Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Пермский государственный аграрно-технологический университет
имени академика Д. Н. Прянишникова»

Кафедра информационных технологий и программной инженерии

ОТЧЁТ О ТЕХНОЛОГИЧЕСКОЙ ПРАКТИКЕ

Выполнил:
Студент группы ПИм-11
направления подготовки
09.04.03 «Прикладная информатика»
Хайрутдинов Кирилл Михайлович

Проверил:
доцент кафедры ИТиПИ, к.т.н.,
Беляков Андрей Юрьевич
Оценка/Дата_____

Пермь 2023

СОДЕРЖАНИЕ

Введение

На ряду с информативной наполненностью, актуальностью, достоверностью не менее важным признаком хорошего сайта является скорость загрузки, особенно это касается сайтов, занимающихся продажами, клиент не будет ждать 15–20 секунд, пока загружается сайт, и через 3 секунды уйдет на сайт конкурентов.

Одним из видов деятельности компании ООО «ОМЕГА» является разработка рекламных лендингов для многих автодилеров России и других компаний. Для этих сайтов удержание пользователей имеет решающее значение для повышения конверсии. Медленная скорость сайтов негативно влияет на доход, а быстрые сайты повышают коэффициент конверсии.

В настоящее время на нескольких десятках лендингов, находящихся на поддержке компании, наблюдается медленная первичная загрузка, исходя из этого перед разработчиками стоит задача оптимизировать скорость загрузки.

Целью учебной технологической практики является разработка программного решения для оптимизации скорости загрузки веб-страницы.

Задачами учебной технологической практики являются:

- постановка проблемы исследования;
- анализ существующих библиотек для отложенной загрузки;
- обоснование выбора технологий проектирования;
- описание методики проведения испытаний эффективности разработки;
- разработка прототипа системы.

1. Постановка задачи на проектирование информационной системы

Наиболее частыми причинами медленной загрузки [1] сайта являются:

1) медленное время ответа сервера;
2) блокирующие рендеринг JavaScript и CSS;
3) медленное время загрузки ресурсов;
4) клиентский рендеринг.

Если же рассматривать 3 пункт подробнее, то можно выделить 2 способа оптимизации загрузки ресурсов – это сжатие и отложенная загрузка.

Проведя простой анализ, можно предположить, что применение даже одного метода, отложенной загрузки, может сильно сократить время первичной загрузки, ведь браузеру при отображении страницы необходимо будет вместо всех медиаресурсов, загрузить только те, которые отображаются на первом экране и видны пользователю сразу после захода на сайт.

Одним из методов, направленных на повышение скорости первичной загрузки сайта, является отложенная загрузка изображений, использование которой позволяет загружать только те изображения, которые попали в видимую область экрана.

Существует множество библиотек, реализующих отложенную загрузку изображений, например lazysizes, vanilla-lazyload и lozad.js, но их использование может быть проблематичным по следующим причинам:

1) размер сторонней библиотеки может превышать размер самостоятельно написанного кода, так как помимо необходимого функционала она может содержать другие функции, которые являются излишними;

2) чем больше сторонних библиотек используется в проекте, тем больше риск появления зависимостей в приложении.

По этим причинам необходимо разработать свое программное решение, реализующее отложенную загрузку изображений. Программное решение

должно быть легковесным, безопасным и поддерживать различные способы встраивания изображений на веб страницу.

Предположительно, использование программного решения для отложенной загрузки изображений должно заметно повысить скорость первичной загрузки веб-страницы, что положительно скажется на конверсии.

2. Анализ технологий проектирования

Можно выделить 2 основных способа реализации отложенной загрузки изображений:

- использование отложенной загрузки на уровне браузера;
- использование Intersection Observer.

У первого способа есть 2 минуса, во первых, его нельзя применить к фоновым изображениям CSS (свойство background и другие), во вторых, на текущий момент, у него низкая поддержка браузерами, тем временем второй способ, а именно Intersection Observer API, согласно сервису Can I Use, поддерживается почти всеми современными браузерами, что несомненно является плюсом.

Intersection Observer API позволяет веб-приложениям асинхронно следить за изменением пересечения элемента с его родителем или областью видимости документа viewport.

