Министерство образования и науки
федеральное государственное автономное образовательное учреждение
высшего образования
«Национальный исследовательский университет ИТМО»

Факультет инфокоммуникационных технологий

Отчет по дисциплине: **«Тестирование программного обеспечения»**

Лабораторная работа 2

Выполнила:

Королева А.С.

Группа:

K3322

Проверил:

Кочубеев Н.С.

Санкт-Петербург,

2024

Цель: нужно научиться разрабатывать и применять интеграционные тесты для проверки взаимодействия между компонентами в существующем проекте.

Задачи:

- Выбрать репозиторий для тестирования с GitHub;
- Провести анализ взаимодействий;
- Написать интеграционные тесты;
- Подготовить отчет о проделанной работе.

Ход работы

Ссылка на репозиторий: https://github.com/nastyakrlv/Testing_PO

Выбор репозитория для тестирования с GitHub

Проект в рамках лабораторной работы использует алгоритмы для обработки лабиринтов. Это отличный пример для тестирования, так как он включает различные сценарии и условия.

Краткое описание проекта:

Реализована задача по анализу 2D массивов, представляющих собой лабиринты, где 1 обозначает проход, а 0 — стену. Программа считывает данные из файла tasks/labyrinths/text.txt, анализирует лабиринты и определяет их корректность по заданным критериям.

Проект написан на JavaScript, тестирование произведено на Jest.

Анализ взаимодействий между модулями

1. Основной поток управления (main.js)

- Ключевая роль: модуль связывает все остальные части системы.
- Интеграции:

  С read.js: считывание данных из файла (getDataFromFile()).

  С scan.js: анализ данных лабиринтов через функцию scan().

  С send.js: отправка результатов работы на сервер (sendDataToServer()).

2. Модуль read.js

- Ключевая роль: обеспечивает считывание данных из текстового файла и их преобразование в формат JSON.
- Интеграции:
  Используется модулем main.js для получения входных данных.

3. Модуль scan.js

- Ключевая роль: выполняет анализ лабиринтов, определяя их свойства (например, количество изолированных областей или проходов).
- Интеграции:
  Используется в main.js для обработки каждого лабиринта.
  Взаимодействует с модулем queue.js для реализации алгоритма BFS.

4. Модуль queue.js

- Ключевая роль: предоставляет структуру данных для реализации BFS (очередь).
- Интеграции:
  Используется модулем scan.js для управления обработкой точек лабиринта.

5. Модуль send.js

- Ключевая роль: отправляет данные на сервер через HTTP-запрос.
- Интеграции:
  Используется main.js для отправки данных после анализа.

6. Модуль start.js

- Ключевая роль: точка входа в приложение, запускающая весь процесс.
- Интеграции:

Поднимает мок-сервер с использованием библиотеки msw для тестирования отправки данных.

Запускает функцию main() и выводит результат.

Важные сценарии взаимодействия

Чтение и преобразование данных:

- main.js → вызывает read.js для получения массива лабиринтов.

Анализ данных:

- main.js → передаёт каждый лабиринт в scan.js, где выполняется BFS с использованием очереди из queue.js.

Отправка результатов:

- main.js → вызывает send.js, отправляя итоговый результат (успешный или неуспешный анализ).

Тестирование взаимодействия:

- start.js поднимает мок-сервер и проверяет корректность работы с API.

Критические части системы

- Считывание данных (read.js):
  Неправильный формат данных может привести к сбою.
- Алгоритм BFS (scan.js):
  Ошибки в логике обхода лабиринта могут исказить результаты анализа.
- HTTP-запросы (send.js):
  Ошибки в конфигурации HTTP-запроса или его параметрах могут повлиять на корректную передачу информации о результате проверки лабиринта на сервер.

