

"НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО"
(УНИВЕРСИТЕТ ИТМО)**ЗАДАНИЕ НА КУРСОВОЙ ПРОЕКТ (РАБОТУ)**

Студент Летон Г.
(Фамилия, И., О.)

Факультет ПИиКТ Группа Р41071

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: Анализ и сравнение метрик при рендеринге таблиц и ленты новостей в трёх фреймворках
(Наименование сайта)

Задание Разработать веб-приложения с реализацией в них эксперимента по оптимизации рендеринга компонентов ленты новостей и таблиц.

Краткие методические указания (задачи работы)

Исследовать актуальные фронт-энд фреймворки и методы оптимизации рендеринга.
Разработать приложения на React, Vue и Svelte, которые будут отображать ленту новостей и таблицы выбранного размера.
Создать веб-приложение на NodeJS, которое будет использовать Jest и Puppeteer для тестирования эффективности рендеринга в фреймворках.
Внутри приложения создать команды для настройки тестирования.
Использовать паттерн MVC в веб-приложениях.
Использовать контейнеры для сборки и хранения приложений.
Выбрать метрики для анализа эффективности рендеринга веб-страниц.
Создать универсальный алгоритм для проведения серии экспериментов для определения самого эффективного фреймворка.
Разработать страницы отчета с результатами исследования.

Содержание пояснительной записки

Оглавление. Введение. Ход выполнения работы — описание хода разработки веб-приложений и проведения эксперимента. Заключение. Список использованной литературы. Приложение — графики производительности каждого эксперимента в работе.

Руководитель

И.Б.Государев

Студент

(подпись)

(подпись)

Летон Г.
(Фамилия И.О.)

"НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО"
(УНИВЕРСИТЕТ ИТМО)

ГРАФИК КУРСОВОГО ПРОЕКТА (РАБОТЫ)

Студент Летон Г.
(Фамилия, И., О.)

Факультет ПИиКТ Группа Р41071

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: Анализ и сравнение метрик при рендеринге таблиц и ленты новостей в трёх фреймворках
(Наименование темы)

Студент

№ п/п	Наименование этапа	Дата завершения		Оценка и подпись руководителя
		Планируемая	Фактическая	
1.	Провести исследование по теме рендеринга компонентов. Научиться работать с выбранными фреймворками. Разработать алгоритм сбора метрик.	апрель	май	
2.	Реализовать веб-приложение, контент которого заполняется случайными данными в виде ленты и таблиц. Написание отчета. Защита проекта.	май	май	

Руководитель

(подпись)

И.Б.Государев

Студент

(подпись)

Летон Г.

(Фамилия И.О.)

"НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО"
(УНИВЕРСИТЕТ ИТМО)

АННОТАЦИЯ К КУРСОВОМУ ПРОЕКТУ (РАБОТЕ)

Студент Летон Г.
(Фамилия, И., О.)

Факультет ПИиКТ Группа Р41071

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: Анализ и сравнение метрик при рендеринге таблиц и ленты новостей в трёх фреймворках
(Наименование темы)

ХАРАКТЕРИСТИКА КУРСОВОГО ПРОЕКТА (РАБОТЫ)

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

Цель работы — проанализировать эффективность рендеринга одинаковых компонентов и в одинаковых условиях разными фреймворками.

Задачи работы:

1. После исследования выбранных фреймворков разработать фронт-энд приложения для проведения эксперимента.
2. Создать алгоритм тестирования и сбора полезных метрик рендеринга.
3. Разработать эксперимент по определению оптимального по производительности фреймворка для рендеринга ленты новостей и таблиц.
4. Разработать приложение для тестирования.
5. Контейнеризовать разработанные фронт-энд приложения и разместить их на сервере.
6. Провести эксперимент и проанализировать полученные данные.

2. Характер работы

☐ Расчет ☐ Конструирование ☐ Моделирование ☒ Другое

3. Содержание работы

Использованы полученные знания по дисциплине для создания веб-приложений

Разработано приложение для тестирования рендеринга страниц и проведен эксперимент, который позволил найти оптимальный фреймворк для рендеринга ленты новостей и таблиц.

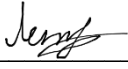
4. Выводы

Требования к проекту реализованы

Руководитель

И.Б. Государев

Студент


(подпись)

Летон Г.
(Фамилия И.О.)

СОДЕРЖАНИЕ

1 ОСНОВНЫЕ ПОНЯТИЯ	8
2 РАЗРАБОТКА ВЕБ-ПРИЛОЖЕНИЙ И ПРИЛОЖЕНИЯ ДЛЯ АНАЛИЗА	10
2.1 Общая структура проекта	10
2.2 Веб-приложение на React.....	11
2.3 Веб-приложение на Vue	13
2.4 Веб-приложение на Svelte	15
2.5 Приложение на Node для проведения эксперимента.....	16
2.6 Контейнеризация приложений.....	20
2.7 Эксперимент.....	20
2.8 Результаты эксперимента	21
ЗАКЛЮЧЕНИЕ	30
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	31

ВВЕДЕНИЕ

Эффективность рендеринга – это проблема для современного веб-разработчика, вне зависимости от его опыта или области работы. Исследования и общедоступные данные указывают на то, что скорость рендеринга значительно влияет на выручку от веб-приложения и на впечатления пользователей от работы с сайтом. Некоторые статистические данные указывают на нетерпеливость современного пользователя. Например, каждый четвёртый посетитель скорее всего покинет страницу, если она не загрузится за четыре секунды, а четверо из десяти откажутся от входа в онлайн-магазин, который загружается дольше трёх секунд. Это усугубляется тем, что онлайн-магазины обычно содержат много картинок, сильно замедляющих рендеринг.

Последствия подобной нетерпеливости – крупные убытки компаний. Так например компания Amazon подсчитала, что если их страницы будут загружаться на секунду дольше, это будет им стоить более, чем полтора миллиарда долларов в год. А Google утверждает, что замедление выдачи поиска даже на 0.4 секунды будет стоить восемь миллионов запросов в день. Для подсчёта такой статистики используется особый термин – показатель отказов, обозначающий процентное соотношение количества посетителей, посетивших не более одной страницы сайта, к общему числу посещений. Чем ниже показатель, тем больше шанс, что пользователь посетит все важные страницы сайта и вернётся к нему в будущем.

Чтобы оценить различные аспекты того, как быстро загружается веб-приложение, разработчики прибегают к инструментам веб-аудита, указывающие на то, какие части сайта можно улучшить, чтобы ускорить его работу. Современный разработчик может выбирать из множества инструментов аудита, хотя самым быстрым и простым является Google Lighthouse, встроенный прямо в Chrome и оценивающий SEO, доступность и, самое главное, эффективность, в том числе полезные метрики, такие как

первое существенное отображение, индекс скорости и общее время блокировки, из которых затем складывается общая оценка страницы. Другими подходами и инструментами веб-аудита, среди прочих, являются: анализ использования HTTPS, время загрузки, анализ памяти, анализ с помощью WebPagetest для оценки с разных точек Земли.

Чтобы улучшить показатели по приведённым выше метрикам, разработчики оптимизируют рендеринг веб-приложений при помощи различных техник, рассмотрение которых выходит за рамки данного исследования. В то же время, для ускорения процесса работы, веб-разработчики используют веб фреймворки, такие как Vue.js, React или Angular. Фреймворки между собой отличаются не только синтаксисом и предлагаемым функционалом, но и показателями эффективности рендеринга и своими собственными техниками оптимизации, а значит одно и то же приложение, написанное на разных фреймворках может иметь разные показатели метрик.

