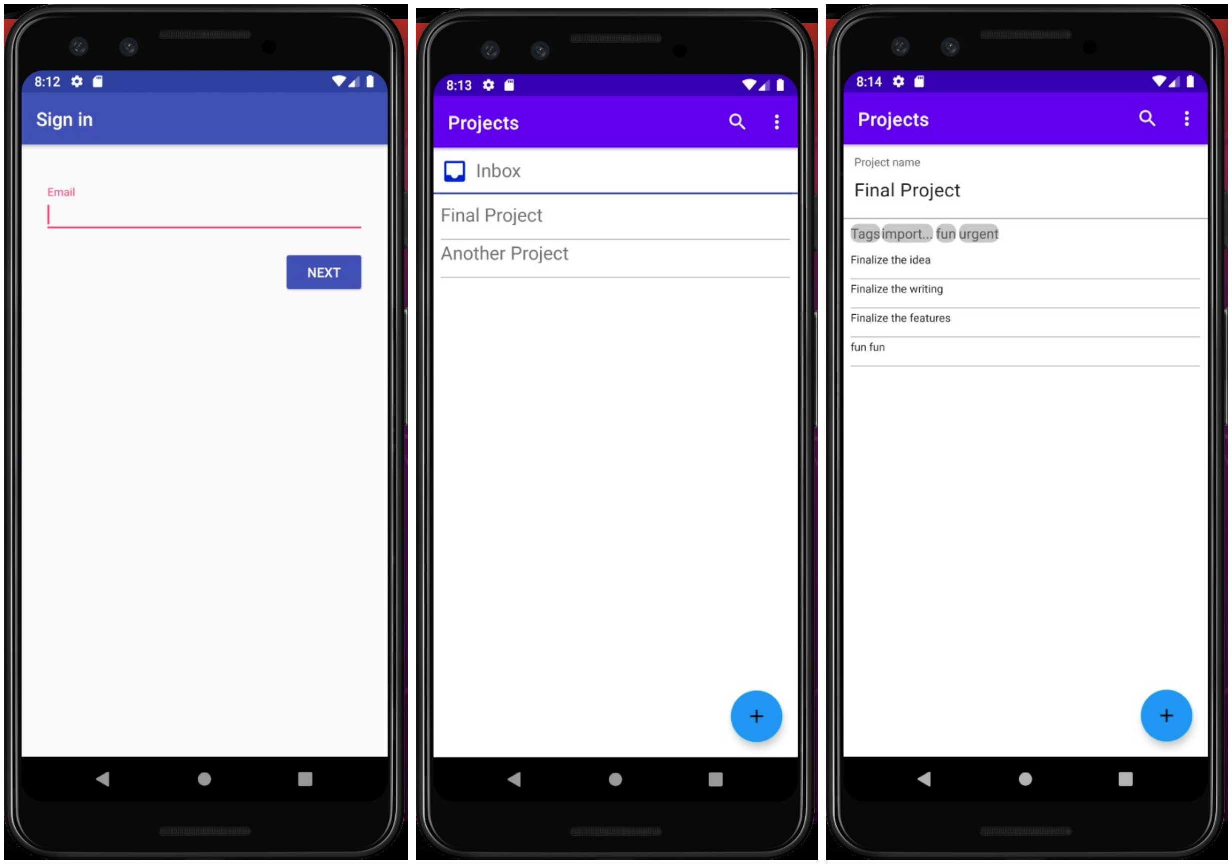
Listman

Manage all your lists

By Sriram Rajaraman

Introduction

Listman is a simple-to-use application that manages lists. Once the user is logged in, the user can create projects, assign the projects tags and create items underneath it.