Intersection Observer API позволяет указать функцию, которая будет вызвана всякий раз для элемента (target) при пересечении его с областью видимости документа (по умолчанию) или заданным элементом (root) [2].

Целевой элемент для просмотра (target) можно определить с помощью CSS класса или идентификатора и в дальнейшем найти его с использованием функции document.querySelector, но это для обычных приложений.

Для приложений написанных на JavaScript-фреймворке Vue.js, именно он используется в большинстве случаев для разработки в компании ООО

«ОМЕГА», следует использовать другой способ, описанный в официальной документации к фреймворку:

Custom directives are mainly intended for reusing logic that involves low-level DOM access on plain elements [3]. В рамках нашей задачи по ленивой загрузке изображений как раз нужен низкоуровневый доступ к DOM, так как один из способов отложить загрузку это – в нужный момент, а именно в момент пересечения целевого объекта (DOM элемента, содержащего изображение) с заданным элементом, вставить ссылку на изображение в атрубут src, который до этого момента должен оставаться пустым или в лучшем случае содержать ссылку на заглушку, для лучшего пользовательского опыта.

Заглушка позволяет пользователю понять, что на этом месте должен быть какой-то элемент и все работает как должно, ею может служить CSS анимация загрузки, GIF анимация или же какие-то более сложные реализации, например использование фигуры цвета, определенного как наиболее используемый в оригинальном изображении (пример на рисунке 1) или размытое изображение худшего качества, такое изображение будет иметь ощутимо меньший размер, как и любой из вышеописанных вариантов.

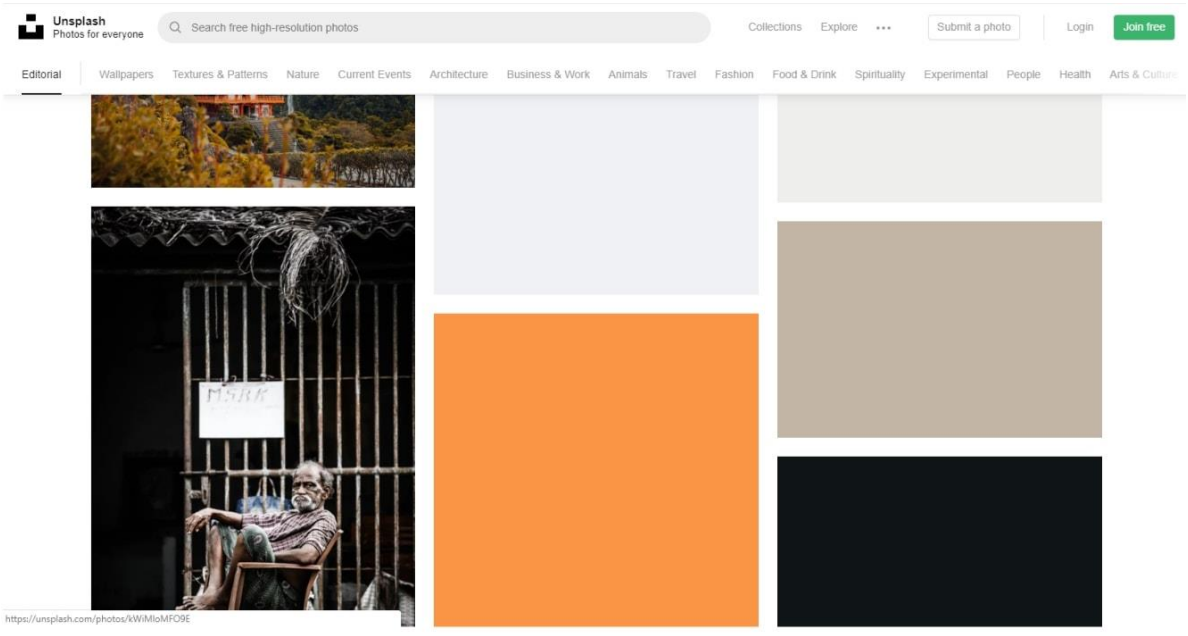

Рисунок 1 – Пример заглушки на сайте unsplash.com [4]

3. Методика проведения испытаний информационной системы

Основным критерием эффективности работы отложенной загрузки изображений является время первичной загрузки, но наравне с временем производительность следует рассматривать с точки зрения объективных критериев, которые можно измерить количественно с достаточной степенью точности. Эти критерии известны как метрики.

