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

Design 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

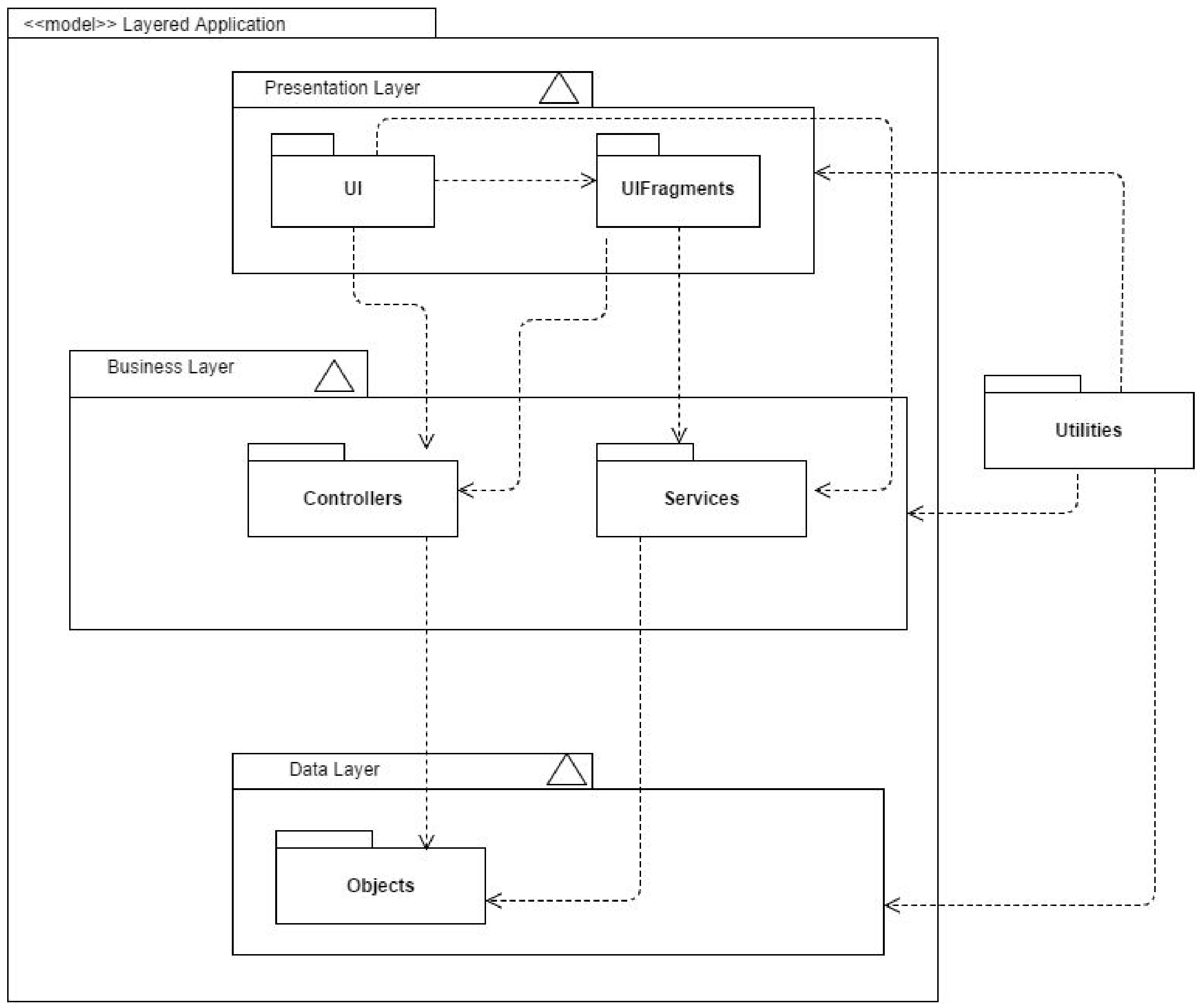
The purpose of this document is to explain the system architecture and design for STREAM.

SYSTEM ARCHITECTURE AND RATIONALE

The application consists of an Android Application connected to a database to manage the information pertinent to each project like tasks, users and the calendar.

System Architecture

The application makes use of the layered architecture style to separate the database, business logic, and the UI. Our packages are organized in the manner shown in the static view below:



As shown above, we have distinguished our packages by layers to separate the UI, Business Logic, and Data. Each layer does interact with the layer below it; the UI retrieves information from the controllers, which are updated real time using listeners, to display data and sends notifications using the contents of the services package. Both services and controllers use the data objects, which are a representation of the data stored in Firebase. The Utilities package exists as an external component that is used by each layer, as it contains tools to assist all layers. Utilities includes eums, listeners, and methods for interacting with the database.

