**CS673 Software Engineering** 

**Team 2 - SoloSavings**

**Software Test Document**

| Team Member | Role(s) | Signature | Date |
| --- | --- | --- | --- |
| Will Aftring | Team Leader | *William Aftring* | 9/25/2023 |
|  |  |  |  |
| Xin Zhao | Flex Leader | *Xin Zhao* | 9/25/2023 |
| Whitney To | QA Leader | *Whitney To* | 9/25/2023 |
| Christan Pratt | Requirements Leader | *Christan Pratt* | 9/25/2023 |
| Hanwen Zhang | Security Leader | *Hanwen Zhang* | 9/17/2023 |
| Andrew Ouellette | Configuration Leader | *Andrew Ouellette* | 9/25/2023 |
| Maryam Hussein | Desing and Implementation Leader | *Maryam Hussein* | 9/25/2023 |

**Revision history**

| **Version** | **Author** | **Date** | **Change** |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |

[Testing Summary](#_heading=h.gjdgxs)

[Manuel Tests Reports](#_heading=h.30j0zll)

[Automated Testing Reports](#_heading=h.1fob9te)

[Testing Metrics](#_heading=h.3znysh7)

[References](#_heading=h.2et92p0)

[Glossary](#_heading=h.tyjcwt)

# Testing Summary

In this iteration, we have created unit tests for our application as well as performed manual testing from the UI to check for UI component functionality and connectivity to backend, performed manual testing with postman to call for the endpoints created from spring-boot application to test for the backend functionalities and expected behavior. Some automated integration testings were done but overall automated integration testing is still in development for the added features for our application. Due to the time limit, the integration tests for transaction features will likely be included in the next iteration report as well as for other features added for the next iteration.

**1. Unit Testing:**

* **What was tested:** Individual components or units of our application, such as methods and classes, were tested in isolation of dependencies to ensure each unit functions correctly.
* **Who is involved in testing:** All team members are primarily responsible for unit testing, each team member will write unit tests to cover methods/functionalities that they created.
* **Testing Techniques Used**: we write and run test cases using frameworks including JUnit and Mockito. Test cases are designed to cover different scenarios and edge cases within the individual units.
* **Testing Result**: we have a 100% pass rate for the unit tests which tested 84% of the classes, 75% of the method and 84% of lines in our application. We would like to increase these numbers and cover more methods created in the application as we develop and create more features in the future.

**2. Manual Testing with Postman:**

* **What was tested:** we tested the endpoints created in our application and confirmed the flow and the behavior of the application by comparing the expected response from the API call.
* **Who is involved in testing:** All team members are responsible for manually testing the functionality they develop, each team member will make API call from postman to a specific endpoint and they will confirm or adjust the logic in code to ensure the expected behavior.
* **Testing Techniques Used**: Testers will use postman to make API calls while the application is running locally. Testers will provide the correct setup URL with any other necessary information such as a body request for a POST API call to postman. And the tester will observe the output, the logs in the application itself and confirm the flow.
* **Testing Result**: We have tested controller methods with methods by calling endpoints from postman and we got 14/15 tests passed initially. The failed test expected error from adding expense but did not see the correct response but this was fixed and updated. Overall, all manual tests now pass with covering all the endpoints in controller classes except for those meant for routing in UI.

**2. Functional/Manual Testing with UI components:**

* **What was tested:** we tested functionalities of the application as a whole by testing from the UI and expect both frontend and backend work well together.
* **Who is involved in testing:** All team members take part in manually testing the functionality they develop, each team member will test from the UI and test from the client point of view.
* **Testing Techniques Used**: With the application running on the backend, test can access localhost url to access the UI of the application. Testers will go through the whole flow of registering a new user, login with user created, add income and expense and be able to see the expected behavior reflected on the UI page with correct information and landing page, routing pages.
* **Testing Result**: For this iteration, functionalities that are working include login, register new user, add income, add expense, be able to see the updated balance on the dashboard. Things to be improved for the future include showing correct error when wrong input is given, correct message showing and page routing for edge case scenarios.

