EzParking Myadmin

Code Testing Plan and Result

SSE Capstone project – University of Regina

Team Apollo

Winter 2023

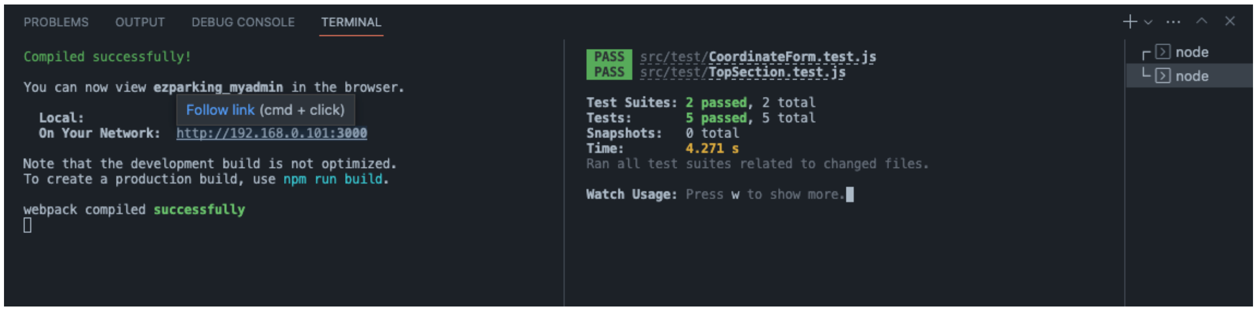
Yilin Ren

Executive Summary

The “EzParking MyAdmin” React application is designed to simplify the process of managing parking lot waypoint graph in variety facilities including education institution, shopping mall and large parking lot network. This test plan carries out the steps required to ensure the application meets its functional requirement. The test plan also talks about the testing methodology that were used. The report demonstrated that the application functions as intended and meets our initial expectation.

Our code testing focused on testing each component of the application by using “React Test Library “to perform unit test automatically, and make sure the component is render correctly. I also perform integration testing by one of our group members to implement tasks to trigger some edge case and check if the application catch exceptions and perform corrections.

Our test script including 5 unit test and 19 integration tests. The test ran successfully, and the application worked as expected. An example of the unit test result is posted below (Fig 1)



```
Compiled successfully!

You can now view ezparking_myadmin in the browser.

  Local:
  On Your Network:  http://192.168.0.101:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

```
 PASS  src/test/CoordinateForm.test.js
 PASS  src/test/TopSection.test.js

Test Suites: 2 passed, 2 total
Tests:       5 passed, 5 total
Snapshots:   0 total
Time:        4.271 s
Ran all test suites related to changed files.

Watch Usage: Press w to show more.
```

Fig 1. Final Test Result : All unit test case passed

Table of Contents

Introduction

Purpose:

This test plan includes the strategy, process, and results of our testing of the EzParking Myadmin program. This article will contain the following:

1. Testing Strategy: The purpose of testing the program and the testing methodology.

2. Test Plan: A detailed plan for testing the application, including the testing tool used, and the specific features to be tested.

3. Test Cases: A detailed list of test cases that will be executed during the testing phase, including inputs, expected outputs, and pass/fail criteria.

4. Test Results: The results of the testing, including pass/fail status for each test case, any issues identified.

5. Conclusion: A summary of the testing process and outlines any follow-up actions required.

Project:

EzParking Myadmin is a react application that acts as a visual solution for managing the database in the EzParking project. In this background planform, the administrator are able to tell how many parking spaces and total parking spaces are currently available in the parking lot network. At the same time, the administrator can add, delete and modify waypoints in the parking lot network through the graphical interface.

Testing Strategy:

The purpose of code testing is to find potential problems through code testing and to ensure that the program runs correctly. I will test this program through two aspects, first is the unit test, the unit test will ensure that each component is working properly, and then I will conduct the integration test, the integration test will be a little complicated. This is because I want to make sure that the coordination between all units is correct.

EzParking Myadmin consists of three main components,

1. User operation panel (coordinator form component)
2. Information notification panel (Top section component)
3. Visual map panel (Map view component).

I will unit test the first two panels, and for the final visual map panel, since it is too complex, I will perform integration tests.

In order to implement unit testing, I will use the React-test-library for unit testing, and I will customize the test program for each react component. Before testing, each test program will have an expected target. After the test program is run, the expected target will be compared with the program running results. Any incorrect results will cause the failure of the entire test.

For integration test. I will design some test scenarios/tasks, and the testers will manually complete these tasks. During the process of completing these tasks, the testers need to record whether the program complies with the pre-designed operation process. If not, the test will fail.

Test Plan:

1. Coordinator form component:

I will use the react-test-library for testing. Through this library, I can simulate rendering a react component. When the component is rendered successfully, I will test whether the DOM argument of this component is consistent with the expected variable. After this component is generated at the beginning, users are prohibited from clicking to upload the waypoint map. Because the waypoint map does not change at this time, repeated uploads will only increase the server load. When the user modifies the waypoint graph, the user is allowed to upload the waypoint graph. I will test that this component works as expected.

2. Top Section components:

I will also use the react-test-library for testing. When this component is generated at the beginning, it will display the information of the current parking lot according to the data provided by the server. I will test that this component displays this information correctly.

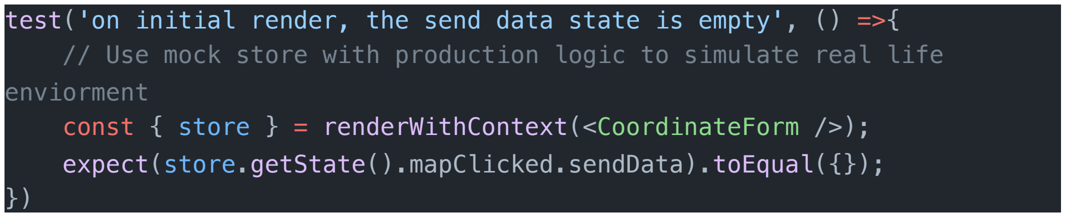
3. Map View component:

I will invite testers to conduct integration test. When this component rendered, it will load all the waypoints already in the database, after that, it will allow the tester to add, delete or modify the waypoint map. I will design a test flow and each condition that required to meet in order to pass the test. Testers will perform these tests and validate whether the results meet the test criteria.

Test Cases:

1. Coordinator form component:

1.1 Initial state: User didn't perform any map edit, the send data object is empty.



