Group 11 Test Reports

Unit Test Report

The purpose of unit tests is to test individual components of the app such as the web components, backend servers, data storage, etc. However, there was a major issue. Jest has problems interacting with pouchDB. By design, Jest cannot access pouchDB imports. Since our local database is dependent on pouchDB, it is impossible to use those two together. Many of our components involve pouchDB so we've decided to leave those to manual testing. Our unit tests consist only of components or part of elements that do not involve any database.

In the backend, we have three security functions:

passHash(String);

Purpose:

To hash a password (string) into a hashed value.

Requirement:

1. hashed value must be different than the original string.
2. The same string must produce the same hashed value, no matter how many times you hash it.

Set up:

- We chose the string "how are you God?"
- Test 1 - compared hashed value with original string: should be different
- Test 2 - hashed the string 2 different times and compare results: should be the same

Result: Passed on both.

Encrypt(string, string);

Purpose:

Take a string(salt) to turn the data(string) into a different form. The process is reversible; We must be able to get the data back after encrypting it.

Requirement:

Encrypted string must be different from the original string.

Set Up:

- Salt is "how are you God?"
- Data is "please help me"
- Test: Encrypt data and compare result with input: result must be different.

Result: Pass, result is different.

Decrypt (String, String)

Purpose:

Take the result from Encrypt(), encrypted string, find and return the original data(string).

Requirement:

Decrypted data must equal to original data

Set up:

- Salt is “how are you God?”
- Data is “please help me”
- Test: decrypt data and compare result with original data: result should be the same.

Result: Pass, results are the same.

Summary:

All tests ran and passed as expected. There is however, a Type error which we suspect has something to do with our jest configuration. The error is as follows:

TypeError: Cannot read property 'JSON' of null

We have not been able to resolve the issue though it does not actually interfere with our tests. It has been postponed as we have been working on solving more pressing matters.

Front End Testing Report

Purpose:

The purpose of this test was to individually test the web components of our application. In the front end we tested the functionalities of the web components using Jest unit testing. We weren't able to access fetch nor use it because we didn't have a server that we were pointing at that should be serving the components/block.html file. To consolidate this we had to work around the limitations of Jest puppeteer to test the functionalities of our web component using Jest unit testing. So, we tweaked some of the files to thoroughly test our web component. We pulled in the block.html straight into the block.js and use that instead of what would be pulled from the fetch and then continue on as though the fetch doesn't even exist. This approach solved the issue that we had testing the block.js file.

Requirements**:**

- Test the setup of the text block styling for different pages of our application.
- Test the note text function whether it properly adds bullet points.
- Test the event text functionalities and check whether the event date and time parsing work properly.
- Test the setup task function for the task test and add a task check off block.

Set up:

- First run beforeAll Jest built in function before tests can be run in our file. This beoreAll function should return a promise that jest would wait until it resolves its promise and starts running tests.
- Make sure our Text block constructor is initialized properly.
- Invoke the *expect* and to *beTrue* Jest builtin functions to test the header functions to see that our tests return the same result as described by the test functions.
- Get the attribute of the text block and make sure that the placeholder attribute is equal to the events.
- Internally install npm to test its output.
- Check whether the equality of the functions in our file and the tests return the same result.
- Report the result

Result:

- All of the web component functions returned the correct result.

Summary:

- Overall, we were successful in overcoming the limitations of Jest and testing our web components.

Manual Testing Report

The purpose of this section is to cover components we couldn't do with unitests. It is most involved with CRUD operations with the database. For all of our users their name/email is going to be Test## where ## is a number such as 01, 13, etc. Their password was by convection "qwert" except for a few variants for testing purposes. We also used "Postman API" to check if the backend servers are working properly.

Test1

We created a new user named Test1 and stored it on the database. We checked for its existence and tried to receive it. The first time with an incorrect password. It failed as expected. The second time we used the correct password and the userObject was returned properly.

Test2

We created user Test2 and repeated the same procedures in Test1. In addition, we also updated new elements in Test2, namely a future log and a collection. Upon testing we found that the new elements were not stored. The issue was with the local and remote databases not communicating properly.

Test3-7

Tests 3 to 9 used the same procedures as Test2 but with various elements. We add/remove/modified Futurelogs, Monthly, Tasks, Themes and other elements. All elements were stored/read properly.

Test8

We implemented a navigation bar and a drop down bar in Test8. Through them we added new collections to Futurelogs and checked if the elements were saving on local and remote databases. The results came back positive, everything was saved.

Test8-13

They were given to group members to play around with. They would add/remove as they see fit and try a combination of buttons to try and break the app.

The list of bugs found (and fixed) were:

- Monthly Logs did not generate days properly; Dates and days did not match.
- Calendar drop down menu did not serve a purpose.
- ~~Text editor did not show up under daily Logs.~~
- ~~Task list spacing was too far to the right~~
- ~~Theme selector had blank options~~
- ~~Remote server shuts down when clicking log in too much~~

Summary:

There are still minor issues that need to be fixed but core functions are now operational

and the release date is coming up soon. Not everything can be tended to in time so we will focus on things that have high priorities.

Puppeteer Test Report

The purpose of using Puppeteer to test our bujo app is to provide end to end testing. Using Puppeteer allows us to interact with the web page by navigating to its URL and interacting with the DOM. End to end testing is a method that helps us automate the test cases involving emulating the user actions, from start to finish.

Our main file for end to end testing using Puppeteer is frontend.test.js, which is in our frontend testing folder.

Requirements:

- Test navigating through our Bullet Journal App
    - For this, we tried things like checking the functionality of the settings menu and changing our app's theme.
- Test Login and Create Account to make sure they are working properly
    - Our app should not allow a user to create an account if an account with the same credentials already exists
    - Login username and password(hashed) must match with the database.
    - The app will also not allow accounts to be created if any of the required fields are missing, ie, username and password.
    - When we login to the bujo app, we should be directed to the home page
- Test Logout
    - When we log out of our account, we should be directed back to the login page

Set Up:

We must first install the puppeteer packages and configure the project in package.json and a puppeteer configuration file. Next, we run our app and go to the directory that holds our end to end tests. Before writing the tests, we should call the beforeAll function to set up our desired environment. This includes going to our app's landing page and setting a timeout. After writing our tests, we can run them using `npm test` and we will see whether our tests are working or not.

Result: All tests passed

Summary: Our end to end testing is working as we expected. Our app should be working well as performing these tests helped us to minimize bugs. In the future, we can work to be even more thorough with our end to end testing to make sure users get the best experience with our app.