

**МИНОБРНАУКИ РОССИИ**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Казанский национальный исследовательский технологический университет»**

**(ФГБОУ ВО «КНИТУ»)**

Кафедра «Интеллектуальных систем и управления информационными ресурсами»

Направление «02.03.03 – Математическое обеспечение и администрирование информационных систем»

Профиль «Информационные системы и базы данных»

Группа 4311-21

**КУРСОВОЙ ПРОЕКТ**

1. по дисциплине «Разработка пользовательского интерфейса»

на тему «Разработка клиентской части для инструмента анализа покрытия автоматизированными тестами»

|  |  |  |  |
| --- | --- | --- | --- |
| Исполнитель |  |  | Козлов Илья Валерьевич |
|  | (дата, подпись) |  | (Ф.И.О.) |
| Руководитель |  |  | доцент кафедры ИСУИР,  к.ф.-м.н. Мангушева А.Р. |
|  | (дата, подпись) |  | должность, (Ф.И.О.) |

|  |  |
| --- | --- |
| Проект защищен с оценкой |  |

|  |  |
| --- | --- |
| Руководитель |  |

Казань, 2024

СОДЕРЖАНИЕ

[ЗАДАНИЕ 3](#_Toc185357965)

[ВВЕДЕНИЕ 4](#_Toc185357966)

[1 АНАЛИЗ ТРЕБОВАНИЙ, ВЫДВИГАЕМЫХ К ПРОДУКТУ 6](#_Toc185357967)

[1.1 Функциональные требования 6](#_Toc185357968)

[1.2 Пользовательские истории 7](#_Toc185357969)

[1.3 Технические требования 7](#_Toc185357970)

[2 РАЗРАБОТКА МАКЕТА ВЕБ-ПРИЛОЖЕНИЯ 9](#_Toc185357971)

[3 ВЕРСТКА САЙТА ПО МАКЕТУ 13](#_Toc185357972)

[3.1 Создание структуры сайта 13](#_Toc185357973)

[3.2 Добавление стилизации 15](#_Toc185357974)

[4 ВНЕДРЕНИЕ ИНТЕРАКТИВНОСТИ 19](#_Toc185357975)

[5. СВЯЗЬ ФУНКЦИОНАЛА С СЕРВЕРНОЙ ЧАСТЬЮ 24](#_Toc185357976)

[ЗАКЛЮЧЕНИЕ 27](#_Toc185357977)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 28](#_Toc185357978)

[ПРИЛОЖЕНИЕ А 30](#_Toc185357979)

ЗАДАНИЕ

на курсовой проект студенту кафедры «Интеллектуальных систем и управления информационными ресурсами»

Тема проекта: «Разработка клиентской части для инструмента анализа покрытия автоматизированными тестами»

Исходные данные к проекту: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Содержание расчетно-пояснительной записки (включая перечень подлежащих разработке вопросов, включая вопросы стандартизации и контроля качества):

ВВЕДЕНИЕ

1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

2 АНАЛИЗ ТРЕБОВАНИЙ, ВЫДВИГАЕМЫХ К ПРОДУКТУ

3 ВЕРСТКА САЙТА ПО МАКЕТУ

4 ВНЕДРЕНИЕ ИНТЕРАКТИВНОСТИ

5 СВЯЗЬ ФУНКЦИОНАЛА С СЕРВЕРНОЙ ЧАСТЬЮ

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

ПРИЛОЖЕНИЕ А

Перечень графического материала (схемной документации) рисунки, таблицы

Консультанты по проекту (с указанием относящихся к ним разделов) \_

Дата выдачи задания «\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_г.

Руководитель проекта \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(Мангушева А.Р. )

ВВЕДЕНИЕ

В современном информационном обществе, где технологические инновации становятся неотъемлемой частью повседневной жизни, понижается порог входа в профессии, связанные с информационными технологиями (ИТ). В связи с этим возрастает потребность в простых, удобных и функциональных интерфейсах не только со стороны конечного пользователя, но и со стороны команды, разрабатывающей продукт. К примеру, чтобы претендовать на должность стажера по ручному тестированию, достаточно пройти курсы продолжительностью в 1 месяц. Специалисту с подобным уровнем подготовки необходим наглядный и понятный интерфейс, отображающий статистику и позволяющий производить верхнеуровневый анализ покрытия автоматизированным тестами.

Создание веб-интерфейса, адаптированного к уникальным требованиям узконаправленных специалистов, может не только обеспечить удобство работы с данными о покрытии тестами, но также повысить производительность команды, снизить количество ошибок при интерпретации данных и упростить процесс принятия решений.

Целью данного курсового проекта является создание веб-интерфейса для инструмента, специально адаптированного под анализ покрытия автоматизированными тестами, который будет интуитивно понятен и функционален.

Для достижения поставленной цели выдвигаются следующие задачи:

1. Сформулировать требования, выдвигаемые к продукту.

2. Разработать макет клиентской части, отвечающий сформулированным функциональным требованиям.

3. Создать визуальную структуру сайта, используя простые языки разметки.

4. Внедрить интерактивность с жестко прописанными в программном коде данными.

5. Добавить связку клиентской части с серверной.

1 АНАЛИЗ ТРЕБОВАНИЙ, ВЫДВИГАЕМЫХ К ПРОДУКТУ

1.1 Функциональные требования

Разрабатываемое веб-приложение должно предоставлять пользователю возможность увидеть сводную статистику по покрытию автоматизированными тестами, которая передается с бэкенд части посредствам Application Programming Interface (API). Каждая метрика должна быть представлена прогресс-баром для упрощения визуального анализа.

Кроме общей статистики, необходимо дать возможность просматривать статистику по каждому запуску инструмента для подсчета покрытия. Информация о каждом запуске хранится в базе данных (БД) и отправляется на клиентскую часть сервером с помощью API[3].

Должна быть возможность увидеть полную информацию о файле, хранящем метрики, а также необходимо учесть, что возможно потребуется скачать этот файл на компьютер пользователя.

В статистике должны отображаться следующие данные:

- покрытие строк кода;

- покрытие функций;

- покрытие ветвлений;

-покрытие операторов.

Для упрощения навигации по результатам запусков необходимо добавить фильтрацию данных по уровню покрытия по метрикам и по дате загрузки файла.

Также необходимо предоставить возможность загружать файл с данными о покрытии тестами в формате JSON.

Интерфейс должен корректно отображаться на устройствах с различными разрешениями экрана, включая мобильные телефоны, планшеты и настольные компьютеры.

Все элементы интерфейса должны быть легко доступными и понятными даже для пользователей с минимальной подготовкой.

1.2 Пользовательские истории

1. Пользователь открывает приложение и видит визуально отделенные друг от друга секции загрузки файла, сводной статистики, фильтрации данных, информации по всем файлам.

