SW Engineering CSC648/848

Project Name: globetrottr

Section Number: 01, Team Number: 03

| Name: | Role: |
| --- | --- |
| Jay Gupta | Team Lead |
| Devin Kern | GitHub Master |
| Essa Husary | Scrum Master |
| Justin Wang | Backend Lead |
| Kevin Liu | Frontend Lead |
| Brandon Khuu | Backend/Frontend Developer |

Milestone 4

Date: 4/25/2023

1. QA Testing
   1. Unit Test (Jay, Justin) (Jest)
      1. 5 P1 features & List of Corresponding Source Files:
         1. Registering/Logging into Account:
            1. [Controller/userController.js](https://github.com/CSC-648-SFSU/csc648-01-sp23-team03/blob/unit-testing/application/backend/Controller/userController.js)
            2. [Model/userModel.js](https://github.com/CSC-648-SFSU/csc648-01-sp23-team03/blob/unit-testing/application/backend/Model/userModel.js)
            3. [routes/user.js](https://github.com/CSC-648-SFSU/csc648-01-sp23-team03/blob/unit-testing/application/backend/routes/user.js)
         2. Schedule Generation based on Input Variables:
            1. [Controller/scheduleController.js](https://github.com/CSC-648-SFSU/csc648-01-sp23-team03/blob/unit-testing/application/backend/Controller/scheduleController.js)
            2. [Model/scheduleModel.js](https://github.com/CSC-648-SFSU/csc648-01-sp23-team03/blob/unit-testing/application/backend/Model/scheduleModel.js)
            3. [routes/schedule.js](https://github.com/CSC-648-SFSU/csc648-01-sp23-team03/blob/unit-testing/application/backend/routes/schedule.js)
         3. Adding/Removing Schedules to Dashboard:
            1. [Controller/scheduleController.js](https://github.com/CSC-648-SFSU/csc648-01-sp23-team03/blob/unit-testing/application/backend/Controller/scheduleController.js)
            2. [Model/scheduleModel.js](https://github.com/CSC-648-SFSU/csc648-01-sp23-team03/blob/unit-testing/application/backend/Model/scheduleModel.js)
            3. [routes/schedule.js](https://github.com/CSC-648-SFSU/csc648-01-sp23-team03/blob/unit-testing/application/backend/routes/schedule.js)
         4. Search Functionality & Search History (for Queries and Search Terms):
            1. [Controller/queryController.js](https://github.com/CSC-648-SFSU/csc648-01-sp23-team03/blob/unit-testing/application/backend/Controller/queryController.js)
            2. [Model/queryModel.js](https://github.com/CSC-648-SFSU/csc648-01-sp23-team03/blob/unit-testing/application/backend/Model/queryModel.js)
            3. [routes/query.js](https://github.com/CSC-648-SFSU/csc648-01-sp23-team03/blob/unit-testing/application/backend/routes/query.js)
         5. Important Additional Schedule Info:
            1. [Controller/scheduleController.js](https://github.com/CSC-648-SFSU/csc648-01-sp23-team03/blob/unit-testing/application/backend/Controller/scheduleController.js)
            2. [Model/scheduleModel.js](https://github.com/CSC-648-SFSU/csc648-01-sp23-team03/blob/unit-testing/application/backend/Model/scheduleModel.js)
      2. Test Cases Directory: <https://github.com/CSC-648-SFSU/csc648-01-sp23-team03/tree/development/application/backend/tests>
      3. Description of Functional & Statement Coverage: The backend/routes directory, which contains all of the routes of the application, has a statement coverage of 78.31%, and a functional coverage of 100%, which are above the recommended guidelines. The backend/Controller files, as it relates to the P1 features, has a statement coverage of 85%, and a functional coverage of about 95%, which are also above the recommended guidelines.
      4. Description of Integrated CI Work Flow: We have a workflow that occurs on every push. It setups node.js, caches node\_modules, and runs mongodb in the background. It runs the Jest unit tests we have, using the environment variables set up in the repository. Workflow runs can be seen in the [Actions tab of the repository](https://github.com/CSC-648-SFSU/csc648-01-sp23-team03/actions).
   2. Integration Test
      1. Description of test cases and test results for each P1 feature:  
           
