

**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 |  |  | д-т кафедры ИСУИР, к.ф.-м.н. Мангушева А.Р. |
|  | (дата, подпись) |  | должность, (Ф.И.О.) |
| Нормоконтролер |  |  |  |
|  | (дата, подпись) |  | должность, (Ф.И.О.) |

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

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

Казань, 2024

СОДЕРЖАНИЕ

[ЗАДАНИЕ 3](#_Toc184770575)

[ЛИСТ НОРМОКОНТРОЛЕРА 4](#_Toc184770576)

[ВВЕДЕНИЕ 5](#_Toc184770577)

[1 Анализ требований, выдвигаемых к продукту 7](#_Toc184770578)

[2 Разработка макета веб-приложения 10](#_Toc184770579)

[3 Верстка сайта по макету 14](#_Toc184770580)

[4 Внедрение интерактивности 20](#_Toc184770581)

[5. Связь функционала с серверной частью 25](#_Toc184770582)

[ЗАКЛЮЧЕНИЕ 28](#_Toc184770583)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 29](#_Toc184770584)

[ПРИЛОЖЕНИЕ А 30](#_Toc184770585)

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

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. )

ЛИСТ НОРМОКОНТРОЛЕРА

1. Лист является обязательным приложением к пояснительной записке курсового проекта.

2. Нормоконтролер имеет право возвращать документацию без рассмотрения в случаях:

- нарушения установленной комплектности,

- отсутствия обязательных подписей,

- нечеткого выполнения текстового и графического материала.

3. Устранение ошибок, указанных нормоконтролером, обязательно.

Перечень

замечаний и предложений нормоконтролера по курсовому проекту, студента

(группа, инициалы, фамилия)

|  |  |  |
| --- | --- | --- |
| Л(Лист (страница) | Условное обозначение (код ошибок) | Содержание замечаний и предложений со ссылкой на нормативный документ, стандарт или типовую документацию |
|  |  |  |

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

Нормоконтролер \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (инициалы, фамилия)

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.

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.

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 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 (рис. 2.1).

