SW Engineering CSC 648-04

“Track Your Money (TYM)”

Team 6

11 / 28 / 2022

Team members:

Robert Sato (Team Lead),

Siham Argaw (Scrum Master),

Hemanta Thapa (Front-end Lead),

Neel Manthani (Back-end Lead),

Diego Islas (Github Master)

​​Product summary (e.g. how would you market and sell your product – max ¾ page)

## · Name of the product : Track Your Money

* Explicit list of **ALL major committed functions (your FINAL P1 functions you finalized after M3 meeting) your team shall deliver and test for. If you do not deliver the committed P1 functions implementing and passing the tests will result in the reduced grade.**

1. Login User Functionality
2. Register User Functionality
3. Add Income Functionality
4. Add Expense Functionality
5. Retrieve Transactions Based on Date

* What is unique in your product features.

Track Your Money app displays a graph of all expenses and income added per year, which makes it easier for users to track where in the year their income was more than expense and vice versa.

* URL to your product accessible to instructors and TA, working on deployment server.

csc.csc648team06.com

Unit Test

<https://github.com/CSC-648-SFSU/csc648-04-fa22-team06/actions>

<https://github.com/CSC-648-SFSU/csc648-04-fa22-team06/tree/unit_test>

We had 4 unit tests. They tested the key components of our application.

Our app is an expense app so it checks if we can login, insert data, and render the website.

Our unit test included:

User story: User needs to be able to insert an income transaction and make sure its been inserted in the database correctly.

One test was about testing if we could successfully insert an income transaction document into the database. We query an existing user and check if the database and function returns the proper response.We test if a document can be inserted and because we have functions that execute specific tasks when we test the one function that calls those functions we test all the functions

User story: User needs to be able to insert an expense transaction and make sure its been inserted in the database correctly.

The second was about testing if we could successfully insert an expense document into our database. We insert a document and check if the document is inserted successfully. We test if a document can be inserted and because we have functions that execute specific tasks when we test the one function that calls those functions we test all the functions.

User story: User needs to be able to login and make sure if they input their information they will be granted access into the website

Another was a login check where we tested if a pre-existing user would be accepted by our login check function. We insert a document and check if the document is inserted successfully. We have a function called login\_check where we query the database with an existing user and we make sure the function returns true. Since we separated the database functions from the route, we can test 100% from the one function.

User story: Website needs to render correctly and navigate functions need to be of the correct type.

Another was our front end check which test our landing page. It test our initial landing page when you use our application. It test if the page renders successfully and the navigation links are working. It does this because the navigate function provided by react are a child of router so we wrap the render with router. In react we have a function that returns our landing page. We make sure our landing page will render correctly and the navigate functions are of the correct type. We cover the one function so coverage is 100.

We used github actions to automatically trigger our unit test when we push or pull on certain branches. We have a yaml file that github uses to follow our workflow steps. It prints to the console a general report on code coverage. The code coverage report can be deceiving because are npm test – – coverage is ran in the whole project directory. So there are many files (node modules, css files, icon/image files)

Integration Tests

<https://github.com/CSC-648-SFSU/csc648-04-fa22-team06/issues>

<https://docs.google.com/spreadsheets/d/1XVfiZ-L5USyfx91pWJ1zOPbLT56BPBmOHjC8CS11ZIk/edit#gid=0>

We are using github issues with a google sheets to keep track of the issues.

We all test the p1 features in separately

. we test if an user can login and we use an existing user information. And we test if an incorrect user can login. We cover all the features for this by simply logging in with a correct and incorrect user.

. we test if a user can register with a new email. We test if the user is registered if the user exists. It should not be registered if the email already exist. We cover all the features for this by simply registering in with an existing and not existing user. Then check the database.

. we test if the users can insert a income component and it shows up in the database. We check if the information we inputted is in the database with the right user information with it in the document.

. we test if the users can insert a expense component and it shows up in the database

. we test if the correct user data shows up when user logs in. And we test if the user information changes based on the month. We check if the information we inputted is in the database with the right user information with it in the document.

. We test if the insights page shows useful information about the users data. We go to the insights tab and check if the information belongs to the user.

## 

3) Code Review:

a) Coding style: Google coding style, enforced through code review.

When it came to the coding style we used meaningful names for information that was being passed from the user and when we were setting the information. The coding style for the function names was the standard, start with lowercase and use uppercase for other added names: ex. setUser(). This would be an example on how we would format most of our functions. Declaration of the variables and functions were always at the top, although we would have to scroll up to see which variables were declared, we were able to see what was declared and would not have to look anywhere else on the page. As for the formatting, we used visual codes default formatter, in which would indent and align functions and statements. This made it easier to read and see where the function ended and what other information was part of the same function.