Kevin Wong 32105132
Robyn Castro 37283141
Catherine Lee 32556136
Omar Mohammed 52712130
Rohini Goyal 10861145
Jane MacGillivray 19953141
Xingyu Tao 33610149

STREAM

'cuz we've all been there

# Test Plan Document

## Introduction

STREAM is an Android application that allows students to collaborate efficiently and effectively in group projects.

STREAM provides:

1) Secure Login with Facebook or Email
2) Board for teams to add important messages into one area
3) Chat module to communicate with team members
4) Notifications to remind team members on tasks
5) Calendar to view upcoming deadlines
6) Task Manager for users to create, assign and manage tasks for the project

The backend of STREAM will use Firebase, Google's real time database that allows our app to store user data such as tasks, notes, projects, user information.
The purpose of this document is to explain the testing plan for STREAM.

## Verification Strategy

A large part of our planning state was gathering data on potential features that users may want to have available. We asked several of our friends to see what features would have helped them better organize their group projects. Based on the feedback we received, we chose what features to implement in STREAM.

For GUI development, we created mock-ups for how the interface might look like. We showed them to several of our classmates to see if the design was intuitive. Based on their feedback, we modified our mock-ups and went on to implement the app.

After completing the implementation of STREAM, we plan on showing the app to our classmates and collecting their feedback. Any small design changes that can be made will be changed before the final release.

## Non-Functional Testing and Results

The table below shows the non-functional requirements and how they should be tested for our application.

| **Test #** | **Requirement/ Purpose** | **Action** | **Expected Result** | **Notes** |
|---|---|---|---|---|
| **1** | Performance | Using the different features | Data loads quickly and accurately across all devices. Information is synced | It's possible that the chat will perform the worst because it has to continuously sync data |
| **2** | Security | Access database without authentication | Permission will be denied, and no information is shown | |
| **3** | Scalability | Add more users, tasks, pinboard items | Users, tasks, and items should be consistent across each user's device | A suitable upper limit should be considered due to limitations in database usage |
| **4** | Software Quality | Running software similar to JDeodorant to look for bad smells. Or manually comparing Android coding style guides to our application | Overall project meets quality standards | At the moment, we have found a software called SonarQube however we have not tried to run the software yet since our code base is not ready |

## Functional Testing Strategy

For testing our features, we will start off with using JUnit tests to test the individual methods. Since many of Stream’s features require reading and writing to the database, we will create separate functions that will test the reading and writing methods to the database. Instrumented unit tests will also be used to create objects to test the database functionalities.

At the moment, we have been testing the GUI manually and trying edge cases with the interface. If we happen to come across a bug, we fix it right away. We plan on testing the actual software during our last sprint. The reason for testing our software at our last sprint is to eliminate the minor bugs that appear in the software. Since we are testing the GUI regularly, we are ensuring that the methods used for the GUI are working expectedly when we are developing our app; therefore, many bugs will be eliminated from the project as we build it. As a result, the last sprint will simply test the methods through JUnit tests instead of GUI. For our test cases, we will create tests in different orders depending on the importance of the functionality to our application. For instance, testing the projects and tasks will likely be tested first because they are the vital features of the applications. Without these two functionalities, the app is not complete.

We have also implemented integrated tests in the form of JUnit Tests. Nearly all of our tests make use of stubs in the form of a test user and a test project. However, our tests are not confined to a single part of the system; changes to one component of the system affect another, so we do not fully isolate our tests from the rest of the system. For example, tests for the HomeManager affect the project’s TaskGroup and Tasks. We therefore also evaluate the impact  that changes and operations have on other parts of the system when we test one component.

To handle bugs, we will record issues on GitHub’s issue tracking system. In the title, we will describe bugs as three different categories: Low, Medium and High. Low describes bugs that are not extremely vital and do not need to be fixed immediately; even if they are not fixed, it will not be a problem. Medium describes the bugs that need to be fixed but do not need to get fixed immediately. High describes the  bugs that are extremely important for the app. These bugs will cause the main workflow of the application to not function and needs immediate attention and fix. When issues are fixed, one of the developers will provide a comment on what they did to fix the bug and close the issue on GitHub. With GitHub, our team can easily see a history of closed issues and unfixed issues so it will be easier for the team to find and fix current bugs or look up bugs that have been fixed.

## Adequacy Criterion

Our application has many different modules that need to interact with each other seamlessly. In addition to a solid array of unit tests to make sure that our functionalities are independently working, our tests also take into account the interactions that happen between these different modules. In light of this, the main focus of our adequacy criteria is to make sure that our use cases are implemented correctly.

We want to devise at least one test per each use case scenario, and make sure that all of these tests pass. This will ensure that the user experience is smoothly maintained and that all expected behaviours for use cases are followed. We decided to be particularly rigorous with use case scenario tests because these will have the greatest and most direct effects on the user's experience.

For the many unit tests that we are running, we will make sure that there is at least one test for each major functionality. This way, we will make sure that the major internal parts of each class is functioning properly.