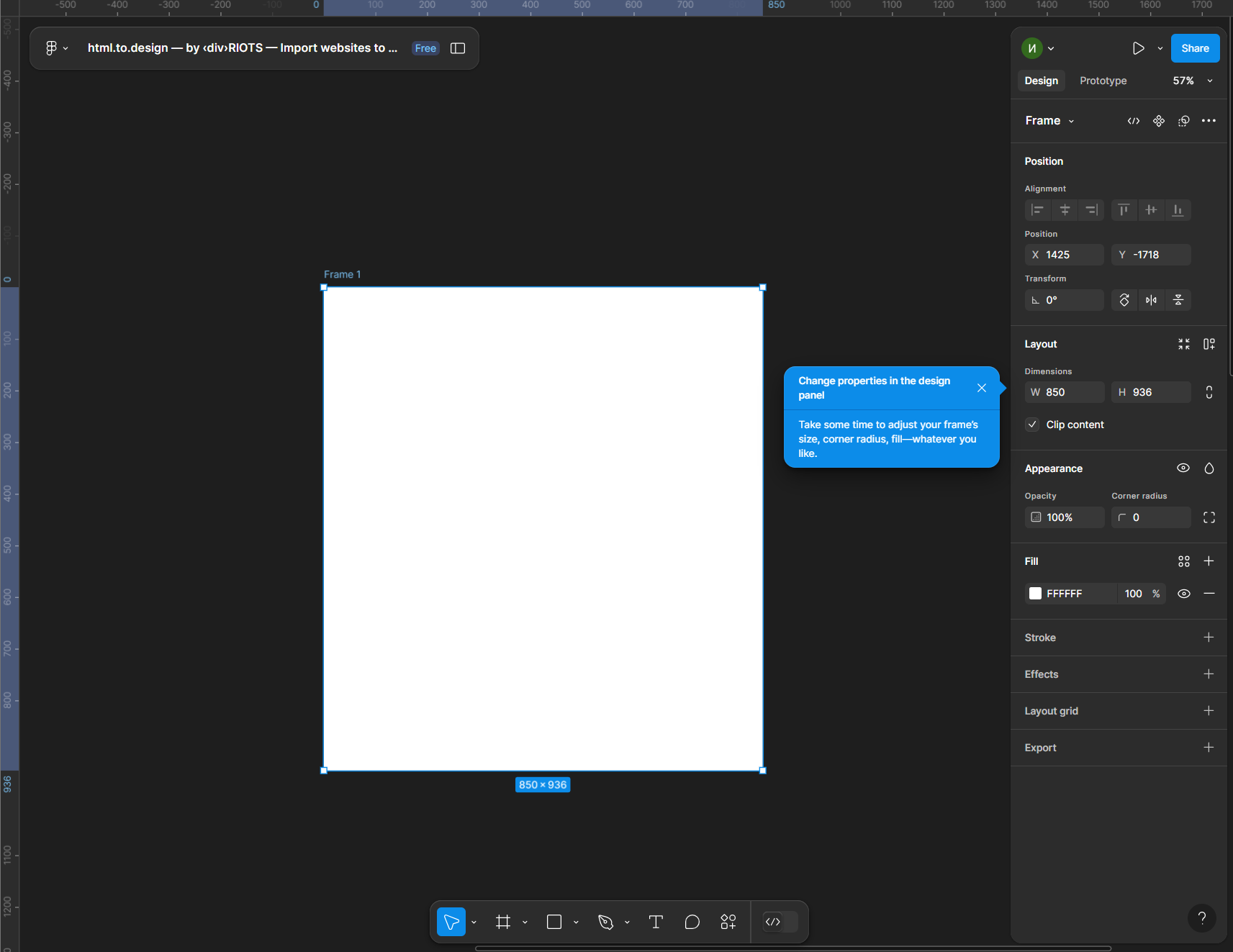


Рисунок 2.1 – Редактор Figma.

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 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.

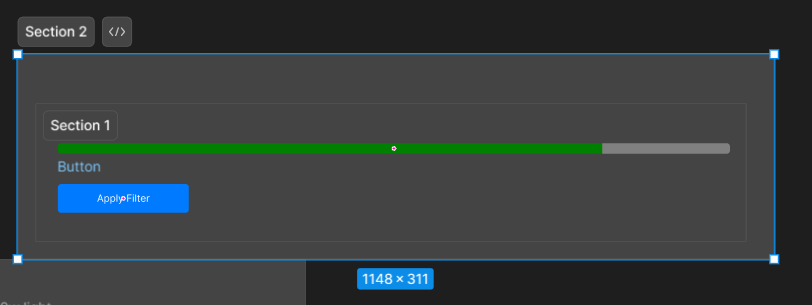


Рисунок 2.2 – Прогресс бар и кнопка.

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

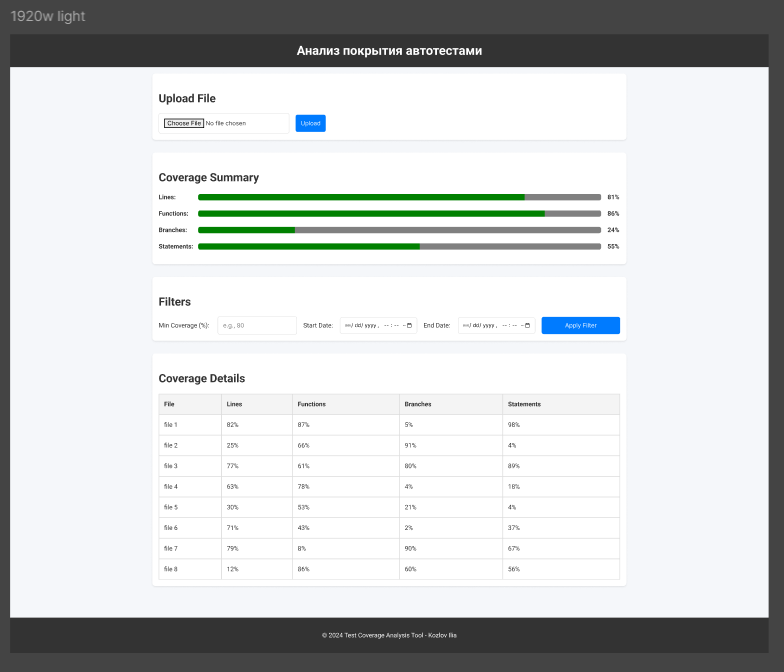


Рисунок 2.3 – Макет сайта для десктопа.