Написание интеграционных тестов

Тест (рисунок 1) проверяет, что в случае, если файл, с которого происходит чтение, оказывается пустым, функция getDataFromFile корректно возвращает пустой массив. Это важно для надежности системы, чтобы убедиться, что программа не выйдет с ошибкой и не будет обрабатывать пустой файл неправильно

```
25
26    test(`Если прочитали пустой файл, то функция возвращает пустой массив`, () => {
27      fs.readFileSync.mockReturnValue("");
28      const expectedResult = [];
29
30      getDataFromFile.mockReturnValue(expectedResult);
31
32      const result = getDataFromFile();
33
34      expect(result).toEqual(expectedResult);
35    });
36
```

Рисунок 1 - тест 1

Тест (рисунок 2) проверяет, что функция scan корректно анализирует лабиринт и возвращает правильные данные, когда в лабиринте есть по одному пути для каждого типа. Это важно для проверки работы алгоритма сканирования, чтобы убедиться, что он правильно идентифицирует различные типы путей и возвращает корректный результат в формате объекта.

```
test(`
  Если вызываем метод сканирования лабиринта и передаем двумерный массив, где есть ровно один путь каждого типа,
  то вернется объект, где все поля будут равны 1
`, () => {
  const labyrinth = [
    [1, 0, 0, 0, 0, 0, 0, 1],
    [1, 1, 0, 0, 0, 0, 0, 1],
    [1, 1, 0, 0, 0, 0, 0, 1],
    [0, 0, 0, 0, 0, 0, 0, 1],
    [1, 0, 1, 1, 0, 0, 0, 1],
    [1, 0, 0, 0, 0, 0, 0, 1]
  ];
  const expectedResult = { ceil: 1, floor: 1, both: 1, isolated: 1 };

  const result = scan(labyrinth);

  expect(result).toStrictEqual(expectedResult);
});
```

Рисунок 2 - тест 2

Тест (рисунок 3) служит для проверки корректности логики, которая анализирует лабиринт. В данном случае основное внимание уделяется тому, как

лабиринт анализируется на основе путей, соприкасающихся с полом и потолком, и правильно ли функция решает, что лабиринт правильный, если таких путей больше, чем других.

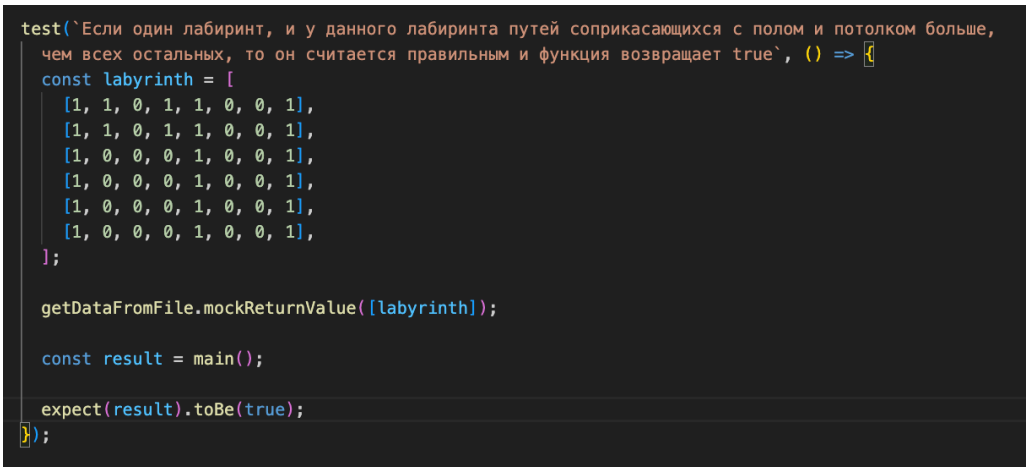

Рисунок 3 - тест 3

Тест (рисунок 4) предназначен для проверки работы функции, которая должна принимать массив лабиринтов, анализировать их и возвращать результат в зависимости от того, больше ли неправильных лабиринтов, чем правильных. В данном тесте, поскольку неправильных лабиринтов два, а правильных один, ожидается, что функция вернет false.