Несмотря на то, что в доступе есть несколько статей, рассматривающих проблему эффективности рендеринга, среди русской литературы данные по современным фреймворкам, подкреплённые экспериментами, скудны и зачастую не актуальны.

Цель работы - проведение эксперимента для формирования доказательной базы и ответы на следующие вопросы:

- 1) Какой из выделенных клиентских фреймворков лучше оптимизирует рендеринг компонентов?
- 2) Как различные фреймворки соотносятся друг к другу в разных условиях эксперимента?
- 3) Какие признаки оптимизации рендеринга компонентов можно выделить в фреймворках?

Опираясь на результаты, можно будет рассчитывать на выведение теорий как об эффективности фреймворков в целом, так и тех или иных оптимизаций рендеринга.

В ходе эксперимента, небольшие приложения на выделенных фреймворках будут подвергнуты испытаниям на рендеринг с варьирующимся количеством компонентов и варьирующимися типами составляющих этих компонентов:

1) Рендеринг компонента с таблицей содержащей 2500 строк и 7500 строк;

2) Рендеринг списка, состоящего из 1000 одинаковых компонентов, то есть симуляция ленты в социальной сети. Скорость замены и добавления такого же количества экземпляров компонентов;

3) Измерение полезных метрик, таких как время до первой полезной отрисовки, время до загрузки содержимого DOM и объём приложения.

Для обеспечения наиболее равных условий, объективности измерений, абстрагированных от платформы как аппаратно, так и программно, эксперименты будут проводиться на виртуальном сервере сервиса Yandex Compute Cloud. Приложения на фреймворках будут развёрнуты через Docker-образы, метрики измерены при помощи внутренних инструментов профилирования NodeJS, инструментов профилирования в браузерах и сервисе Lighthouse от Google.

Результаты эксперимента и микроисследования должны хотя бы частично ответить на поставленные вопросы и выделить фреймворки с наиболее оптимизированным рендерингом компонентов.

1 ОСНОВНЫЕ ПОНЯТИЯ

Веб-приложение (или «web app») – это любая компьютерная программа, которая выполняет определенную функцию, используя в качестве клиента веб-браузер.

При разработке веб-приложения и проведения эксперимента были использованы следующие понятия:

- Node.js — это программная платформа, основанная на «движке» Google Chrome V8, которая превращает JavaScript из узкоспециализированного языка в универсальный язык программирования.
- NPM (аббр. node package manager) — это стандартный менеджер пакетов, автоматически устанавливающийся вместе с Node.js. Он используется для скачивания пакетов из облачного сервера npm, либо для загрузки пакетов на эти сервера.
- Yarn — это альтернативный npm-клиент для работы в качестве пакетного менеджера JavaScript, совместно созданный Facebook, Google, Exponent и Tilde. Этот менеджер пакетов ускоряет сборку пакетов и делает её более безопасной.
- ESLint — это утилита, которая может анализировать написанный код. Фактически, это статический анализатор кода, и он может находить синтаксические ошибки, баги или неточности форматирования.
- Контейнер — стандартная единица программного обеспечения, в которую упаковано приложение со всеми необходимыми для его работы зависимостями — кодом приложения, средой запуска, системными инструментами, библиотеками и настройками.
- Рендеринг — процесс получения изображения с помощью компьютерной программы.
- Puppeteer — библиотека Node.js, которая позволяет автоматизировать процессы в Chromium браузере при помощи API высокого уровня.

- Jest — это JavaScript-тестер, то есть фреймворк JavaScript для создания, запуска и структурирования тестов.
- Метрика — мера, позволяющая получить численное значение некоторого свойства программного обеспечения или его спецификаций.
- Профилирование — сбор характеристик работы программы, таких как время выполнения отдельных фрагментов (обычно подпрограмм)
- DOM (Document Object Model) - объектная модель документа - это объектная модель документа, которую браузер создаёт в памяти компьютера на основании HTML-разметки.

•

2 РАЗРАБОТКА ВЕБ-ПРИЛОЖЕНИЙ И ПРИЛОЖЕНИЯ ДЛЯ АНАЛИЗА

2.1 Общая структура проекта

Весь проект эксперимента было решено разделить на несколько отдельных проектов, или так называемых подпроектов. Такое решение было принято для реализации воспроизводимого микросервисного подхода, особенно важного в описываемом эксперименте, так как при нём приложения, разработанные на разных фреймворках, можно будет разместить на одном сервере в одинаковых условиях, а затем проанализировать по принципу, не зависящему от платформы. Итого, проект будет разделён на четыре части:

- 1) Веб-приложение на React;
- 2) Веб-приложение на Vue 3;
- 3) Веб-приложение на Svelte;
- 4) Приложение, использующее Jest, Puppeteer и библиотеки Node для теста, анализа и аудита трёх веб-приложений с формированием отчёта.

React и Vue были выбраны как два популярных в широком кругу фреймворка, а Svelte – как новая многообещающая технология.

Готовые приложения, после проведения предварительного эксперимента на локальной машине, будут контейнеризованы с обратным прокси-сервером Nginx и расположены на сервере VScale. Контейнеризация с Docker и Docker compose позволит ещё дальше развить идею воспроизводимости эксперимента с использованием последних технологий в веб-разработке. Каждый контейнер с веб-приложением можно будет быстро запустить, провести эксперимент, и переключиться на другой контейнер.

Так как проект будет реализован на программной платформе NodeJS, то будет использован стандартный пакетный менеджер NodeJS NPM. Средой разработки служит IDE WebStorm от JetBrains.

Следующие части работы описывают разработку веб-приложений на выбранных фреймворках.

2.2 Веб-приложение на React

Первым из разработанных приложений станет такое на React, наименее известной для исследователя технологии. Инициализация проекта производится командой `npm create-react-app my-app --use-npm`. Флаг `--use-npm` предназначен для того, чтобы отдавать предпочтение пакетному менеджеру `npm` вместо стандартного для React `Yarn`. `create-react-app` – это официально поддерживаемый способ создания базового одностраничного React приложения.

Помимо необходимых для функционирования React пакетов, будут установлены следующие дополнительные пакеты:

1) `react-router` – пакет компонентов и настроек для реализации декларативной маршрутизации в React. Используется для добавления различного рода навигации в приложениях.

2) `react-router-dom` – DOM привязка для `react-router`.

3) `faker` – пакет для настраиваемой генерации большого количества ненастоящей информации-заполнителя. В некотором роде аналог известного заполнителя `lorem ipsum`, но гораздо более гибкий, так как формирует не просто неразборчивый текст, а правдоподобную информацию, например имена, адреса или электронную почту.

4) Набор пакетов `testing-library` (`jest-dom`, `react`, `user-event`) – пакеты для упрощения процесса тестирования React приложений.

5) `web-vitals` – маленькая библиотека для расчёта важных веб-метрик (CLS, FID, LCP и другие) по принципу Google.

Для того, чтобы повысить качество кода будет использовано семейство пакетов `ESLint`, которое в реальном времени анализирует написанный код на наличие проблемных и потенциально проблемных конструкций.

