ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

"НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО"
(УНИВЕРСИТЕТ ИТМО)

ЗАДАНИЕ НА КУРСОВОЙ ПРОЕКТ (РАБОТУ)

Студент Катюшин Станислав Владимирович
(Фамилия, И., О.)

Факультет ПИиКТ Группа P41071

Направление (специальность) 09.04.04 Программная инженерия

Руководитель Государев И.Б., к.п.н., доцент
(Фамилия, И., О., должность,)

Дисциплина Проектирование и анализ языков веб-решений

Наименование темы: Сравнительный анализ скорости рендеринга приложений, использующих CSR и SSR, на примере React и Next.js

Задание Разработать идентичные веб-приложения на React и Next.js, провести сравнительный анализ их производительности

Краткие методические указания (задачи работы)

Исследовать различия Client-Side Rendering (CSR) и Server-Side Rendering (SSR).

Описать задачи, решаемые React и Next.js, а также выявить их различия.

Исследовать особенности Server-Side Rendering (SSR) в Next.js.

Описать создаваемое веб-приложение.

Разработать веб-приложение на React.

Разработать веб-приложение на Next.js.

Описать различия созданных приложений, связанные с особенностями используемых технологий.

Использовать Google Lighthouse для оценки производительности веб-приложений.

Провести сравнительный анализ результатов и выявить закономерности

Разработать страницы отчета с результатами исследования.

Содержание пояснительной записки

Оглавление. Введение. Ход выполнения работы — описание хода исследования, разработки веб-приложений и проведения эксперимента. Заключение. Список использованной литературы. Приложение — скриншоты страниц созданных веб-приложений, а также результатов тестирования.

Руководитель _____ И.Б.Государев
(подпись)

Студент TRACE Катюшин С.В.
(подпись) (Фамилия И.О.)

"НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО"
(УНИВЕРСИТЕТ ИТМО)

ГРАФИК КУРСОВОГО ПРОЕКТА (РАБОТЫ)

Студент Катюшин Станислав Владимирович
(Фамилия, И., О.)

Факультет ПИиКТ Группа P41071

Направление (специальность) 09.04.04 Программная инженерия

Руководитель Государев И.Б., к.п.н., доцент
(Фамилия, И., О., должность)

Дисциплина Проектирование и анализ языков веб-решений

Наименование темы: Сравнительный анализ скорости рендеринга приложений, использующих CSR и SSR, на примере React и Next.js

№ п/п	Наименование этапа	Дата завершения		Оценка и подпись руководителя
		Планируемая	Фактическая	
1.	Исследование различий клиентского и серверного рендеринга. Описание задач, решаемых React и Next.js. Исследование особенностей серверного рендеринга в Next.js. Описание создаваемого веб-приложения.	март	март	
2.	Разработка веб-приложений на React и Next.js. Описание различий приложений, связанных с особенностями используемых технологий. Оценка производительности приложений с помощью Google Lighthouse. Сравнительный анализ результатов. Написание отчета. Защита проекта.	апрель	апрель	

Руководитель _____ И.Б.Государев
(подпись)

Студент TRACE Катюшин С.В.

"НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО"
(УНИВЕРСИТЕТ ИТМО)

АННОТАЦИЯ К КУРСОВОМУ ПРОЕКТУ (РАБОТЕ)

Студент Катюшин Станислав Владимирович
(Фамилия, И., О.)

Факультет ПИиКТ Группа P41071

Направление (специальность) 09.04.04 Программная инженерия

Руководитель Государев И.Б., к.п.н., доцент
(Фамилия, И., О., должность)

Дисциплина Проектирование и анализ языков веб-решений

Наименование темы: Сравнительный анализ скорости рендеринга приложений, использующих CSR и SSR, на примере React и Next.js

ХАРАКТЕРИСТИКА КУРСОВОГО ПРОЕКТА (РАБОТЫ)

1. Цель и задачи работы ☒ Предложены студентом ☐ Определены руководителем

Цель работы — разработать идентичные веб-приложения на React и Next.js, сравнить различия в их производительности с помощью Google Lighthouse

Задачи работы:

1. Исследовать Client-Side Rendering (CSR) и Server-Side Rendering (SSR), а также их различия
2. Описать задачи, решаемые React и Next.js, а также выявить их различия.
3. Исследовать особенности Server-Side Rendering (SSR) в Next.js.
4. Описать создаваемое веб-приложение.
5. Разработать веб-приложение на React.
6. Разработать веб-приложение на Next.js.
7. Описать различия созданных приложений, связанные с особенностями используемых технологий.
8. Использовать Google Lighthouse для оценки производительности веб-приложений.
9. Провести сравнительный анализ результатов и выявить закономерности.
10. Разработать страницы отчета с результатами исследования.

2. Характер работы

☐ Расчет ☐ Конструирование ☐ Моделирование ☒ Другое

3. Содержание работы

Используя полученные знания по дисциплине, созданы идентичные приложения на React и Next.js. Проведён сравнительный анализ их производительности с помощью Google Lighthouse. Сделаны выводы касательно необходимости использования технологий в различных случаях.

4. Выводы

Требования к проекту реализованы

Руководитель _____ И.Б. Государев
(подпись)

Студент TRACE Катюшин С.В.
(подпись)

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

"НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО"
(УНИВЕРСИТЕТ ИТМО)

Факультет программной инженерии и компьютерной техники

Направление (специальность) — 09.04.04 Программная инженерия

Образовательная программа — Веб-технологии

Дисциплина — Проектирование и анализ языков веб-решений

Курсовой проект (работа)

ТЕМА: Сравнительный анализ скорости рендеринга приложений, использующих CSR и SSR, на примере React и Next.js

ВЫПОЛНИЛ

Студент группы P41071 TRACE Катюшин С.В.

№ группы подпись, дата ФИО

ПРОВЕРИЛ к.п.н., доцент Государев И.Б.

должность подпись, дата ФИО

САНКТ-ПЕТЕРБУРГ

2021 г

Содержание

Введение

Прошло немало времени с того момента, как появился протокол передачи данных HTTP. Изменились цели и задачи, которые человечество хотело бы решить с помощью сети Интернет. Выросла целая индустрия, завязанная на веб-технологиях, и едва ли можно встретить человека, не использующего их в своей повседневной жизни.

Требования и пожелания пользователей увеличиваются с каждым годом. Всё это заставляет разработчиков искать новые способы разработки, ускоряющие и улучшающие взаимодействие пользователей с приложениями.

В основе всех веб-приложений лежит клиент-серверная архитектура, позволяющая пользователям сети иметь доступ к удаленным ресурсам, расположенным на удалённом сервере.

Изоморфные веб-фреймворки – тренд последних лет в сфере веб-разработки. Данная технология продолжает философию одностраничного приложения (SPA) - веб-приложения или веб-сайта, использующего единственный HTML-документ как оболочку для всех веб-страниц и организующий взаимодействие с пользователем через динамически подгружаемые HTML, CSS, JavaScript, обычно посредством AJAX – однако решает ряд присущих ей проблем.

Говоря об изоморфных веб-приложениях, мы подразумеваем изоморфный JavaScript, также известный как универсальный JavaScript, - JavaScript-приложения, работающие как на клиенте, так и на сервере.

Изоморфные веб-приложения стали доступны благодаря появлению Node.js. Данная программная платформа, основанная на движке V8 (транслирующем JavaScript в машинный код), превращает JavaScript из узкоспециализированного языка в язык общего назначения.

Многие изоморфные веб-фреймворки находятся в стадии активной разработки, однако крайне популярны в среде разработчиков и в особенности у крупных высокопроизводительных сервисов (Ozon, Netflix, Airbnb, Twitch, Github, Gitlab, Docker, Uber, Hulu, Starbucks, IKEA и другие).

Изоморфные веб-фреймворки позволяют создавать веб-приложения с

улучшенной производительностью и улучшенным пользовательским опытом с помощью дополнительных функций предварительного рендеринга, таких как полноценный рендеринг на стороне сервера (SSR) и статическая генерация страниц (SSG).

