**Budget Management**

# Iteration 2 Progress Report

Authors: Anna Kim, Ali Zaidi, Na Li, Jack Long, Joon Lim, Andy Boho

Android User Interface Implementation Report

**Main Difficulties**

We have been struggling to connect the MySQL database with the servers and spent quite some time to figure it out. We also need to figure out how to integrate with the client.

We were able to connect the database and server in another platform, but had trouble implementing it on Android Studio. We ended up finding that despite our server and client working in eclipse and in the terminal, integrating into

We also had trouble merging all the separate branches into master. We didn’t start using .gitignore file until this iteration so there were lots of unnecessary files in all the branches. So, there were lots of merge conflicts whenever we tried to gather all the files into master.

Since the server and database connection was not stable on the iteration 2, we tested our backend code by putting hard-coded data. This will be improved in the beginning of the last iteration by implementing the server into the entire project. For now, we successfully connect the server to the project and implement the server on the login system. So, connecting the server with the rest of activities will be done in the beginning of iteration 3.

**Features not implemented as planned**

We planned to allow users to have their own options to choose three questions from a list of security questions when creating an account of the app. In the current iteration, we mainly focused on the implementation of the connection between database, and server, as well as the testing of the systems, we decided to keep other features simple. Thus, our current version only provides three pre-set security questions. If time allows, we will implement this feature so that users can have more options for security questions, which also enhances the security of using the app.

We are pushing some features in settings to later iteration. Features like user guide and application info are not essential parts of the application. Moreover, they don’t interact with other parts of the app and server. We will also push the goal page to later iterations. There were difficulties in setting up server and client, so we are focussing on essential features that are prime functions of the application.

**JUnit test and Code Coverage**

Backend implementation test

Since we mainly use JAVA, we used JUnit as the unit test framework to test our program.

**Login activity**

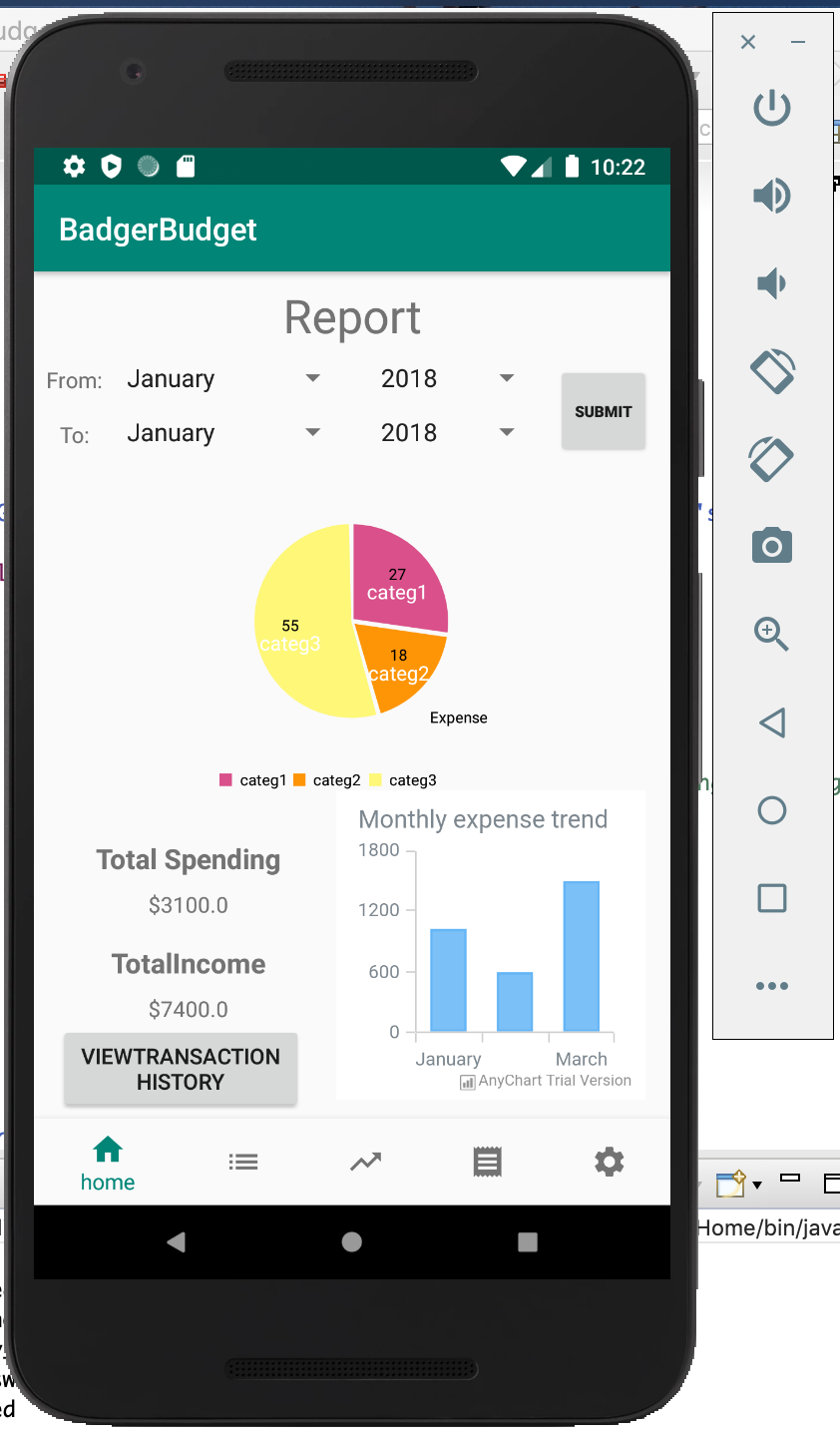
The login page is really only meant to either get an existing user’s account, or guide them to the create account page. As such, the queries that we needed to test via the server and client were already tested in the server section below. Once the login activity backend was completely implemented we manually tested the features of the page, trying to login with existing dummy users, which would then take the user to the home page. We would then try to login with non-existing users which would prompt the page to display via a Toast message that no such account existed. We did write some Junit tests for the login activity, though they are very similar to the server tests that are listed later in this document.