All this allows STREAM to maintain up-to-date data on objects we store in the database, including users, projects, and data corresponding to each project. The objects in the Data Layer are a manifestation of information stored in the database. These objects are constantly fetched by the controllers to ensure that the objects are up to date. Finally, the UI displays the up-to-date information fetched by the controllers.

Android Studio and Android

STREAM is built on one of the most popular mobile platforms, Android. It was chosen for several reasons:

1. **Familiarity with Java and Android**: Everyone on the team is familiar with Java, which is used to develop Android applications. Additionally, some team members have experience working with Android so that the learning technologies portion of the project can be reduced to focus on implementation.
2. **Android is open source:** Android being open source and popular among the community means that there is more support for development. Our team can ask for help through stackoverflow and use pre-built libraries built by other developers. The community support is a huge reason for choosing Android since it can reduce our team's work time by simply using a library instead of trying to implement all features ourselves.
3. **Android provides a distinction between frontend and backend:** Android does not enforce the MVC pattern but it does divide the software application to several parts. The user interface is is implemented with XML files and the business logic of the application is implemented in Java files. Having this clear distinction between frontend and backend allows our team to divide ourselves to work on different areas of the application without interfering with other team members. This process also allows us to merge our code more easily to complete the project even faster.
4. **Accessibility:** Since we are developing a mobile application, it is imperative that our team creates the application for a platform that has the most users. The Android operating system is running on over 80% of the smartphones in the market; therefore, our team decided to build the application on Android hoping that our application can reach more users.

Firebase

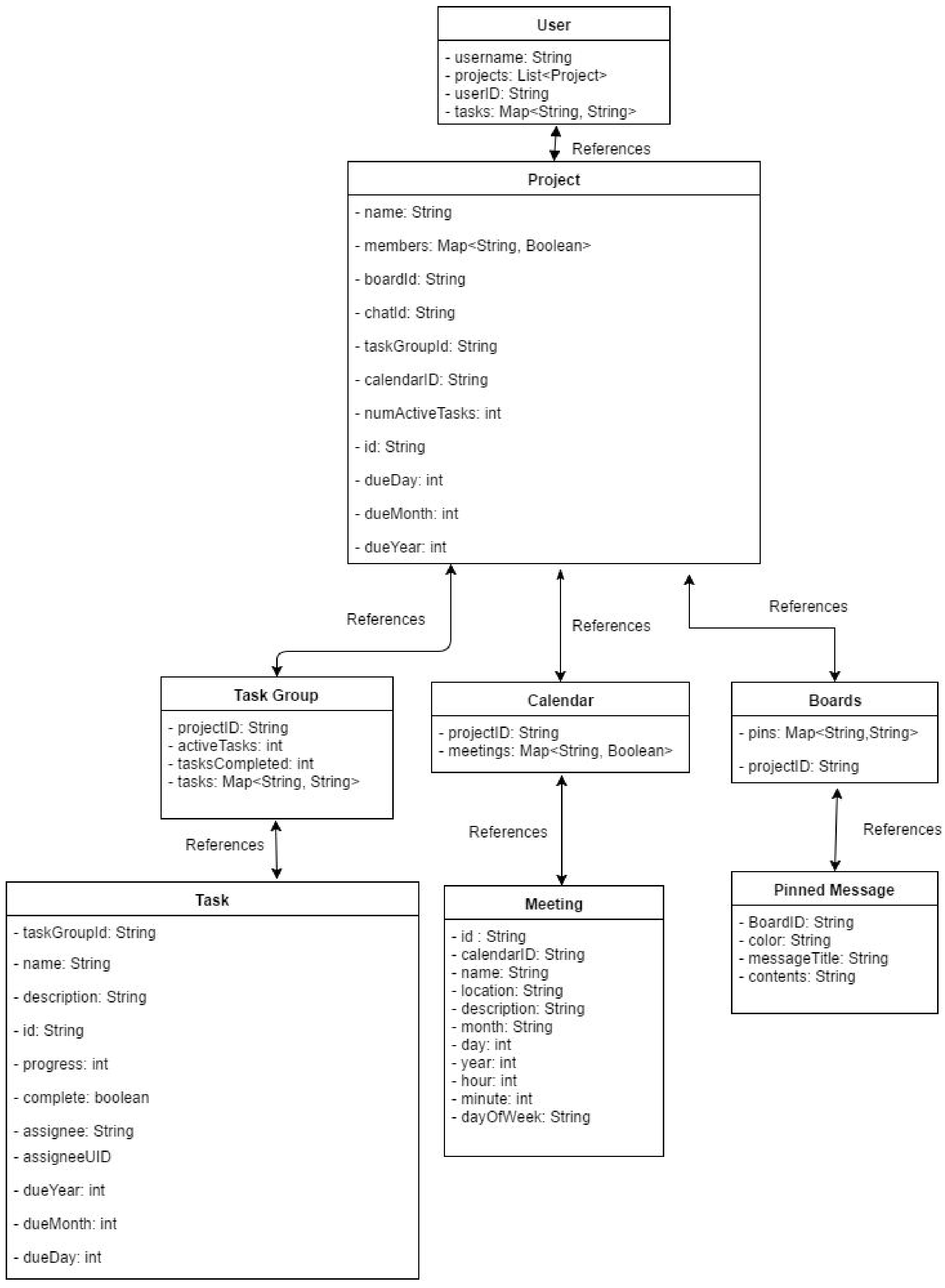
Firebase is a cloud application platform that abstracts read/write functions, user authentication, and file storage making it an excellent choice for our project; for our project, we need to identify users, write and fetch information to and from the database, and also store uploaded files to a Board functionality we planned. The following qualities have made Firebase the desired choice for our project:

1. **Built-in support for Firebase** in Android Studio, Greatly simplifying the setup process.
2. **Extensive documentation** for integrating Firebase. Hundreds of tutorials and ready-made examples for features we intend to include in our app such as authentication and chat. All available on the Firebase website.
3. **Simple NoSQL database**. Unlike relational databases, Firebase NoSQL databases are object oriented, making data storage flexible and easy to visualize.

4. **Cross-platform support**. Firebase can be integrated into any Android, iOS or Web application. While we only intend to build Stream for Android, Using Firebase will make any future migrations to other platforms significantly easier.
5. **Secure Login.** Firebase allows user accounts to be created and hide sensitive information on users, such as passwords. Identification of users is changed from login information to a unique User Token which means that users logging into Stream do not have to worry about how secure passwords they provide are. Neither developers or Stream nor other users will be able to easily access user accounts.

DATA

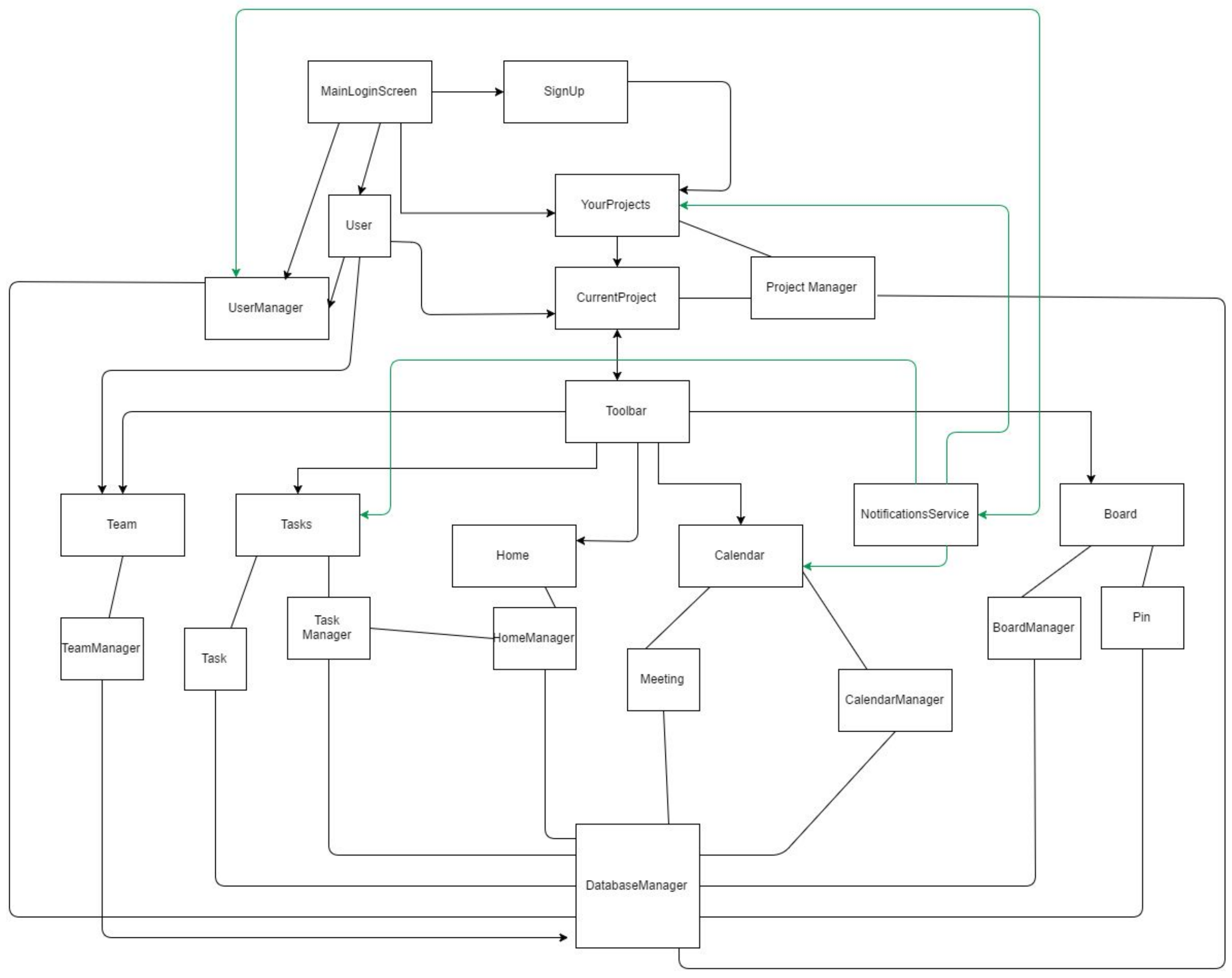
The persistent data of our application is mainly stored on Firebase. As Firebase is a NoSQL database, any data structures and features of our application are represented as objects in the database. We have several types of objects to implement functionality for Users, Projects, Tasks, Chat, and a Board. For users, we simple need a user object to represent each application user. Projects will be modeled as a single project object that will then contain a means to retrieve a task group, chat group, and board. The task group encapsulates all task objects of the project, while the chat group will hold all the channels, which contain messages from members of the team. The board will list all files and messages that are pinned in the project to be quickly accessed.