При инициализации проекта в папке `src` создаётся несколько файлов, среди которых `index.js` – входная точка веб-приложения. В ней при помощи

функции `ReactDOM.render` на странице рендерится передаваемый в неё HTML. Благодаря JavaScript XML или просто JSX в JavaScript функции можно передавать разметку HTML и в ней использовать логику JavaScript. В стандартном случае в функцию `render` передаётся компонент `App`, который мы рассмотрим позже. В нашем приложении компонент `App` будет обернут в компонент `BrowserRouter`, который импортируется под именем `Router`. Компонент `Router` обеспечивает работу навигации внутри компонентов, которые в него обернуты. `Router` в свою очередь обернут в компонент `React.StrictMode`. Он не отображает видимый интерфейс, но служит как дополнительный инструмент для поиска ошибок в React. Например, он может указать на компоненты с небезопасными жизненными циклами или неожиданными побочными эффектами при рендеринге, например если одна часть жизненного цикла вызовется дважды. Второй аргумент функции `render` - `document.getElementById('root')`, он указывает на нужный компонент в html шаблоне, где будет рендериться контент приложения

В файле также импортируется и вызывается функция `reportWebVitals`, в которую передаётся `console.log`. Эта функция использует библиотеку `web-vitals` для сбора важных метрик при загрузке приложения. Передача `console.log` значит, что результат будет выведен в консоль браузера.

В компоненте `App` описываются необходимые для эксперимента логика и разметка. В функции `App` содержится главный блок приложения. Компоненты `Switch`, `Route` и `Redirect` определяют маршрутизацию приложения. В приложении три маршрута, соответствующих этапам эксперимента. В маршруте с адресом `feed` имитируется лента социальной сети, состоящая из постов пользователей с текстом. Массив `feedData` заполняется тысячей наборов случайных данных, генерируемых библиотекой `faker`. Каждый набор данных – объект с `id`, названием, текстом, именем и фамилией. В функции `Feed` производится mapping массива `feedData` и на выходе получается массив постов с готовой разметкой JSX. Функция `Feed` передаётся как компонент в основной компонент `App` и рендерит поле для

ввода и ленту. Отметим использование в разметке `<>` - это так называемый «фрагмент» React, служащий абстракцией, благодаря которой на верхнем уровне компонента может быть больше одного элемента.

В оставшихся двух маршрутах происходит рендеринг таблицы на 7500 и 2500 строк. В них используется функция `Table`, принимающая на вход массив данных. Этими массивами служат `tableData` и `tableDataSmall`. Первый массив состоит из 7500 объектов, содержащих `id`, случайные имя, фамилию и профессию. Этот массив затем обрезается до 2500 элементов методом `slice` и присваивается массиву `tableDataSmall`. Функция `Table` передаётся как компонент в основной компонент `App` и рендерит поле для ввода и таблицу.

Помимо компонентов маршрутов в `Switch` добавлен компонент `Redirect`, который определяет на какой адрес автоматически переходить, если он не указан, в нашем случае это адрес будет с лентой.

Теперь рассмотрим такое же приложение на `Vue`.

2.3 Веб-приложение на Vue

В Приложение на `Vue` было решено разрабатывать на новой, третьей версии `Vue`. `Vue 3` привносит множество новшеств во `Vue`, среди которых можно выделить:

- `Composition API`, предназначение которого – дать возможность разработчику расположить и сгруппировать код в компоненте соответственно его логике, освобождать от необходимости постоянно прыгать между блоками с переменными, методами или жизненными циклами;
- Фрагменты, как и в `React`, дают возможность нескольким элементам выступать в качестве корневых в объекте;
- `Teleport`, с помощью которого часть шаблона компонента можно вынести за пределы места в `DOM`, куда вставлен сам компонент, например в `body`.

Чтобы инициализировать приложение на `Vue` можно использовать команду `“npm init @vitejs/app hello-vue3”`. Заметим, что использование `Vite`

вместо уже стандартного Webpack значительно ускоряет процесс сборки приложения.

Структура приложения несколько отличается от React. В первую очередь входной точкой приложения является `main.js`, где приложение инициализируется функцией `createApp`. Но перед тем как инициализировать приложение, в файл импортируется библиотека `vue-router`, отвечающая за маршрутизацию в веб-приложении.

В логике навигации также есть различия. Вместо использования компонентов прямо в разметке главного компонента `App`, `Vue-router` использует один компонент `router-view`, который в нашем случае будет единственным компонентом, содержащимся в `App`. Его особенность в том, что он динамически изменяет своё наполнение в зависимости от выбранного адреса. Конфигурация маршрутизации производится при помощи функции `createRouter`, возвращающей экземпляр маршрутизатора. В эту функцию передаётся массив объектов маршрутов, обладающих гибкими опциями и дополнительные настройки, например тип используемой истории. В объекте с маршрутами можно указать путь и компонент, который будет отображаться по этому адресу. Ключ `redirect` обозначает путь, на который следует перенаправлять из указанного, в нашем случае корневого. В нашем маршрутизаторе будет два маршрута: `feed` и `table`. Булевым ключом `props` можно разрешить использовать в маршрутах свойства и передать их в компоненты, поэтому в `table` мы дополнительно обозначаем опциональный маршрут, указывающий на количество строк в таблице, например по маршруту `"/table/3000"` отобразится таблица на три тысячи строк.

После создания экземпляра маршрутизатора мы инициализируем приложение, указывая в функции `createApp` корневой компонент. Методом `use` экземпляра приложения мы указываем ему использовать экземпляр маршрутизатора и, наконец, методом `mount` указываем к какому элементу в `index.html` следует привязать приложение.

Рассмотрим компоненты, используемые в приложении. Их итоговые логика и разметка идентичны React, но их определения заметно отличаются из-за особенностей Vue. Во-первых, компоненты здесь не представлены функциями, расширяющими базовый компонент React, а являются обособленными сущностями и поэтому расположены в отдельных файлах. В компоненте ленты Feed в блоке script происходит заполнение данными публикаций в ленте, а в блоке template определяется разметка компонента. Отметим использование конструкции v-for для рендеринга частей разметки в цикле вместо map в React. В компоненте следует выделить определение свойства size через Composition API, которое берётся из адреса и передаётся в функцию заполнения данных, определяя размер массива.

2.4 Веб-приложение на Svelte

Svelte в своей стандартной комплектации – чрезвычайно легковесный фреймворк, выступающий в роли компилятора приложения и представляющий новый взгляд на оптимизацию, избавляя от необходимости использовать виртуальный DOM. Чтобы инициализировать проект на Svelte достаточно использовать команду “`npx degit sveltejs/template название проекта`” и установить зависимости. В отличие от Vue, в стандартном Svelte в качестве сборщика используется библиотека rollup, а за отдачу статических файлов отвечает sirv. Заметим, что новый расширенный фреймворк на основе Svelte – SvelteKit использует уже Vite.

Синтаксически Svelte скорее похож на Vue с его отдельными блоками для разметки, исполняемого кода и стилей, но с некоторыми важными различиями. Например, блок-цикл в разметке реализуется с помощью конструкции “`{#each x as y}{/each}`” прямо в разметке, а не через свойство v-for компонента. В приложении на Svelte у нас, так же как и на Vue, имеется три компонента: основной – App, Feed для отображения ленты и Table для таблицы.

В роли маршрутизатора в нашем приложении выступает библиотека svelte-navigator. Маршрутизация реализуется довольно просто, достаточно

импортировать компоненты Router и Route из библиотеки и добавить их в нужный, в нашем случае входной компонент App. Компонент Router оборачивает маршруты Route, в которых можно указать необходимый путь и отображаемые в них разметку или компонент. Маршруты поддерживают передачу свойств как и Vue, но процесс реализован ещё проще, так как не нужно указывать использование свойств в настройках маршрута, все части пути, перед которыми стоит двоеточие, доступны в компоненте в качестве свойств, достаточно лишь объявить в компоненте переменную с таким же именем и добавить к ней ключевое слово `export`. Однако с базовыми настройками маршрутизация не работает, так как `serv` изначально не считает приложение одностраничным. Чтобы это исправить достаточно добавить ключ `“-s”` к команде `start` в `package.json`, которая является частью сборки приложения.