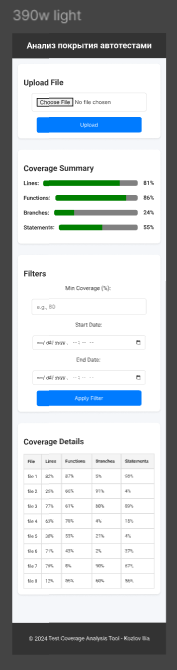


Рисунок 2.4 – Макет для мобильного веба.

**3 Верстка сайта по макету**

3.1 Создание структуры сайта

Для создания каркаса сайта, основанного на готовом макете, необходимо использовать язык разметки HTML (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 разметки выделить блоки будущего веб-приложения (рис. 3.1).

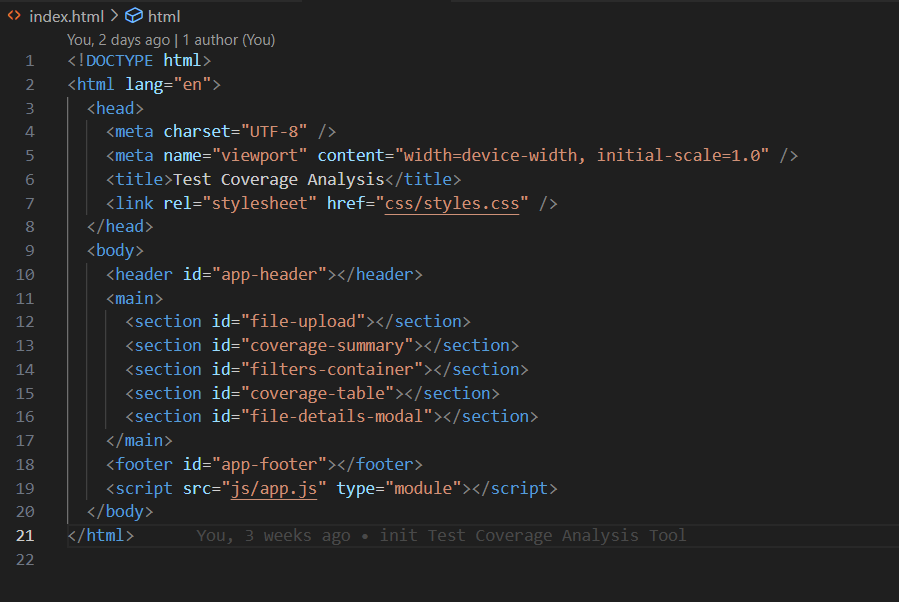


Рисунок 3.1 – Создание структуры сайта на с использованием HTML.

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.