User
- username: String
- projects: List<Project>
- userID: String
- tasks: Map<String, String>
References
Project
- name: String
- members: Map<String, Boolean>
- boardId: String
- chatId: String
- taskGroupId: String
- calendarID: String
- numActiveTasks: int
- id: String
- dueDay: int
- dueMonth: int
- dueYear: int
References
References
References
Task Group
- projectID: String
- activeTasks: int
- tasksCompleted: int
- tasks: Map<String, String>
Calendar
- projectID: String
- meetings: Map<String, Boolean>
Boards
- pins: Map<String,String>
- projectID: String
References
References
References
Task
- taskGroupId: String
- name: String
- description: String
- id: String
- progress: int
- complete: boolean
- assignee: String
- assigneeUID
- dueYear: int
- dueMonth: int
- dueDay: int
Meeting
- id : String
- calendarID: String
- name: String
- location: String
- description: String
- month: String
- day: int
- year: int
- hour: int
- minute: int
- dayOfWeek: String
Pinned Message
- BoardID: String
- color: String
- messageTitle: String
- contents: String

DETAILED DESIGN

Detailed Design Diagram

The diagram below illustrates the overarching design for STREAM: how all of the functionalities are connected and the data structures needed for each.



After logging in or creating a new account, the user will be taken to their current projects. Upon choosing one of them, the user will have the option of seeing all of the tasks associated with that project, set up a meeting with their group members, or pin a quick note for everyone to see. The data for each of these features will be fetched and updated by each component's corresponding controllers, which are UserManager, ProjectManager, TaskManager, CalendarManager, and BoardManager. Each DataManager fetches objects from the Database with the help of the DatabaseManager class.

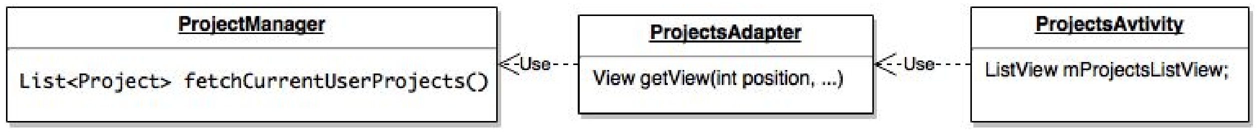
Design Patterns

Singleton Pattern

All of our DataManagers are only instantiated once, and can then be accessed throughout our project. This reduces the amount of data retrieval requests.

