**Tutor by Request**

**Progress Report - Iteration 1**

**4/9/2021**

**Team members:**

**Azzed Khursheed**

**Edwin Solache**

**Evan Winckler**

**Harrison Bell**

**Ryan Fries**

1. **What were the main difficulties so far?**

One the biggest difficulties has been setting up and implementing a remote database. This is in large part because none of us have experience with it. UI testing was very difficult because of the way we set up the underlying structure of the software. Most of the testing needs to be done on fragments and a lot of the natural testing integrated in android studio is geared towards activities. We had a lot of fragments with complicated layering which made it difficult to test the user interface. Automated testing was difficult.

1. **Were there any features you did not implement as planned, and why? Are you pushing some features to later iterations, and if so, why?**

We’re being forced to push setting up the remote database because it’s a far larger undertaking than we were prepared for. There’s been a significant amount of progress however we must push this to the next iteration while we learn PHP, translate all of the SQL queries, integrate the implementation into the current code, and run tests on the implementation.

1. **What tests did you prepare for this iteration, and what are they covering? Did the tests you wrote deviate from your plan? What features are you not testing yet? Did you use any test frameworks, such as JUnit, the Android Monkey, Selenium, etc.?**

For this iteration we wanted to test all of the control flow and navigation to make sure that each type of user could get where they needed to be. We completed all of that and were able to automate UI tests in order to make this happen. In order to test our code, we used JUnit4 tests for backend code such as database calls and business logic. For the front end, we used a library called Espresso, this allowed us to automate UI testing with some nifty syntax. We are able to replicate the movements of a user and make sure that they are indeed getting to the correct page. We also used junit to get near 100% code coverage for the local database.

1. **Give a sample of code coverage tool output both before and after adding to your test suite. What did you learn from the code coverage data? How did you use this information to expand your test suite and improve coverage?**

For our code coverage we used a service provided in Android Studio called createDebugCoverageReport. This service runs through tests and tracks the percentage of code in our repo used. It was really helpful in giving us insight into where our tests were lacking and where we had thoroughly tested. It mostly showed how a lot of the underlying simple classes we created lacked any sort of direct testing. The first run of the tool we used had us at around 35% coverage. After rearranging the structure of our code and adding UI testing we landed at roughly 76% code coverage.The tool allowed us to see exactly where in the control flow of the software we were failing to test and add specific test cases for these scenarios.

1. **Optionally give a URL and instructions for using your application in the current stage. This makes sense for purely web-based projects, but it may be impractical for projects that must be installed on a client device.**

Follow the instructions in the repo for importing the project. Run the app once before testing then be sure to wipe data for the AVD device in between test runs.