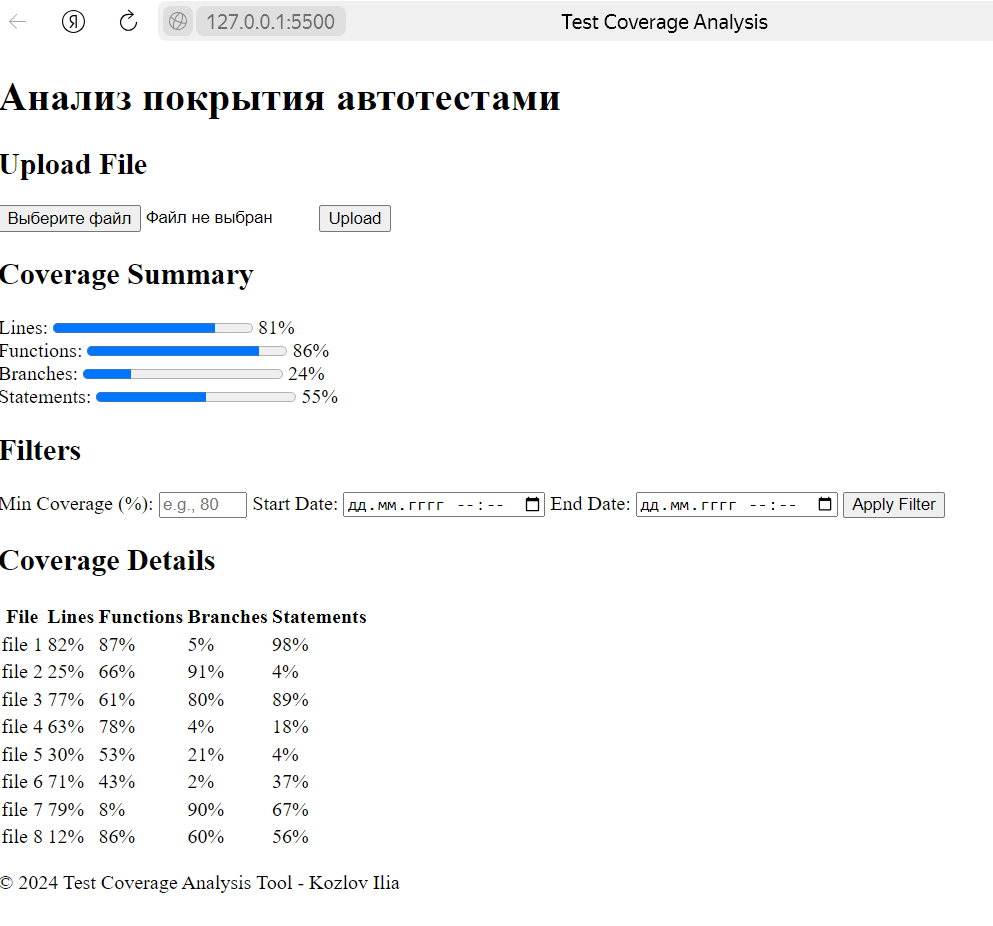


Рисунок 3.3 – Веб-интерфейс, построенный с помощью HTML.

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.

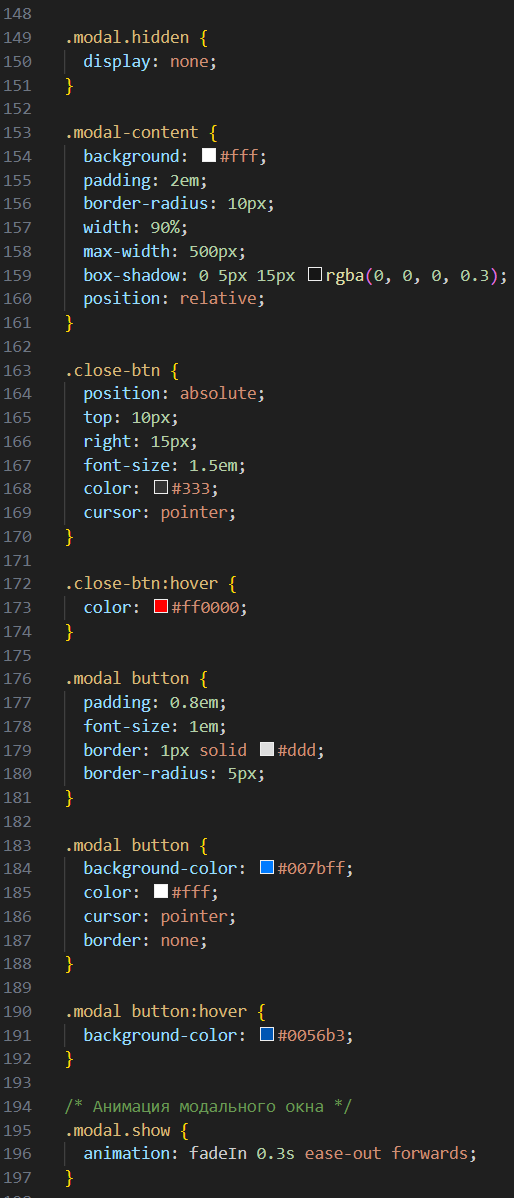


Рисунок 3.3 – Пример стилизации элементов интерфейса.

