Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет информационных технологий и управления

Кафедра информационных технологий автоматизированных систем

Отчёт по лабораторной работе №8
Вариант 5

по дисциплине
«Интернет-технологии и веб-программирование»

ТЕСТИРОВАНИЕ ФРОНТЕНД-ПРИЛОЖЕНИЯ

Выполнили: студ. гр 320604
Колодич А.Д.

Проверила: Хаджинова Н.В.

Минск 2025

ОГЛАВЛЕНИЕ

1 ЦЕЛЬ РАБОТЫ

Изучение методов тестирования и отладки *React*-приложений на примере каталога фильмов. Приобретение навыков написания модульных и компонентных тестов с использованием *Jest* и *Testing Library*, анализа покрытия кода, а также применения встроенных инструментов разработчика для диагностики и оптимизации веб-приложений.

2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Тестирование программного обеспечения проверяет соответствие кода ожидаемому поведению и позволяет автоматически выявлять ошибки при изменениях. В *React*-проектах применяют модульные тесты для чистых функций и компонентные тесты для проверки рендеринга и реакции на действия пользователя. Основным инструментом тестирования выступает фреймворк *Jest*, который предоставляет функции группировки тестов, отдельные проверки и утверждения. Для работы с *React*-компонентами используется библиотека *Testing Library*, позволяющая рендерить компоненты в виртуальном *DOM*, искать элементы по тексту, роли или атрибутам и симулировать события. Мок-функции *Jest* дают возможность проверять факт и параметры вызова колбэков.

Покрытие кода показывает, какая доля инструкций, ветвлений, функций и строк была выполнена во время тестов. Высокий процент покрытия снижает риск скрытых дефектов, но не заменяет проверку логической корректности. Встроенные инструменты разработчика браузера включают консоль для логирования, вкладку *Sources* для пошаговой отладки с точками останова, *Network* для анализа загрузки ресурсов, *Performance* для профилирования производительности, а также расширение *React Developer Tools* для просмотра дерева компонентов и их пропсов. *Lighthouse* оценивает качество страницы по метрикам скорости загрузки и стабильности интерфейса.

3 ХОД РАБОТЫ

В начале работы было настроено тестовое окружение проекта «Каталог фильмов». Через терминал установлены пакеты *jest*, *jest-environment-jsdom*, *@testing-library/react*, *@testing-library/jest-dom*, *@testing-library/user-event*, *babel-jest*, *@babel/preset-react*, *@babel/preset-env* и *identity-obj-proxy*. Созданы конфигурационные файлы *jest.config.js* и *babel.config.json*, а также заглушки для импорта стилей и изображений. Проверка правильности установки выполнена запуском тестовой команды, все утилиты отработали без ошибок.

Далее разработаны модульные тесты для функций утилит в файле *movieUtils.test.js* и компонентные тесты для *RatingStars*, *MovieCard*, *GenreFilter*, *Button* и *SearchBox* с использованием *Testing Library*. Модульные тесты проверили форматирование длительности, вычисление рейтинга и фильтрацию по жанру. Компонентные тесты подтвердили корректность рендеринга, передачу пропсов и реакцию на события: отображение названий, жанров, звёзд рейтинга, вызов колбэков при кликах.

После прогона всех тестов сформирован отчёт о покрытии кода. Достигнутые показатели: инструкции — 86.53%, ветвления — 72.72%, функции — 88.23%, строки — 90.24%, что превышает пороговые значения. Непокрытыми остались редко используемые ветки (значения пропсов по умолчанию, условие отсутствия *onDetailsClick*). Таблица покрытия по файлам представлена на рисунке 1.