## Test Cases and Results

The table below shows some of the tests we have implemented along with the JUnit file this test is confirmed in:

| *Test #* | *Requirement Purpose* | *Action/input* | *Expected Result* | *Implementation* |
|---|---|---|---|---|
| **1** | Create projects | Create new project and add collaborators from the new project screen | Project added to the database;<br>Feedback given to user;<br>Collaborators notified; | addProject() in ProjectsTest |
| **2** | Join project | User 1 creates a project as in test #1, and adds user 2 as a collaborator. | User 2 can go to the app and accept the invitation to join the project;<br>User 2 has access to the project; | joinProject() in ProjectsTest |

| | | | | |
|---|---|---|---|---|
| **3** | Edit project details | User changes details of existing project (i.e due date, collaborators) | Project status is updated in database; | editProject() in ProjectsTest |
| **4** | Add task | User creates a task. And assigns the task to collaborator(s). | Task is added to the database;<br>Collaborators can view the task in the task list;<br>Task contains the information provided by the user; | addTask(), assignTask(), and taskDetails() in TasksTest |
| **5** | Complete task | User marks task as complete. | Task status is updated in database; | setTaskCompletion() in TasksTest |
| **6** | Delete task | User deletes a task | Task is removed from the database; | deleteTask() in TasksTest |
| **7** | Edit task | User changes details of existing task (i.e due date, assignees) | Task contents are updated in the database; | editTask() and changeTaskAssignee () in TasksTest |
| **8** | Add item to pinboard | User Uploads an item to a project's pinboard view | Item uploaded to database;<br>Project collaborators can view the item from their pinboard view;<br>Pin contains the information provided by the user; | addPin() and pinDetails() in BoardTest |
| **9** | Delete item from pinboard | User Uploads item to pinboard. Same user later deletes the item. | Item is deleted from database;<br>Collaborators no longer have access to the item; | deletePin() in BoardTest |
| **10** | Edit Pin details | User changes details of existing pin | Pin updated in database; | editPin() in BoardTest |
| **11** | Meeting can be added to Calendar | User adds a new meeting to the team calendar. | Meeting is added to the database;<br>Meeting contains the details inputted by the user.<br>Calendar status is updated in | addMeeting() and meetingDatails() in CalendarTest |

| | | | | |
|---|---|---|---|---|
| | | | database; | |
| **12** | Delete meeting from calendar | User deletes an existing meeting from the team calendar. | Meeting is removed from the database;<br>Calendar status is updated in the database; | deleteMeeting() in CalendarTest |
| **13** | Edit meeting details (Not implemented in current product, but is tested for future extensibility) | User modified an existing meeting in the team calendar | Meeting details are modified according to values provided by user;<br>Meeting is updated in the database; | editMeeting() in CalendarTest |
| **14** | Add members to an existing project. | Users in a team can add new Stream users to an existing project | The member is added to the project;<br>The project is made available to the user added; | addRemoveMember () in TeamTest |
| **15** | Remove members from an existing project. | Users already in a project can be removed from the project. | The member is removed from the project;<br>The project is no longer available to the member; | addRemoveMember () in TeamTest |
| **16** | Show a user’s own tasks in the home screen. | Users will view only their own tasks from the home screen. | Tasks assigned to the user can be retrieved using the HomeManager;<br>Tasks assigned to other users are not retrieved; | verifyOwnTasks() in HomeTest |
| **17** | Calculate a user’s progress in their team. | The number of tasks that the user completes will be calculated over the total number of tasks. | When tasks are completed the user’s completion progress can be retrieved;<br>Progress is calculated as incomplete tasks/all tasks; | verifyUserProgress() in HomeTest |
| **18** | Add and edit tasks from the home screen (Not implemented | Users can edit tasks assigned to them or add new tasks to assign to themselves using | New and corrected tasks are shown on the home screen;<br>The TaskGroup and Tasks of the projects are updated; | editTask() and addTask() in HomeTest |

| | | | | |
|---|---|---|---|---|
| | in this release, but tested for future extensibility) | the home screen (HomeManager). | | |
| **19** | Set task status to complete from Home screen | Users can choose to complete the tasks displayed on the home screen | The status of the task is updated to complete;<br>The task is no longer visible to the user an active task; | markTaskComplete() in HomeTest() |
| **20** | Create a User with Email | Users can sign up using an email and gain an account to access the application with | The user is authenticated using Firebase login;<br>The user's token and information is recorded on Firebase; | createUserAndLogIn() and userDetails() in UserTest() |
| **21** | Retrieve Users from the Database | Users can be retrieved from the database either using their UID (firebase user token) or Username | The user is retrieved using either their UID or username; | fetchByUsername() and fetchByUID() in UserTest() |
| **22** | Log out a user | Users can log out of the application | The user is logged out of firebase;<br>The user object is no longer stored; | logout() in UserTest() |

These tests effectively cover all the back end functionality for all of the major functionalities of Stream's system: Users, Projects, Tasks, Calendar, and Pin Board.

## User Testing

Our team has given the application to our friends and family to test. Some of their feedback are recorded in the following link:

https://github.com/omarbakker/Stream-app/blob/master/docs/Updated%20Design%20-%20Nov%2027%20Deadline/STREAM-CustomerFeedback.pdf

Generally, our friends and family like the application and it's features. They think our application is simple and clean. We have fixed some of the comments mentioned by our users.

Their main issue was with pinboard, the user interface was not ideal. We have since updated the pinboard to have a better user interface that looks like sticky notes and allowing users to choose the colors of the note.