2. Пользователь локально запускает инструмент по подсчету покрытия, у него должна быть возможность загрузить файл с данными на сервер. Обновленные данные должны отображаться в интерфейсе[1].

3. Пользователь просматривает сводную статистику, которая подсчитывается на сервере.

4. Пользователь фильтрует результаты прогонов по минимальному покрытию и по дате генерации отчета с помощью блока фильтрации.

5. Пользователь просматривает детальную информацию о файле, кликнув по интересующей его строке.

6. Пользователь скачивает детальную информацию о прогоне в виде JSON файла на локальный компьютер.

1.3 Технические требования

- Используемые технологии: HTML, CSS, JavaScript.

- Структура данных файла:

{

"file": "file 1",

"lines": 82,

"functions": 87,

"branches": 5,

"statements": 98,

"uploadTime": 1729774461458,

"id": "1"

}

- Загруженные данные должны сохраняться на сервере через соответствующий API-эндпоинт.

- Веб-приложение[2] должно взаимодействовать с внешним API для получения и обновления данных о покрытии тестами.

- Приложение должно корректно работать в современных браузерах (Chrome, Firefox, Safari).

2 РАЗРАБОТКА МАКЕТА ВЕБ-ПРИЛОЖЕНИЯ

Для верстки веб-страницы необходимо скомпоновать текстовые и графические элементы – создать макет. Для проектирования макета выбран графический редактор Figma. Он имеет понятный, легкий в осваивании интерфейс, а также позволяет отрисовать элементы сайта и веб-приложения, иллюстрации и векторную графику.

После создания нового проекта открывается редактор, позволяющий отрисовать будущий сайт[4] (рис. 2.1).

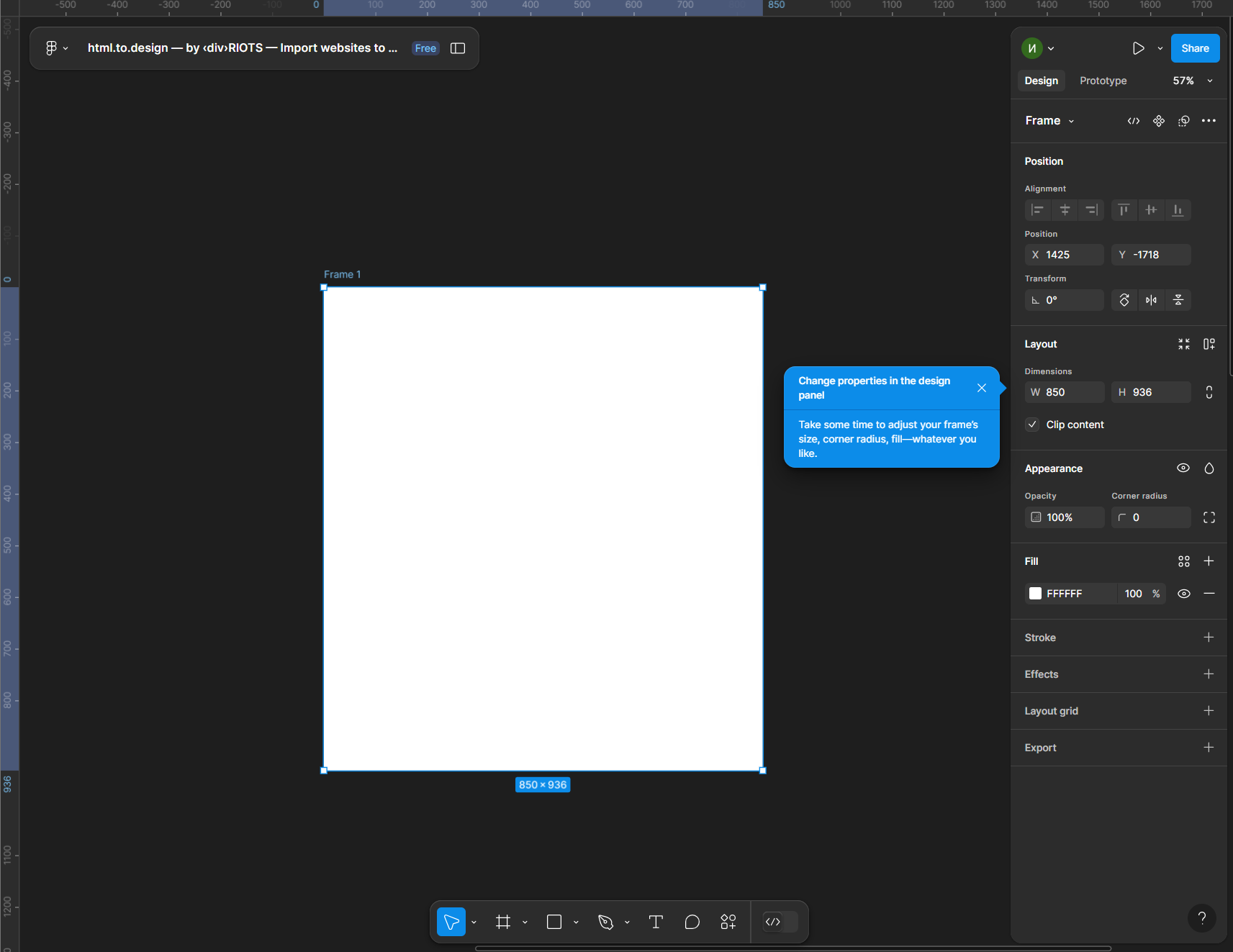


Рисунок 2.1 – Редактор Figma.

При создании макета необходимо опираться на проведенный ранее анализ требований. Все элементы необходимо располагать таким образом, чтобы среднестатистическому пользователю было очевидно, к чему они относятся. Блоки необходимо отделять друг от друга визуально и озаглавливать.

Исходя из функциональных требований и с учетом пользовательских сценариев была разработана следующая структура сайта:

1. Хедер, содержащий информацию о названии и назначении веб-приложения.

2. Блок загрузки файла, имеющий поле типа input для загрузки файла и кнопку для подтверждения действия.

3. Секция с суммарной статистикой по покрытию тестов. Для каждой из метрик необходим прогресс-бар для лучшего визуального восприятия.

4. Блок с фильтрами для облегчения навигации. Фильтр по минимальному покрытию, фильтр по датам и кнопка для подтверждения действия.

5. Таблица, содержащая детали о каждом файле.

Разработанный макет сайта содержит стандартные шрифты, которые будут читаться в разных браузерах. Для поддержки адаптивности[5] отрисованы две версии сайта:

- десктопная с шириной 1920 пикселей;

- мобильная с шириной 390 пикселей.