Adapter Pattern

Whenever we needed to populate a view, we sometimes used the adapter pattern. This provided a clean interface between two rather incompatible app modules, such as ProjectsActivity and ProjectManager.


ProjectManager
List<Project> fetchCurrentUserProjects()
Use
ProjectsAdapter
View getView(int position, ...)
Use
ProjectsAvtivity
ListView mProjectsListView;

Fulfillment of Use Cases

Our requirements include the following use cases: User Authentication, Project Creation, Task Creation, Adding Meetings to the Calendar, Posting to Board, Sending reminders, and Adding Users to an Existing Project.

The first, User Authentication, is fulfilled using the Main Login Screen. The user can access is directed to the Main Login Screen when they first enter the application and can choose to log in, which the allows UserManager to fetch the user object on Firebase corresponding to the application user. The sign up extension is covered with through the option to go to the sign up screen. Once users sign up for a project, they are logged in and brought to their projects.

To create a project and other components, users simply need to create objects from the corresponding screens. Users can create a project from the “Your Projects” screen, which implements the Create Projects use case. The ProjectsManager ensures the new project is made available to users. Users can then browse to the project’s tasks, board, and calendar using the Toolbar.

Tasks can be created in the “Tasks” screen. The TaskManager will ensure that the new task is available after creation through the UI. The same is true for creation of meetings in the Calendar and Posting pins to the Board.

Reminders to users are sent using NotificationService. NotificationService is linked to Users, Projects, Tasks, and Calendar. This thus allows notifications to be sent to the user with information on the relevant user and project from either the Tasks screen or the Calendar screen, fulfilling our use case to send reminders to users.

Finally, adding Users to an existing project is made possible by accessing the Team page from the Toolbar. The Team page allows modifications to members in the project through TeamManager.

Fulfillment of Non-Functional Requirements

Performance Requirements

Our JSON tree does not have that many layers so when data is retrieved, there are not many children to be loaded with it. Since there is less information to load, lag is reduced significantly. We also use the singleton pattern so data retrieval requests are reduced, consequently lowering the server load.

Safety Requirements

Users must be able to move through the app without encountering bugs that would cause the application to crash or lose important information. If bugs are encountered the app should be able to recover quickly without losing information, because all information is stored on the database.

Security Requirements

Users cannot modify the database directly, because our system handles all the database modifications instead. Therefore, user cannot modify or access other’s data, unless they use our own regulated functions. Furthermore, our database is private and can only be accessed by our team.

Scalability Requirements

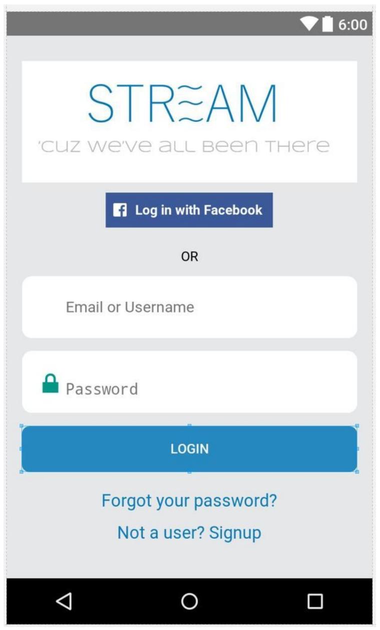
As mentioned above, our JSON tree does not have many layers. This gives further separation to what each fragment/activity loads. Therefore, even if a lot of information has to be loaded in one part of our app, it will not affect the rest of our app. Furthermore, if our database is reaching maximum capacity, we can easily upgrade our storage.

Software Quality Requirements

We used the adapter pattern, and singleton pattern as described above. This ensures that our code is easier to understand, because it follows a pattern that many are already familiar with. Furthermore, all of our functions have Javadocs and any ambiguous statements are commented to ensure all our functions are understood.

GUI

The user interface is being created in Android Studio, which uses a combination XML and its own drag and drop formating.