**Main activity**

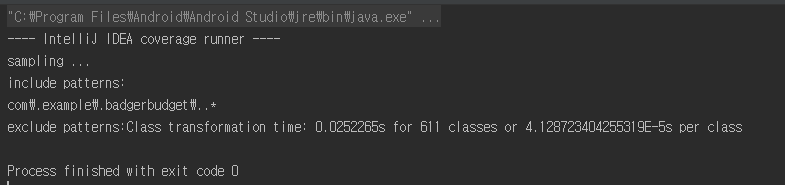
The Main page is for showing current balance and overall expense which is a main function of the app. It also enables users to add new transactions. We manually tested the app by changing input values of Textviews of current balance, current expense, and target expense. We also checked whether values are correctly added to the spinner. We also tested when the “add” button is clicked. Whenever the essential part like type of transaction, category, and balance is not corrected submitted, it will show failure to add.

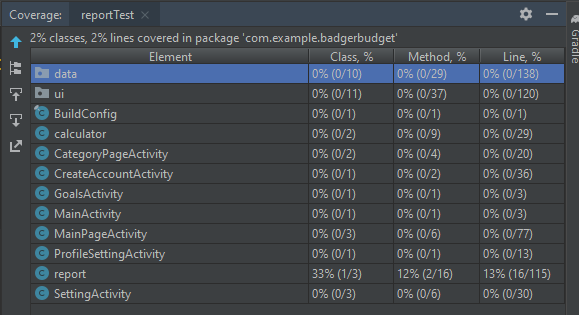
**Report activity**

Since the report page is for displaying user-input data with a user-given condition, features like bar chart and pie chart are tested by running the project on the Android Virtual Device.



The backend implementations we tested are calculating total expense and income and adding data into the array set to be used when displaying the pie chart and the bar chart. In order to display the total income and total expense, the basic addition function is also tested.





The code coverage is low as we expected. This is because we did not write test code for the methods that display charts and text on the user interface. Instead, we tested them on the interface by running the project, as described above. We believe that code coverage will improve as we further develop the project by bringing actual user input data using queries and using methods within the class to retrieve the data.

In the next iteration, we will mainly test that the report class correctly retrieves month and year data with the corresponding expense, category and income data. The retrieved category, expense, and date will be stored in HashMap. So, we will test the data added in Hashmap. As a regression test, we kept and will keep using the previous JUnit tests when we tested other features or after we made changes in our code.