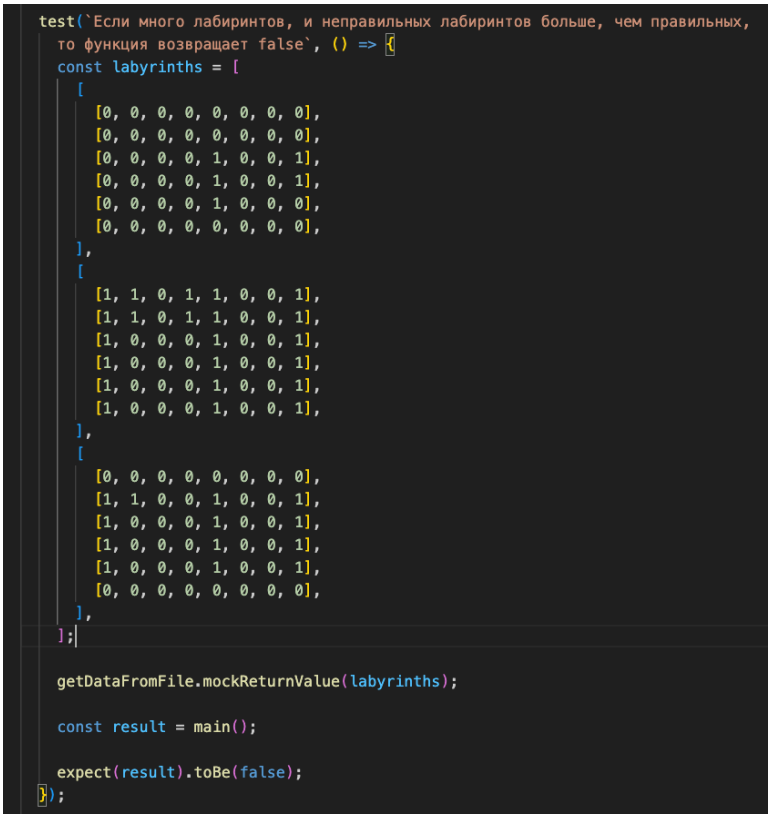

Рисунок 4 - тест 4

Тест (рисунок 5) предназначен для проверки того, что функция main() правильно обрабатывает правильный лабиринт и отправляет соответствующие данные на сервер в случае успеха. В данном случае, если лабиринт правильный, функция должна передать { passed: true } как данные для отправки на сервер.

```
test(`Если правильный лабиринт, то происходит запрос на сервер с передачей информации
  о результате функции - { passed: true }`, () => {
  const labyrinth = [
    [
      [1, 1, 0, 1, 1, 0, 0, 1],
      [1, 1, 0, 1, 1, 0, 0, 1],
      [1, 0, 0, 0, 1, 0, 0, 1],
      [1, 0, 0, 0, 1, 0, 0, 1],
      [1, 0, 0, 0, 1, 0, 0, 1],
      [1, 0, 0, 0, 1, 0, 0, 1],
    ],
  ];

  getDataFromFile.mockReturnValue(labyrinth);

  main();

  expect(sendDataToServer).toHaveBeenCalledWith({ passed: true });
});
```

Рисунок 5 - тест 5

Тест проверяет важный аспект логики программы — правильность обработки неправильных лабиринтов. Если лабиринт не соответствует ожидаемому формату или не является валидным, программа не должна отправлять данные на сервер, и тест подтверждает, что в таких случаях запрос не выполняется.

```
test(`Если неправильный лабиринт, то запрос на сервер с передачей информации
  не происходит `, () => {
  const labyrinth = [
    [
      [0, 0, 0, 0, 0, 0, 0, 0],
      [0, 0, 0, 0, 0, 0, 0, 0],
      [0, 0, 0, 0, 1, 0, 0, 1],
      [0, 0, 0, 0, 1, 0, 0, 1],
      [0, 0, 0, 0, 1, 0, 0, 0],
      [0, 0, 0, 0, 0, 0, 0, 0],
    ],
  ];

  getDataFromFile.mockReturnValue(labyrinth);

  main();

  expect(sendDataToServer).toHaveBeenCalledTimes(0);
});
```

Рисунок 6 - тест 6

Результаты тестирования

В ходе тестирования были проверены различные аспекты функциональности программы, включая обработку правильных и неправильных лабиринтов, взаимодействие с сервером и корректность выполнения логики программы.

Корректность логики: Тесты показали, что логика программы (определение правильных и неправильных лабиринтов) корректно работает, и в зависимости от результатов происходит либо отправка данных на сервер, либо их отсутствие. Это подтверждает, что модуль, отвечающий за анализ лабиринтов, работает надежно и правильно.

Взаимодействие с сервером: Взаимодействие с сервером происходит корректно только в случае правильных лабиринтов. Когда лабиринт неправильный, запрос на сервер не отправляется, что является ожидаемым и правильным поведением.

Потенциальные проблемы и узкие места: Если сервис, на который отправляются данные, недоступен (например, из-за сетевой ошибки или серверного сбоя) или исходные данные содержат ошибку в синтаксисе, программа может завершиться с ошибкой.

Вывод: был проведен анализ взаимодействий в проекте, написаны интеграционные тесты, на основании этого подготовлен отчет о проделанной работе.