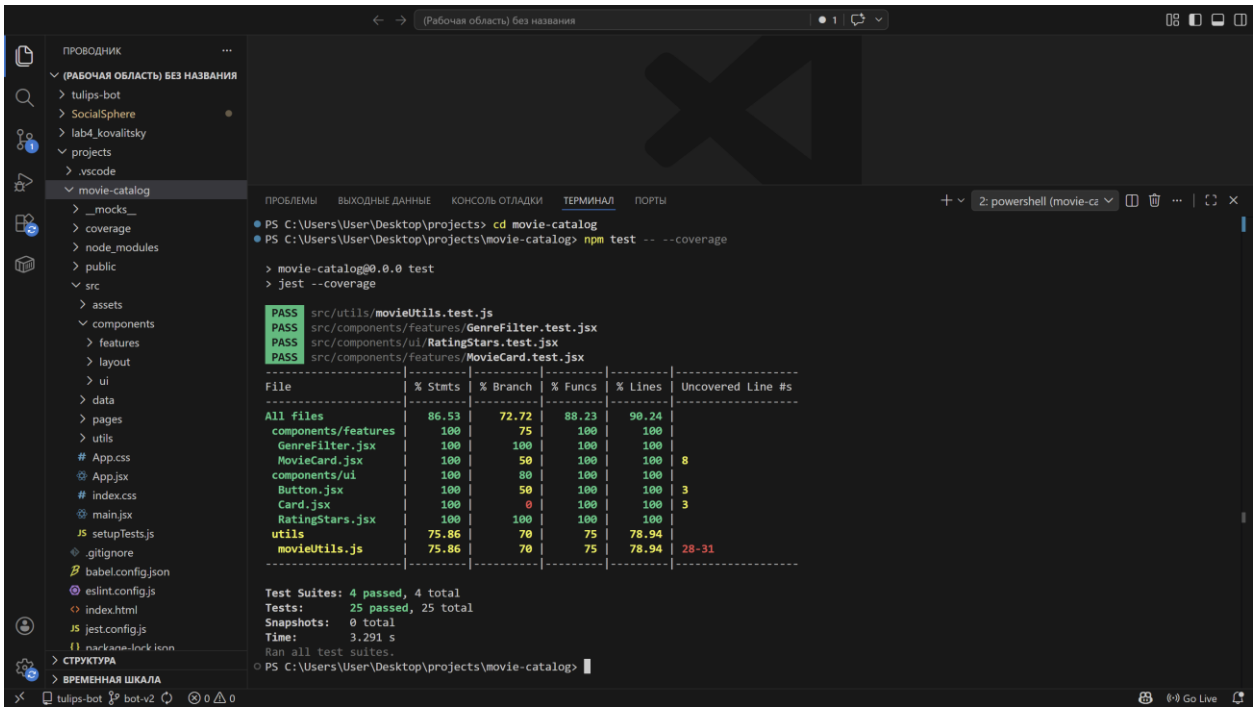

Рисунок 1 – Покрытие кода

Параллельно применялись инструменты разработчика *Chrome*. В компонент *App* добавлены отладочные инструкции *console.group*, *console.log* и

console.table для наблюдения за переменными состояния и содержимым отфильтрованного массива. В обработчик клика по фильму помещена команда *debugger*, позволившая приостановить выполнение во вкладке *Sources*, изучить локальные переменные и пройти код по шагам. Вкладка *Network* использована для анализа загрузки постеров: все изображения получены со статусом 200, определены их размеры и время загрузки. Во вкладке *Performance* записан сценарий поиска фильма — на записи видно, что *React* обновляет только изменившиеся карточки, не перерисовывая всю страницу. Расширение *React Developer Tools* отобразило дерево компонентов и пропсы. Аудит *Lighthouse* показал высокие оценки: *FCP* около 0.5 с, *LCP* около 1.5 с, *CLS* равен 0, *TTI* около 1 с. Результаты аудита и дерево компонентов отражены на рисунке 2, а пример отладки в *Sources* — на рисунке 3.

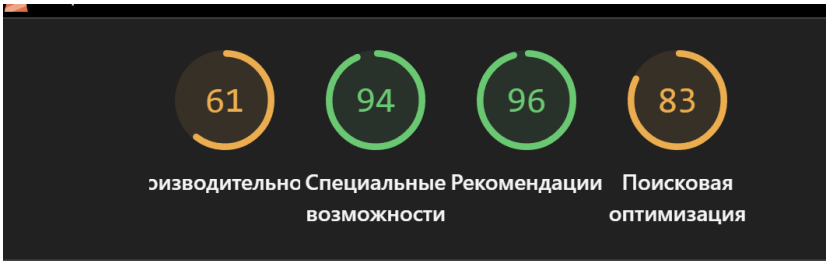

Рисунок 2 – Результаты аудита *Lighthouse* и инспектирование компонентов в *React Developer Tools*

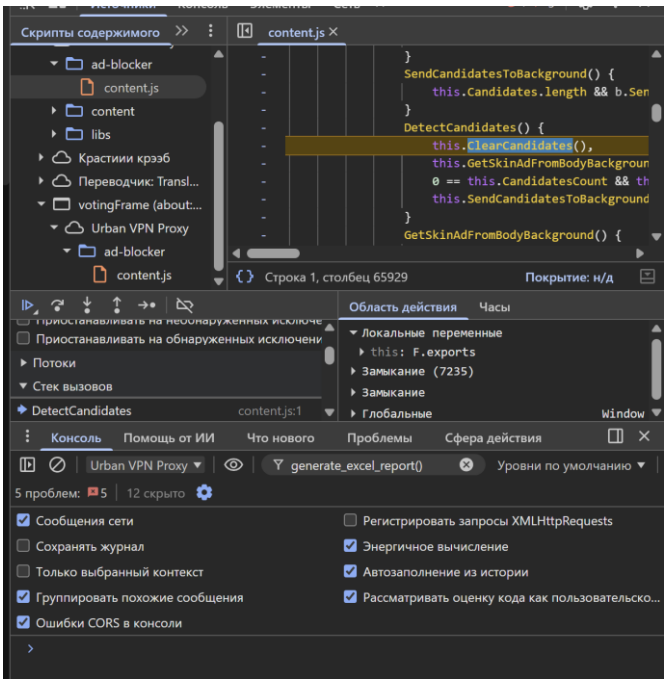

Рисунок 3 – Пошаговая отладка в инструментах разработчика с точкой останова

ВЫВОД

В ходе лабораторной работы освоены методы модульного и компонентного тестирования *React*-приложений. Настроено окружение на базе *Jest* и *Testing Library*, написаны и выполнены тесты для функций-утилит и ключевых компонентов каталога фильмов. Проанализирован отчёт о покрытии кода, определены непокрытые участки, не влияющие на работоспособность. Практически применены инструменты разработчика: консоль, пошаговая отладка, сетевой мониторинг, профилирование производительности и аудит *Lighthouse*. Полученные навыки подтверждают важность автоматического тестирования и владения средствами диагностики для создания стабильных и производительных веб-приложений.