Цель работы — разработать идентичные веб-приложения на React и Next.js, использующие разные подходы к рендерингу, сравнить различия в их производительности с помощью Google Lighthouse.

Задачи работы:

- Исследовать Client-Side Rendering (CSR) и Server-Side Rendering (SSR), а также их различия
- Описать задачи, решаемые React и Next.js, а также выявить их различия.
- Исследовать особенности Server-Side Rendering (SSR) в Next.js.
- Описать создаваемое веб-приложение.
- Разработать веб-приложение на React.
- Разработать веб-приложение на Next.js.
- Описать различия созданных приложений, связанные с особенностями используемых технологий.
- Использовать Google Lighthouse для оценки производительности веб-приложений.
- Провести сравнительный анализ результатов и выявить закономерности.
- Разработать страницы отчета с результатами исследования.

1 Основные понятия

- HTML (HyperText Markup Language — «язык гипертекстовой разметки») — стандартизированный язык разметки документов во Всемирной паутине. Большинство веб-страниц содержат описание разметки на языке HTML (или XHTML). Язык HTML интерпретируется браузерами; полученный в результате интерпретации форматированный текст отображается на экране монитора компьютера или мобильного устройства.
- JavaScript — мультипарадигменный язык программирования. Поддерживает объектно-ориентированный, императивный и функциональный стили. Является реализацией спецификации ECMAScript. JavaScript обычно используется как встраиваемый язык для программного доступа к объектам приложений. Наиболее широкое применение находит в браузерах как язык сценариев для придания интерактивности веб-страницам.
- Server-Side Rendering (SSR) - рендеринг на стороне сервера - рендеринг клиентского или универсального (изоморфного) приложения в HTML на сервере.
- Client-Side Rendering (CSR) - рендеринг на стороне клиента - рендеринг приложения в браузере, обычно с использованием DOM.
- Регидратация/гидратация - «загрузка» представлений JavaScript на клиенте так, чтобы они повторно использовали дерево DOM и данные HTML, представленные сервером.
- Предварительный рендеринг - запуск приложения на стороне клиента во время сборки для захвата его исходного состояния в виде статического HTML.
- Time to First Byte (TTFB) - время до первого байта - рассматривается как время между нажатием на ссылку и первым поступающим контентом.
- First Paint (FP) - первый раз, когда любой пиксель становится видимым для пользователя.
- First Contentful Paint (FCP) – время, когда запрашиваемые контент (тело статьи и т.д.) становится видимым.
- Time To Interactive (TTI) – время, когда страница становится интерактивной

(события связаны и т.д.)

- Google Lighthouse – автоматизированный инструмент с открытым исходным кодом для измерения качества веб-страниц. Он может быть запущен на любой веб-странице, общедоступной или требующей аутентификации. Google Lighthouse проверяет производительность, доступность и поисковую оптимизацию страниц.
- React — это декларативная, эффективная и гибкая JavaScript библиотека для создания пользовательских интерфейсов.
- JSX – JavaScript XML (eXtensible Markup Language). JSX упрощает написание и добавление HTML-кода в React.
- Next.js — бесплатный и открытый JavaScript фреймворк, созданный поверх React.js для создания SSR-приложений, созданный компанией Vercel. Помогает создавать пользовательский интерфейс приложений (чаще всего, с помощью React, не придерживаясь его принципа — SPA (Single Page Application)).
- Static Site Generation (SSG) – создание HTML-страницы из шаблонов или компонентов и заданного источника контента на этапе сборки приложения.
- Сеть доставки содержимого (Content Delivery Network или Content Distribution Network, CDN) — географически распределённая сетевая инфраструктура, позволяющая оптимизировать доставку и дистрибуцию содержимого конечным пользователям в сети Интернет. Использование контент-провайдерами CDN способствует увеличению скорости загрузки интернет-пользователями аудио-, видео-, программного, игрового и других видов цифрового содержимого в точках присутствия сети CDN.
- Model-View-Controller (MVC, «Модель-Представление-Контроллер», «Модель-Вид-Контроллер») — схема разделения данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: модель, представление и контроллер — таким образом, что модификация каждого компонента может осуществляться независимо. Модель (Model) предоставляет данные и реагирует на команды контроллера, изменяя своё состояние. Представление (View) отвечает за отображение данных модели пользователю, реагируя на изменения модели. Контроллер (Controller) интерпретирует

действия пользователя, оповещая модель о необходимости изменений.

- DOM (Document Object Model — «объектная модель документа») — это независящий от платформы и языка программный интерфейс, позволяющий программам и скриптам получить доступ к содержимому HTML-, XHTML- и XML-документов, а также изменять содержимое, структуру и оформление таких документов.
- Виртуальный DOM (VDOM) - концепция программирования, в которой идеальное или «виртуальное» представление пользовательского интерфейса хранится в памяти и синхронизируется с «реальным» DOM с помощью библиотеки, такой как ReactDOM. Данный процесс называется согласованием.
- API (программный интерфейс приложения) (application programming interface, API) — описание способов (набор классов, процедур, функций, структур или констант), которыми одна компьютерная программа может взаимодействовать с другой программой.
- Node или Node.js — программная платформа, основанная на движке V8 (транслирующем JavaScript в машинный код), превращающая JavaScript из узкоспециализированного языка в язык общего назначения. Node.js добавляет возможность JavaScript взаимодействовать с устройствами ввода-вывода через свой API, написанный на C++, подключать другие внешние библиотеки, написанные на разных языках, обеспечивая вызовы к ним из JavaScript-кода.
- Ошибка 404 или Not Found («не найдено») — стандартный код ответа HTTP о том, что клиент был в состоянии общаться с сервером, но сервер не может найти данные согласно запросу.
- Унифицированный указатель ресурса (Uniform Resource Locator, сокр. URL)— система унифицированных адресов электронных ресурсов, или единообразный определитель местонахождения ресурса (файла).
- «Клиент — сервер» (client–server) — вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределены между поставщиками услуг, называемыми серверами, и заказчиками услуг, называемыми клиентами. Фактически клиент и сервер — это программное обеспечение. Обычно эти программы расположены на разных вычислительных машинах и

взаимодействуют между собой через вычислительную сеть посредством сетевых протоколов.

- Поисковая оптимизация (search engine optimization, SEO) — комплекс мероприятий по внутренней и внешней оптимизации для поднятия позиций сайта в результатах выдачи поисковых систем по определённым запросам пользователей, с целью увеличения сетевого трафика (для информационных ресурсов) и потенциальных клиентов (для коммерческих ресурсов) и последующей монетизации (получение дохода) этого трафика.

2 Теоретическое описание проблемы

2.1 Исследование CSR и SSR, а также их различий

Для достижения цели работы, прежде всего, следует проанализировать рендеринг в современной веб-разработке. При выборе подхода для рендеринга нужно понимать разницу между возможными вариантами, чтобы не прогадать с производительностью.

Серверный рендеринг

При серверном рендеринге в ответ на запрос на сервере генерируется весь HTML страницы. Это исключает необходимость дополнительных запросов данных со стороны клиента, так как сервер берёт всю работу на себя, прежде чем отправить ответ.

Такой подход позволяет добиться быстрой первой отрисовки (First Paint) и первой содержательной отрисовки (First Contentful Paint). Выполнение логики страницы и рендеринг на сервере позволяют избежать отправки клиенту большого количества JavaScript, что приводит к меньшему времени до интерактивности. И это логично, ведь при серверном рендеринге пользователю отсылаются только текст и ссылки. Этот подход хорошо сработает на широком диапазоне устройств и сетевых условий, а также откроет возможности для интересных браузерных оптимизаций вроде потокового парсинга документа.

Однако у этого подхода есть один существенный недостаток: формирование страницы на сервере занимает определённое время, что может привести к большему времени до первого байта (Time to First Byte).

Сообщество давно дискутирует на тему правильного применения серверного рендеринга против клиентского, но важно помнить, что для одних страниц использовать серверный рендеринг можно, а для других — нет. Некоторые сайты успешно используют смешанный рендеринг.