Таким образом, у нас получается три идентичных по функционалу и стилю приложения, готовых к проведению эксперимента вне контейнеров. Далее мы рассмотрим разработку приложения, управляющего самим экспериментом.

2.5 Приложение на Node для проведения эксперимента

Для сравнения эффективности рендеринга выбранных фреймворков нужно было разработать небольшое приложение, которое могло бы провести серию тестов на страницах с лентой и таблицами, агрегировать интересующие нас метрики (время до первой полезной отрисовки, время до загрузки содержимого DOM, время до интерактивности DOM, объём приложения и другие), и, проведя с ними некоторые манипуляции, вывести их в доступной форме. Для этой цели прекрасно подходит сочетание таких фреймворков-инструментов, как Jest, Puppeteer и PageSpeedInsights с Lighthouse.

Jest — поддерживаемый компанией Facebook фреймворк для тестирования приложений любого масштаба. Он удобен тем, что несмотря на обширный функционал, его очень легко настроить и освоить на базовом

уровне. Такое же описание подходит и для Puppeteer – библиотеки от Google, позволяющей управлять браузером Chrome через протокол DevTools, причём браузер может быть как с интерфейсом, так и без него. С Puppeteer можно, среди прочего, имитировать действия пользователя, тестировать интерфейс, отправлять формы или генерировать скриншоты. В сочетании с Jest он становится мощнейшим инструментом для тестирования. Мы будем использовать Jest с Puppeteer для автоматизации процесса тестирования и сбора метрик из браузера.

PageSpeedInsights и Lighthouse – это также поддерживаемые Google технологии, которые обычно работают в тандеме. PageSpeedInsights известен по одноимённому онлайн-инструменту, который использует Lighthouse для формирования отчёта по выбранной странице в сети интернет. Отчёт содержит в себе важные метрики скорости, на основе которых выдаётся оценка, а также список аудитов с предложениями как можно улучшить работу страницы. Эти библиотеки будут использованы для сбора метрик, недоступных Puppeteer.

Сначала мы инициализируем простой проект на Node командой “npm init” и устанавливаем нужные пакеты командой “npm i puppeteer jest jest-puppeteer psi lighthouse”. Далее нужно настроить работу Jest. Настройки jest можно хранить как в отдельном файле, так и прямо в package.json. Базовая настройка Jest для работы с Puppeteer тривиальна и заключается в выборе предустановки jest-puppeteer из одноимённого пакета.

```
"name": "experiment-orchestrator",  
"jest": {  
  "preset": "jest-puppeteer"  
},
```

Рисунок 1 – Зависимости проекта

В список скриптов также добавляем новый скрипт test, который начинает цикл тестирования вызовом jest. Теперь у нас всё готово чтобы создать тест.

Определение теста будет находиться в файле `index.test.js`, где `test` в названии явно сообщает `jest`, что в нём нужно искать описание теста. Блок тестов определяется функцией `jest` под названием `describe`, в которую передаются название и функция, выступающая этим блоком. Сами единичные тесты определяются функцией `it`, в которую так же передаются название и тело теста. Функция `describe` очень полезна, когда нужно не просто объединить несколько тестов, но и определить циклы блока тестов.

В нашем блоке под названием “Performance Test” определяются переменные, которые будут хранить клиент, браузер, открытую страницу и наборы собранных метрик. Переменным `iterations` и `webpage` присваиваются переменные окружения, которые в будущем будут передаваться как аргументы в команду `npm`, а иначе им задаются стандартные значения в 25 циклов и страница с таблицей на 7500 элементов. Дальше в блоке определено несколько функций, отвечающих за жизненные циклы блока тестов:

1) `beforeAll` - вызывается перед началом запуска тестов, в ней запускается браузер в режиме `headless`, то есть без интерфейса;

2) `beforeEach` – вызывается перед каждым тестом, в ней открывается новая вкладка, устанавливается лимит времени на тест, открывается сессия сбора метрик и совершается переход на нужную страницу;

3) `afterEach` – вызывается после каждого теста, в ней просто закрывается только что протестированная вкладка;

4) `afterAll` – вызывается после завершения всех тестов, в ней закрывается браузер, рассчитывается среднее, максимум и минимум метрик. Затем все метрики собираются в JSON объекты и записываются в локальные файлы, а процесс тестирования завершается.

После циклов следуют определения двух тестов, один собирает метрики инструментами, доступными в самом браузере: первая значимая отрисовка, первая отрисовка с контентом, время до начала отрисовки, время до ответа на запрос, время до загрузки ответа на запрос, время до интерактивности DOM, время до загрузки DOM. К сожалению, метрики,

собираемые браузером обозначаются моментами во времени, а не продолжительностью, поэтому в файл помимо модулей импортируются вспомогательные функции, которые переводят собранные метрики в читаемый формат, например функция `extractDataFromPerformanceTiming` проходит по выбранным метрикам и отнимает от них `navigationStart`, возвращая продолжительность измерений в миллисекундах. Этот тест оборачивается в цикл, количество циклов которого определяется переменной `iterations`. Другой тест запускает сессию Lighthouse и генерирует отчёт в формате `json`. Отчёт Lighthouse содержит время до интерактивности, время до первой отрисовки контента, время блокирования основного потока до интерактивности и размер страницы.

Для более предсказуемого проведения эксперимента и независимости сбора метрик от потоков, команда Jest будет запускаться с ключом `--runInBand` с которым тесты запускаются синхронно.

Приложение было решено не помещать в контейнер, так как работе с Puppeteer в контейнере свойственно множество проблем, особенно с установкой и работой с файловой системой, а плюсы от работы в контейнере в данном случае не стоят усложнений. Puppeteer сам автоматически устанавливает свежую версию браузера Chromium, а остальные части приложения не зависят от окружения, не считая версии Node. Чтобы сохранить работу с переменными окружения без создания контейнера, в приложение было добавлен дополнительный `npm` скрипт и два связанных с ним JavaScript скрипта. При запуске `npm setup` в скрипт можно передать три переменных: `iterations`, `page` и `framework`. `Npm setup` через Node запускает скрипт `Setvars.js` и передаёт туда переменные, в нём они считываются и записываются в `jestVars.js` как переменные окружения, отвечающие соответственно за количество циклов теста, за тестируемую страницу и идентификацию фреймворка. Затем файл `jestVars.js` указывается в настройках Jest как дополнительный файл настройки и таким образом переменные попадают в тест.

2.6 Контейнеризация приложений

Как уже упоминалось ранее, для обеспечения равных, абстрагированных от платформы условий, приложения на React, Vue и Svelte были помещены в контейнеры. Для создания контейнеров использовались рекомендованные Dockerfile файлы, в которые было внесено несколько изменений, например использованные менеджеры пакетов и использованные версии Node, а React с маршрутизатором React-router требует дополнительных настроек выдачи страниц. Командой `docker build` создаются образы с передаваемым под ключом `-t` тегом, которым для удобства выступает ссылка на репозиторий Docker в `cr.yandex` (Yandex Container Registry). После сборки образа он полностью готов к запуску командой `docker run`, в которую достаточно передать открытый порт через ключ `-p`, где через двоеточие нужно указать с каким портом внутри контейнера его нужно связать, то есть с портом, где расположено приложение. Ещё одним удобством контейнеров является то, что они почти не оставляют после себя следов, достаточно остановить их, а затем удалить контейнер и образ, после чего останется только кэш в самом Docker, который также можно очистить.