Кроме того, в процессе создания макета отрисованы элементы прогресс-бара и кнопки в уникальном стиле, они представлены на рисунке 2.2. Также на макете представлены все размеры в пикселях, чтобы при разработке было проще сверстать сайт.

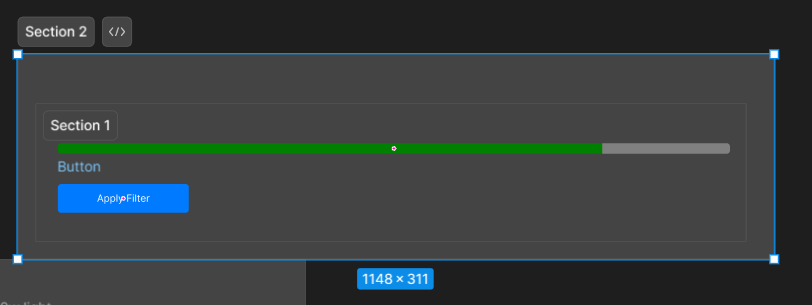


Рисунок 2.2 – Прогресс бар и кнопка.

На рисунке 2.3 представлена окончательная версия макета сайта для десктопа, а на рисунке 2.4 - окончательная версия макета сайта для мобильного веба.

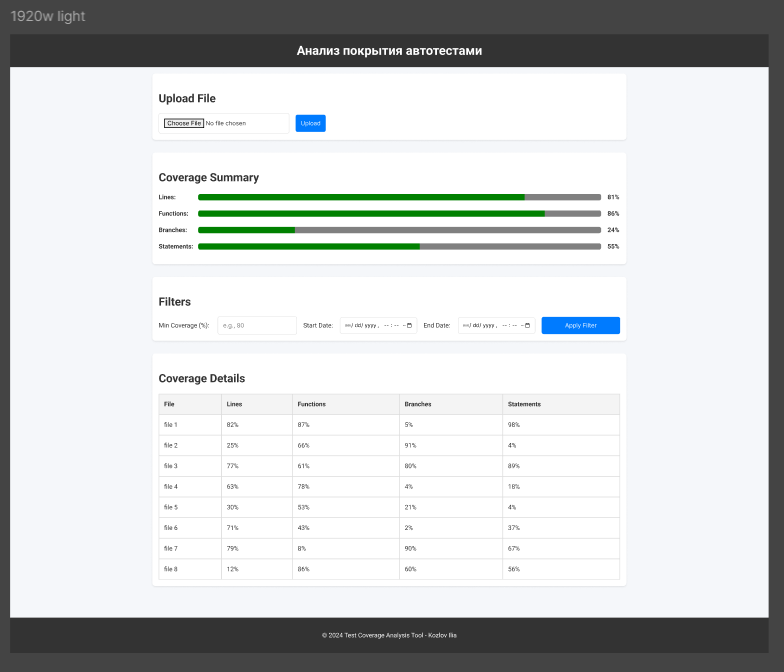


Рисунок 2.3 – Макет сайта для десктопа.

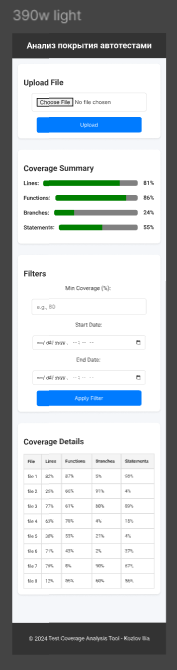


Рисунок 2.4 – Макет для мобильного веба.

3 ВЕРСТКА САЙТА ПО МАКЕТУ

3.1 Создание структуры сайта

Для создания каркаса сайта, основанного на готовом макете, необходимо использовать язык разметки HTML[6] (HyperText Markup Language — «язык гипертекстовой разметки»). Этот язык поддерживается всеми браузерами и позволяет создать блочную структуру с элементами, заранее определенными при проектировании макета.

Изначально необходимо создать «скелет» будущего сайта – исходя из определенной ранее структуры, с помощью тегов[7] разметки выделить блоки будущего веб-приложения (рис. 3.1).

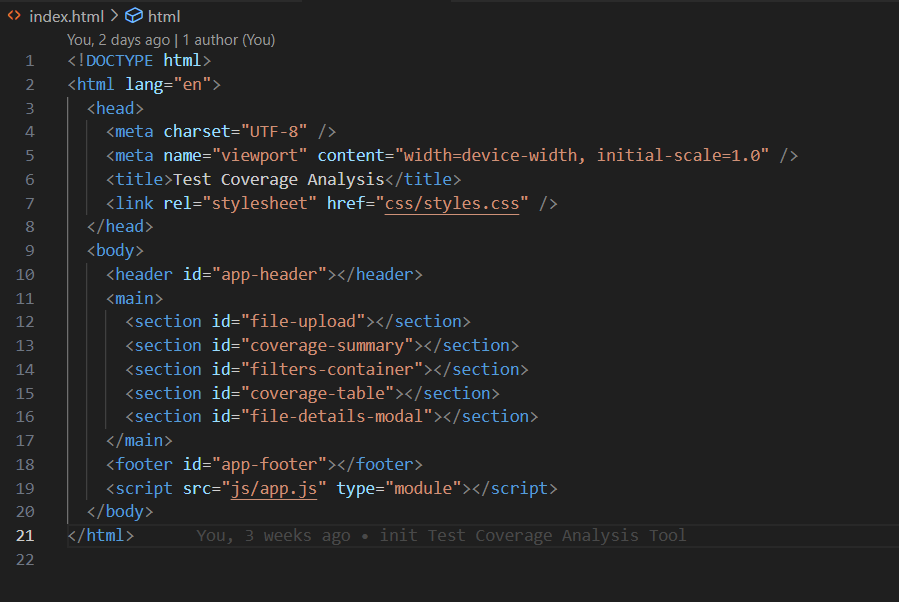


Рисунок 3.1 – Создание структуры сайта на с использованием HTML.

Далее с использованием стандартных тегов HTML формируется полная структура клиентской части (рис. 3.2). Полный код файла index.hml приведен в Приложении А. Результат проделанной работы представлен на рисунке 3.3.



Рисунок 3.2 – Стандартные HTML теги, составляющие структуру сайта.