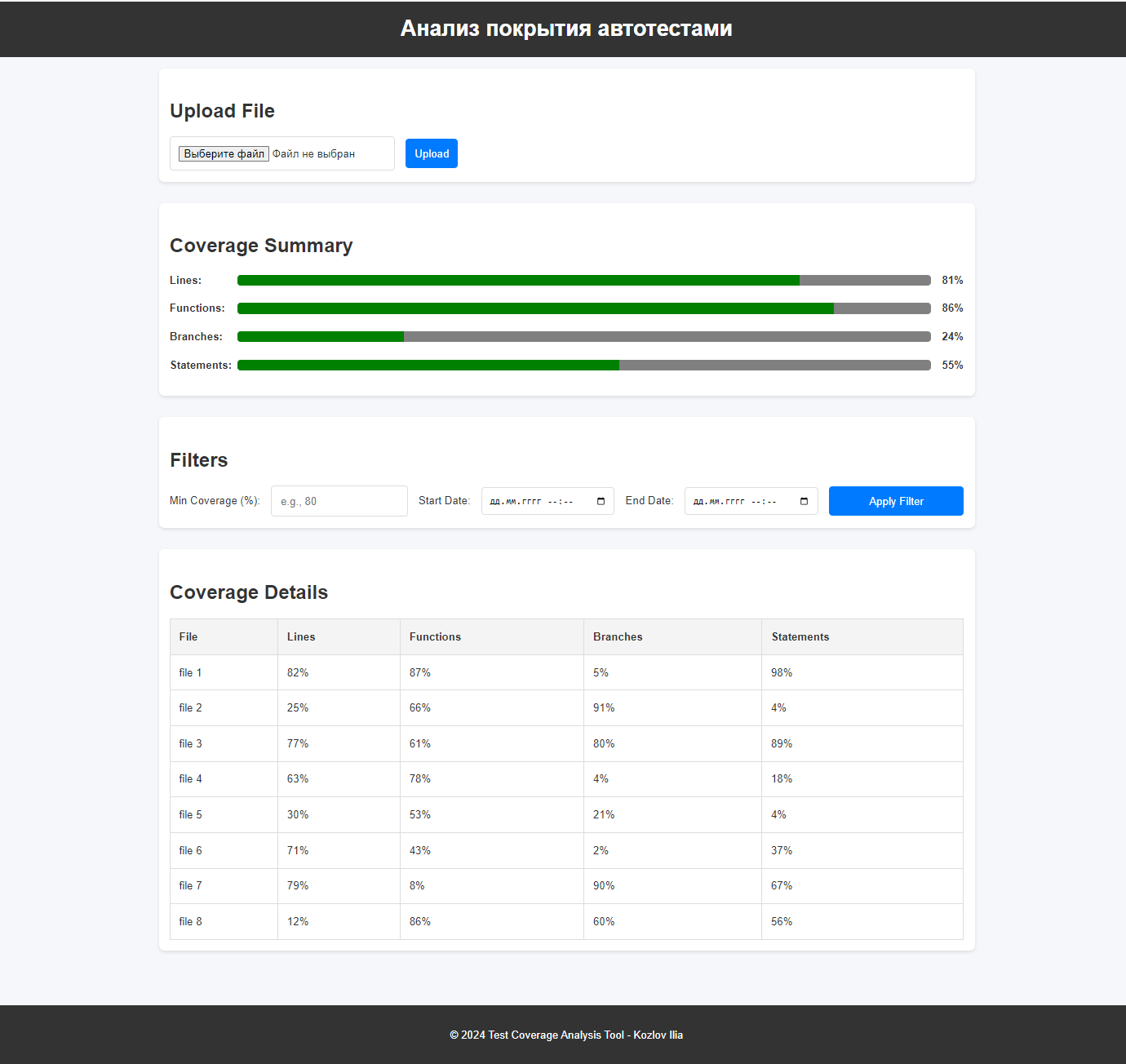


Рисунок 3.4 – Сайт с добавленной стилизацией.