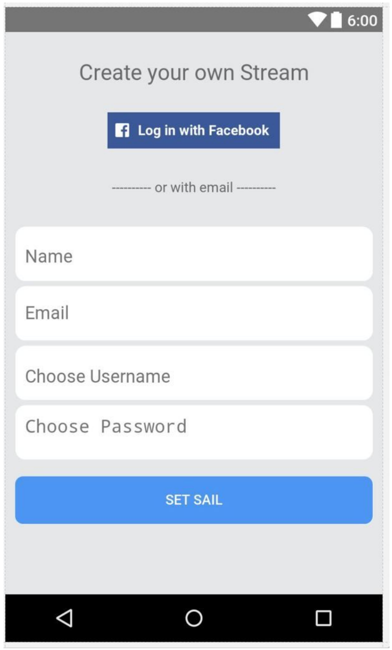
LOGIN

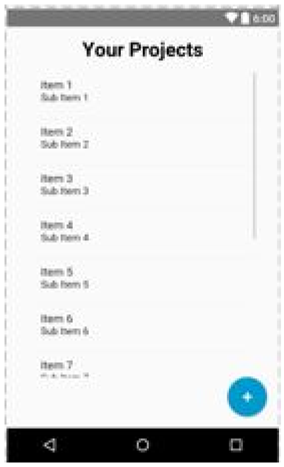
The user will have to ability to log in to STREAM using either Facebook or email. Facebook implements OAuth for login authentication. Both of these options will be immediately presented to the user when the app starts up. We also implemented a Remember me option to let the user use the app more easily once they have an account.

Forgot Password allow users to retrieve passwords for their accounts and Sign Up will take the user to a page where they create a new account.

SIGNUP

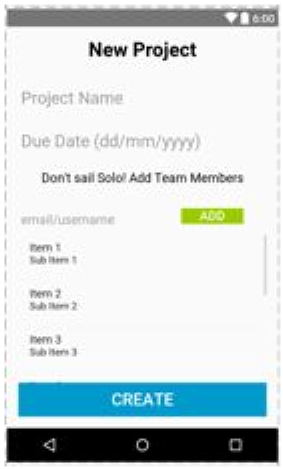
Users will "Create their own STREAM" by entering in their name and email and creating a username and password. Users will also have the ability to check their password using the eye icon.





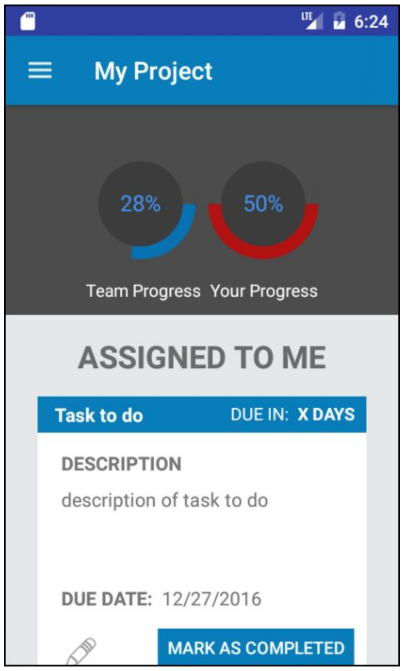
PROJECT LIST VIEW

Unless the user only has one project, after the user has either signed up or logged in they will be directed to the Projects List. This list will display all the projects the user currently has on the go.



CREATE A PROJECT

The user will have the ability to create a new project from the project list view. The user will be prompted to enter a project name and add other STREAM users either from facebook or with their STREAM username.



HOME PAGE

The Home Page will allow user to look at their project in a glance. Personal and Team Progress bars will be shown along with the users assigned tasks.