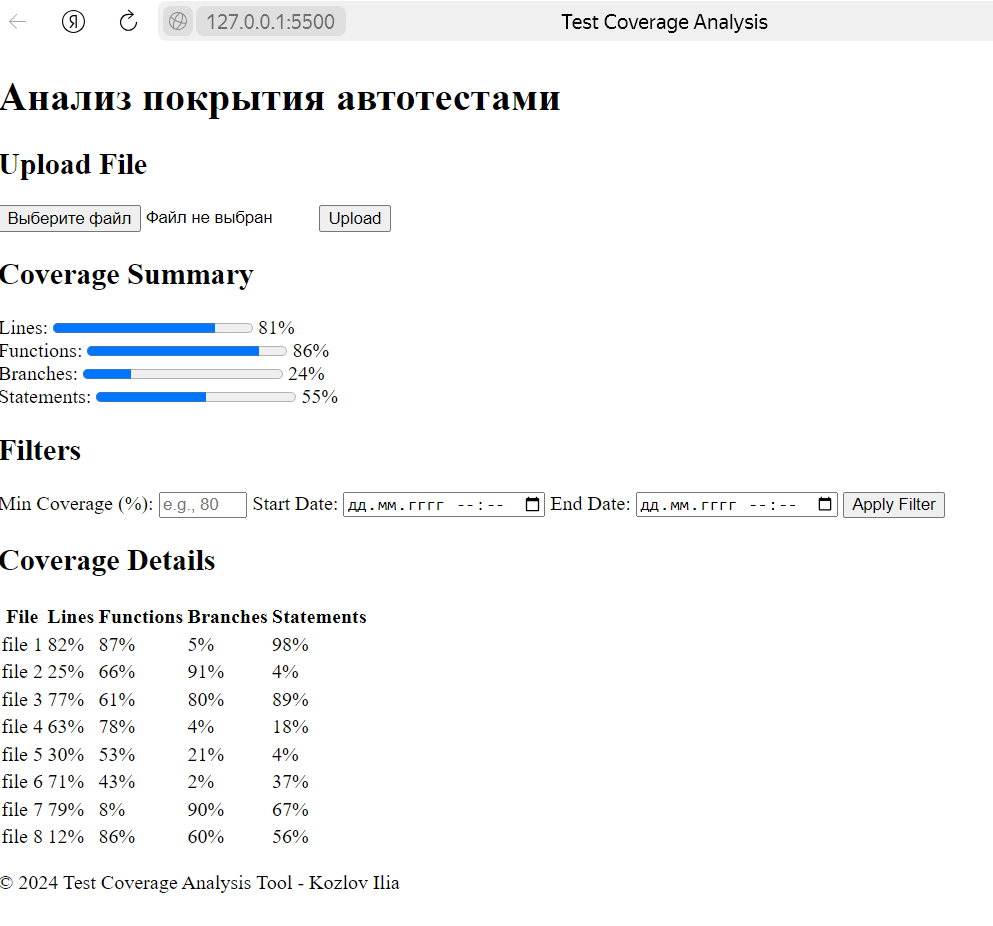


Рисунок 3.3 – Веб-интерфейс, построенный с помощью HTML.

3.2 Добавление стилизации

Для формирования внешнего вида страницы, написанной с использованием языка разметки HTML необходимо использовать формальный язык CSS (Cascading Style Sheets, «каскадные таблицы стилей»). Объявление стиля состоит из следующих частей:

- селектор – например, селектор h1 обозначает, что описанный стиль будет применен к любому тегу h1 на странице;

- свойство – например, font-size отвечает за размар текта;

- значение – например, 2em означает, что размер текста будет в 2 раза больше стандартного, используемого браузером.

Полностью стиль заголовка будет выглядеть следующим образом:

h1 {

margin: 0;

font-size: 2em;

}

Аналогичным образом необходимо стилизовать остальные элементы интерфейса (рис. 3.3). Полученный результат представлен на рисунке 3.4.

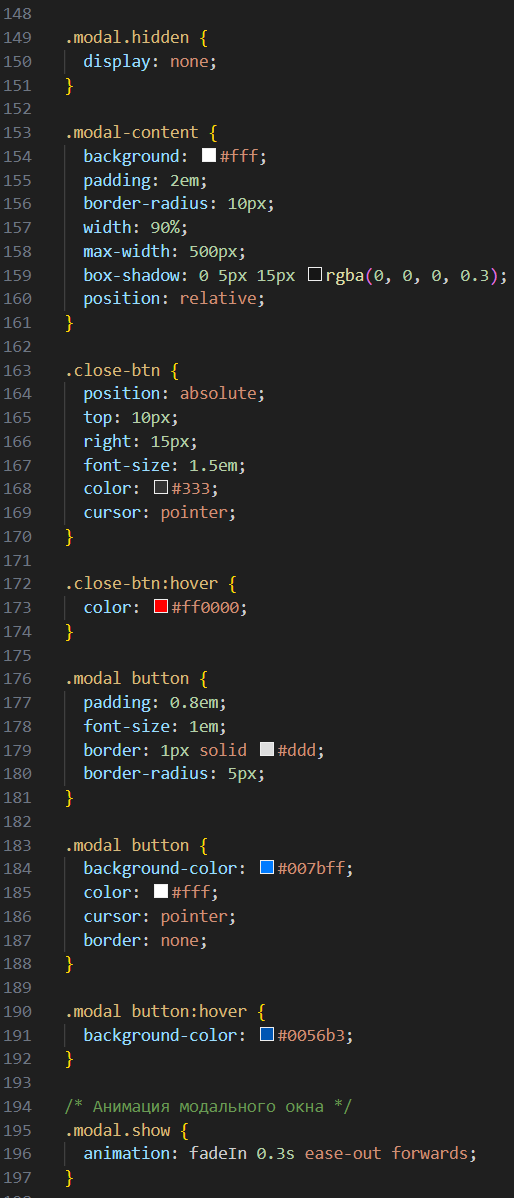


Рисунок 3.3 – Пример стилизации элементов интерфейса.