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

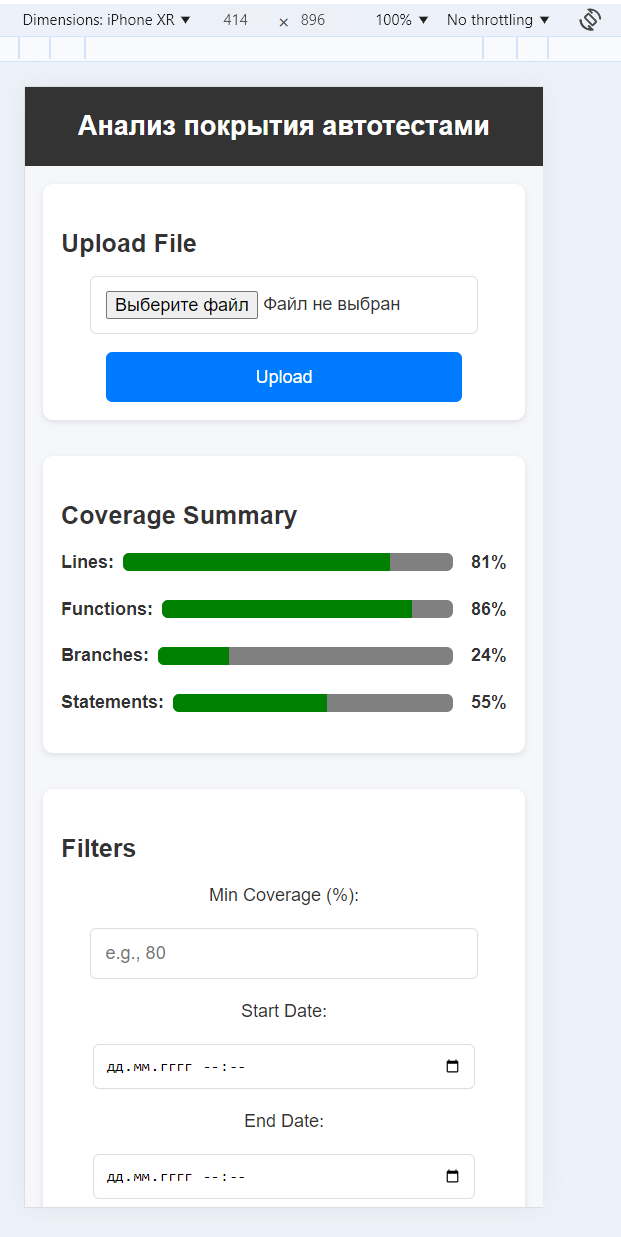


Рисунок 3.5 – Адаптивная верстка.

**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). Данная архитектура позволяет легко добавлять новые функции 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, представленная на рисунке 4.1

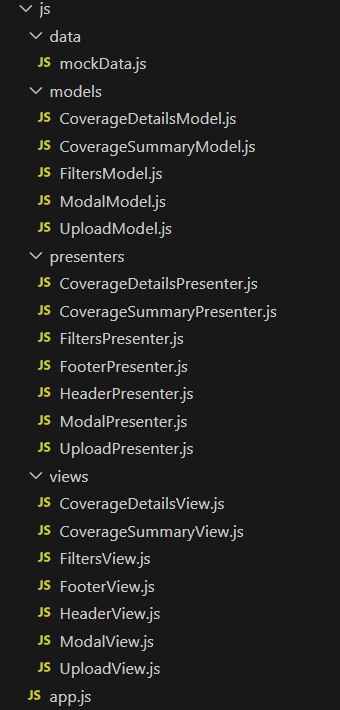


Рисунок 4.1 – Структура директорий и файлов.

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, 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, 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;

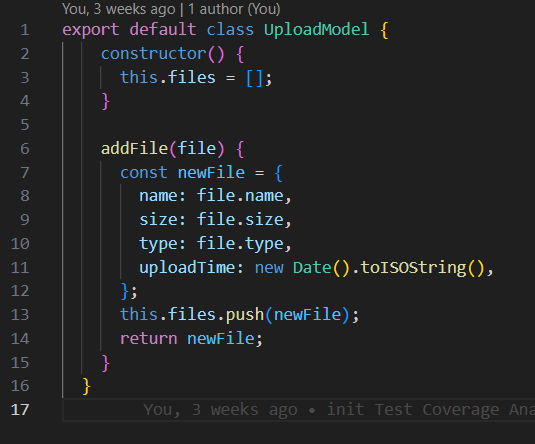


Рисунок 4.2 – Код класса UploadModel.

- в классе UploadView (рис.4.3) содержится функция render(), 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;

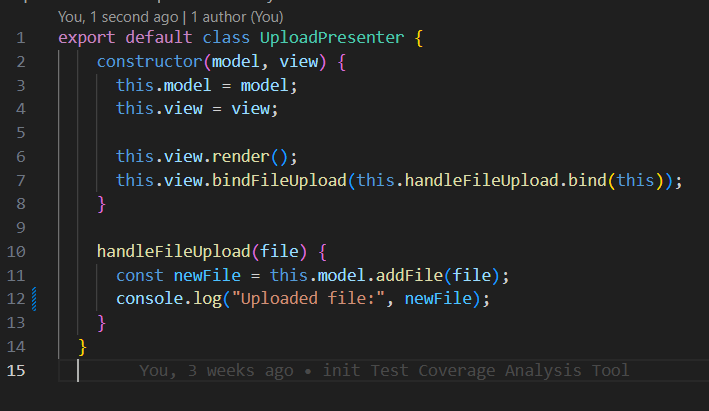


Рисунок 4.4 – Код класса UploadPresenter.