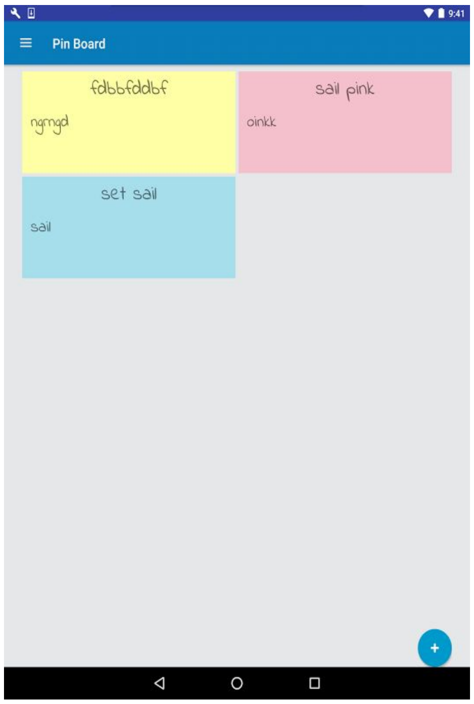
TASKS

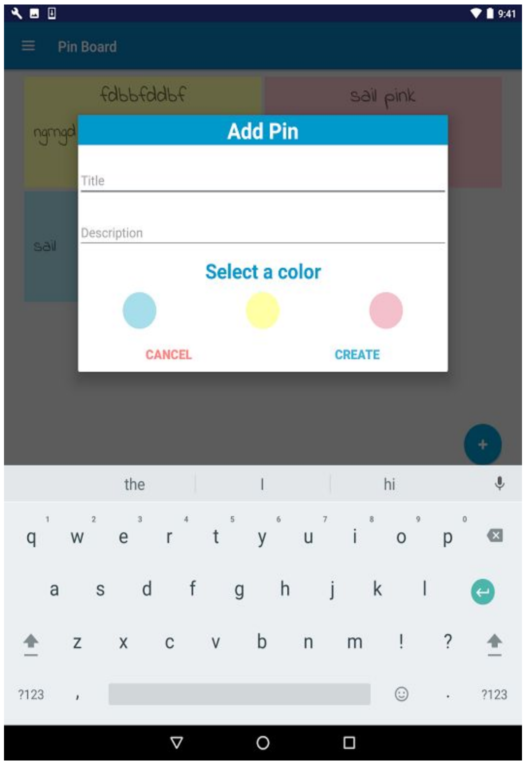
This interface allows the user to have a more direct view of the tasks related to their project. Tasks will be grouped by the user that the task is assigned to. Each user will be assigned a hex colour and the task button will be set to that colour. This will allow users to easily visualize what tasks are assigned to which users.

When the task is pressed on the main task page the user will be move to an expanded view of the task that contains all of the relevant information. From here the user will be able to delete the task, mark the task a complete, or send notifications to the tasks assignee. The desired final UI is shown below.

BOARD

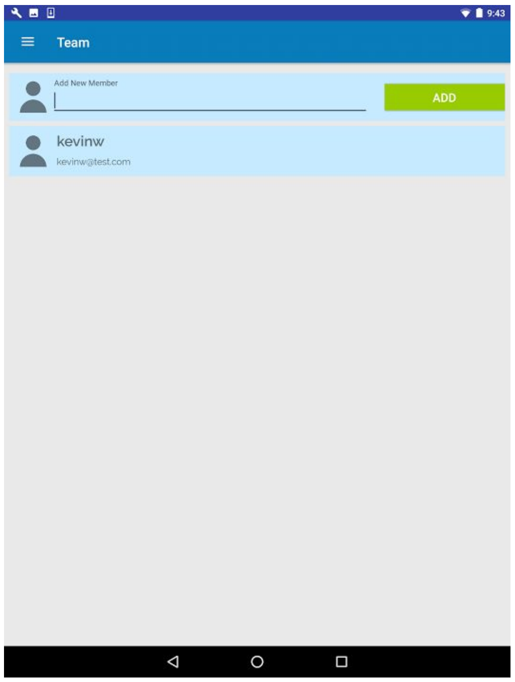
The below images display the how the notes on the board will appear on the screen with the current implementation.





TEAM PAGE

The Team page will allow users to see the current users of a project and add more members. Below is the current implementation of the Team page.



PROJECT SETTINGS

Project settings will give the user the ability to adjust the settings of their app. For example changing notifications, sound vibration and in-app notifications.

NOTIFICATIONS

One of the main features that consumers said they would most appreciate was notifications. STREAM will push banner notifications to the users phone to alert them of upcoming task due-dates, messages in chat, or review notes from other team members.

VALIDATION

STREAM is a project management software that will assist students in managing their group projects. Prior to starting STREAM, we interviewed approximately 20 students to determine which features particularly interest them and are a priority for our target market. Their feedback is recorded in the Customer Feedback document on our Github. Having worked on group projects during their education, most students expressed a high level of interest in being able to assign tasks and enforce deadlines with their team members. There was also a fair amount of interest in providing file storage system for the team.

Given the results from our potential users, we decided to put much emphasis on allowing groups to create and assign tasks. Notifications are used to help enforce deadlines, while a Board is available to all teams to store important messages and files.

In addition to determining our users' interests, we also validated the usability of our application with GUI prototypes. We created several mockups of our eventual UI, and reviewed the usability of screens within the team and with potential users. Using the feedback provided, we made some changes to our UI to ensure that the app would be easy to use. In particular, we decided on a navigation bar on the top left corner of the screen, as both our team and potential users felt that it was more intuitive than most other positions.

During the course of creating the app we have been validating potential design plan changes and UI with potential clients.