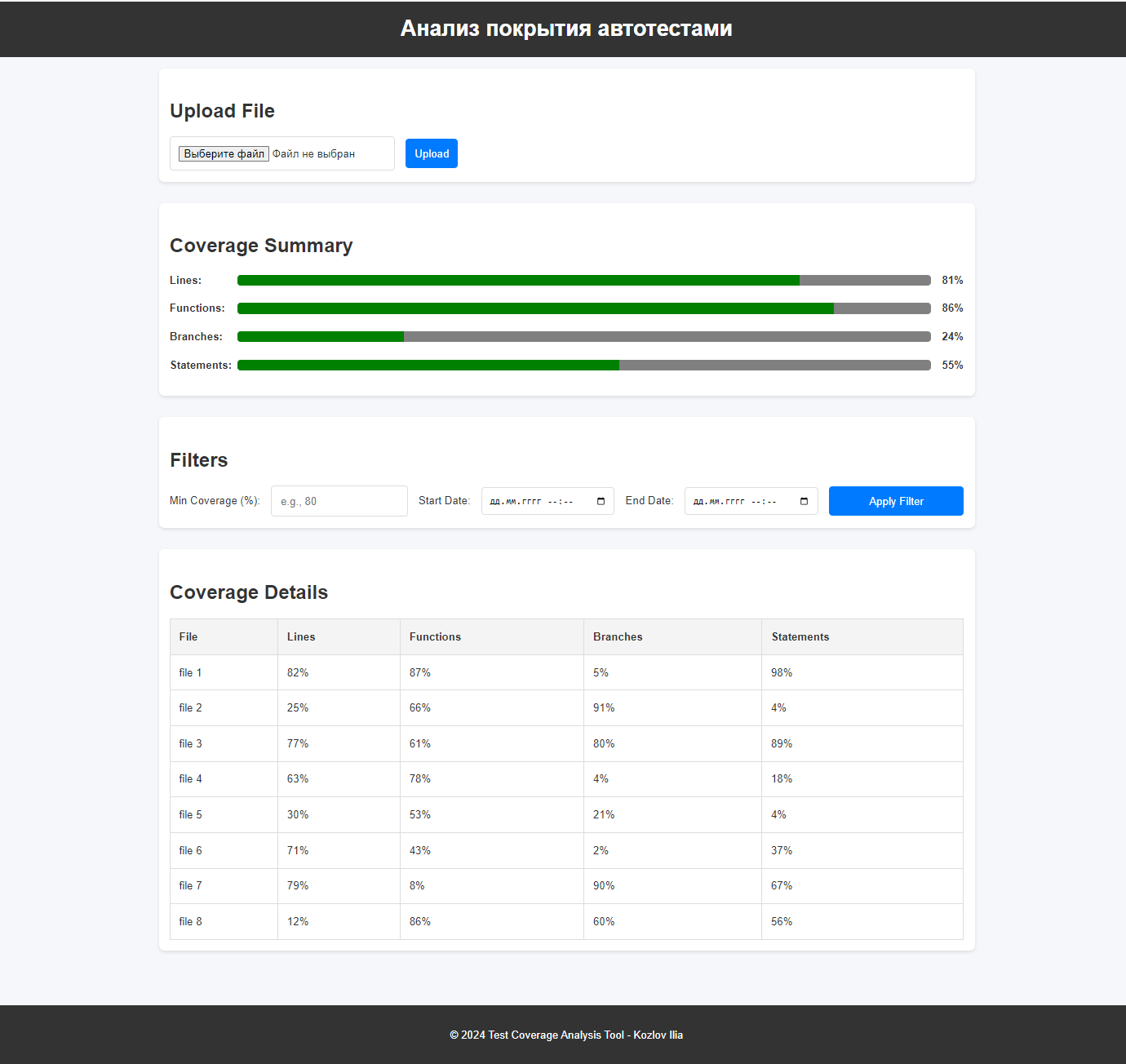


Рисунок 3.4 – Сайт с добавленной стилизацией.

Также согласно проведенному анализу требований и составленному макету, необходимо добавить адаптивность для получившегося сайта (рис.3.5).

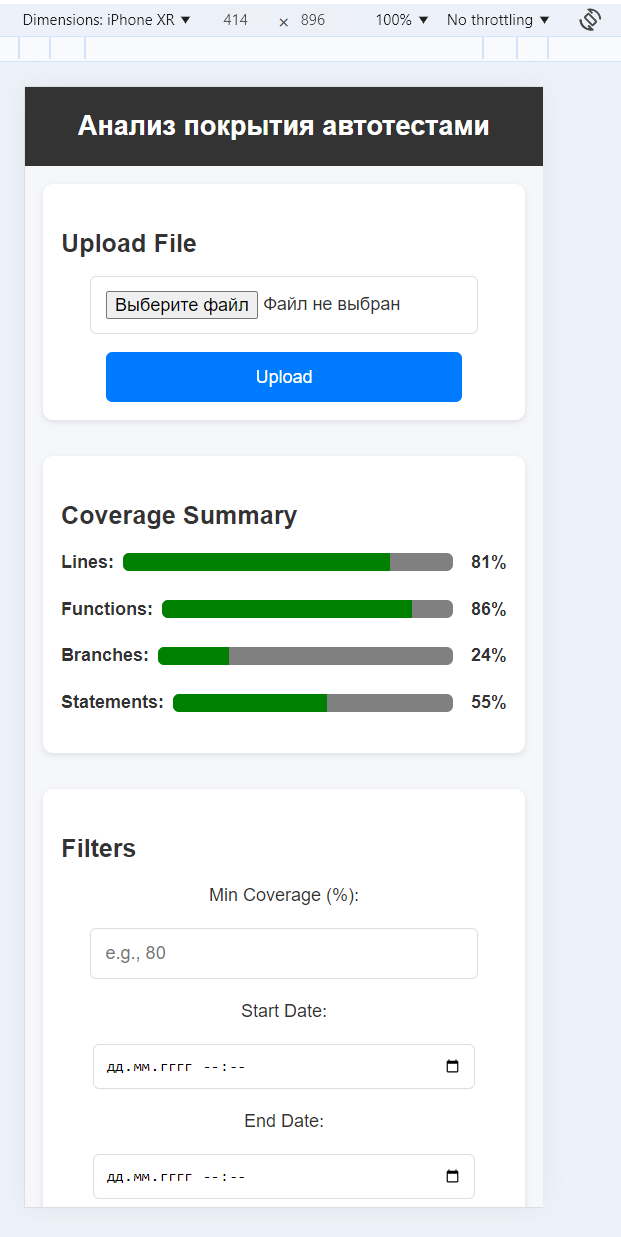


Рисунок 3.5 – Адаптивная верстка.

4 ВНЕДРЕНИЕ ИНТЕРАКТИВНОСТИ

Процесс внесения изменений в HTML документ крайне трудозатратен и неэффективен. Полученная ранее версия веб-приложения не зависит от данных на сервере и не поддерживает взаимодействие с пользователем. По этой причине необходимо внедрить интерактивность в приложение.

Для отделения логики приложения от пользовательского интерфейса и упрощения разработки был выбран язык прогаммирования JavaScript и архитектура MVP (Model-View-Presenter)[7]. Данная архитектура позволяет легко добавлять новые функции[8] или изменять существующие, не затрагивая другие части приложения, что обеспечивает простоту тестирования, поскольку каждый модуль изолирован.

Модель MVP предполагает разделение ответственности:

- model отвечает за данные и логику их обработки;

- view отвечает только за отображение данных;

- presenterуправляет взаимодействием между model и view.