Server Implementation Report

**Main difficulties**

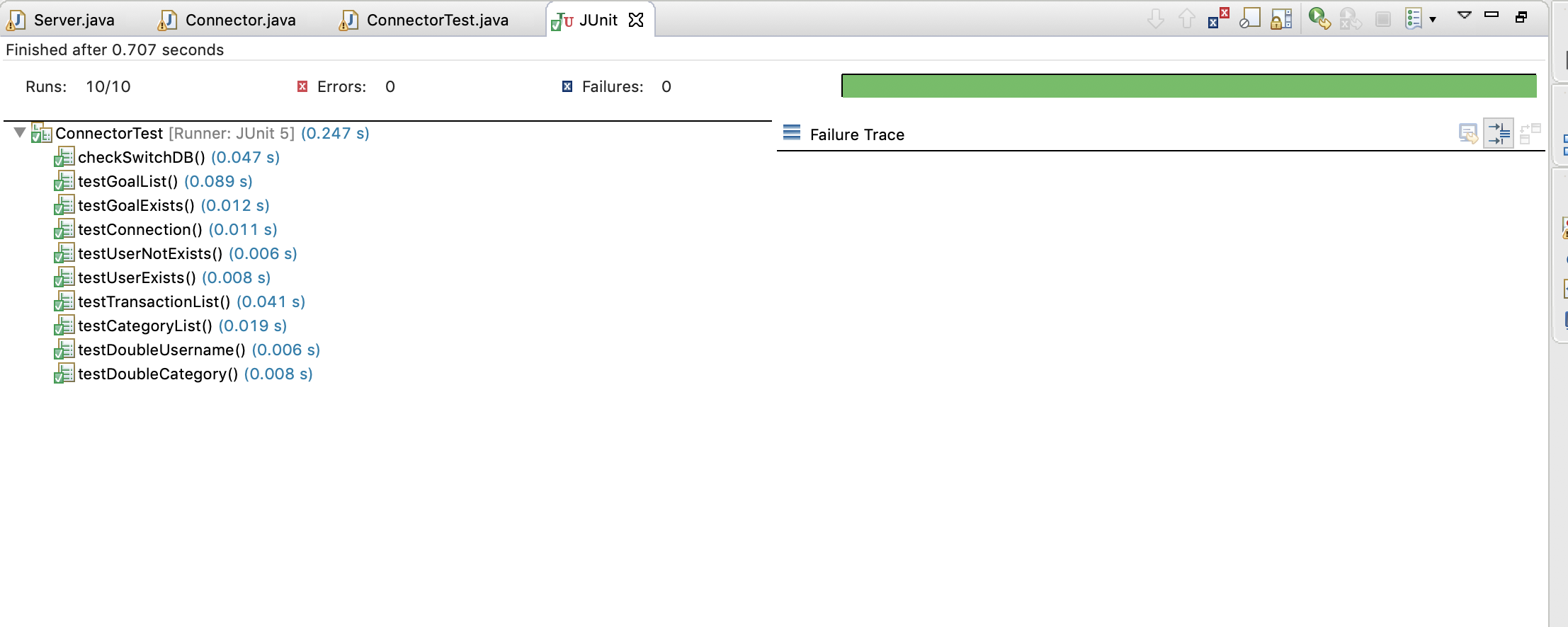
During the beginning of the iteration, for some reason, I couldn’t get my XAMPP server to connect to my java socket server. For this reason, we made the switch to just use MySQL Workbench. After that, the java socket server was able to successfully connect to the MySQL Database. Integrating the socket server proved to be a challenge and took us much longer than expected. We didn’t realize that we would need to run the server on the emulator’s local ip “10.0.2.2”. Once we came to that conclusion, we were able to successfully do basic user tasks like creating a user and logging into the app.