Главные метрики для отслеживания:

Первая отрисовка контента (FCP): измеряет время от начала загрузки страницы до отображения на экране любой части содержимого страницы.

Скорость загрузки основного контента (LCP): измеряет время от начала загрузки страницы до момента, когда на экране отображается самый большой текстовый блок или элемент изображения.

Время ожидания до первого взаимодействия с контентом (FID): измеряет время от момента, когда пользователь впервые взаимодействует с сайтом (то есть, когда он щелкает ссылку, нажимает кнопку или использует настраиваемый элемент управления на основе JavaScript) до момента, когда браузер действительно сможет ответить на это взаимодействие.

Время до интерактивности (TTI): измеряет время с начала загрузки страницы до момента ее визуального отображения, загрузки начальных сценариев (если таковые имеются) и способности страницы быстро и надежно реагировать на ввод пользователя.

Общее время блокировки (TBT): измеряет общее количество времени между FCP и TTI, когда основной поток был заблокирован на достаточно долгое время, чтобы предотвратить реакцию на ввод.

Совокупное смещение макета (CLS): измеряет совокупную оценку всех неожиданных сдвигов макета, которые происходят между моментом начала загрузки страницы и изменением ее состояния жизненного цикла на скрытое [6].

Вышеописанные метрики можно рассчитать с помощью ресурса PageSpeed Insights или встроенной в Google Chrome панели Lighthouse, она позволяет определить FCP, LCP, TBT, CLS, Speed Index и общий коэффициент производительности, который рассчитывается из этих метрик [5].

Для проведения испытаний необходимо сделать как минимум 2 замера в одинаковых условиях, первый – без использования отложенной загрузки, второй – с использованием. Также для точности измерений, веб-страница должна быть наполнена одним и тем же контентом, тестирование должно проводиться на одном и том же устройстве.

Проводить испытание будем на одностраничном сайте одного и того же содержания без сторонних скриптов.

4. Разработка и тестирование прототипа информационной системы

Пользовательские директивы в JavaScript-фреймворке Vue.js работают с помощью хуков жизненного цикла, таких как:

- created;
- beforeMount;
- mounted;
- beforeUpdate;
- updated;
- beforeUnmount;
- unmounted;

A custom directive is defined as an object containing lifecycle hooks similar to those of a component. The hooks receive the element the directive is bound to [3].

Основные хуки жизненного цикла, которые необходимы в разработке директивы для отложенной загрузки изображений – это created, mounted и updated, рассмотрим их использование подробнее.

В хуке created, который вызывается до того, как элемент вмонтирован в DOM-дерево, можно добавить Inline стиль background: none, так как, указанные таким образом, стили имеют самый высокий приоритет [7], можно на этапе создания DOM-элемента обнулить свойство background, тем самым браузеру будет нечего отображать и никакое изображение не загрузиться сразу. Код представлен в листинге 1.

Листинг 1 – Код хука created

```
created: (el) => {
  el.style.background = 'none'
}
```

В хуке mounted, который вызывается после created, можно указать функцию, которая будет выполняться при пересечении элементом области видимости, а также указать другие свойства объекта Intersection Observer, такие как rootMargin и threshold. Код представлен в листинге 2.

Листинг 2 – Код хука mounted

```
mounted: (el) => {
  function loadImage() {
    if (el.dataset.src) {
      el.src = el.dataset.src
    }

    el.style.removeProperty('background')
  }

  function callback(entries, observer) {
    entries.forEach(entry => {
      if (entry.isIntersecting) {
        loadImage()
        observer.unobserve(el)
      }
    })
  }

  function createIntersectionObserver() {
    const options = {
      root: null
    }

    const observer = new IntersectionObserver(callback, options)
    observer.observe(el)
```

```
    }

    if (!window['IntersectionObserver']) {
      loadImage()
    } else {
      createIntersectionObserver()
    }
  },
```

Сначала программа проверяет, поддерживает ли браузер Intersection Observer API, если нет, то запускает функцию loadImage, которая вставляет значение из data-src в src, тем самым позволяет браузеру загрузить изображение. Здесь data-src является data-атрибутом [8], позволяющим хранить информацию о пути к изображению. В этой же функции удаляется Inline стиль background: none.