Для проекта была разработана структура директорий и файлов[9], представленная на рисунке 4.1

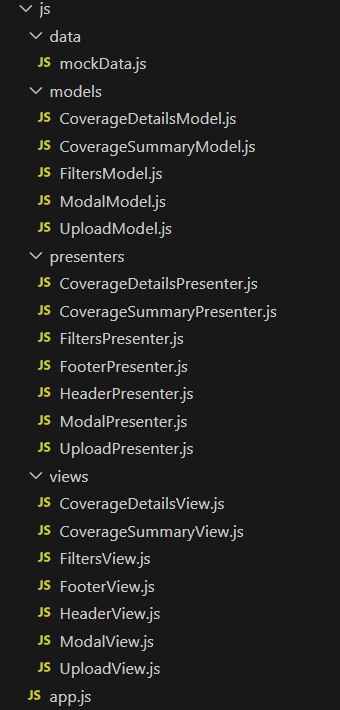


Рисунок 4.1 – Структура директорий и файлов.

В папке data временно хранятся жестко закодированные данные в формате JSON, необходимые для построения таблиц.

В папке models хранятся все модели для управления данными приложения. Также в модели можно выполнять операции с API[15], такие как загрузка или сохранение данных.

В папке views лежат файлы, генерирующие HTML на основе предоставленных данных, а также обрабатывающие взаимодействие пользователя с интерфейсом.

В папке presenters хранятся файлы, реализующие бизнес-логику приложения. Также в них обрабатываются пользовательские действия посредствам запросов данных у models и передачи их во views.

App.js – это центральная точка приложения[10], в ней инициализируются все компоненты и связываются между собой.

Рассмотрим пример выноса секции загрузки файлов из index.html:

- в классе UploadModel, приведенном на рисунке 4.2 содержится переменная files, хранящая массив загружаемых файлов, а также функция addFile, которая добавляет файл в этот массив;

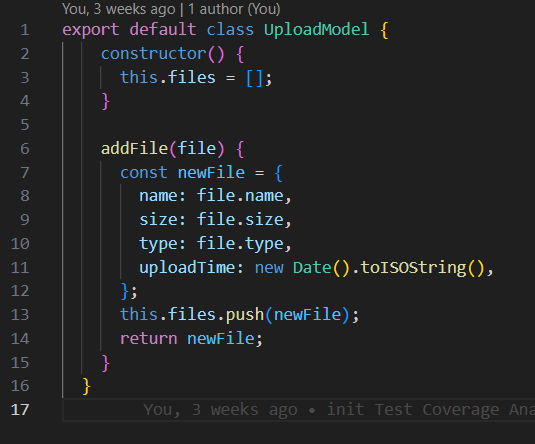


Рисунок 4.2 – Код класса UploadModel.

- в классе UploadView (рис.4.3) содержится функция render()[14], позволяющая отрисовать элемент в интерфейсе. За контейнер берется родительский элемент всего блока. Далее в контейнер с помощью изменения свойства innerHTML помещается код разметки HTML, написанный при верске сайта по макету. Также в классе есть функция bindFileUpload, в которой добавляется реакция на событие submit, которое возникает при нажатии кнопки;



Рисунок 4.3 – Код класса UploadView.

- в классе UploadPresenter реализована связь между моделью и представлением. При загрузке файла и последующем клике по кнопке Submit происходит вывод в консоль информации о файле. В дальнейшем содержимое файла будет отправляться на сервер для обновления таблицы;

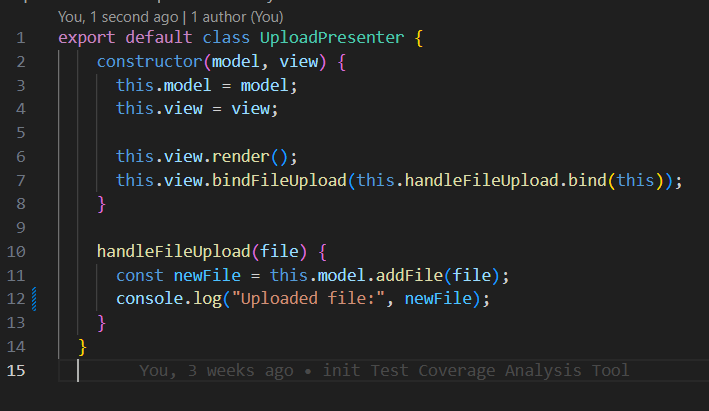


Рисунок 4.4 – Код класса UploadPresenter.

- инициализация всех ранее описанных классов происходит в файле app.js (рис. 4.5);

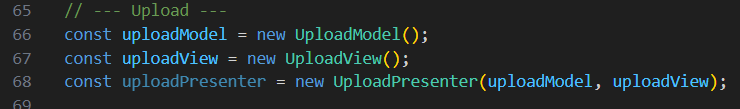


Рисунок 4.5 – Инициализация классов.

- после проделанной работы можно убрать содержимое секции file-upload из файла index.html. Обновленный тег выглядит следующим образом: <section id="file-upload"></section>.

Проделав аналогичные шаги с остальными секциями, можно добавить интерактивность для всех блоков из index.html. Финальный код этого файла представлен на рисунке 4.6. Теперь при изменении блока нет необходимости менять данный файл. Отображение веб-интерфейса при этом осталось прежним.

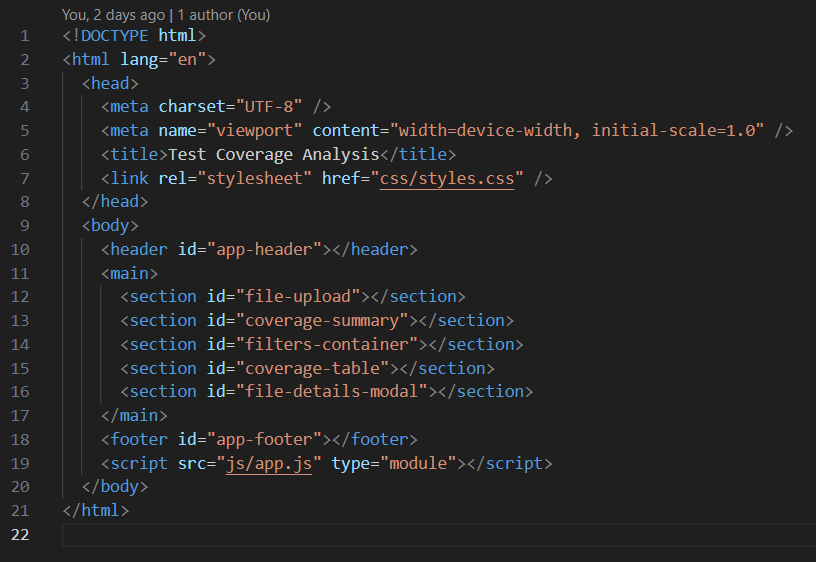


Рисунок 4.6 – Оптимизированный код файла index.html.