```
test('on initial render, the send data state is empty', () =>{
    // Use mock store with production logic to simulate real life
enviorment
    const { store } = renderWithContext(<CoordinateForm />);
    expect(store.getState().mapClicked.sendData).toEqual({});
})
```

1.2 At initial state, user didn't made any change to the graph, the send data object is empty, which lead the send graph button is disabled.

```
test('on initial render, the send graph button is disabled', () =>{
    render(
        <Provider store={store}>
            <CoordinateForm />
        </Provider>
    );
    expect(screen.getByRole('button', { name: /Send
Graph/i})).toBeDisabled;
})
```

1.3 After user made change to the graph, the send data object will be re generated, and the send graph button is enabled.

```
test('After user add a waypoint, the send data state is updated, send
data button is ebabled', () =>{
    // Use mock store with production logic to simulate real life
enviorment
    const mockStore = configureStore()

    render(
        <Provider store={mockStore({mapClicked: {sendData:
'mockData'}})}>
            <CoordinateForm />
        </Provider>
    );
    expect(screen.getByRole('button', { name: /Send
Graph/i})).toBeEnabled;
})
```

2. Top Section components:

2.1 Renders a top section, check if the top section title is correct, I don't need to care what API will provide to the client, it will be care out on the integration test stage.

```
test("renders a top section, check if the title is correct", async () =>
{
    render(<Provider store={store}> <TopSection itemName={"In Use"}
itemValue={123}/> </Provider>);
    expect(screen.getByText("In Use"))
  });
```

2.2 Renders a top section, check if the top section information is correct.