Многие современные фреймворки, библиотеки и архитектуры позволяют рендерить одно и то же приложение как на клиенте, так и на сервере. Их возможности можно использовать и для серверного рендеринга, однако важно отметить, что архитектуры, в которых рендеринг происходит и на клиенте, и на

сервере, являются отдельным классом решений со своими характеристиками производительности и недостатками.

Статический рендеринг

Статический рендеринг происходит на этапе сборки и предоставляет быструю первую отрисовку (First Paint), первую содержательную отрисовку (First Contentful Paint) и время до интерактивности (Time To Interactive) — при условии, что количество клиентского JavaScript ограничено. В отличие от серверного рендеринга здесь удаётся добиться стабильно быстрого времени до первого байта (Time To First Byte), так как HTML-код страницы не должен генерироваться на лету. Как правило, статический рендеринг подразумевает предварительное создание отдельного HTML-файла для каждого URL. Поскольку HTML-ответы созданы заранее, статический рендеринг можно развернуть на нескольких CDN, чтобы воспользоваться преимуществом кеширования.

Но у такого способа рендеринга есть один недостаток — необходимо заранее создать HTML-файлы для всех возможных URL. Это может быть очень сложно или даже невыполнимо, если вы не можете заранее сказать, какие URL возможны, или если у вас сайт с большим количеством уникальных страниц.

Однако важно понимать разницу между статическим рендерингом и пререндерингом: статически отрендеренные страницы не нуждаются в выполнении большого количества клиентского JS для интерактивности, в то время как пререндеринг улучшает первую (содержательную) отрисовку одностраничного приложения, которое должно быть загружено на клиент, чтобы страницы были действительно интерактивными.

Клиентский рендеринг

Клиентский рендеринг подразумевает рендеринг страниц прямо в браузере с помощью JavaScript. Вся логика, получение данных, шаблонизация и маршрутизация обрабатываются на клиенте, а не на сервере.

Основной недостаток клиентского рендеринга заключается в том, что количество необходимого JavaScript обычно увеличивается вместе с ростом приложения. Ситуация ухудшается с подключением новых JavaScript-библиотек,

полифиллов и прочего стороннего кода, который соревнуется между собой за вычислительные мощности и часто требует обработки, прежде чем содержимое страницы можно будет отобразить. Решениям с клиентским рендерингом, которые полагаются на большие JavaScript-бандлы, стоит рассмотреть сильное разделение кода и ленивую загрузку JavaScript — «загружайте только то, что вам нужно и только когда это нужно». Для решений с минимумом интерактивности или её отсутствием серверный рендеринг может предоставить более масштабируемое решение этих проблем.

Совмещение серверного и клиентского рендеринга с помощью регидратации

Универсальный рендеринг (или просто «SSR») пытается устранить недостатки серверного и клиентского рендеринга, используя оба подхода. Навигационные запросы вроде полной загрузки или перезагрузки страницы обрабатываются сервером, который рендерит приложение в HTML, затем JavaScript и данные, используемые для рендеринга, встраиваются в итоговый документ. При правильной реализации время первой содержательной отрисовки будет как при серверном рендеринге, а повторный рендеринг будет производиться на клиенте с помощью техники, называемой (ре)гидратацией. Это новое решение, тем не менее не лишённое определённых проблем с производительностью.

Основной недостаток универсального рендеринга с регидратацией заключается в том, что такой подход может очень негативно повлиять на время до интерактивности (Time To Interactive) даже при улучшении первой отрисовки (First Paint). Страницы часто выглядят обманчиво готовыми и интерактивными, но по факту не могут никак реагировать на действия пользователя до выполнения JS на стороне клиента и присоединения обработчиков событий. Это может занять несколько секунд или даже минут на мобильных устройствах.

2.2 React и Next.js. Решаемые задачи и различия

React

React – самая популярная библиотека JavaScript для разработки пользовательского интерфейса (UI). Компоненты этого инструмента были разработаны Facebook.

React используется сотнями крупных компаний по всему миру, включая Netflix, Airbnb, American Express, Facebook, WhatsApp, eBay и Instagram.

React – это JavaScript библиотека с открытым исходным кодом, сосредоточенная на одной конкретной цели – эффективном выполнении задач в рамках разработки пользовательского интерфейса. Его можно отнести к категории “V” в архитектурном шаблоне MVC (модель-вид-контроллер).

React позволяет повторно использовать компоненты, которые были разработаны в других приложениях, использующих ту же функцию.

Компонент React создать проще, поскольку он использует JSX, опциональное расширение синтаксиса JavaScript, которое позволяет комбинировать HTML с JavaScript.

JSX – это отличная смесь JavaScript и HTML. Оно делает весь процесс написания структуры сайта более понятным. Хотя JSX может быть не самым популярным расширением синтаксиса, оно доказало свою эффективность при разработке специальных компонентов или приложений большого объема.

React эффективно обновляет процесс DOM (объектная модель документа). React позволяет создавать виртуальные DOM и размещать их в памяти. В результате каждый раз, когда происходит изменение в реальном DOM, виртуальный меняется немедленно.

«Виртуальный DOM» является скорее паттерном, чем конкретной технологией. В мире React термин «виртуальный DOM» обычно ассоциируется с элементами React, поскольку они являются объектами, представляющими пользовательский интерфейс. React, однако, также использует внутренние объекты, называемые «волокнами» («fibers»), для хранения дополнительной информации о дереве компонентов. Они также могут считаться частью реализации «виртуального DOM» в React.

Next.js

Next.js позволяет создавать приложения на порядок проще, чем с помощью React, потому что предоставляет решения для общих проблем и с которыми можно столкнуться в каждом проекте на React.

На официальном сайте сказано, что Next.js является The React Framework for Production. В этой фразе 3 ключевых слова: React, Framework, Production. React – потому что по-прежнему пишется код на React, создаются React компоненты, используется функционал React (props, state, context). Framework – потому что Next, по сути, расширяет приложения на React, приносит новый функционал и паттерны решения общих проблем (например, файловая маршрутизация). Production – потому что Next.js решает множество проблем «из коробки», с которыми можно столкнуться в каждом проекте на React (предлагает решения для оптимизации изображений, аутентификации).

В парадигме React приложений клиент изначально получает только скелет страницы с входными точками, в которые загружается само приложение. И дальше пройдёт какое-то время, чтобы загрузить необходимый контент. Данный подход называется клиентским рендерингом. И это может быть проблемой для пользовательского опыта. Разумеется, не всегда, но в некоторых случаях определенно, если есть необходимость и бизнес-логика отдавать отрендеренный контент сразу же с загрузкой страницы. Ещё более важно, клиентский рендеринг может быть проблем для SEO - поисковой оптимизации, так как клиент, в том числе и поисковые роботы, изначально получает пустую и не заполненную контентом страницу. Опять же не всегда есть необходимость сразу же отдавать заполненную страницу, в случае с какой-нибудь необщедоступной информацией или информацией, требующей аутентификации, но в случае с, например, новостными порталами или сервисами объявлений есть необходимость сделать страницы с распознаваемыми поисковыми роботами контентом.

2.3 Особенности Server-Side Rendering (SSR) в Next.js

Server-Side Rendering позволяет Next.js пререндерить React компоненты на сервере (стандартное поведение фреймворка). Конечно, существуют способы добиться серверного рендеринга компонентов и средствами React, однако Next.js позволяет всё это настроить достаточно просто и «из коробки». И в то же время мы не лишаемся всех преимуществ React, связанных с клиентским рендерингом благодаря, так называемому, процессу гидратации (или же регидратации). Пройдёт некоторое время, прежде чем страница станет интерактивной, однако это имеет значение только для пользователя, а не поискового робота. А непосредственно пререндеринг касается только изначальной загрузки или обновления страницы.