# Manual Testing Report

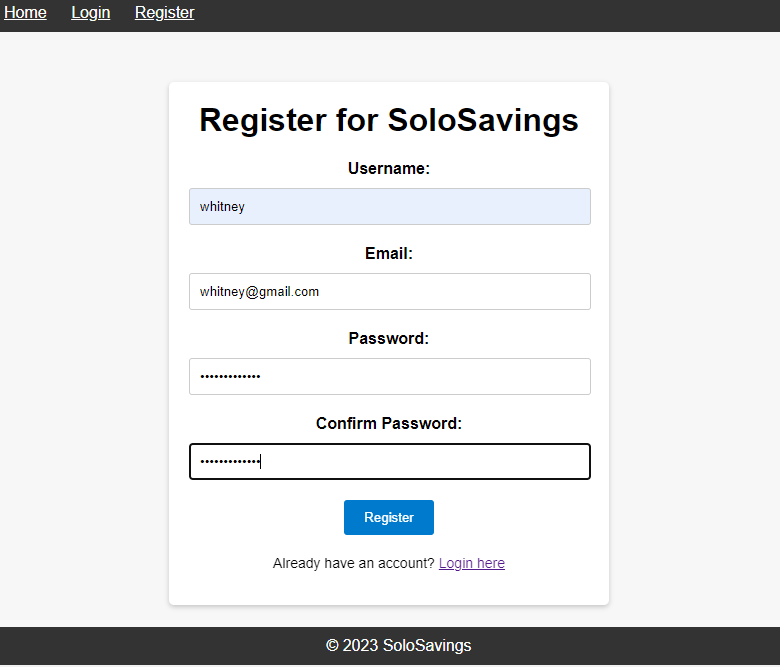
We performed manual testing using postman to test endpoints created from our application. The report can be found in the below Google SpreadSheet link:

[CS673\_Manual Testing\_Team2](https://docs.google.com/spreadsheets/d/1lnsNEZebDdWaqx8FMv4HBKB5OR5NM2IkYowBZ5pT2-o/edit?usp=sharing)

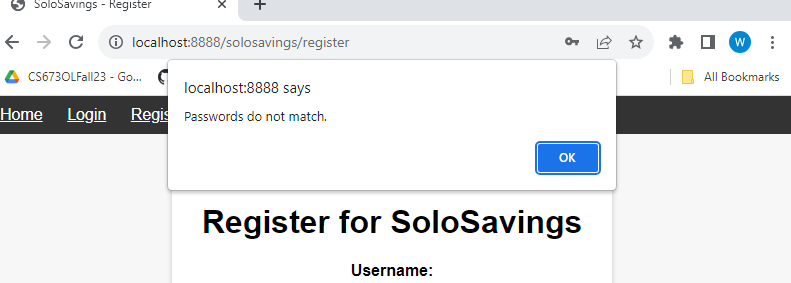
We also performed manual testing from the UI components to test the UI functionality and the connectivity to the backend, below are the screenshots for the UI manual testing:

**Acceptance (Manual) Testing:**

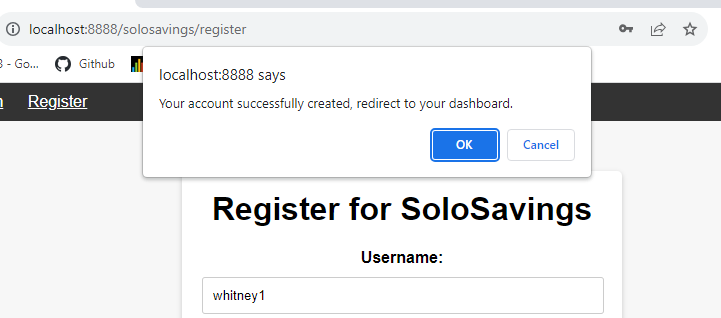
1. Page for register new user



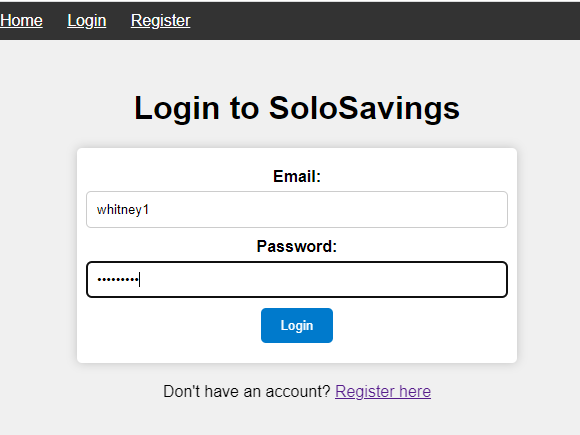
1. When register new user, but the password not matching:



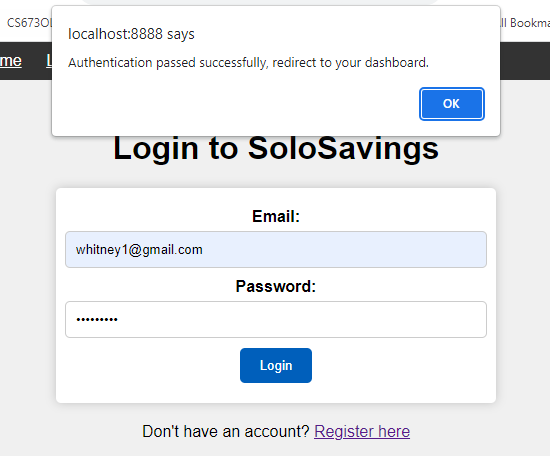
1. When all the given fields are correct and user has not registered before