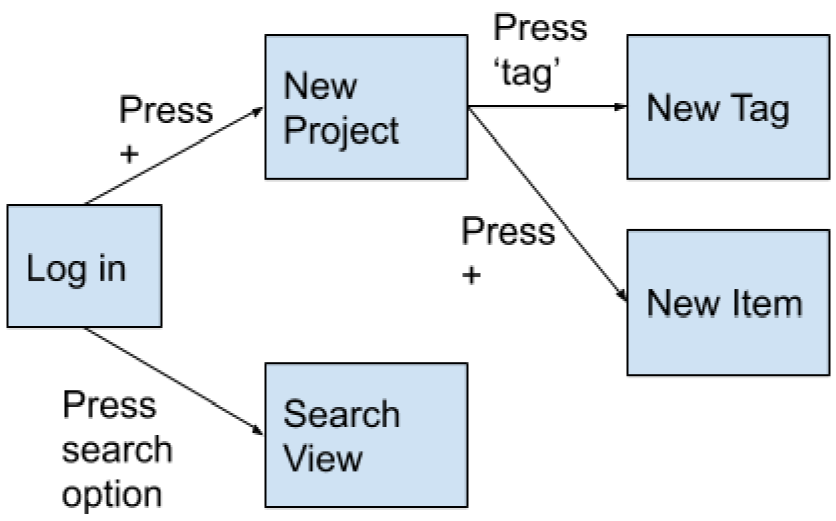


As the user enters the information, the data is automatically synced to the Firebase database. After entering the items, the user can swipe items from left to right to move it to another project. Also, the user can swipe from right to left to delete any items. By pressing back, the user can move the project view where they can dive into another project or create a project or jump into the inbox. The inbox is a default project where items can be added if they aren’t related to a project. Finally, there are action bar options that allow the user to search or log out.

Libraries/API Used

- Firebase Auth
- Firebase Firestore (explained below)
- Material UI for Floating Button (explained below)
- Actionbar (explained below)
- ItemTouchHelper for Gestures (explained below)
- Normal Android UI (explained below)
    - RecyclerView
    - TextView
    - EditText
    - Styles

User Flow



Note: the log out option is not shown here

Note: search view is accessible from any fragment but lines are not drawn for simplicity

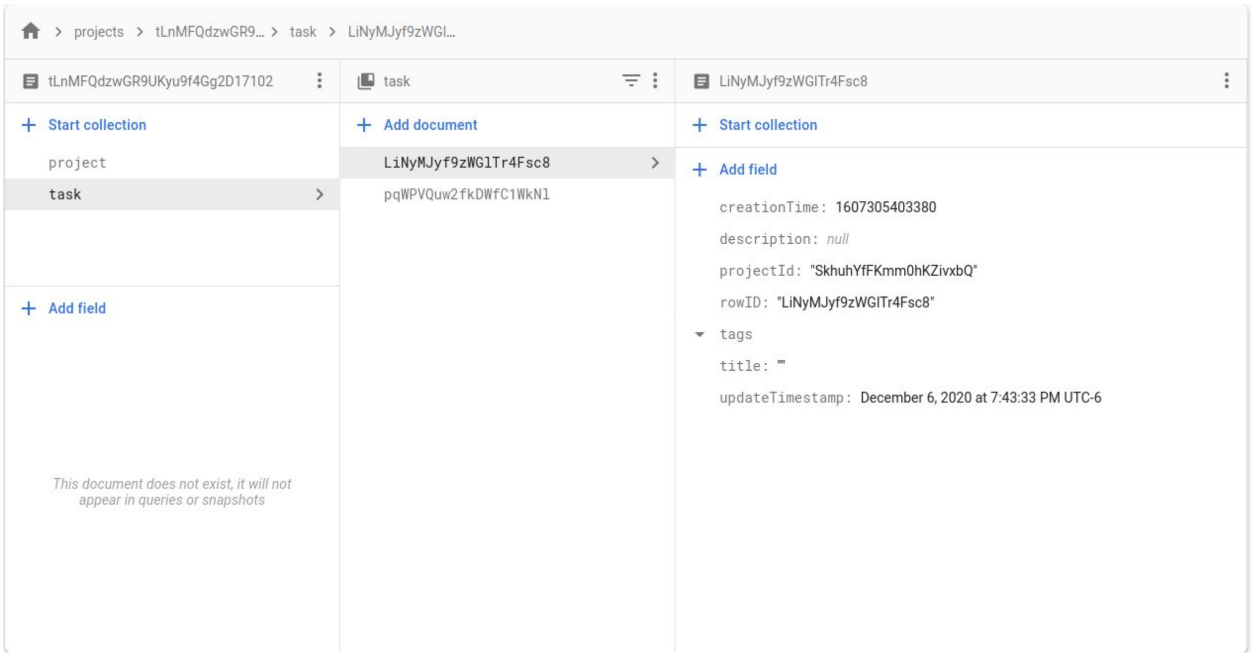
Database Schema

I narrowed down the schema design to two options: heavily-nested and semi-relational.