Две формы пререндеринга в Next.js:

- Статическая генерация (Static Generation)
- Рендеринг на сервере (Server-Side Rendering)

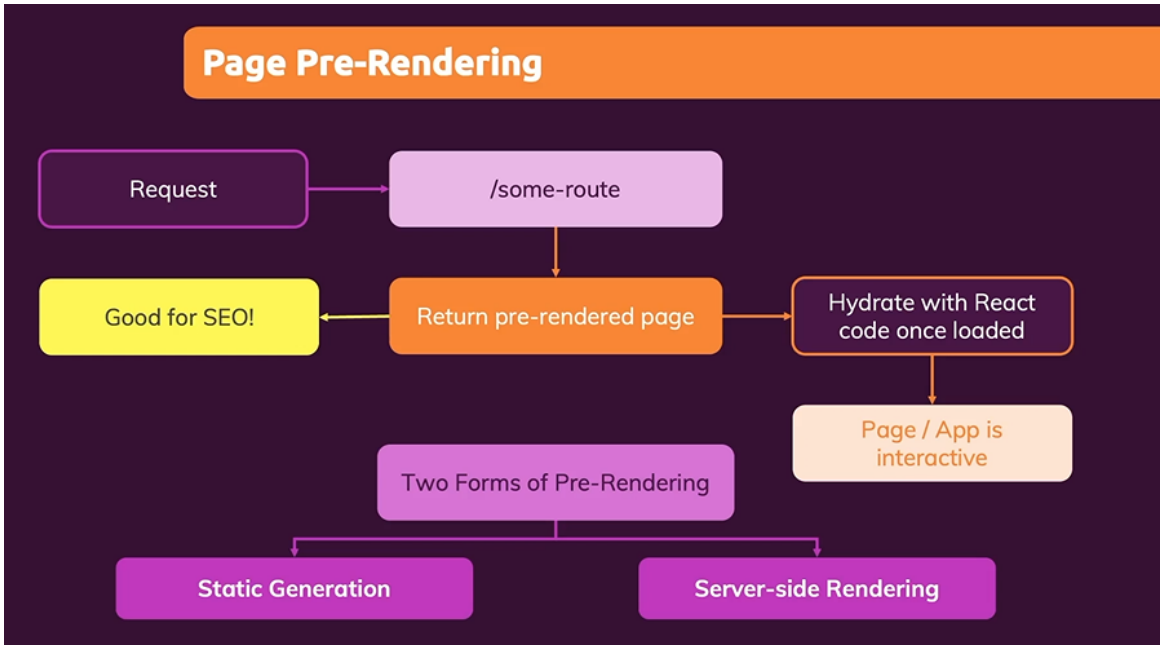


Рис. 1 – Пререндеринг в Next.js

Статическая генерация

При статической генерации страница будет пререндерена во время сборки приложения. Тем самым, страница будет заготовлена заранее, может быть закэширована сервером или CDN, обслуживающим сервер.

Для этого существует специальная зарезервированная функция getStaticProps:

```
export async function getStaticProps() {
    const filePath = path.join(process.cwd(), "data", "dummy-backend.json");
    const jsonData = await fs.readFile(filePath);
    const data = JSON.parse(jsonData);

    return {
        props: {
            products: data.products,
        },
    };
}
```

Рис. 2 – пример функции getStaticProps

Данная функция будет исполнена исключительно на сервере, потому также имеет доступ к пакетам node.js, не доступных в браузере (fs, path, например). Она позволяет подготовить данные для компонента и передать их в качестве props.

Если данные на пререндеренной странице не изменяются, то это превосходный вариант. Однако статическая генерация не подходит, если данные всё же динамически изменяются. В таком случае существуют два решения:

- Подгружать статичную страницу, далее в фоновом режиме на стороне клиента подгрузить обновленные данные.
- Использовать инкрементальную статическую генерацию.

  Инкрементальная статическая генерация (Incremental Static Generation)

  Можно установить определенный «таймер» для частоты обновления страницы, использующей getStaticProps. Если к моменту нового запроса, количество времени, прошедшего с последнего пререндеринга страницы, не пересекло обозначенный лимит, то клиенту отдастся ранее пререндеренная страница. В обратном же случае генерируется, хранится и отдаётся новая страница.

  Для этого достаточно внутри возвращаемого объекта функции getStaticProps указать значение revalidate.

  Кроме входных данных для компонента (props) и значения revalidate, можно возвращать значение notFound: true, что обернётся возвращением 404 ошибки, или же redirect, позволяющую перенаправить запрос на другую

страницу.

Также у данной функции есть доступ к контексту приложения и в том числе к параметрам URL.

Для динамических страниц в Next.js стандартным поведением является не не генерировать страницу заранее. Добавляя getStaticProps к компоненту, мы обозначаем намерение сгенерировать данную страницу во время развертывания приложения. Возникает очевидная проблема с динамическими страницами, не имеющих строгих URL заранее и даже примерного количества страниц, которое необходимо сгенерировать заранее. Для таких случаев существует ещё одна функция getStaticPaths.

```
export async function getStaticPaths() {
  return {
    paths: [
      { params: { pid: 'p1' } },
      { params: { pid: 'p2' } },
      { params: { pid: 'p3' } }
    ],
    fallback: false
  };
}
```

Рис. 3 – Пример функции getStaticPaths

Данная функция позволяет определить параметры тех страниц с динамическими URL, которые необходимо сгенерировать заранее.

Также можно контролировать поведения React компонента с помощью параметра fallback, описывающего поведение в случае поступившего запроса с параметрами, не указанными в getStaticPaths. При значении fallback: false будет возвращаться 404 ошибка, fallback: true – произведена попытка сгенерировать страницу «на лету», прежде отдав клиенту компонент без подгруженных данных из getStaticProp, а в случае fallback: ‘blocking’ – Next.js будет ожидать подгрузки всех необходимых данных на сервере, и только после их получения отдаст пользователю готовую страницу.

Server-Side Rendering

Иногда есть необходимость генерировать страницу каждый раз, когда

поступает соответствующий запрос, или же иметь доступ к объекту запроса (например, для доступа к cookies). В таком случае в Next.js существует специальная функция getServerSideProps.

```
// for every incoming request
export async function getServerSideProps(context) {
  return {
    props: {
      username: 'Max'
    }
  };
}
```

Рис.4 – Пример функции getServerSideProps

Данная функция очень похожа на getStatisProps с отличием в рендеринге и расширенном доступе к контекстной информации, имея доступ к объектам req и res.

Также можно совмещать подгрузку данных на сервере и на клиенте в зависимости от задач и бизнес-логика, стоящей перед приложением.

На рисунке ниже представлен консольный вывод при развертывании Next.js приложения. В данном выводе всегда предоставляется исчерпывающая информация относительно всех страниц приложения, в том числе список всех заранее сгенерированных, их вес.

```
Page                                Size     First Load JS
┌ ● /                               775 B           109 kB
├   /_app                           0 B             102 kB
├ ⧗ /404                            3.68 kB         106 kB
└ ● /blocking                       396 B           108 kB
+ First Load JS shared by all       102 kB
  ├ chunks/168.e0b106.js            7.14 kB
  ├ chunks/296.355108.js            20.2 kB
  ├ chunks/433.0e5935.js            11.1 kB
  ├ chunks/725.9bba23.js            17.8 kB
  ├ chunks/757.4111c4.js            2.43 kB
  ├ chunks/framework.4dbbad.js      42 kB
  ├ chunks/main.3a5d5f.js           158 B
  ├ chunks/pages/_app.339083.js     545 B
  └ chunks/webpack.6cc645.js        1.03 kB

λ  (Server)  server-side renders at runtime (uses getInitialProps or getServerSideProps)
⧗  (Static)  automatically rendered as static HTML (uses no initial props)
●  (SSG)     automatically generated as static HTML + JSON (uses getStaticProps)
   (ISR)     incremental static regeneration (uses revalidate in getStaticProps)
```

Рис.5 – Консольный вывод при развёртывании Next.js приложения

2.4 Описание создаваемого приложения

В ходе данной курсовой работы будут созданы 2 идентичных приложения на React и Next.js для проведения эксперимента и сравнения их производительности. В каждом приложении будет 2 страницы одинакового содержимого, но с разными подходами к рендерингу.

Для тестовых данных будет использоваться сторонний API SWAPI (The Star Wars API), а именно запрос https://swapi.dev/api/people, возвращающий постранично персонажей вселенной «Звёздных воин».