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

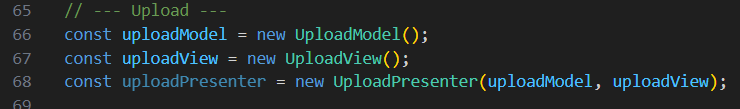


Рисунок 4.5 – Инициализация классов.

- 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.

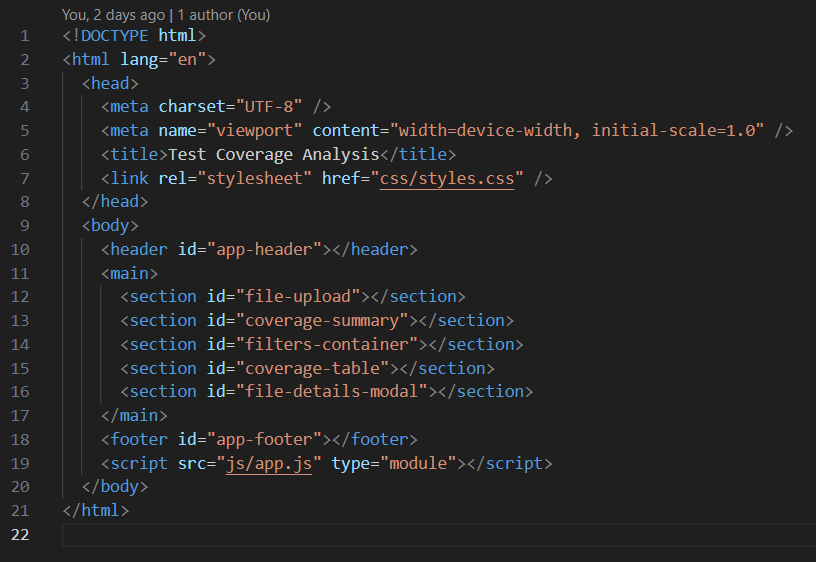


Рисунок 4.6 – Оптимизированный код файла index.html.

**5. Связь функционала с серверной частью**

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

1. Создание и использование эндпоинтов API. 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.