**Features not implemented as planned**

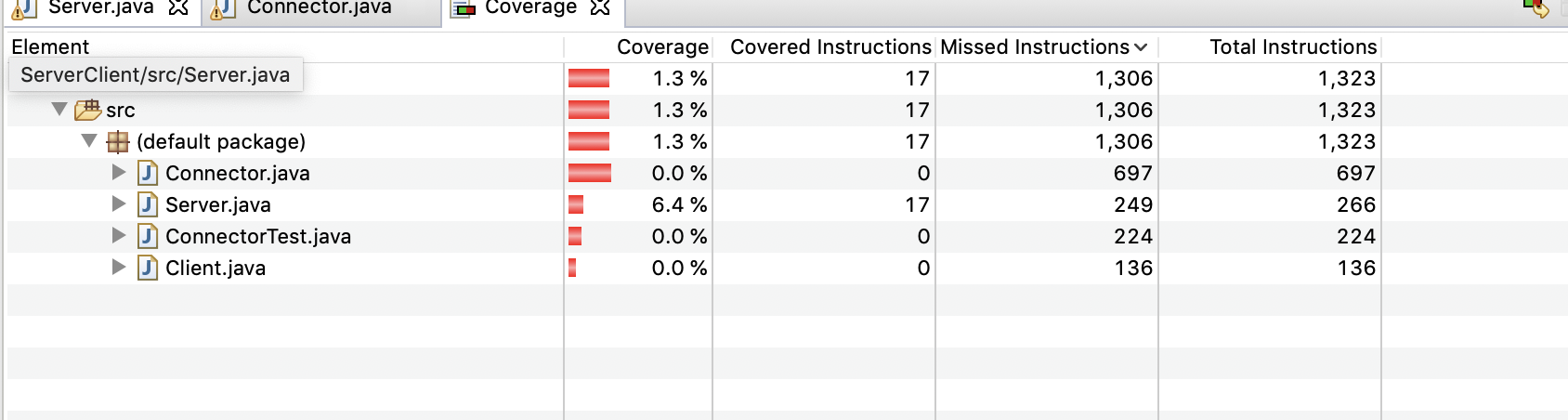
We didn’t get as much of the backend and frontend connected as we liked. It took a little bit longer than we expected to get the server connected to the database, so we were only able to implement “login” as well as “create user” for this iteration. We do have full backend features now; however, most of them aren’t actually connected to the frontend.

**Server Test**

We used mainly JUnit for running Unit Tests as we are coding in Java. Since the backend stores all the data, we wrote JUnit tests for testing to see whether the data retrieved from the database. The first test we wrote made sure that the connection to the MySQL server was successful. For login validation, we tested whether or not a user could be implemented into the database more than once as well as if their password matched the username. For the user’s information, we tested whether or not the MySQL server successfully switched from the ‘Users’ database to the user’s information database. We also implemented tests for the user’s data. We placed values in the user’s Goal List, Transaction List and the Category List. We compared what the database returned to the expected value. We also implemented tests to make sure that the user couldn’t enter a particular goal or category more than once.

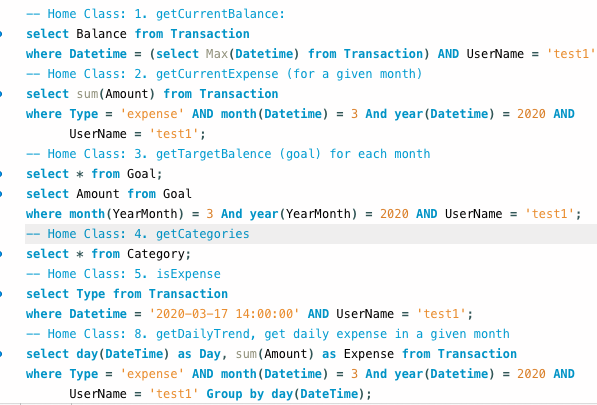


For the server and connector classes in Eclipse, we used Eclemma, a code coverage tool that comes with Eclipse. Running the code coverage made us realize how many branches that we were missing in the Connector and Server classes. Going forward, it is something that we really need to work on. The coverage was low for these methods because we encountered a lot of errors while writing the connector and server classes. We were more focused on making sure that we could get everything done then worrying about code coverage; however, now that we have things up and running it is something that we’re paying closer attention to.



**Database Test**

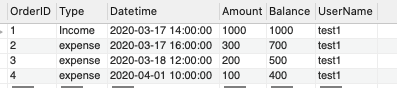
We created a MySQL database and wrote the corresponding queries that realize the functionalities of our Budget Management app. To test the accuracy of the queries, we created sample data for all the entities. We also tested the queries when we implemented the connector between server and client. Below shows part of the queries tested.



Take the Home Class function 8 -- getDailyTrend in 03/2020 for example.

The Transaction table is below. The expected output should have row 2 and 3 with column Amount and Datetime for the transaction.

**Transaction Table Actual output**

Submission

Our github repository, BudgetManagementApplication, has been shared with our TA Siyang Chen. The branch for iteration 2 has been tagged. In order to check the code and run the application for iteration 2, you can follow the instructions in the README file.