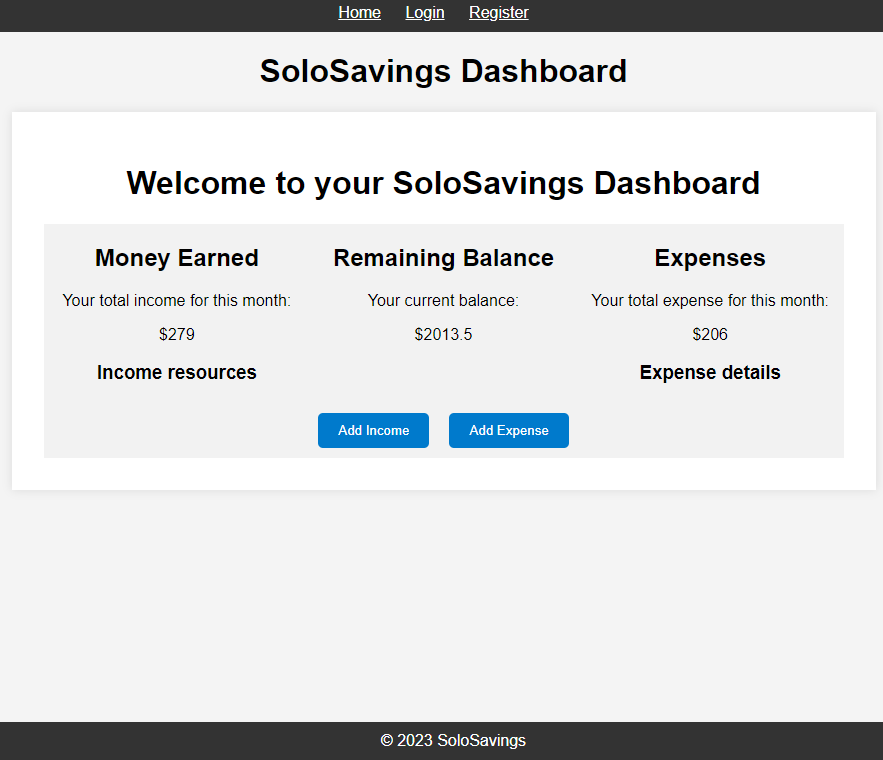
1. After registration, user can login on this page



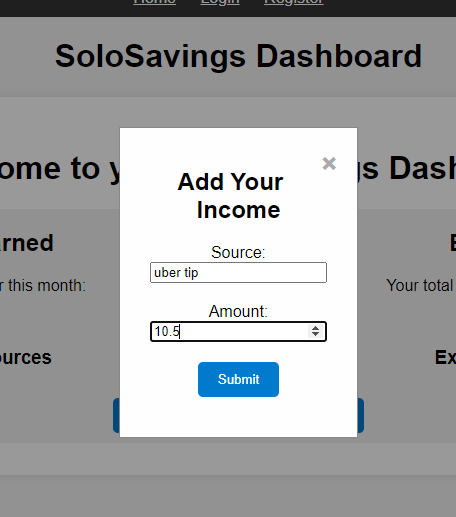
1. User login successfully with correct credential



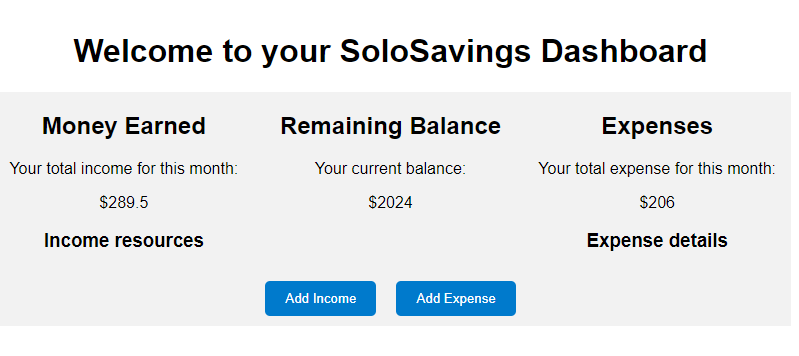
1. After login, user can see their dashboard