После авторизации в `cr.yandex` остаётся отправить их в репозиторий командой `docker push` и они готовы к загрузке на любой машине с установленным Docker.

2.7 Эксперимент

На основе описанных выше приложений и алгоритмов тестирования и сбора метрик в рамках эксперимента будет проведена серия тестов над идентичными страницами на разных фреймворках в таком порядке:

1. Рендеринг и сбор метрик страниц на React:
 - 1.1. Страница с лентой новостей;
 - 1.2. Страница с таблицей, содержащей 2500 строк;
 - 1.3. Страница с таблицей, содержащей 7500 строк;
2. Рендеринг и сбор метрик страниц на Vue:
 - 2.1. Страница с лентой новостей;

- 2.2. Страница с таблицей, содержащей 2500 строк;
- 2.3. Страница с таблицей, содержащей 7500 строк;
- 3. Рендеринг и сбор метрик страниц на Svelte:
 - 3.1. Страница с лентой новостей;
 - 3.2. Страница с таблицей, содержащей 2500 строк;
 - 3.3. Страница с таблицей, содержащей 7500 строк;

На выходе получится три набора файлов, в каждом из которых будет по четыре файла с измерениями: файл с максимальными значениями метрик, файл с минимальными значениями метрик, файл со средними значениями метрик и файл с метриками из отчёта Lighthouse. Файлы различимы по префиксу названием фреймворка (если оно было передано) и по постфиксу с количеством циклов тестирования. Полученные результаты будут проанализированы и сравнены с помощью таблиц и инструмента создания графиков Anychart.

2.8 Результаты эксперимента

Было проведено 9 серий испытаний, в каждом из которых проводилось 100 циклов тестирования с использованием сервера на сервисе Yandex Compute Cloud.

Рассматриваемые метрики:

- First Contentful Paint – время, за которое браузер отображает первую часть данных из DOM, сообщая пользователю, что страница действительно загружается.
- DOM Content Loaded – время, за которое браузер загружает и парсит первоначальный HTML документ, не дожидаясь стилей или изображений.
- Load end – время, за которое загружается вся отображаемая страница, включая стили и изображения.
- Time to Interactive – сколько времени требуется странице, чтобы стать полностью интерактивной. Это значит, на странице уже начали отображаться данные из DOM (значит First Contentful Paint уже

произошло), было зарегистрировано большинство обработчиков событий у видимых элементов, и страница отвечает на ввод пользователя за 50 миллисекунд или быстрее.

- Total Blocking Time – время между First Contentful Paint и Time to Interactive, когда основной поток блокируется и страница плохо реагирует на ввод.
- Byte Weight – общий размер всех ресурсов, запрошенных страницей.

Результаты экспериментов (единицы измерения представлены на рисунках):

1. Серия испытаний рендеринга ленты новостей показала следующие результаты:

1.1. В среднем наблюдается высокая производительность React и Vue при первичной загрузке страницы с небольшим отставанием React. Svelte значительно отстаёт в полной загрузке данных страницы, но это связано с большим объёмом самого приложения. При этом Svelte быстрее загружает документ страницы. Также следует отметить, что разрыв между тремя метриками у React практически отсутствует, в то время как у Vue и Svelte разные стадии загрузки заметно разбросаны

1.2. Средняя производительность фреймворков показана на рисунке 2.

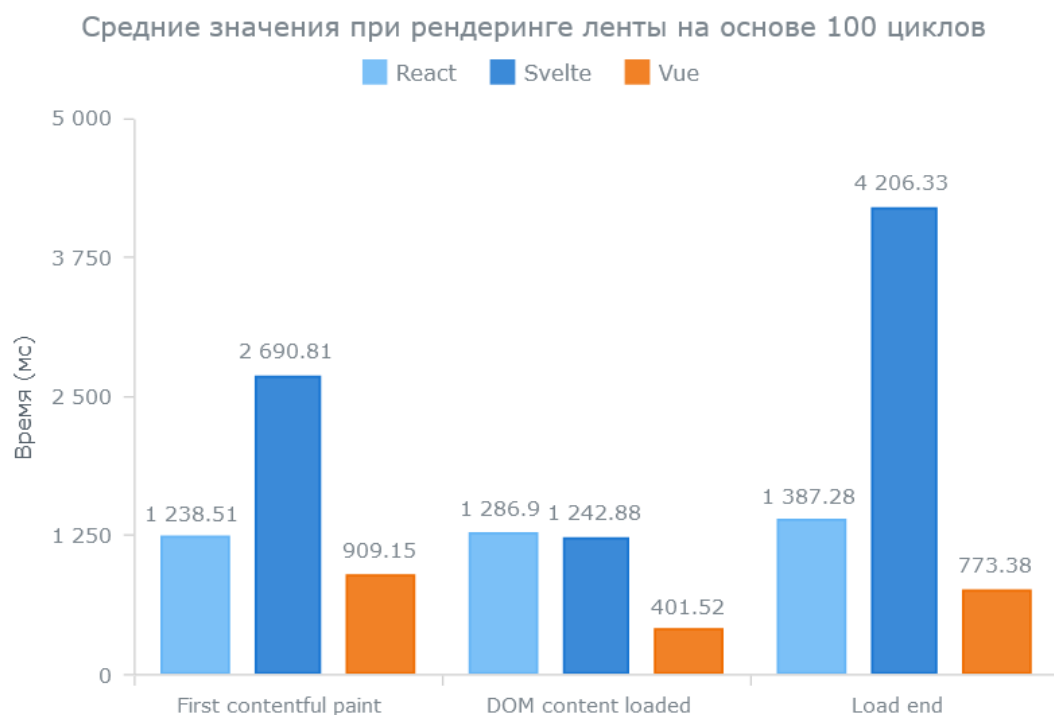


Рисунок 2 – Средние значения при рендеринге ленты

2. Серия испытаний рендеринга таблицы на 2500 строк показала следующие результаты:

2.1. В отличие от загрузки ленты новостей, Vue не настолько эффективно начинает рисовать значимую информацию, но, с другой стороны, быстрее загружает документ страницы. а результаты. Результаты Svelte сопоставимы с крупными фреймворками, не считая полной загрузки.

2.2. Средняя производительность фреймворков показана на рисунке

3.