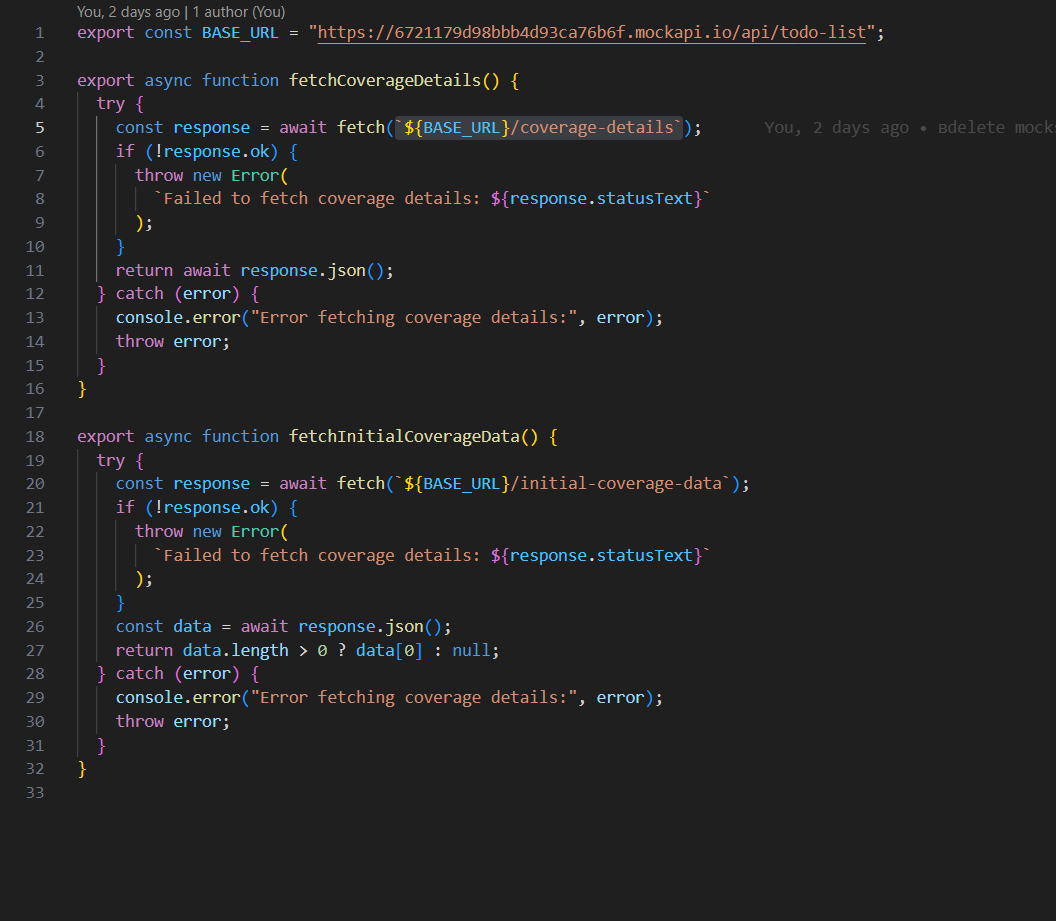


Рисунок 5.1 – Состав файла ApiService.js

4. Интеграция с функционалом. При загрузке файла из интерфейса он обрабатывается презентером и сохраняется на сервер 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.

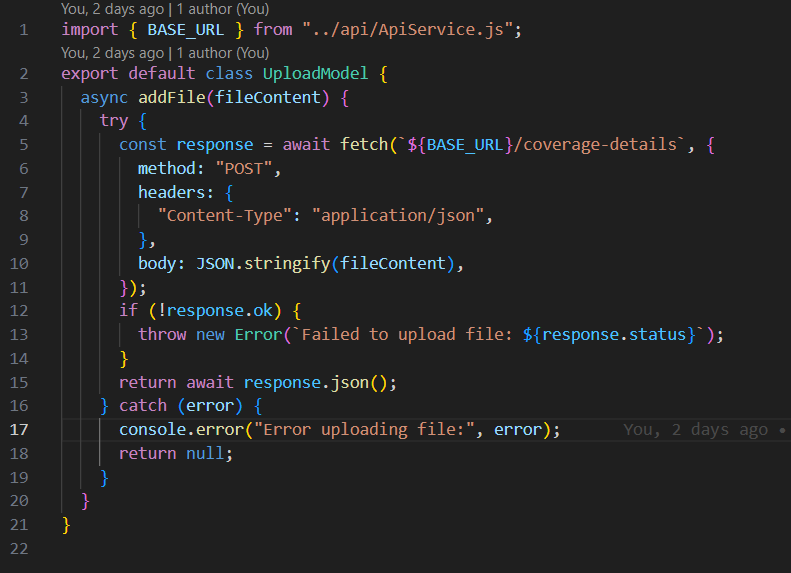


Рисунок 5.5 – Обработка загружаемого файла.

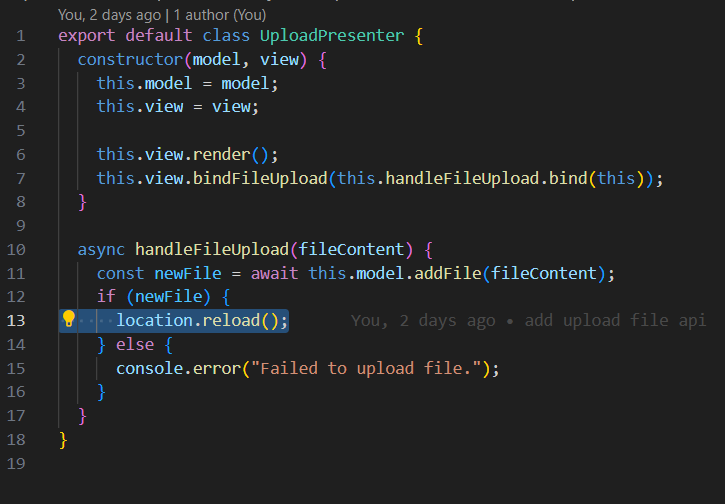


Рисунок 5.6 – Перезагрузка страницы после отправки данных на сервер.

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.

Благодаря реализации динамического 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.

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

ПРИЛОЖЕНИЕ А