```
test("renders a top section, check if the number is correct", async () => {
    render(<Provider store={store}> <TopSection itemName={"In Use"}
itemValue={123}/> </Provider>);
    expect(screen.getByText(123))
  });
```

3. Map view components:

3.1 The tester will perform the following test and validate the respond of the application

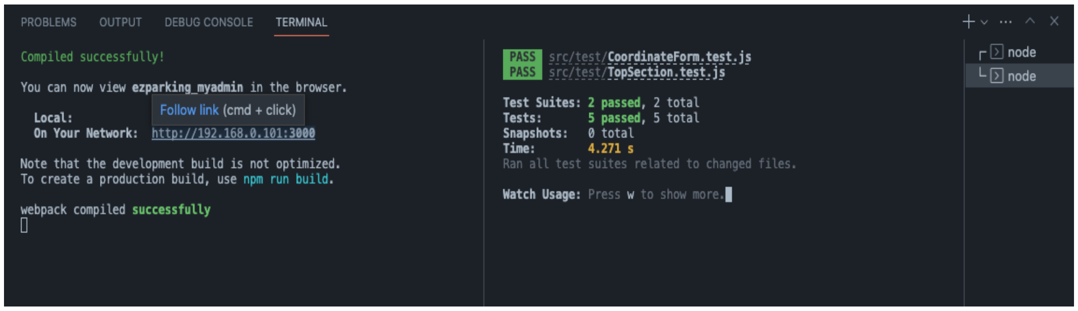
Test #	Required Task	Application Respond	Pass/Fail
1	Create a waypoint	Waypoint is created	
2	Create a parking lot	A parking lot waypoint is created and wait the user to assign a parking lot name and available spot	
3	Entry -1 as available spot	Catch an error, the available should not be negative, ask user re-entry	
4	Entry “test” as available spot	Catch an error, the available should be number only, ask user re-entry	
5	Entry 5 as available spot	Ask user continue entry parking lot name	
6	Entry “ ” as name	Catch an error, the name should not contain space, ask user re-entry	
7	Entry “ test” as name	Catch an error, the name should not start with space, ask user re-entry	
8	Entry “Lot 5” as name	A parking lot waypoint is created	
9	Create a destination waypoint	A destination waypoint is created and wait the user to assign a destination name	
10	Entry “ ” as name	Catch an error, the name should not contain space, ask user re-entry	
11	Entry “ test” as name	Catch an error, the name should not start with space, ask user re-entry	
12	Entry “Classroom” as name	A destination waypoint is created	

13	User click a waypoint	A info window is showing up with latitude and longitude	
14	User click a parking lot waypoint	A info window is showing up with latitude, longitude, parking lot name and available spot	
15	User click a destination waypoint	A info window is showing up with latitude, longitude and destination name	
16	User double click a waypoint	The waypoint start a jumping animation	
17	User double click another waypoint	Another waypoint start the jumping animation and a connection line is created between those two waypoints.	
18	User click a parking lot waypoint and select update the waypoint	A dialog window pops out and ask user to entry new available amount	
19	User click a parking lot waypoint and select delete the waypoint	The selected waypoint is deleted and all path associated with it is also deleted.	

Test Results:

1. Unit Test:

The application pass all unit test, this is means each signal unit is working correctly.



```
Compiled successfully!

You can now view ezparking_myadmin in the browser.

  Local:            Follow link (cmd + click)
  On Your Network:  http://192.168.0.101:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

```
PASS  src/test/CoordinateForm.test.js
PASS  src/test/TopSection.test.js

Test Suites: 2 passed, 2 total
Tests:       5 passed, 5 total
Snapshots:   0 total
Time:        4.271 s
Ran all test suites related to changed files.

Watch Usage: Press w to show more.
```

Fig 2. Final Test Result : All unit test case passed

2. Integration Test

The application also passes all integration test, Which is means all the components working correctly when it combined together.

Test #	Required Task	Application Respond	Pass/Fail
1	Create a waypoint	Waypoint is created	Pass
2	Create a parking lot	A parking lot waypoint is created and wait the user to assign a parking lot name and available spot	Pass
3	Entry -1 as available spot	Catch an error, the available should not be negative, ask user re-entry	Pass
4	Entry “test” as available spot	Catch an error, the available should be number only, ask user re-entry	Pass
5	Entry 5 as available spot	Ask user continue entry parking lot name	Pass
6	Entry “ ” as name	Catch an error, the name should not contain space, ask user re-entry	Pass
7	Entry “ test” as name	Catch an error, the name should not start with space, ask user re-entry	Pass
8	Entry “Lot 5” as name	A parking lot waypoint is created	Pass
9	Create a destination waypoint	A destination waypoint is created and wait the user to assign a destination name	Pass
10	Entry “ ” as name	Catch an error, the name should not contain space, ask user re-entry	Pass
11	Entry “ test” as name	Catch an error, the name should not start with space, ask user re-entry	Pass
12	Entry “Classroom” as name	A destination waypoint is created	Pass
13	User click a waypoint	A info window is showing up with latitude and longitude	Pass
14	User click a parking lot waypoint	A info window is showing up with latitude, longitude, parking lot name and available spot	Pass

15	User click a destination waypoint	A info window is showing up with latitude, longitude and destination name	Pass
16	User double click a waypoint	The waypoint start a jumping animation	Pass
17	User double click another waypoint	Another waypoint start the jumping animation and a connection line is created between those two waypoints.	Pass
18	User click a parking lot waypoint and select update the waypoint	A dialog window pops out and ask user to entry new available amount	Pass
19	User click a parking lot waypoint and select delete the waypoint	The selected waypoint is deleted and all path associated with it is also deleted.	Pass

Conclusion:

In conclusion, the EzParking MyAdmin React application has been thoroughly tested using React Test Library and Integration test methodology, which enabled effective and efficient testing, ensuring that the application is robust and reliable. The testing process has helped to identify and resolve any issues.

By analyse the testing result, The application meets the specified function requirements and provides a positive user experience.