**Software Implementation and Testing Document**

**For**

**Group AssignmentBuddy**

Version 3.0

**Authors**:

Ashley Ellis

Jeffrey Knappman

Analise Saunders

Virginia Sicuriello

**1.** **Programming Languages (5 points)**

The programming languages used throughout the project are **Java**, **XML,** and **SQL**.

In our project,we used **Java** to invoke activities within our Android application. An activity provides the window in which our app will draw its UI; one activity implements one screen in an app. Developing a .java file is necessary for developing activities within our application. Our group chose to use Java because it is the language primarily used in Android development and it is supported by Android Studio (IDE where we are building our app on). Java utilizes object-oriented design, which all members of our group are familiar with; and it contains many frameworks and class features that are beneficial in the development of our project.

We used **XML** in conjunction with java to implement activities within our app. XML was used to store data about our activity - such as activity size, layout, etc. Since XML makes data storage easier, we used it to store necessary user information. We chose XML because it is a necessary aspect in developing the UI for our app, and it contains several predefined widgets in android that aided in our project’s development.

**SQL** was used as a means to manage data that we wanted to store in our application. SQL was primarily used in overseeing data management of user login information and other necessary user data that relate to the user’s personal data, such as university courses that were stored within the app. We chose SQL because it's great for querying data - which is important. After all, our app relied on quite a bit of data storage. It was also used in conjunction with XML.

# **2.** **Platforms, APIs, Databases, and other technologies used (5 points)**

The platform for the project was **Android Studio**, it was the IDE where our app was being developed on. The emulator that our application runs on is **Pixel 3 API 15**. We used this emulator due to this preloaded hardware profile - this relates primarily to the applications UI. The API of the emulator that was being used to run our application affected how our XML files were developed. Android Studio provides a template-based setting which made it easier to visualize how the interface will look like. We also used **SQLite API** in Java in order to create and maintain database tables within our application. SQLite was primarily used in relation to user login information, including data concerning registered courses and assignments. In addition, it was used to help store the classes pertaining to each user, and the grades within each class and assignment.

**3. Execution-based Functional Testing (10 points)**

Unit testing was the functional testing used for this project. Each member had their personal branch in order to test the part of the software they were working on, this way each unit was tested separately checking if the activities were performing as expected. The group decided that only the units that are working correctly will be uploaded to the master branch therefore, this branch has the latest version of the code without errors. In addition, the emulator was also used as a visual way to check that everything is working correctly, and the layouts are centered. Before every increment, each member checked all units to attribute in them or help if needed.

Additionally, with the implementation of a few databases and tables, our group members were able to make data entries into our application and test whether any of the user’s input could possibly ‘break’ our application - this also tests whether our database’s relational design has been implemented well and effectively. For example, the account screen was useful to test the functionality of several activities such as the courses, the todo list, the chat messenger, and the calendar to check how they are interacting with one another and with the application as a whole. Using this method helped us to determine what modules are working as expected once they are integrated. Making sure that an account was able to be created and stored within the database to get to the activity page was also another important test we had to complete.

**4. Execution-based Non-Functional Testing (10 points)**

Two non-functional requirements that were of primary importance was performance and security. As a team, we determined test cases where security would be a primary concern within our application. After running the app and walking through each screen page, we concluded that the login and register screen was the most vulnerable to security threats. To safeguard our application, we made certain that password inputs be of input type “textPassword”. This allows for password input by the user to be hidden (i.e. \*\*\*).

Testing for security required a combination of integrated & system testing because a lot of our checks and user verification occurred in conjunction with our database. Test cases included verifying that users could not login to the application without first being registered – this is authenticated by the database that checks to see if the provided username and password is stored as a registered user.

As for performance, we tested this simultaneously along with its scalability and reliability. As mentioned before, our application relies heavily on data management and manipulation. We had to ensure that our application (namely, our database) did not get too bogged down by the influx of registered users on the app. Volume and stress testing was our methodology to assess this aspect of our application – team members registered several ‘test users’ in a short interval of time to determine if our database could handle that amount of data processing. This was something we tested towards the end of our implementation phase, when all our database tables and heavy running features (i.e. messenger) were implemented and executed.

Another approach we used to test performance was load testing. Each member tried to run several functions within our application simultaneously. It was important for us to assess whether our application could perform under anticipated user loads.

Overall, our application performed well on both performance and security. We did notice that our application ran a little slower whenever the user tried to switch between several different functions simultaneously (i.e. adding calendar events, tasks, messaging all at once), but we concluded that we couldn’t improve this without compromising the functionality of our application.

**5. Non-Execution-based Testing (10 points)**

Our team began iteration 3 with a technical review of our revised R&D document for this increment. We held a conference call to assess our application’s technical specifications. This was to verify that the requirements outlined were suitable for our final stage of development and to determine if we had a solid foundation, in terms of code implementation, to proceed with the remaining technical provisions. This first step of testing was crucial to identifying errors early during our developmental process of our third iteration. Through our technical review, our team identified a discrepancy between our design and specification documentation. This saved us a great deal of time because we realized, despite our developmental standards, that we did not have to create individualized activity screens for either user (student/professor). Instead, both users can view the same activity screen, and a simple hidden button could be used to differentiate between either user’s functionalities/permission controls.

Throughout the duration of this iteration, our team performed several static code reviews or “white box testing”. This involved making certain that team members adhered to coding standards – namely documentation to prevent knowledge silos between members who had overlapping code. It was also done passively, when group members had to resolve conflicts when merging their branch into master – this included syntax checking (ensure changes to syntax were consistent between both branches).

Before submission, our team held a walkthrough, where each member explained a portion of the application. Team members asked questions and made documentation of their comments. At the end of the walkthrough, we complied the most pressing concerns and made adjustments with the given time we had left. Fortunately, many of the comments consisted of minor user interface design suggestions (i.e. placement of buttons) and not flaws in our applications functionality.