Если же браузер поддерживает Intersection Observer API, то вызывается функция для создания объекта Intersection Observer для каждого целевого элемента. Сам элемент передается как аргумент, в хук mounted.

При пересечении целевого элемента с элементом root запускается функция loadImage.

Хук updated запускается при обновлении целевого элемента или его атрибутов, например, его можно использовать для замены изображения, при динамическом обновлении пути к изображению в атрибуте src. Код представлен в листинге 3.

Листинг 3 – Код хука updated

```
updated: (el) => {
  el.src = el.dataset.src
},
```

Веб-приложения, разрабатываемые в компании ООО «ОМЕГА», в основном, имеют схожее наполнение, поэтому возьмем для тестирования один из них. Результаты тестов без использования директивы для отложенной загрузки представлены на рисунке 2.

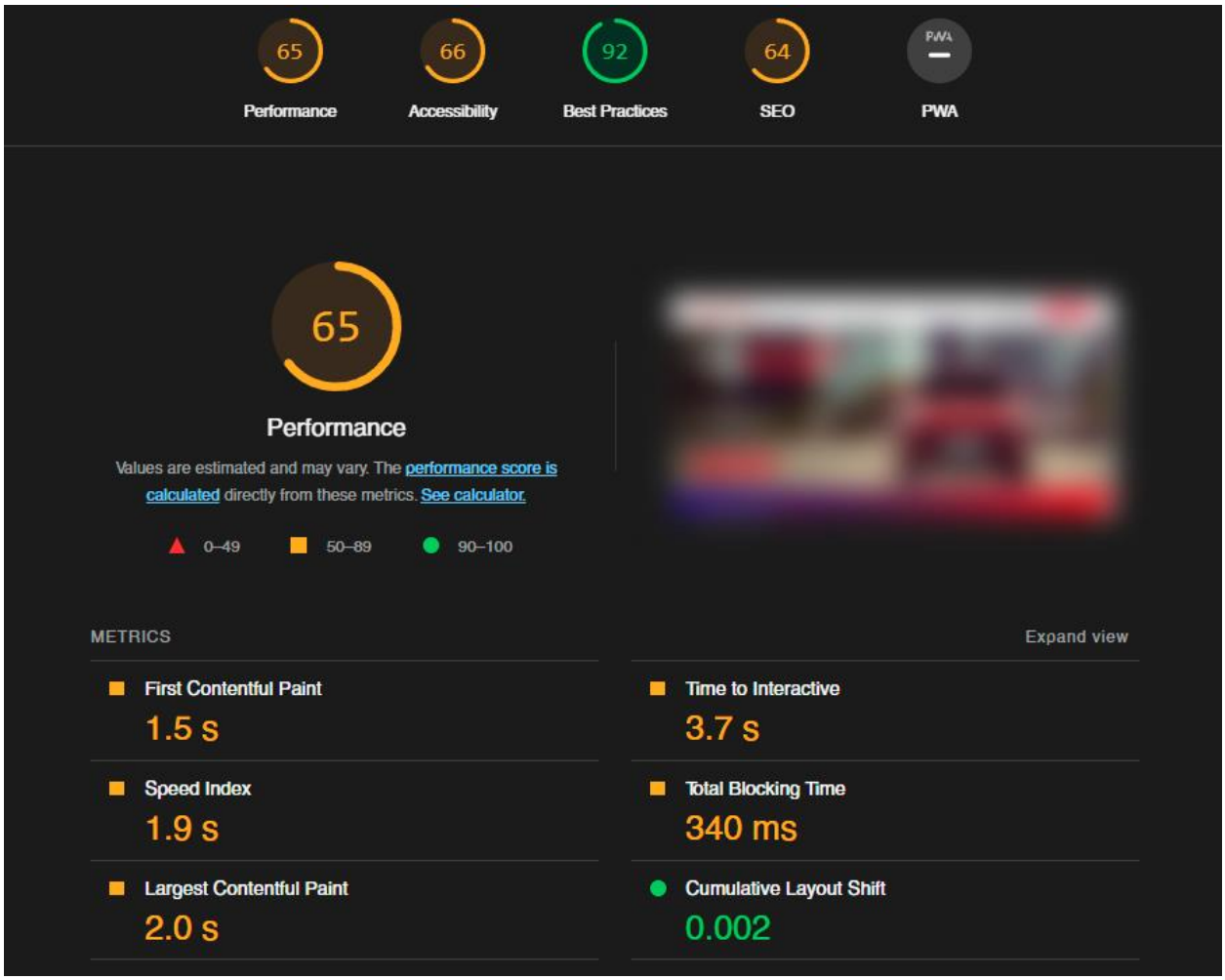

Рисунок 2 – Результаты аудита Lighthouse без применения директивы

Первая отрисовка контента происходит через 1,5 секунды после открытия сайта, а самый большой элемент отображается после 1,9 секунд, когда время до интерактивности составляет 3,7 секунды, что не критично, но достаточно много.

Результаты тестов с использованием директивы представлены на рисунке 3.