Средние значения при рендеринге таблицы на 2500 элементов на основе 100 циклов

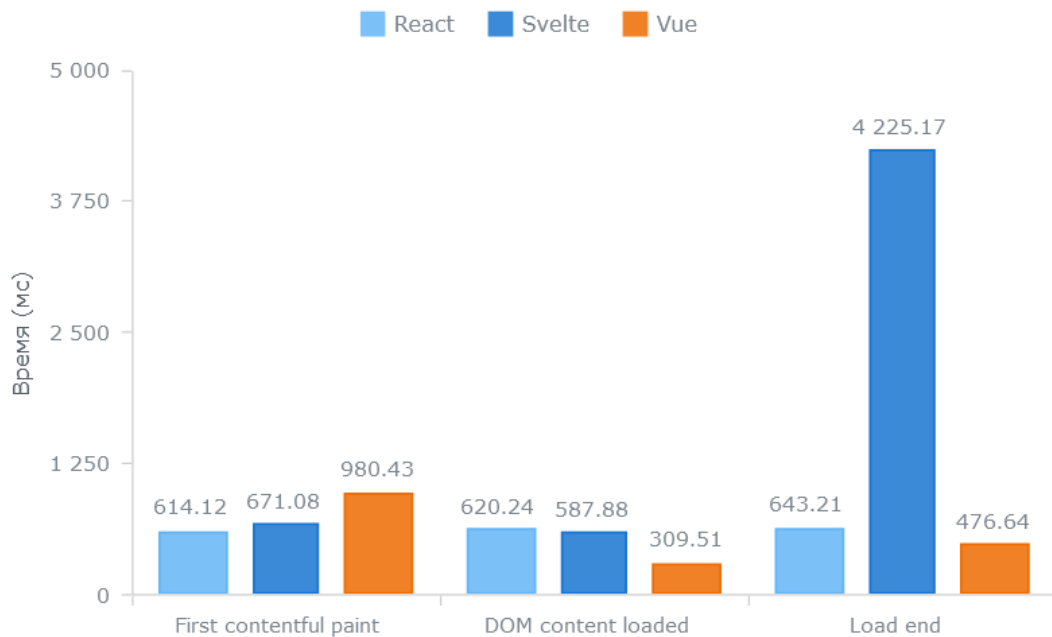


Рисунок 3 – Средние значения при рендеринге таблицы на 2500 строк

3. Серия испытаний рендеринга таблицы на 7500 строк показала следующие результаты:

3.1. С увеличением числа строк в таблице Vue сильнее отстаёт в первой полезной загрузке, но загружает документ ещё эффективнее на фоне конкурентов. React показывает стабильные результаты, незначительно теряя в скорости при увеличении отображаемых элементов

3.2. Средняя производительность фреймворков показана на рисунке 4.

Средние значения при рендеринге таблицы на 7500 элементов на основе 100 циклов

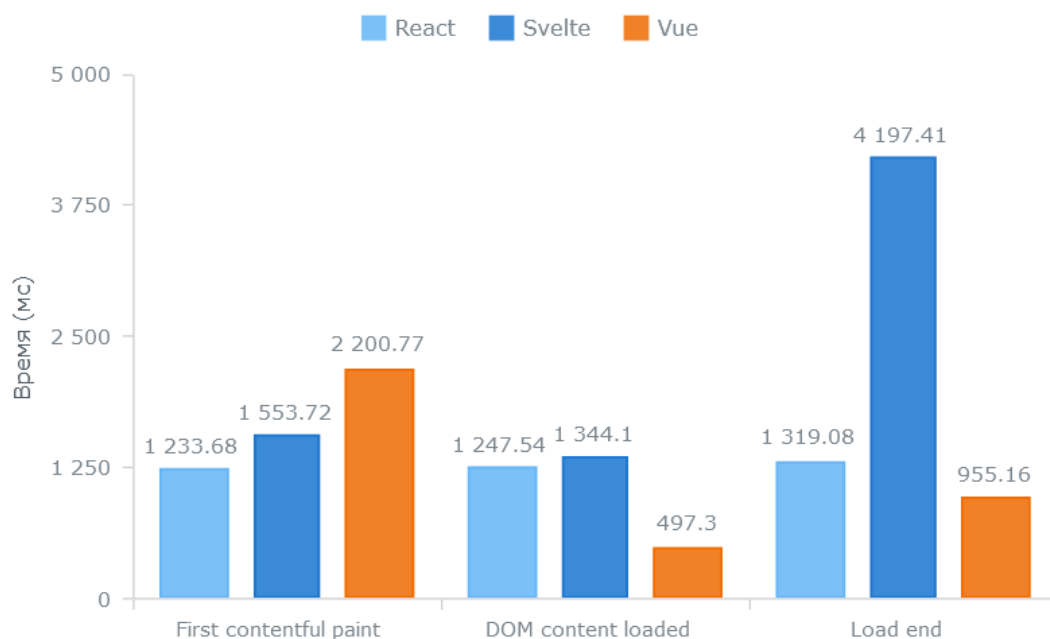


Рисунок 4 – Средние значения при рендеринге таблицы на 7500 строк

4. Для всех трёх серий экспериментов были зафиксированы минимальные и максимальные значения метрик:

4.1. Проанализировав максимальные и минимальные значения метрик на протяжении трёх серий экспериментов, можно прийти к выводу, что наименее эффективной частью Vue является момент начала рендеринга контента на странице, но остальных сферах он даже в самых медленных циклах выдал высокие показатели. Худшая производительность React с таблицей на 2500 строк примерно сопоставима с лучшими показателями при 7500 строках. Svelte показал очень низкие результаты полной загрузки страницы, что скорее всего связано с работой библиотеки, генерирующей случайные данные для компонентов.

4.2. Минимальная производительность фреймворков показана на рисунках 5 и 6, максимальная – на рисунке 7.