DEVELOPER SECTION

How to find source code

All source code is located in the directory app/src/main on GitHub:
https://github.com/omarbakker/Stream-app/tree/master/app/src/main/

How to check out source code and run project

1. Install Android Studio on computer
2. Set up Emulator or have an Android Device
3. Clone the repository from GitHub by: git clone https://github.com/omarbakker/Stream-app on Terminal/ Git Bash
4. Open the project on Android Studio and wait for Gradle to sync
5. Run the Android app on emulator or plug in Android Device and run app on device

How to run tests

1. Clone the repo from GitHub by: git clone https://github.com/omarbakker/Stream-app on Terminal/ Git Bash
2. Open the project on Android Studio and wait for Gradle to sync
3. Go to directory app/src/androidTest/java/com/test/stream/stream to see all Unit Tests
4. Open the Unit Test file e.g. ProjectsTest.java
5. Right click on the class name and click Run. For instance if you are trying to run ProjectsTest.java right click ProjectsTest and click "Run ProjectsTest"

Structure of source code directory

- app/src/main/assets folder includes all fonts used for the project
- AndroidManifest.xml contains settings for the project such as listing all Activities and Fragments as well as themes
- app/src/main/res folder contains all images used in the application. Images will be stored on drawable or mipmap folder
- app/src/main/res/layout folder contains all the xml files for the application. These xml files are inflated to display the UI of the application such as PinBoard, Tasks, Projects, etc
- app/src/main/java/com/test/stream/stream/Controllers contains all the Java files to interact with Firebase
- app/src/main/java/com/test/stream/stream/Objects contains all the object declarations of the project. For instance, PinMessage will have string properties to store title, subtitle and description. These objects will make it easier to store data into the database.
- app/src/main/java/com/test/stream/stream/UI contains all the activities of our application. These activities display all the UI such as buttons, lists, popup dialogs and many more so that the users can interact with the application
- app/src/main/java/com/test/stream/stream/UIFragments contains all the fragments for navigation when clicking on the specified button on the toolbar
- app/src/main/java/com/test/stream/stream/Utilities contains all the enums, callbacks and general database helper functions for the application

Design Patterns used

- We are using the Singleton design pattern for the database Managers (BoardManager, CalendaManager, DataManager, ProjectManager, TaskManager, UserManager) because there are a lot of dependencies with the data in the database. Since we want to be able to access objects while browsing through the application, we applied the Singleton design pattern to these classes.

DESIGN CHANGES AND RATIONALE

IOS and Web Application Implementation

During our meetings with Farshid, the topic of implementing our application on IOS and Web platforms was discussed. After the discussion, the team analyzed how much we had left to do on the Android application and decided to focus our efforts on create a better functioning Android application rather than multiple different applications that did not fully implement all of our use cases. Additionally, most of the team did not know technologies to create Web applications and IOS applications so the time to learn the technologies would set the team back even more.

PinBoard Changes
Some of the changes implemented on PinBoard was the elimination of the PinFiles. As a team, we decided that storing Files into Firebase would consume a lot of our time. Due to time constraints, we decided to eliminate PinFiles. Instead, we will focus on PinMessages allowing users to store a pin's title, subtitle and description. Moreover, the UI preview of the PinBoard under the GUI title of this report has been changed to reflect current implementation of Pinboard.

Tasks Changes
Tasks was originally planned to have a drop down function when a task was planned, but we decided a new full screen would better facilitate the functions we wanted to implement. We needed to allow the user to edit the task, delete the task, mark the task as complete, allow the user to send notifications and reminders to others, while displaying all of the tasks information. We decided a drop down would make all of these functionalities to compacted and would be a better graphic design implementation in its own page.

The original UI plan was to have the tasks separated by user, and having the user's name as a title for each section. We decided that having the users assigned colours and have the tasks be coloured based on their user was a simpler and more streamlined implementation.

Removed Chat
A chat function was created, however it was very basic and would have required major modifications in order to function in an effective way with the app and the database. We decided as a team to remove the chat functionality from the app, and instead spend the time on improving other, more important functionalities rather than spending the time re-working chat.

Added Managers to Most Data Objects
We created data managers were created for most objects to act as a controller for the UI since we wanted the ability to update project content in real time without cluttering the UI with business logic. The data controllers become responsible for maintaining up-to-date data and ensuring that the UI is updated when information on the database has changed. We decided that allowing real time editing will allow teams to more effectively collaborate with each other.

Added New Parameters to Data Objects
For several objects, we added an id parameter to store the Firebase id of the object. This is done so that we can more easily identify the object from the database, therefore more quickly updating the object. Some other parameters added include a task list within the user data object. We wanted users to be able to view their tasks for all projects from their home screen, so we decided that providing a means to associating tasks with users without a need for constant querying would help us achieve this.