Санкт-Петербургский Национальный Исследовательский Университет Информационных Технологий, Механики и Оптики

Факультет Инфокоммуникационных Технологий

Лабораторная работа №2

по дисциплине «Тестирование программного обеспечения»

Выполнила:

Соколова Дарья Максимовна

Группа: K3323

Проверил:

Кочубеев Николай Сергеевич

Санкт-Петербург,

2024

Цель работы

Научиться разрабатывать и применять интеграционные тесты для проверки взаимодействия между компонентами в существующем проекте, провести анализ взаимодействий и подготовить отчёт о проведённом тестировании.

Задачи

1. Выбрать репозиторий с GitHub;
2. Проанализировать взаимодействия между модулями;
3. Написать интеграционные тесты;
4. Составить отчет по выполненной работе.

Ход работы

1. Выбор репозитория с GitHub

Для выполнения лабораторной работы был выбран проект https://github.com/harshit0075/movie-search. Этот проект написан на React, представляет собой поисковик фильмов, который взаимодействует с внешним API OMDB (Open Movie Database) для получения информации о фильмах.

2. Анализ взаимодействий между модулями

К основным компонентам проекта относятся:

- Клиент для работы с API (Основной модуль, который отправляет запросы к OMDB API и обрабатывает полученные данные);
- Интерфейс пользователя (Компонент, который позволяет пользователям вводить запросы и отображает результаты поиска);
- Механизм обработки ошибок (Логика, обеспечивающая корректную реакцию на ошибки и исключительные ситуации при взаимодействии с API).

Ключевые точки интеграции:

- Отправление запросов к внешнему API (API OMDB) для получения информации о фильмах на основе введенного пользователем названия;
- Обработка полученных данных и формата ошибок при неверных запросах.

К **критическим частям** системы относятся:

- Правильная обработка ответов от API;
- Корректное формирование запросов с необходимыми параметрами.

Отметим **важные сценарии взаимодействия**:

- Выполнение GET-запросов к OMDB API для получения информации о фильмах, проверка как успешных, так и неуспешных запросов;
- Сценарии, связанные с обработкой ошибок, такие как вызов API с неправильными параметрами или отсутствием обязательных параметров.

3. Написание тестов

Тесты были написаны с использованием библиотеки Jest. Ниже представлены созданные тесты:

1. Проверка на успешность запроса к API. Тест отправляет GET-запрос к валидному эндпоинту и проверяет, что статус ответа равен 200, что указывает на успешное выполнение запроса (рисунок 1).

Рисунок 1 - Проверка на успешность запроса к API

2. Проверка на корректность структуры данных, получаемых от API. Тест проверяет, что ответ содержит свойство Search, что подтверждает наличие ожидаемой структуры данных при успешном запросе (рисунок 2).

```
test('should return correct data structure for valid request', async () => {
  const response = await axios.get(`${BASE_URL}&s=Harry`);
  expect(response.data).toHaveProperty('Search');
});
```

Рисунок 2 - Проверка на корректность структуры данных

3. Проверка на корректность обработки ошибок для недоступного эндпоинта. Тест отправляет запрос к несуществующему эндпоинту и проверяет, что статус ответа равен 404, что указывает на ошибку (рисунок 3).

```
test('should return 404 for invalid endpoint', async () => {
  try {
    await axios.get(`${BASE_URL}&=invalid`);
  } catch (error) {
    expect(error.response.status).toBe(404);
  }
});
```

Рисунок 3 - Запрос с недоступным эндпоинтом

4. Проверка на обработку ошибок при отсутствии обязательных параметров. Тест отправляет запрос без обязательного параметра s и проверяет, что ответ содержит свойство Response со значением False, а также наличие свойства Error (рисунок 4).

```
test('should return error message for missing required parameters', async () => {
  try {
    await axios.get(`${BASE_URL}`);
  } catch (error) {
    expect(error.response.status).toBe(200);
    expect(error.response.data).toHaveProperty('Response', 'False');
    expect(error.response.data).toHaveProperty('Error');
  }
});
```

Рисунок 4 – Запрос с отсутствием обязательных параметров

5. Проверка на корректность обработки ошибок при передаче несуществующего названия фильма. Тест отправляет запрос с невалидным названием и проверяет, что ответ содержит свойство Response со значением False, свойство Error с сообщением "Movie not found!", и что статус ответа равен 200 (рисунок 5).

```
test('should return error message for invalid search term', async () => {
  const response = await axios.get(`${BASE_URL}&s=invalidName`);
  expect(response.status).toBe(200);
  expect(response.data).toHaveProperty('Response', "False");
  expect(response.data).toHaveProperty('Error', 'Movie not found!');
});
```

Рисунок 5 – Запрос с несуществующим названием фильма

Видим, что тесты выполнились успешно (рисунок 6), что подтверждает правильность обработки запросов и ответов от OMDB API. К потенциальным проблемам можно отнести то, что так как приложение зависит от OMDB API, любые изменения в его структуре или доступности могут привести к сбоям, поэтому стоит реализовать дополнительные механизмы обработки ошибок.

```
 PASS  tests/api.test.js
  API Integration Tests
    ✓ should return 200 for valid endpoint (234 ms)
    ✓ should return correct data structure for valid request (73 ms)
    ✓ should return 404 for invalid endpoint (60 ms)
    ✓ should return error message for missing required parameters (74 ms)
    ✓ should return error message for invalid search term (68 ms)

Test Suites: 1 passed, 1 total
Tests:       5 passed, 5 total
Snapshots:   0 total
Time:        0.733 s, estimated 2 s
Ran all test suites.
```

Рисунок 6 – Запуск тестов

Вывод

В данной лабораторной работе был выбран репозиторий с GitHub, проанализированы взаимодействия между модулями, написаны интеграционные тесты, а также составлен отчет по выполненной работе.

Ссылка на репозиторий с тестами:
https://github.com/daryasokolova04/testing_PO