и другими сервисами. Важным этапом является договоренность между фронтэнд- и бекэнд-разработчиками о формате взаимодействия, для чего обычно используется REST API. Это позволяет обеспечить согласованность в обмене данными между клиентской и серверной частями приложения.

После завершения разработки статические файлы фронтэнда размещаются на веб-сервере, таком как Nginx. Стоит отметить, что этот сервер предназначен для обслуживания статических файлов и не следует путать его с бекэнд-сервером, который обрабатывает динамические запросы и выполняет бизнес-логику приложения.



В рамках моей дипломной работы был выбраны архитектура и стек технологий согласно изображению выше.

Одним из ключевых факторов, влияющих на скорость загрузки страницы, является размер передаваемых файлов. Сокращение размера статических файлов путем сжатия позволяет уменьшить время загрузки и, как следствие, улучшить показатели LCP (Largest Contentful Paint) — метрики, отражающей время загрузки наибольшего видимого элемента на странице. Однако использование сжатия требует учета различных факторов. При быстрой скорости интернета и слабой вычислительной мощности устройства пользователя мы можем пожертвовать степенью сжатия для уменьшения времени на декодирование.

При динамическом сжатии, происходящем во время запроса, необходимо также анализировать нагрузку на физический сервер, чтобы избежать снижения производительности из-за дополнительных вычислительных затрат.

Таким образом, эффективное управление степенью сжатия статических файлов в зависимости от текущих параметров системы и запросов пользователей становится важной задачей для оптимизации работы веб-приложения. Динамическое регулирование полосы пропускания позволяет найти баланс между скоростью загрузки и нагрузкой на серверные ресурсы.

В рамках данной работы будет рассмотрена реализация алгоритма динамического управления полосой пропускания. Будут проанализированы существующие методы сжатия, исследованы их влияния на показатели производительности и предложено решение, позволяющее оптимизировать скорость загрузки страниц без ущерба для стабильности и эффективности работы сервера.

**Техническое описание железа пока не добавил. Думаю, стоит упомянуть, когда начну измерять загрузку сервера.**

# Практическая часть

В ходе прохождения практики я занимался оптимизацией фронтенд-части веб-приложения с целью улучшения его производительности и пользовательского опыта. Основное внимание было уделено исследованию влияния степени сжатия статических файлов на время загрузки страницы и ключевые показатели производительности, такие как LCP (Largest Contentful Paint).

## 1. Сборка фронтенд-части приложения

Я начал с подготовки фронтенд-части проекта:

- Компиляция исходных файлов: из исходного кода приложения были скомпилированы статические файлы HTML, CSS, JavaScript и изображения. Для этого использовались современные инструменты сборки и компиляции, обеспечивающие оптимизацию кода и совместимость с различными браузерами.

- Минификация кода: был применен процесс минификации для удаления из кода незначащих символов, пробелов и комментариев, что позволило сократить размер файлов без изменения их функциональности.

## 2. Размещение статических файлов на сервере Nginx

- Настройка сервера: статические файлы были размещены на веб-сервере Nginx. Я настроил сервер для эффективной раздачи контента, учитывая особенности обработки статических ресурсов.

## 3. Применение сжатия Gzip

- Включение сжатия: я настроил сервер Nginx для сжатия передаваемых данных с помощью алгоритма Gzip, который использует комбинацию методов LZ77 и кодирования Хаффмана. Это позволило уменьшить объем передаваемых данных и ускорить загрузку страницы для пользователей.

## 4. Измерение производительности

- Проведение тестов: после настройки сжатия были проведены измерения времени загрузки страницы и показателя LCP с помощью инструментов веб-разработки, таких как Google Lighthouse и встроенных средств браузера.

- Результаты: были получены данные о времени загрузки страницы и о том, как применение сжатия влияет на эти показатели.

## 5. Моделирование различных условий работы

- Замедление сети: я искусственно замедлил соединение с помощью инструментов разработки браузера, чтобы смоделировать работу приложения при низкой скорости интернет-соединения.

- Троттлинг процессора: был применен троттлинг процессора для имитации работы на менее мощных устройствах или при высокой загрузке системы.

## 6. Анализ результатов и выявленные проблемы

- Нестабильность показателей: при проведении тестов я заметил, что показатели времени загрузки и LCP сильно варьируются между запусками, что затрудняет точный анализ влияния оптимизаций.

- Трудность в измерении времени декодирования: не удалось отдельно измерить время, затраченное на декомпрессию сжатых данных на стороне клиента, что важно для оценки нагрузки на устройство пользователя.

## 7. Ограничения при использовании Brotli в Nginx

- Отсутствие поддержки Brotli в бесплатной версии Nginx: попытка использовать более эффективный алгоритм сжатия Brotli столкнулась с проблемой — в бесплатной версии Nginx нет встроенной поддержки Brotli.

- Стоимость коммерческой версии: для подключения Brotli требуется коммерческая подписка Nginx стоимостью около 2500 долларов, что выходит за рамки доступных ресурсов.

## 8. Выводы

- Необходимость баланса между степенью сжатия и производительностью: экспериментально подтверждено, что оптимизация скорости загрузки сводится к поиску баланса между степенью сжатия файлов и нагрузкой на устройство пользователя.

- Влияние условий сети и мощности устройства: оптимальная степень сжатия зависит от скорости сети и мощности устройства пользователя; при медленном соединении и слабых устройствах эффективность сжатия приобретает большее значение.

## 9. Рекомендации и дальнейшие шаги

- Поиск альтернативных решений: рассмотреть возможность использования других веб-серверов или модулей, поддерживающих Brotli без коммерческих лицензий, например, Apache с модулем brotli или использование сторонних модулей для Nginx.

- Улучшение инструментов измерения: использовать более продвинутые инструменты профилирования и мониторинга для получения стабильных и точных показателей производительности.

- Динамическое управление сжатием: исследовать возможность реализации динамического изменения степени сжатия в зависимости от параметров сети и устройства пользователя с использованием адаптивных алгоритмов.

## 10. Закрепление теоретических знаний

- Изучение алгоритмов сжатия: на практике были применены знания о работе алгоритмов сжатия, таких как LZ77 и кодирование Хаффмана в Gzip.

- Понимание работы веб-серверов: настройка Nginx для раздачи статических файлов и сжатия данных углубила понимание принципов работы веб-серверов и протокола HTTP.

## 11. Использованные инструменты и технологии

- Языки и технологии: HTML, CSS, JavaScript.

- Инструменты сборки и минификации: Vite

- Веб-сервер: Nginx.

- Инструменты измерения производительности: Google Lighthouse, Chrome DevTools, WebPageTest.