5. СВЯЗЬ ФУНКЦИОНАЛА С СЕРВЕРНОЙ ЧАСТЬЮ

В данный момент все данные прописаны заранее (замокированы), что не позволяет динамически взаимодействовать с веб-приложением[13]. Чтобы получать актуальную информацию с сервера и сохранять изменения, необходимо связать клиентскую часть с серверной по следующему алгоритму:

1. Создание и использование эндпоинтов API [11]. Задействован сервер MockAPI для хранения данных о покрытии тестов. Основной эндпоинт: /coverage-details – для работы с данными о покрытии.

2. Запросы к API. Функции, реализующие общение с сервером, представлены в файле ApiService.js (рис. 5.1). Обмен с сервером с помощью методов HTTP:

- GET для получения текущих данных с сервера

- POST для добавления новых файлов.

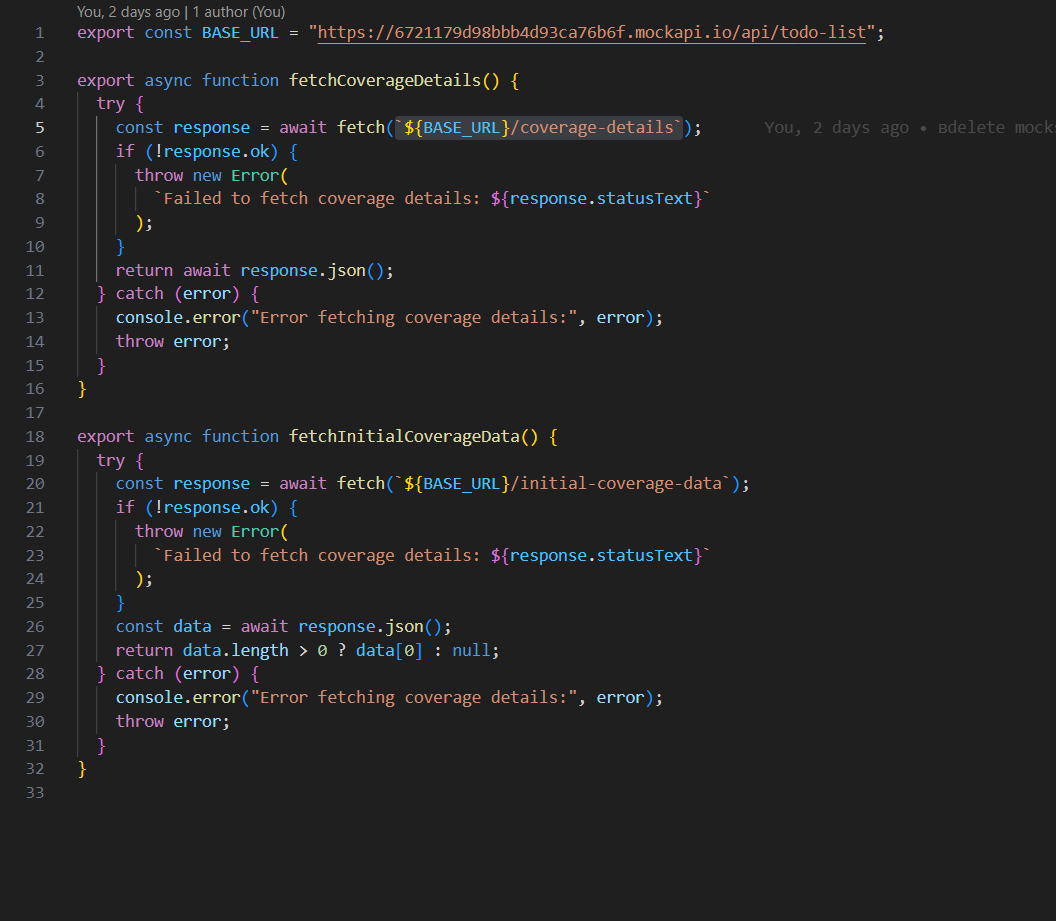


Рисунок 5.1 – Состав файла ApiService.js

4. Интеграция с функционалом. При загрузке файла из интерфейса он обрабатывается презентером и сохраняется на сервер[12] через модель (рис. 5.2). После отправки данных из файла на сервер страница перезагружается и обновленные данные запрашиваются повторно, а далее интерфейс рендерится на основе полученного ответа.

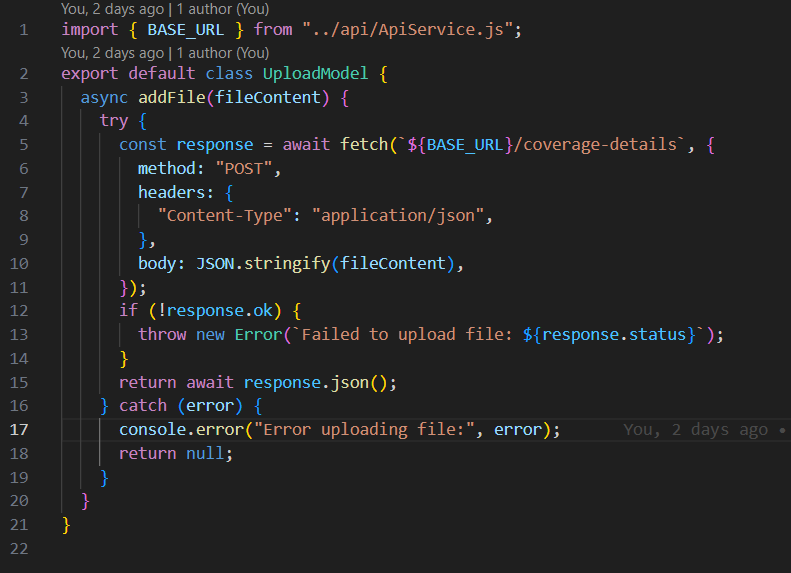


Рисунок 5.5 – Обработка загружаемого файла.

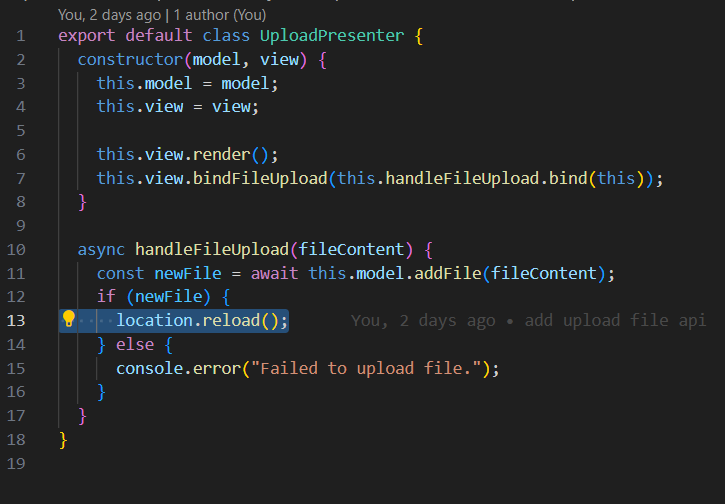


Рисунок 5.6 – Перезагрузка страницы после отправки данных на сервер.