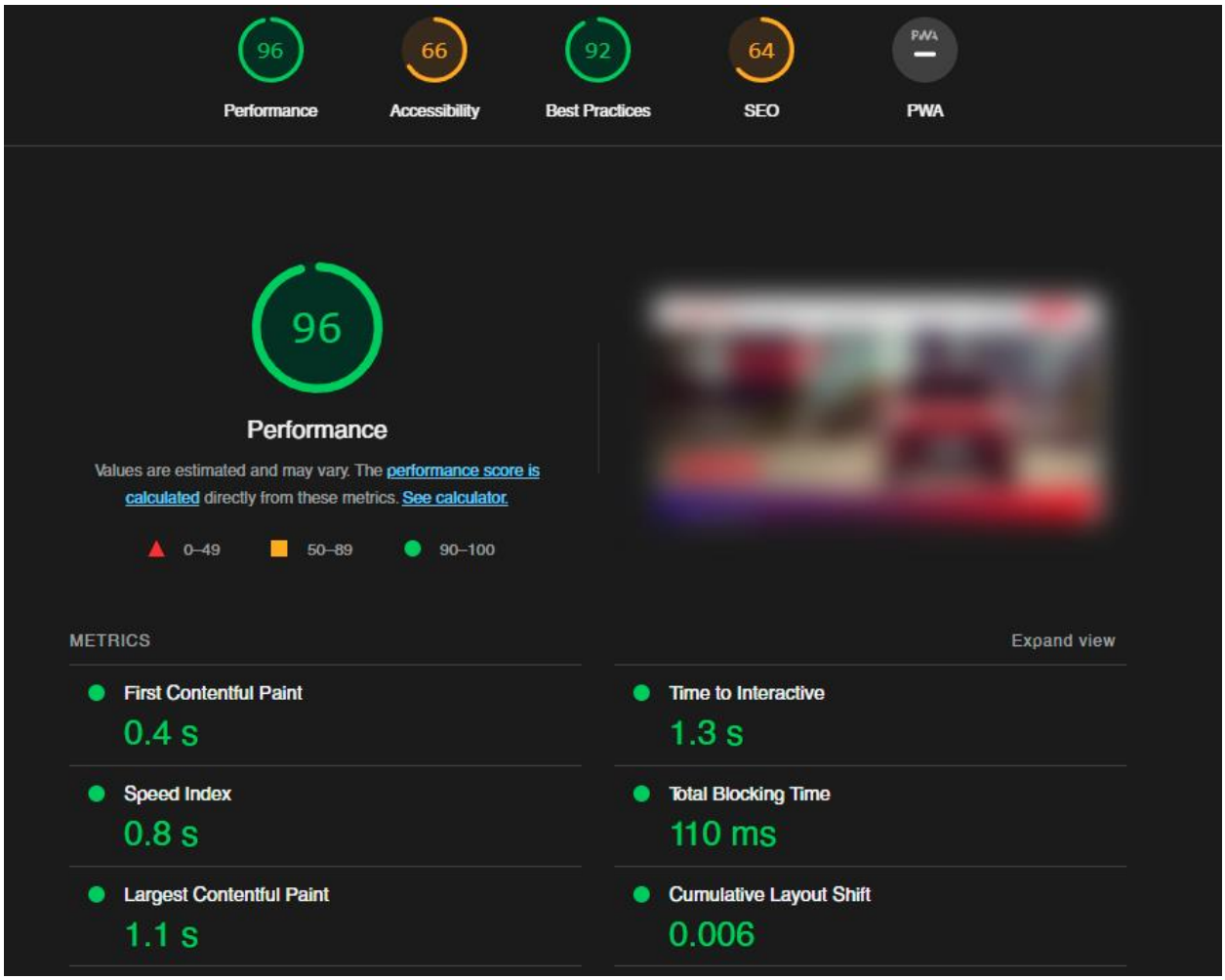

Рисунок 3 – Результаты аудита Lighthouse после применения директивы

Первая отрисовка контента происходит через 0,4 секунды, самый большой элемент через 1,1 секунды, а время до интерактивности занимает 1,3 секунды. Общий индекс производительности вырос на 31 пункт (прирост составил 147%).

Заключение

В рамках учебной технологической практики была постановлена проблемы исследования, проведен анализ существующих библиотек для отложенной загрузки изображений, обоснован выбор технологий проектирования, описана методика проведения испытаний эффективности разработки и разработан прототип системы.

Достигнута цель по разработке программного решения для оптимизации скорости загрузки веб-страницы.

Было проведено успешное испытание, которое показало большой прирост метрик Lighthouse, а именно:

- **FCP**: 1,5 – 0,4 = 1,1 секунды;
- **Speed Index**: 1,9 – 0,8 = 1,1 секунды;
- **LCP**: 2,0 – 1,1 = 0,9 секунды;
- **TTI**: 3,7 – 1,3 = 2,4 секунды;
- **TBT**: 340 – 110 = 230 миллисекунд;
- **CLS**: 0,006 – 0,002 = 0,004;
- **Performance Score**: 96 – 65 = 31.

Гипотеза о том, что применение отложенной загрузки изображений сократит время первичной загрузки подтверждена.

В будущем можно улучить разработку в следующих моментах:

- пересмотреть корректность использования некоторых хуков жизненного цикла в директиве для повышения производительности;