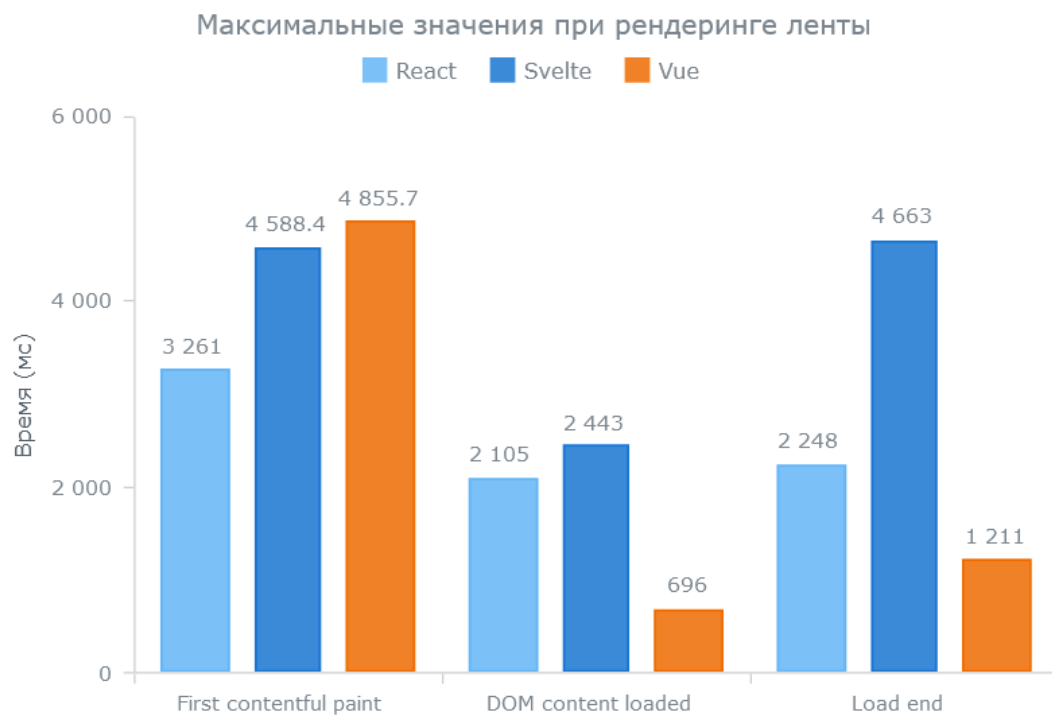


Рисунок 5 – Худшие значения при рендеринге ленты

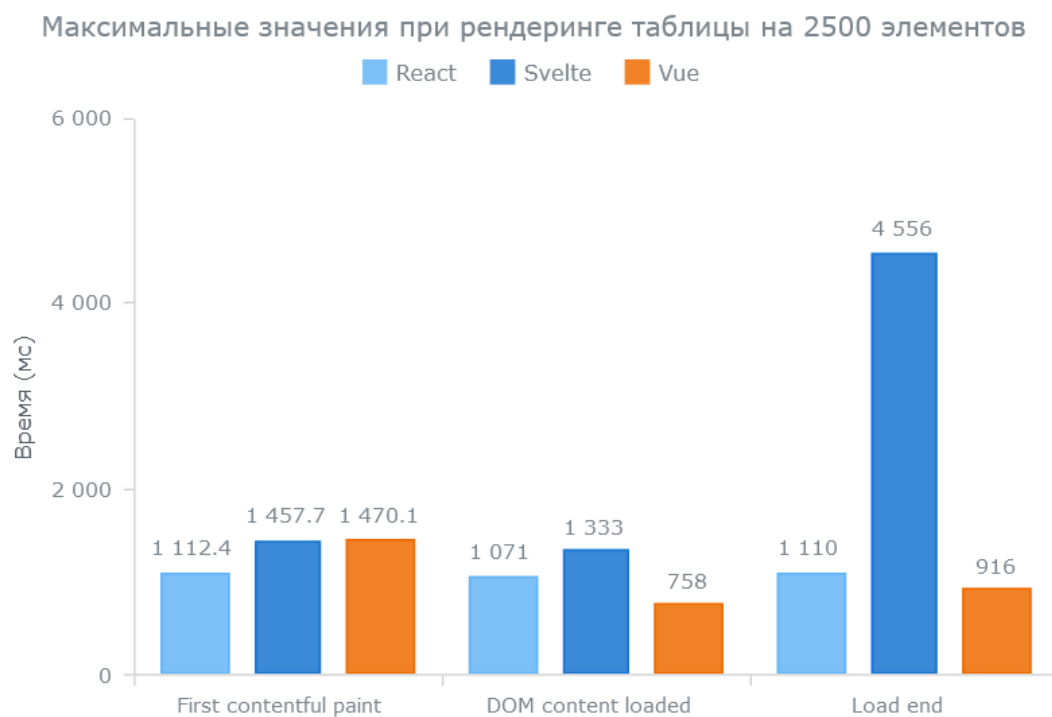


Рисунок 6 – Худшие значения при рендеринге таблицы на 2500 строк

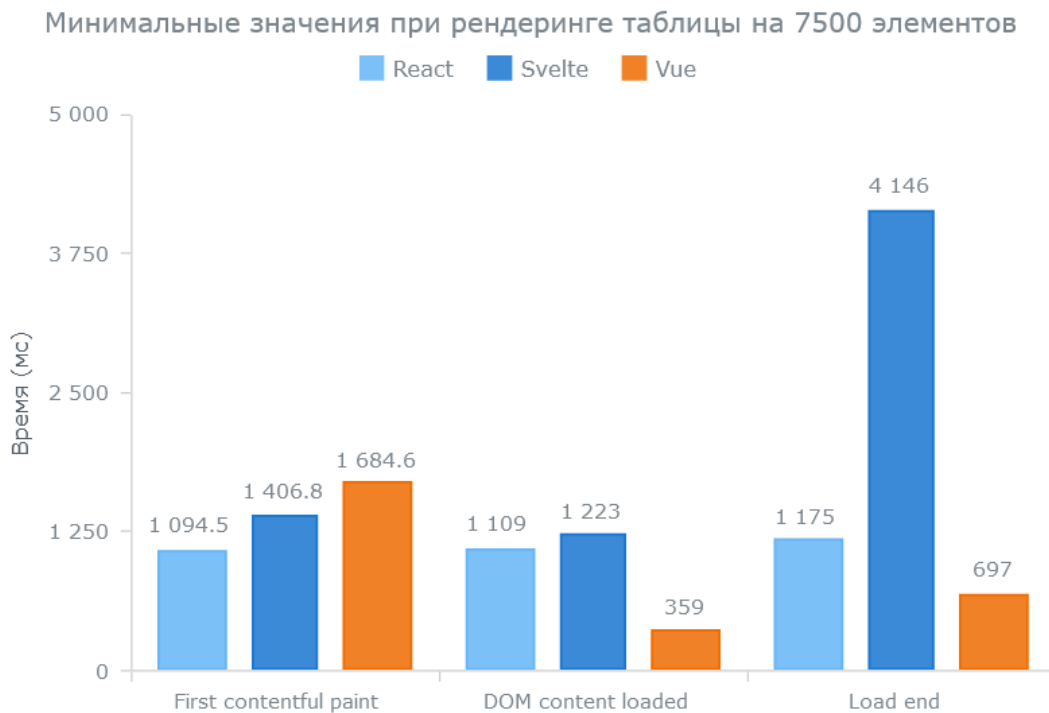


Рисунок 7 – Лучшие значения при рендеринге ленты

5. После каждой серии экспериментов проводился сбор метрик инструментом Lighthouse:

5.1. React значительно опережает другие фреймворки в рамках загрузки основного потока, особенно с таблицей на 2500 строк, а Svelte снова оказался медленнее. Такие же тренды замечены во времени до интерактивности, но в меньших масштабах. У Vue оказалось самое легковесное приложение.

5.2. Результаты Lighthouse показаны на рисунках 8-10.