В рамках эксперимента мы хотели бы отобразить достаточное количество элементов на странице, а также получать данные для этих элементов у стороннего источника ради большей показательности результатов эксперимента.

```
HTTP/1.0 200 OK
Content-Type: application/json
{
    "birth_year": "19 BBY",
    "eye_color": "Blue",
    "films": [
        "https://swapi.dev/api/films/1/",
        ...
    ],
    "gender": "Male",
    "hair_color": "Blond",
    "height": "172",
    "homeworld": "https://swapi.dev/api/planets/1/",
    "mass": "77",
    "name": "Luke Skywalker",
    "skin_color": "Fair",
    "created": "2014-12-09T13:50:51.644000Z",
    "edited": "2014-12-10T13:52:43.172000Z",
    "species": [
        "https://swapi.dev/api/species/1/"
    ],
    "starships": [
        "https://swapi.dev/api/starships/12/",
        ...
    ],
    "url": "https://swapi.dev/api/people/1/",
    "vehicles": [
        "https://swapi.dev/api/vehicles/14/"
        ...
    ]
}
```

Рис.6 – Пример ответа от SWAPI

Наша задача отобразить всех персонажей вселенной на одной странице. На момент написания курсовой работы (13.05.2021) SWAPI возвращал максимум 10 персонажей на один запрос.

В Next.js одна страница будет полностью сгенерирована на сервере, включая совершение всех необходимых запросов также на стороне сервера. На второй же странице заранее сгенерировано будет только то количество элементов, которое возвращается в ответ на один запрос к SWAPI. Все же остальные запросы будут сделаны уже на стороне клиента.

В React же будет похожая логика приложения, однако реализованная с помощью инструментов исключительно React. Все запросы к SWAPI будут осуществляться на стороне клиента. В первом случае любая новая информация, получаемая в ответ на запросы к SWAPI, будет отображаться на странице. На второй странице клиент получит отрендеренную страницу только после того, как будут успешно совершены все запросы. Разумеется, это не лучшая практика и явно не пример того кода, который необходимо использовать в производстве, однако хотелось бы сравнить показатели данной страницы с полной генерацией контента на сервере.

Информация о персонажах вселенной «Звёздных войн», очевидно, не является динамически меняющейся, поэтому в Next.js будет использована статическая генерация с помощью функции getStaticProps.

Для обоих приложений будет использована библиотека компонентов Chakra, которая легко подключается к любым приложениям на базе React, включая Next.js. Она потребуется для отображения контента в удобном и адаптивном для различных устройств виде без какой-либо стилизации с нашей стороны.

Также для React потребуется дополнительная библиотека React Router.

3. Практическая часть

3.1 Разработка веб-приложения на React

Структура проекта на React проста:

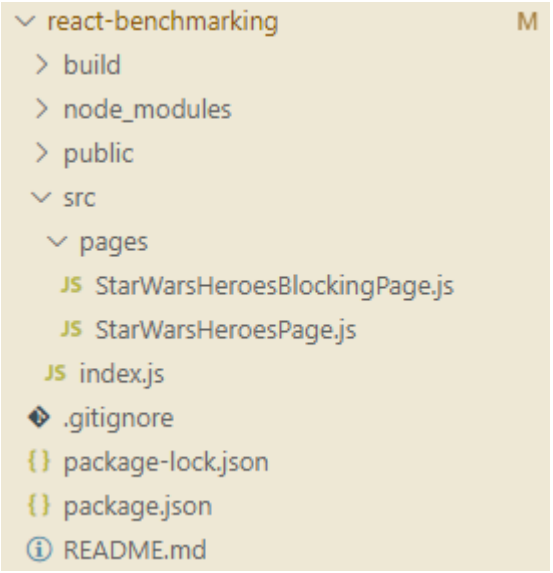


Рис.7 – Структура проекта на React

Дополнительно были установлены только React Router и библиотека компонентов Chakra. В injex.js описана маршрутизация приложения и рендеринг его в документ.

```
import React from "react";
import ReactDOM from "react-dom";
import StarWarsHeroesPage from "./pages/StarWarsHeroesPage";
import StarWarsHeroesBlockingPage from "./pages/StarWarsHeroesBlockingPage";
import { BrowserRouter, Route, Switch } from "react-router-dom";

ReactDOM.render(
    <BrowserRouter>
        <Switch>
            <Route path="/" exact>
                <StarWarsHeroesPage />
            </Route>
            <Route path="/blocking">
                <StarWarsHeroesBlockingPage />
            </Route>
        </Switch>
    </BrowserRouter>,
    document.getElementById("root")
);
```

Рис.8 – index.js-файл приложения на React

Оба компонента страниц возвращают либо null при отсутствии данных, либо

стилизованную таблицу из библиотеки компонентов Chakra, содержащую все данные персонажа (Имя, Рост, Вес).

```
if (data.length === 0) return null;

return (
    <ChakraProvider>
        <Table>
            <Thead>
                <Tr>
                    <Th>Name</Th>
                    <Th isNumeric>Height</Th>
                    <Th isNumeric>Mass</Th>
                </Tr>
            </Thead>
            <Tbody>
                {data.map((item) => (
                    <Tr key={item.url}>
                        <Td>{item.name}</Td>
                        <Td isNumeric>{item.height}</Td>
                        <Td isNumeric>{item.mass}</Td>
                    </Tr>
                ))}
            </Tbody>
        </Table>
    </ChakraProvider>
);
```

Рис.9 – возвращаемые объекты компонентов страниц приложения на React

На одной странице реализована логика последовательной загрузки и отображения содержимого, а на другой отображение содержимого только после того, как все данные будут получены.

```
const [data, setData] = useState([]);
useEffect(() => {
    const fetchData = async () => {
        let results = [];
        let next = "https://swapi.dev/api/people/?page=1";
        while (next) {
            const resp = await fetch(next);
            const respData = await resp.json();
            results = results.concat(respData.results);
            next = respData.next;
            setData(results);
        }
    };
    fetchData();
}, []);
```

Рис.10 – загрузка данных в компоненте стартовой страницы (последовательное отображение)

3.2 Разработка веб-приложения на Next.js

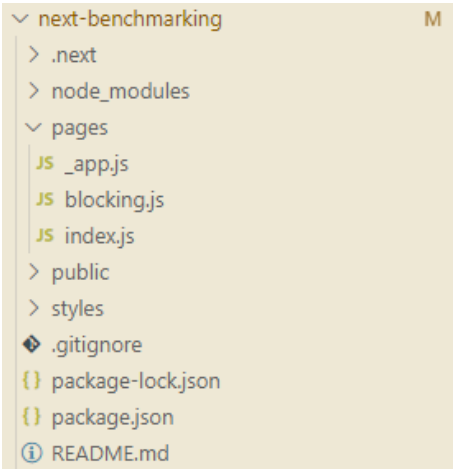


Рис.11 – структура проекта на React

Так как Next.js является фреймворком с файловой маршрутизацией, в приложении не требуется дополнительно подключать какую-либо библиотеку рутинга, а также заниматься программным описанием маршрутизации.

Внутри директории pages файл index.js будет ответственен за главную страницу /, где будет реализован частичный рендеринг (первый запрос на сервере, остальные на клиенте), а blocking.js – за страницу /blocking, где все запросы будут отправлены на сервере и сгенерирован весь контент во время развёртывания приложения. Также мы подключили библиотеку компонентов Chakra.