- рассмотреть возможность добавления полифилла при условиях, что браузер не поддерживает Intersection Observer API;
- рассмотреть возможность добавления заглушки при загрузке изображения.

Список литературы

1) Оптимизация скорости загрузки основного контента [Электронный ресурс]. — URL: https://web.dev/optimize-lcp (дата обращения: 26.05.2023).
2) Intersection Observer API [Электронный ресурс]. — URL: https://developer.mozilla.org/ru/docs/Web/API/Intersection_Observer_API (дата обращения: 26.05.2023).
3) Custom Directives [Электронный ресурс]. — URL: https://vuejs.org/guide/reusability/custom-directives.html (дата обращения: 26.05.2023).
4) Lazy loading или «ленивая загрузка» для изображений [Электронный ресурс]. — URL: https://siteclinic.ru/blog/internal-optimization/lazy-load (дата обращения: 26.05.2023).
5) Lighthouse. Performance Audits [Электронный ресурс]. — URL: https://developer.chrome.com/docs/lighthouse/performance (дата обращения: 26.05.2023).
6) Ориентированные на пользователя показатели производительности [Электронный ресурс]. — URL: https://web.dev/user-centric-performance-metrics/ (дата обращения: 26.05.2023).
7) MDN Web Docs. Specificity [Электронный ресурс]. — URL: https://developer.mozilla.org/en-us/docs/Web/CSS/Specificity (дата обращения: 26.05.2023).
8) Использование data-* атрибутов [Электронный ресурс]. — URL: https://developer.mozilla.org/ru/docs/Learn/HTML/Howto/Use_data_attributes (дата обращения: 26.05.2023).