         For each P1 feature, we have a spreadsheet that describes the function to test, the test data, prerequisites of the test and the steps to test that function. The spreadsheet also records the test results, the assignee (if the test fails) and the URL to the Github issue opened.  
         The spreadsheet can be found at:  
         [848 Integration Testing](https://docs.google.com/spreadsheets/d/1kgBXongdt0aqIeBy3AbkNic__xM2WCu0mQgKkorxe3M/edit?usp=sharing)  
         (<https://docs.google.com/spreadsheets/d/1kgBXongdt0aqIeBy3AbkNic__xM2WCu0mQgKkorxe3M/edit?usp=sharing>)
      2. URL of bug tracker system which records failed integration test results:   
           
         For each failed test result, we open issues on Github and assign personnels who made that specific function. For each failed test, we tested them again after the assignee resolved the bugs. The URL to each Github issue is included in the above spreadsheet, in the cell “Remarks & GitHub Issue Link.”   
         Alternatively, all the issues (now closed since the assignee addressed them before the submission of this milestone) can be found here: <https://github.com/CSC-648-SFSU/csc648-01-sp23-team03/issues?q=is%3Aissue+is%3Aclosed>
      3. Description which analyzes coverage:  
           
         All 7 P1 features we promised in the Milestone 3 document were tested. For each function being tested, the test cases and steps cover how the user would normally interact with the app to achieve that function, and how to verify if the function works correctly.
2. Coding Practices (Essa, Brandon) (ETC: 5/5)
   1. Coding Style (Google JavaScript Coding Style) (Prettier)
      1. Description of Coding Style & how to enforce it by tool or plug-in:

The coding style we chose to follow and enforce was the “Google JavaScript Coding Style” that can be found and referenced through this link here:

[Google JavaScript Style Guide](https://google.github.io/styleguide/jsguide.html#jsdoc)

We used a code formatting tool called “Prettier” which itself enforces a consistent code style by right-clicking on the file one wants to format and then hitting “Format Document With…”. A balance had to be struck when enforcing the Google JS style, running Prettier, and making sure that our code didn’t break all at the same time. What we mean by this is that, for instance, the Google JS Styling Guide requires that all functions be named in camelcase fashion. The issue with this is that certain React functions only work when the function name is in Pascal case, that is, where the first letter of a function name is capitalized as opposed to the camelcase naming convention where the first letter of the function name is lower-case. So in order for our code to not break, it was imperative that we defied the Google JS style for instances such as these. Further, the Google JS style requires that all filenames be lower case, but again, this breeds bugs and errors in our code, so we deemed it imperative to defy this rule as well. When it came to formatting our files with Prettier, things ran smoothly where, for example, semicolons were inserted where they did not exist previously. However, there were times where Prettier itself did not obey the Google Style Guide, as the developers of Prettier themselves say that it is an “opinionated code formatter”. A case where Prettier’s automatic formatting had to be defied was in the case of line wrapping. Google’s JS style requires that there be 4+ indentation if a line of a block comment or even a line of code ran past the 80 column limit. So we had to manually adjust that. So given what has been stated, we can define a hierarchy for how we desired the code to be formatted:

1) Making sure code doesn’t break

2) Enforcing Google’s JS styling principles

3) Running the Prettier formatter on each file

Let’s start with #3. We first run prettier on the file in which we want to enforce the code styling. This allows and ensures proper syntax where it doesn’t already exist. Next, we manually ensure that Google’s JS styling principles are being obeyed as Prettier does not obey that specific style, though it does help with syntax as we just stated. However, and this cannot be emphasized enough, if we find that our code breaks when enforcing one of Google’s principles, we deem it necessary to defy that principle to ensure that our code still works and that there are no bugs. So code working and functioning is king, Google’s JS style is below that, and Prettier is the least of the above three.

* + 1. Source files related to 5 P1 features which demonstrate coding style

The source files are as follows:

Backend:

Backend/index.js

Controller/userController.js

Controller/scheduleController.js

Controller/queryController.js

Controller/index.js

Model/db.js

Model/index.js

Model/queryModel.js

Model/scheduleModel.js

Model/userModel.js

routes/user.js

routes/query.js

routes/schedule.js

routes/index.js

Frontend:

Pages/Login.jsx

Pages/Signup.jsx

Pages/Landing.jsx

Pages/CreateForm.jsx

Pages/Results.jsx

Pages/Schedules.jsx

Pages/ScheduleView.jsx

Components/SchedulePreviewCard.jsx