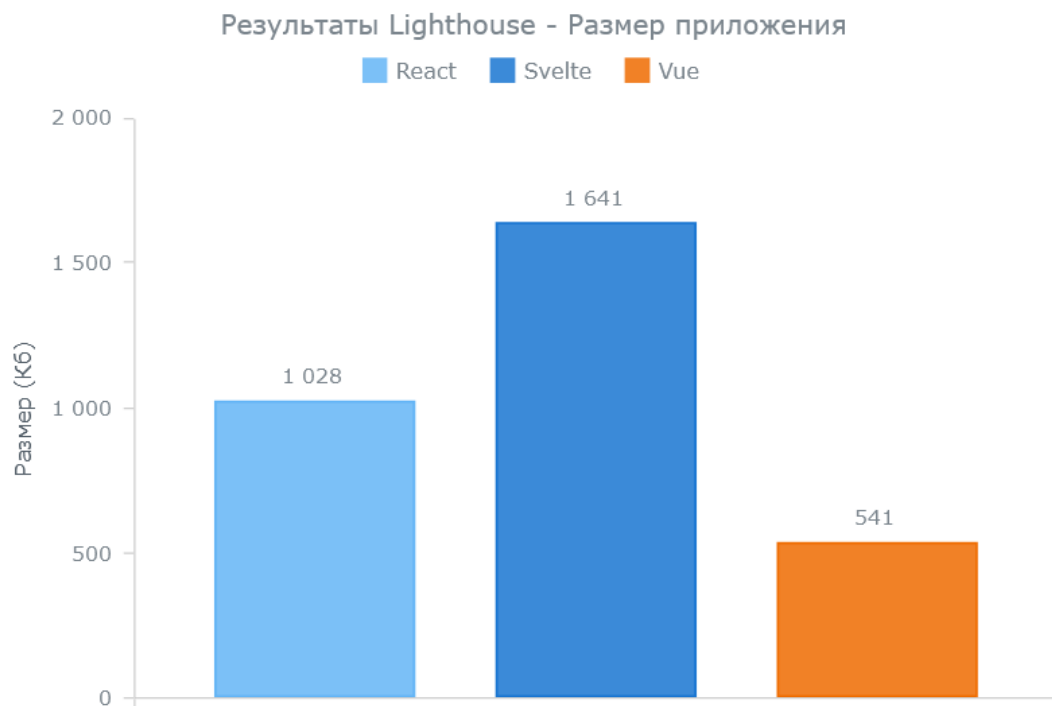


Рисунок 8 – Lighthouse - Размер приложения

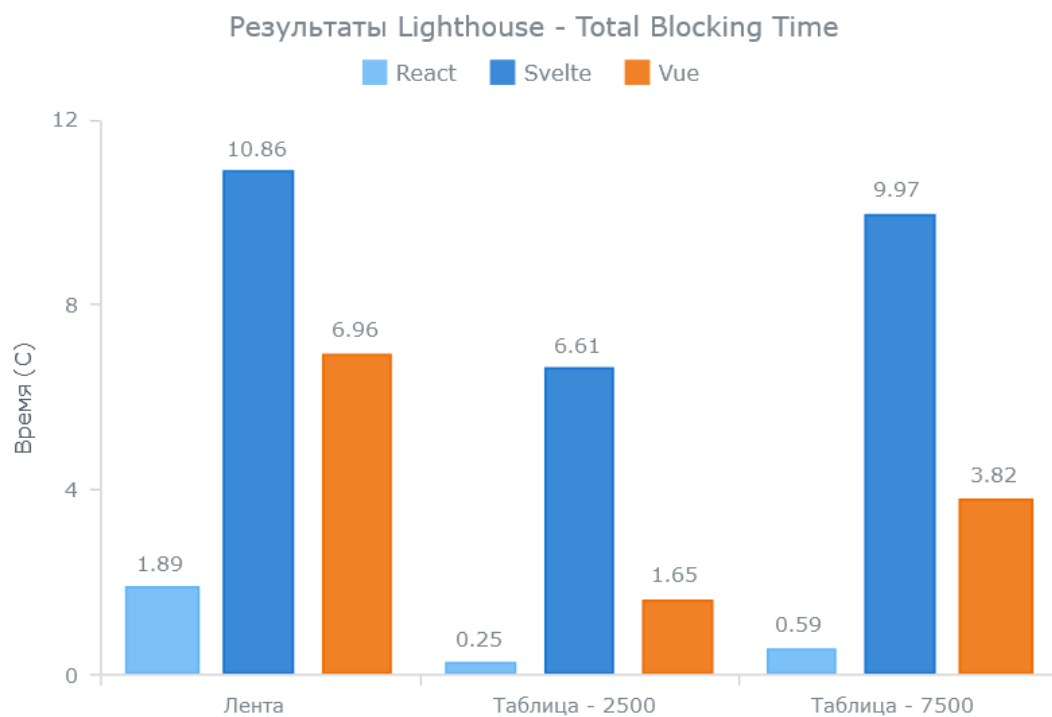


Рисунок 9 – Lighthouse – Общее время блокировки основного потока

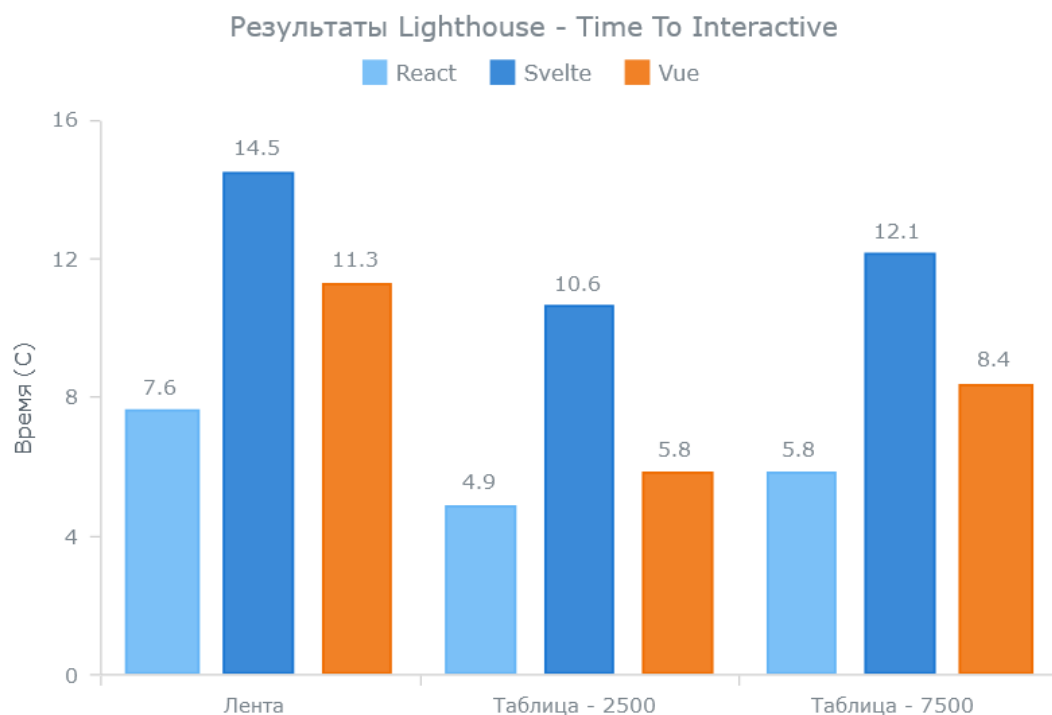


Рисунок 10 – Lighthouse – Время до интерактивности

Общий результат в рамках этого исследования показывает, что базовая комплектация Svelte оказывается медленнее своих конкурентов, но это может быть связано с плохой сборкой. Собранные метрики указывают на то, что разные фреймворки отличаются в эффективности на разных стадиях загрузки страницы, что указывает на различные техники оптимизации рендеринга, используемые в них.

ЗАКЛЮЧЕНИЕ

В ходе реализации курсового проекта были решены все поставленные задачи. Был разработан набор приложений для рендеринга одинаковых компонентов с маршрутизацией, сами приложения были собраны в Docker контейнеры и расположены на независимом сервере. Было разработано также приложение на Node для тестирования и систематизированного сбора метрик, характеризующих эффективность рендеринга веб-страниц. Разработан алгоритм обработки и визуализации полученных результатов. Были получены данные серий экспериментов, а результаты были проанализированы. Созданы страницы отчета с результатами и анализом проведенных экспериментов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Dockerizing a React App [Электронный ресурс] URL: <https://mherman.org/blog/dockerizing-a-react-app/> (дата обращения: 10.05.2021).
2. Google developers [Электронный ресурс] URL: <https://developers.google.com> (дата обращения: 16.05.2021).
3. Web-vitals [Электронный ресурс] URL: <https://web.dev/vitals/> (дата обращения: 17.05.2021).

"НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО"
(УНИВЕРСИТЕТ ИТМО)

ОТЗЫВ РУКОВОДИТЕЛЯ
о выполнении курсового проекта (работы)

Студент Летон Г.
(Фамилия, И., О.)

Факультет ПИиКТ Группа Р41071

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: Анализ и сравнение метрик при рендеринге таблиц и ленты новостей в трёх фреймворках
(Наименование темы)

ОЦЕНКА КУРСОВОГО ПРОЕКТА (РАБОТЫ)

№ п/п	Показатели	Оценка			
		5	4	3	0
1.	Проект создан обучающимся самостоятельно				
2.	Созданные приложения раскрывают тему исследования и позволяют объективно оценить результаты				
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 ответственной, готовой к изучению нового материала. В процессе работы студент подтвердил свои навыки в области проектирования веб-приложений. Студент стремился получить наиболее объективные результаты как при разработке приложений, так и при анализе полученных данных. Эксперимент, проведенный студентом, позволил раскрыть тему исследования.

Отмеченные недостатки:**Заключение:**

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

Руководитель

(подпись)

И.Б. Государев

« ____ » _____ 20__