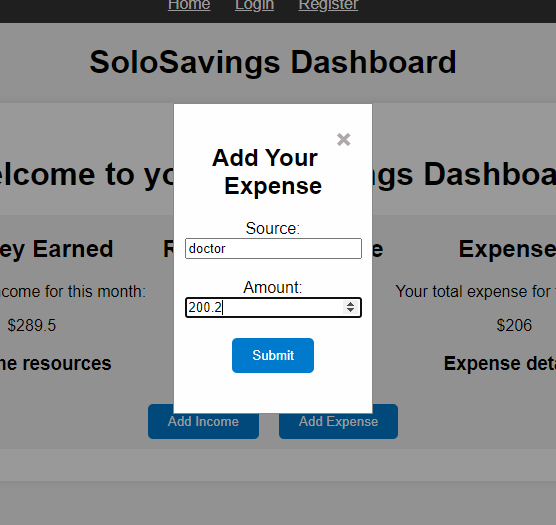
1. User can fill in information if they want to add income



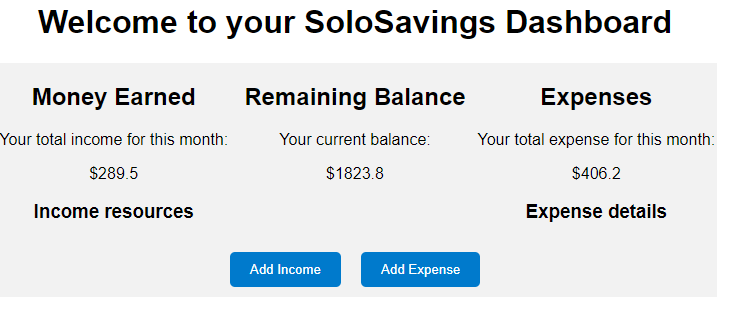
1. User can see the changes on dashboard once hit submit button for add income



1. User can fill in information if they want to add expense



1. User can see the changes on dashboard once they hit submit for add expense

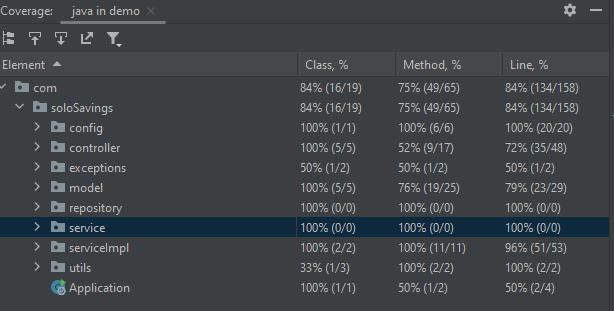


# Automated Testing Report

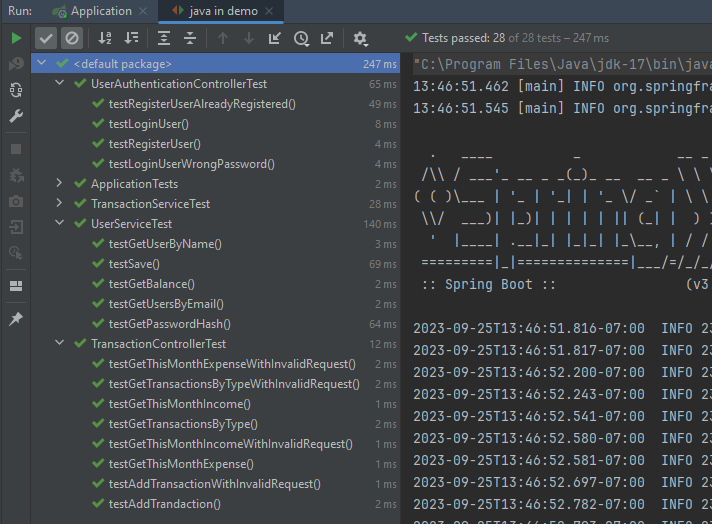
In this iteration, we developed unit tests for each of the functionalities that we implemented. Specifically, we wanted to have unit tests to cover all the methods that contain business logic for our application as well as other methods we created for the application. We used Junit test for unit testing and Mockito for mocking dependencies when needed to isolate the unit of test from dependencies. Below are screenshots of our automated test results including the test coverage, test status and time took to complete the test.

**Test code location within the project**: SoloSavingsApp/src/test/java/com/soloSavings

**Test Coverage:**



**Overall Test Result:**

****

# Testing Metrics

Some metrics used for the evaluation include # of test cases, test coverage, defects rate, etc.

| Metric Name | Description |
| --- | --- |
| # of automated test cases | 28 test cases |
| Automated test pass rate | 100% |
| All test execution time | 247 ms |
| Class coverage | 84% |
| Method coverage | 75% |
| Line coverage | 84% |
| Defects rate Defects / KLOC \* 1000 | 3 / 1176 \* 1000 = 2.6 |
| # of manual test performed (postman) | 15 tests |
| Manual test pass rate (initial) | 14 / 15 passed |
| Manual test pass rate (final) | 15 / 15 passed |
| Estimate time perform manual testing | 3 hours |