Функция компонента очень схожа с приложением на React:

```
function FullPrerenderPage(props) {
    const { data } = props;

    return (
        <Table>
            <Thead>
                <Tr>
                    <Th>Name</Th>
                    <Th isNumeric>Height</Th>
                    <Th isNumeric>Mass</Th>
                </Tr>
            </Thead>
            <Tbody>
                {data.map((item) => (
                    <Tr key={item.url}>
                        <Td>{item.name}</Td>
                        <Td isNumeric>{item.height}</Td>
                        <Td isNumeric>{item.mass}</Td>
                    </Tr>
                ))}
            </Tbody>
        </Table>
    );
}
```

Рис.12 – функция компонента blocking.js (генерация всего контента заранее) приложения на Next.js

Добавилась только функция getStaticProps, позволяющая подгрузить весь необходимый контент (или его часть в случае с главной страницей).

```
export async function getStaticProps() {
    let data = [];
    let next = "https://swapi.dev/api/people/?page=1";
    while (next) {
        const resp = await fetch(next);
        const respData = await resp.json();
        data = data.concat(respData.results);
        next = respData.next;
    }

    return {
        props: {
            data: data,
        },
    };
}
```

Рис. 13 – функция getStaticProps у blocking.js приложения на Next.js

```
export async function getStaticProps() {
    let data = [];
    let next = "https://swapi.dev/api/people/?page=1";
    const resp = await fetch(next);
    const respData = await resp.json();
    data = data.concat(respData.results);
    next = respData.next;

    return {
        props: {
            data: data,
            next: next,
        },
    };
}
```

Рис. 14 – функция getStaticProps у index.js приложения на Next.js

В случае смешанного рендеринга совершается на сервере первый запрос, а в компонент передается URL следующего необходимого запроса. Далее данные последовательно подгружаются и рендерятся уже на клиенте.

```
let { data, next } = props;
const [dataToRender, setData] = useState(data);
useEffect(() => {
    const fetchData = async () => {
        let results = dataToRender;
        while (next) {
            const resp = await fetch(next);
            const respData = await resp.json();
            results = results.concat(respData.results);
            next = respData.next;
            setData(results);
        }
    };
    fetchData();
}, []);
```

Рис. 15 – загрузка и отображение данных на клиенте в случае смешанного рендеринга на главной странице приложения на Next.js

3.3 Проведение эксперимента

Приложения были развернуты на облачном сервисе Vscale от компании Selectel. Для оценки производительности страниц потребовался инструмент Google Lighthouse. Использовался пресет для мобильных устройств. Для каждой страницы производилось 3 замера и посчитано среднее значение.

#	First Contentful Paint	Speed Index	Largest Contentful Paint	Time to Interactive	Performance
1	0,6c	0,8c	0,6c	2,3c	98
2	0,6c	0,7c	0,6c	2,4c	98
3	0,6c	0,7c	0,6c	2,3c	98

Табл. 1 - Результаты производительности страницы приложения Next.js, использующую смешанный рендеринг, по оценке Google Lighthouse

#	First Contentful Paint	Speed Index	Largest Contentful Paint	Time to Interactive	Performance
1	0,6c	0,6c	0,6c	2,1c	100
2	0,6c	0,6c	0,6c	2,1c	100
3	0,6c	0,6c	0,6c	2,1c	100

Табл. 2 - Результаты производительности страницы приложения Next.js, использующую статическую генерацию на стороне сервера, по оценке Google Lighthouse

#	First Contentful Paint	Speed Index	Largest Contentful Paint	Time to Interactive	Performance
1	1,5c	1,6c	2,3c	2,0c	97
2	1,6c	1,6c	2,3c	2,1c	96
3	1,5c	1,6c	2,3c	2,0c	96

Табл. 3 - Результаты производительности страницы приложения React, использующую постепенные загрузку и рендеринг данных на клиенте, по оценке Google Lighthouse

#	First Contentful Paint	Speed Index	Largest Contentful Paint	Time to Interactive	Performance
1	1,6c	2,8c	3,0c	1,7c	94
2	1,6c	3,0c	3,1c	1,7c	93
3	1,5c	3,7c	3,1c	1,6c	92

Табл. 4 - Результаты производительности страницы приложения React, осуществляющую рендеринг данных после их полной загрузки, по оценке Google Lighthouse

С примерами интерфейса и вывода результатов Google Lighthouse можно ознакомиться в приложении.

Пояснение по критериям:

- FCP измеряет, сколько времени требуется браузеру для отображения первого фрагмента содержимого DOM после перехода пользователя на страницу. Изображения, небелые элементы <canvas> и SVG на странице считаются содержимым DOM; все, что находится внутри iframe, не включено.
- LCP измеряет, когда самый большой элемент содержимого отображается на экране. Это примерно соответствует утверждению, когда основное содержимое страницы видно пользователям.
- Speed Index измеряет скорость визуального отображения контента во время загрузки страницы. Сначала Lighthouse захватывает видео загрузки страницы в браузере и вычисляет визуальную прогрессию между кадрами.
- TTI измеряет, сколько времени требуется странице, чтобы стать полностью интерактивной. Страница считается полностью интерактивной, если:

1) Страница визуально заполнена;
2) Обработчики событий зарегистрированы для большинства видимых элементов страницы;
3) Страница реагирует на действия пользователя в течение 50 миллисекунд.

Страница	First Contentful Paint	Speed Index	Largest Contentful Paint	Time to Interactive	Performance	Исходный код страницы
Next.js (SSG + CSR)	0,6с	0,73с	0,6с	2,33с	98	Частично содержательный
Next.js (полностью SSG)	0,6с	0,6с	0,6с	2,1с	100	Содержательный
React (CSR)	1,53с	1,6с	2,3с	2,03с	96,33	Несодержательный
React (блокирующий CSR)	1,57с	3,17с	3,07с	1,67с	93	Несодержательный

Табл. 5 – Итоговая таблица результатов эксперимента

3.4 Анализ результатов эксперимента

По показателю First Contentful Paint лучшие результаты у страниц приложения на Next.js, что было ожидаемо, как и показатели Speed Index и Largest Contentful Paint.

Абсолютным лидером по производительности является страница, которая была полностью сгенерирована на сервере во время развертывания приложения. Все показатели First Contentful Paint, Speed Index и Largest Contentful Paint демонстрируют одинаковое время.

В нашем примере никаких преимуществ не дал смешанный рендеринг на Next.js, а только ухудшил результаты полностью сгенерированный на сервере страницы.

Приложение же на React показало худшие результаты по показателям First Contentful Paint, Speed Index и Largest Contentful Paint. Однако стоит заметить, что у данных страниц есть преимущество в показателе Time to Interactive при том, что на страницах практически нет никаких обработчиков событий, а приложение на Next.js оказалось медленнее в этом вопросе, чем приложение на React. При увеличенном количестве различных обработчиков событий на странице разрыв между приложениями на React и Next.js стал бы увеличиваться.

Также кроме стандартных показателей от Google Lighthouse я добавил столбец с информацией о том, насколько содержательным является исходный код страницы, что важно для поисковой оптимизации (SEO) и поисковых роботов. Это принципиальный вопрос в споре о выборе между React и Next.js.

Данные результаты релевантны страницам приложений с относительно статическим контентом, меняющимся крайне редко или же никогда. Страницы с динамически меняющимся контентом или контентом, требующем аутентификации, необходимо исследовать отдельно.

Заключение

В ходе реализации курсовой работы были решены все поставленные задачи. Исследованы современные технологии для создания SPA приложений (React, Next.js), а также различия между подходами клиентского и серверного рендеринга. Созданы 2 идентичных приложения на React и Next.js, использующие различные подходы в рендеринге, в соответствии с различиями технологий и развернуты на облачном сервере от Vscale. Произведена оценка производительности страниц приложений с помощью продукта Google Lighthouse. Результаты были проанализированы и сделаны соответствующие выводы.

Несмотря на все удобства и преимущества, которые приносят в разработку такие изоморфные фреймворки как Next.js, серверный рендеринг не является панацеей. Его динамическая природа может сопровождаться множественными вычислительными затратами. Тот факт, что серверный рендеринг может показать что-то быстрее, вовсе не означает, что нужно проделать меньше вычислительной работы.

Основной недостаток рендеринга изоморфных фреймворков, использующих регидратацию, заключается в том, что такой подход может очень негативно повлиять на время до интерактивности (Time To Interactive) даже при улучшении первой отрисовки (First Paint). Страницы часто выглядят обманчиво готовыми и интерактивными, но по факту не могут никак реагировать на действия пользователя до выполнения JS на стороне клиента и присоединения обработчиков событий.