Пример потока данных

1. Пользователь загружает файл через форму.

2. Презентер вызывает метод модели для отправки данных на сервер через POST-запрос.

3. Сервер добавляет новые данные и возвращает результат.

4. Приложение получает обновленные данные и рендерит их на странице.

ЗАКЛЮЧЕНИЕ

Во время выполнения курсового проекта были проанализированы функциональные и технические требования. Для разработки пользовательского интерфейса также были проработаны пользовательские истории.

С помощью инструмента Figma, опираясь на ранее составленные требования был спроектирован макет будущего приложения.

Статическая версия клиентской части была написана с использованием HTML и CSS. Она отличается тем, что может лишь показывать заранее запрограммированные данные в ограниченном количестве. На данном этапе поддержан респонсивный дизайн, подразумевающий возможность пользоваться как мобильными, так и стационарными устройствами.

С целью внедрения интерактивности и уменьшения затрат по обновлению HTML верстки проект был переведен на архитектуру MVP. Также данный переход позволил сделать страницу динамически составляемой. При переходе была сформирована структура файлов и папок, а также с помощью высокоуровневого языка JavaScript написаны классы для взаимодействия с данными. На этом этапе клиент хранит данные, обрабатывает и отображает их.

Благодаря реализации динамического клиент-серверного взаимодействия появилась возможность разграничить доступ к данным – клиент отвечает только за отображение данных, а сервер за их обработку и хранение. Также удалось добиться централизации данных, что обеспечивает согласованность и возможность их обновления в реальном времени с нескольких клентов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Пьюривал С. Основы разработки веб-приложений. - СПб.: Питер, 2015. - 272 с.

2. Мейер Эрик А. CSS. Карманный справочник. - 4-е изд. - М.: И.Д. Вильямс, 2016. - 288 с.

3. Вейл Э. HTML5. - СПБ.: Питер, 2015. - 480 с.

4. Скотт Чакон Git для профессионального программиста. - СПБ.: Питер, 2016. - 300 с.

5. Будевич К.В. Архитектуры программирования приложений // 55-я юбилейная научная конференция аспирантов, магистрантов и студентов БГУИР. – 2019

6. Учебник по JavaScript, начиная с основ, включающий в себя много тонкостей и фишек JavaScript/DOM // Современный учебник JavaScript URL: https://learn.javascript.ru/ (дата обращения: 10.11.2024).

7. JavaScript. Освой на примерах. - СПб.:БХВ-Петербург, 2007. - 400 с.

8. Резиг Джон JavaScript для профессионалов. - 2-е изд. - М.: Вильямс, 2016. - 240 с.

9. Браун Э. Изучаем JavaScript: руководство по созданию современных веб-сайтов - 3-е изд. - Санкт-Петербург: ООО «Альфа-книга», 2017. - 368 с.

10. Брокшмидт К. Пользовательский интерфейс приложений для

Windows 8, созданных с использованием HTML, CSS и JavaScript : - 2-е изд. - Москва, 2016. - 395 с.

11. Маккоу А. Веб-приложения на JavaScript - Санкт-Петербург: Питер, 2015. - 288 с.

12. Никольский А. П. JavaScript на примерах. Практика, практика и только практика - Санкт-Петербург: Наука и Техника, 2018. - 272 c.

13. HTML5BOOK.ru — HTML, CSS, JavaScript и jQuery [Электронный ресурс]. — Режим доступа: https://html5book.ru/ (дата обращения: 20.11.2024).

14. Metanit.com — Сайт о программировании [Электронный ресурс]. —

Режим доступа: https://metanit.com/ (дата обращения: 20.10.2024).

15. Wm-school.ru - Сайт для вебмастеров, учебники для вебпрограммистов [Электронный ресурс]. — Режим доступа: http://wm-school.ru/ (дата обращения: 01.12.2024).

ПРИЛОЖЕНИЕ А

Листинг сайта, сверстанного на HTML.

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>Test Coverage Analysis</title>

<link rel="stylesheet" href="css/styles.css" />

<style></style>

</head>

<body>

<header id="app-header">

<h1>Анализ покрытия автотестами</h1>

</header>

<main>

<section id="file-upload">

<h2>Upload File</h2>

<form id="file-upload-form">

<input type="file" id="file-input" accept="application/json" />

<button type="submit">Upload</button>

</form>

</section>

<section id="coverage-summary">

<h2>Coverage Summary</h2>

<div class="stats-bar">

<label>Lines:</label>

<progress id="lines-coverage" max="100" value="81"></progress>

<span id="lines-percentage">81%</span>

</div>

<div class="stats-bar">

<label>Functions:</label>

<progress id="functions-coverage" max="100" value="86"></progress>

<span id="functions-percentage">86%</span>

</div>

<div class="stats-bar">

<label>Branches:</label>

<progress id="branches-coverage" max="100" value="24"></progress>

<span id="branches-percentage">24%</span>

</div>

<div class="stats-bar">

<label>Statements:</label>

<progress id="statements-coverage" max="100" value="55"></progress>

<span id="statements-percentage">55%</span>

</div>

</section>

<section id="filters-container">

<h2>Filters</h2>

<form id="filter-form">

<label for="min-coverage">Min Coverage (%):</label>

<input

type="number"

id="min-coverage"

name="min-coverage"

min="0"

max="100"

step="1"

placeholder="e.g., 80"

/>

<label for="start-date">Start Date:</label>

<input type="datetime-local" id="start-date" name="start-date" />

<label for="end-date">End Date:</label>

<input type="datetime-local" id="end-date" name="end-date" />

<button type="submit">Apply Filter</button>

</form>

</section>

<section id="coverage-table">

<h2>Coverage Details</h2>

<table>

<thead>

<tr>

<th>File</th>

<th>Lines</th>

<th>Functions</th>

<th>Branches</th>

<th>Statements</th>

</tr>

</thead>

<tbody id="coverage-data">

<tr>

<td>file 1</td>

<td>82%</td>

<td>87%</td>

<td>5%</td>

<td>98%</td>

</tr>

</tbody>

</table>

</section>

<section id="file-details-modal" class="modal hidden"></section>

</main>

<footer id="app-footer">

<p>© 2024 Test Coverage Analysis Tool - Kozlov Ilia</p>

</footer>

<script src="js/app.js" type="module"></script>

</body>

</html>