Стоит также не забывать про преимущество содержательного исходного кода, наполненного контентом, у изоморфных веб-фреймворков. Для каких-то страниц это может быть критически важным преимуществом.

Важно выбирать технологии в зависимости от требований, выдвигаемых приложению.

Список литературы

1. Рендеринг в Интернете | Web | Google Developers [Электронный ресурс]. – Режим доступа: https://developers.google.com/web/updates/2019/02/rendering-on-the-web?hl=ru (Дата обращения: 12.05.2021).
2. Google Lighthouse – Wikipedia [Электронный ресурс]. – Режим доступа: https://en.wikipedia.org/wiki/Google_Lighthouse (Дата обращения: 12.05.2021).
3. Введение: Знакомство с React – React [Электронный ресурс]. – Режим доступа: https://ru.reactjs.org/tutorial/tutorial.html#what-is-react (Дата обращения: 12.05.2021).
4. Next.js – Википедия [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Next.js (Дата обращения: 12.05.2021).
5. Серверный или клиентский рендеринг на вебе: что лучше использовать у себя в проекте и почему | Techrocks [Электронный ресурс]. – Режим доступа: https://techrocks.ru/2019/02/24/server-side-and-client-side-rendering/ (Дата обращения: 12.05.2021).
6. Static Site Generator | Gatsby [Электронный ресурс]. – Режим доступа: https://www.gatsbyjs.com/docs/glossary/static-site-generator/ (Дата обращения: 12.05.2021)
7. Content Delivery Network – Википедия [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Content_Delivery_Network (Дата обращения: 12.05.2021)
8. Что Такое React и Как Он Работает на Самом Деле? [Электронный ресурс]. – Режим доступа: https://www.hostinger.ru/rukovodstva/chto-takoe-react (Дата обращения: 12.05.2021)
9. Model-View-Controller – Википедия [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Model-View-Controller (Дата обращения: 12.05.2021)
10. HTML – Википедия [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/HTML (Дата обращения: 12.05.2021)
11. JavaScript – Википедия [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/JavaScript (Дата обращения: 12.05.2021)
12. React JSX [Электронный ресурс]. – Режим доступа: https://www.w3schools.com/react/react_jsx.asp (Дата обращения: 12.05.2021)
13. Виртуальный DOM и внутренние механизмы [Электронный ресурс]. – Режим доступа: https://learn-reactjs.ru/faq/virtual-dom-and-internals (Дата обращения: 12.05.2021)
14. Document Object Model – Википедия [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Document_Object_Model (Дата обращения: 12.05.2021)
15. Next.js by Vercel – The React Framework [Электронный ресурс]. – Режим доступа: https://nextjs.org/ (Дата обращения: 12.05.2021)
16. Node.js – Википедия [Электронный ресурс]. – Режим доступа:

https://ru.wikipedia.org/wiki/Node.js (Дата обращения: 12.05.2021)

17. Ошибка 404 – Википедия [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/%D0%9E%D1%88%D0%B8%D0%B1%D0%BA%D0%B0_404 (Дата обращения : 12.05.2021)

18. URL – Википедия [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/URL (Дата обращения: 12.05.2021)

19. Клиент-сервер – Википедия [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B8%D0%B5%D0%BD%D1%82_%E2%80%94_%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80 (Дата обращения: 12.05.2021)

20. API – Википедия [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/API (Дата обращения: 13.05.2021)

21. Поисковая оптимизация – Википедия [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B8%D1%81%D0%BA%D0%BE%D0%B2%D0%B0%D1%8F_%D0%BE%D0%BF%D1%82%D0%B8%D0%BC%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F (Дата обращения: 14.05.2021)

22. Vscale. Облачные серверы для разработчиков. [Электронный ресурс]. – Режим доступа: https://vscale.io/ru/ (Дата обращения: 14.05.2021)

23. First Contentful Paint [Электронный ресурс]. – Режим доступа: https://web.dev/first-contentful-paint/?utm_source=lighthouse&utm_medium=devtools (Дата обращения: 14.05.2021)

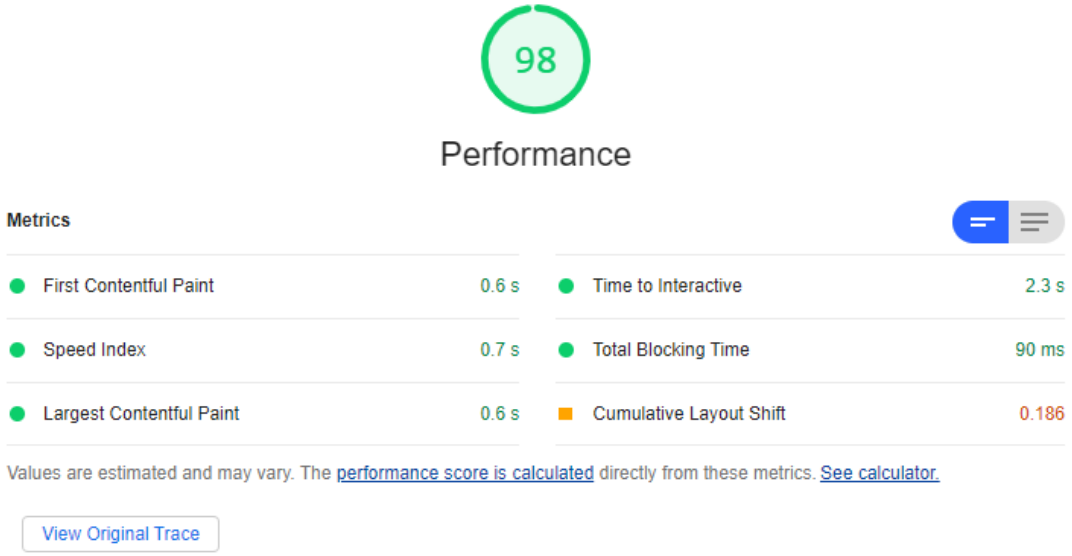
24. Speed Index [Электронный ресурс]. – Режим доступа: https://web.dev/speed-index/?utm_source=lighthouse&utm_medium=devtools (Дата обращения: 14.05.2021)

25. Largest Contentful Paint [Электронный ресурс]. – Режим доступа: https://web.dev/lighthouse-largest-contentful-paint/?utm_source=lighthouse&utm_medium=devtools (Дата обращения: 14.05.2021)

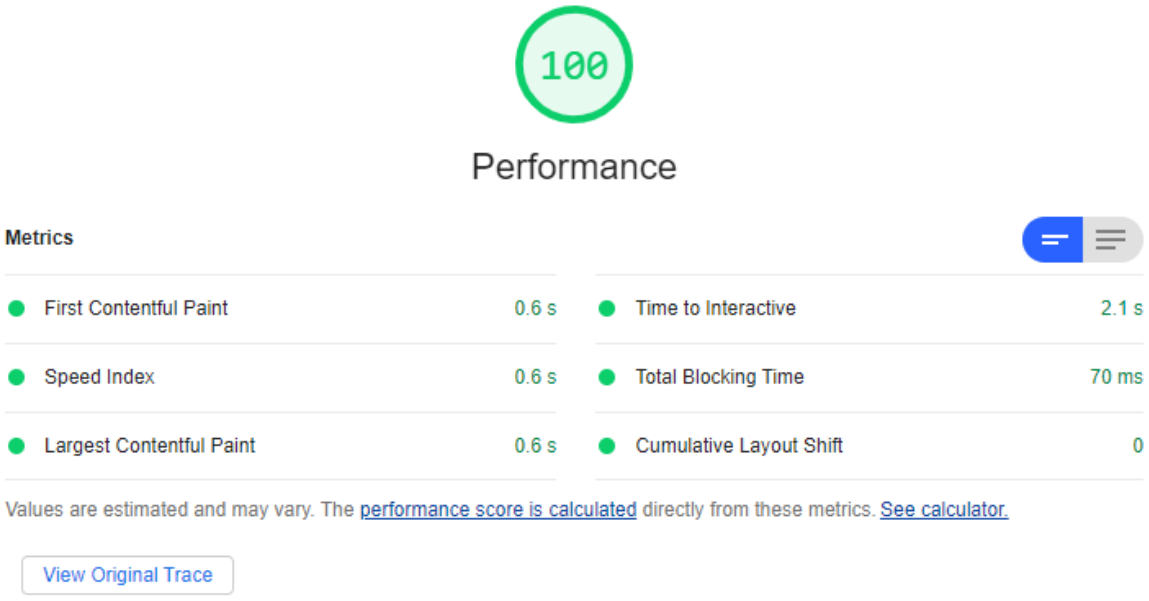
26. Time to Interactive [Электронный ресурс]. – Режим доступа: https://web.dev/interactive/?utm_source=lighthouse&utm_medium=devtools (Дата обращения: 14.05.2021)

Приложение 1

Интерфейс Google Lighthouse после оценки производительности страницы приложения на Next.js, использующей смешанный рендеринг.

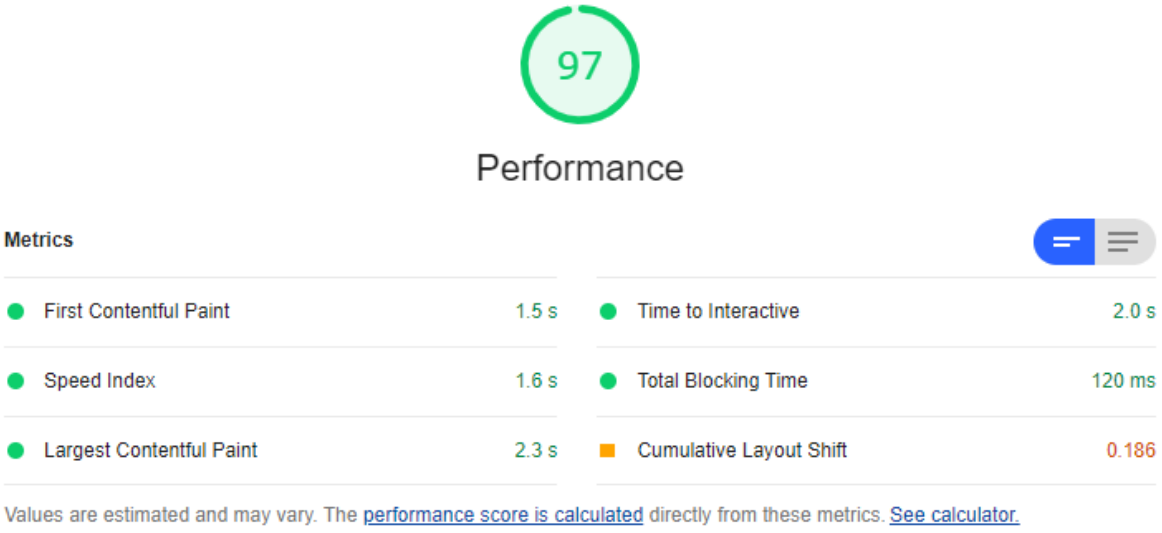


Интерфейс Google Lighthouse после оценки производительности страницы приложения на Next.js, полностью использующей статическую генерацию на стороне сервера.

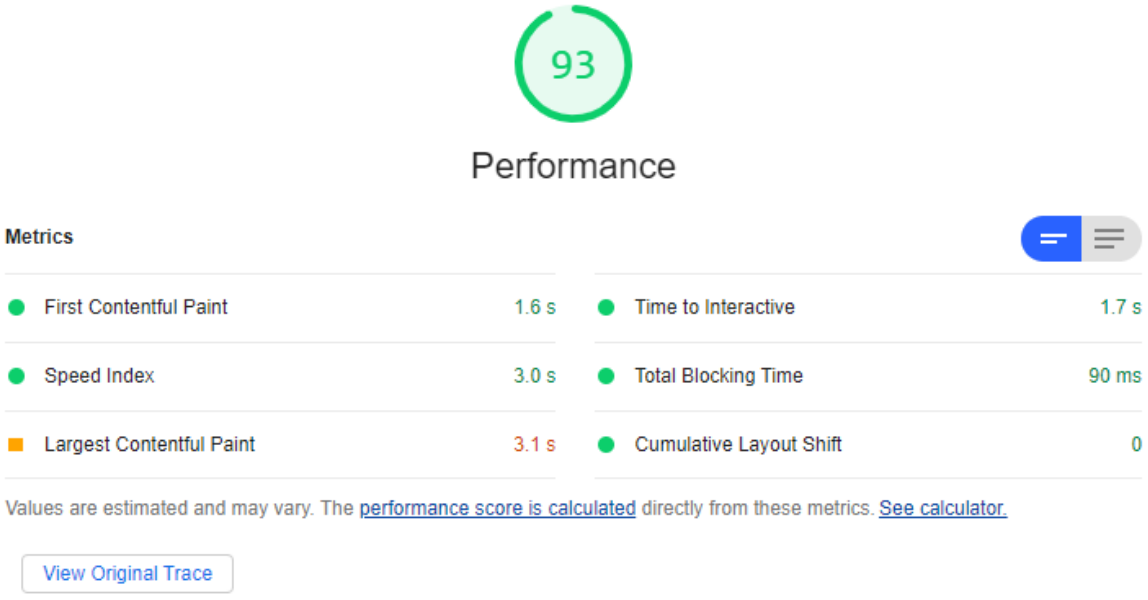


Приложение 2

Интерфейс Google Lighthouse после оценки производительности страницы приложения на React, использующей постепенные загрузку и рендеринг.



Интерфейс Google Lighthouse после оценки производительности страницы приложения на React, рендерящей страницу после загрузки всех данных.



Приложение 3

Скриншот веб-страницы Next.js приложения

Смешанный рендеринг - Полностью SSG

NAME	HEIGHT	MASS
Luke Skywalker	172	77
C-3PO	167	75
R2-D2	96	32
Darth Vader	202	136
Leia Organa	150	49
Owen Lars	178	120
Beru Whitesun lars	165	75
R5-D4	97	32
Biggs Darklighter	183	84
Obi-Wan Kenobi	182	77
Anakin Skywalker	188	84
Wilhuff Tarkin	180	unknown
Chewbacca	228	112
Han Solo	180	80
Greedo	173	74
Jabba Desilijic Tiure	175	1,358
Wedge Antilles	170	77

"НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО"
(УНИВЕРСИТЕТ ИТМО)

ОТЗЫВ РУКОВОДИТЕЛЯ
о выполнении курсового проекта (работы)

Студент Катюшин Станислав Владимирович
(Фамилия, И., О.)

Факультет ПИиКТ Группа P41071

Направление (специальность) 09.04.04 Программная инженерия

Руководитель Государев И.Б., к.п.н., доцент
(Фамилия, И., О., должность)

Дисциплина Проектирование и анализ языков веб-решений

Наименование темы: Сравнительный анализ скорости рендеринга приложений,
(Наименование сайта)
использующих CSR и SSR, на примере React и Next.js

ОЦЕНКА КУРСОВОГО ПРОЕКТА (РАБОТЫ)

№ п/п	Показатели	Оценка			
		5	4	3	0
1.	Проект создан обучающимся самостоятельно				
2.	Созданные элементы сайта раскрывают тематику				
3.	Проект технологически грамотный				
4.	Оформление отвечает требованиям к отчету				
5.	Во время защиты обучающийся показал умение кратко, доступно представить результаты работы, умение анализировать, аргументировать свою точку зрения, делать обобщение и выводы, адекватно ответить на поставленные вопросы.				
ИТОГОВАЯ ОЦЕНКА					

Отмеченные достоинства:

В отчете студента отражены полученные в ходе выполнения проекта навыки, соответствующие компетенциям по данной тематике. Студент показал себя личностью пунктуальной, ответственной, готовной к изучению нового материала. В процессе работы студент подтвердил навыки.

Отмеченные недостатки:

Заключение:

Студент подтвердил навыки, полученные за время обучения по указанной специальности.

Руководитель _____ И.Б. Государев
(подпись)

«_____»_____ 20__