In the heavily-nested option, each user has a list of projects which have a list of tasks. In the semi-relational option, each user has a list of projects and a list of tasks. Each task has a projectId that corresponds to a project's rowId.

The benefit of the first option is that deleting a project will automatically delete the tasks because they are nested. However, the negative is that searching is more complex because of the traversal. The second option allows the application to easily pull all the tasks and projects and join locally. With a large list, this will become an issue but most users would not create more than 10,000+ items. The negative of the second option is that deletion of a project would require multiple calls. The first call will delete the project and then the next calls are needed to delete the tasks.

I selected option two as it was easier to search than option one.

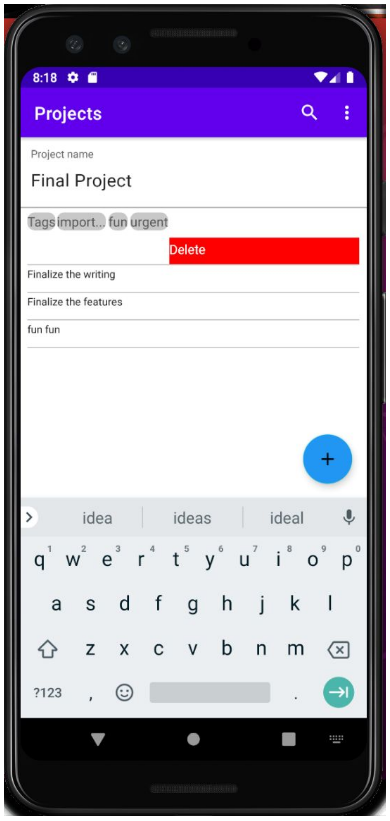


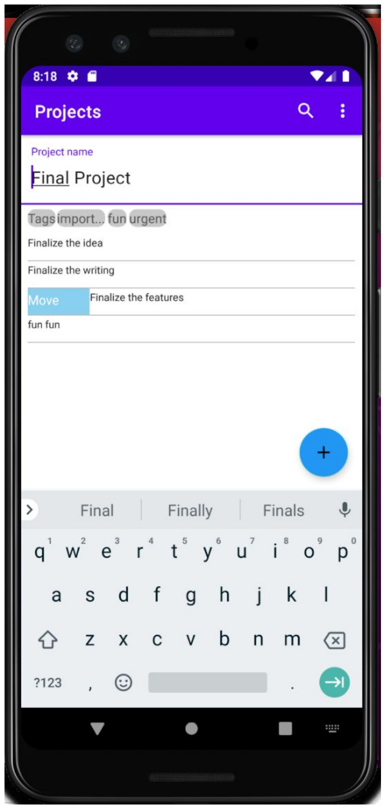
projects > tLnMFQdzwGR9... > task > LiNyMJyf9zWGl...
tLnMFQdzwGR9UKyu9f4Gg2D17102
Start collection
project
task
Add field
This document does not exist, it will not appear in queries or snapshots
task
Add document
LiNyMJyf9zWGlTr4Fsc8
pqWPVQuw2fkDWfC1WkNl
LiNyMJyf9zWGlTr4Fsc8
Start collection
Add field

```
creationTime: 1607305403380
description: null
projectId: "SkhuhYfFKmm0hKZivxbQ"
rowID: "LiNyMJyf9zWGlTr4Fsc8"
tags
title: ""
updateTimestamp: December 6, 2020 at 7:43:33 PM UTC-6
```

Gestures

There are two major but similar gestures in the application: swipe to delete and swipe to move. Both gestures required an implementation of the ItemTouchHelper where the device can detect the gesture's direction and the subsequent action. Implementing the logic to delete a task/project was not difficult. However, adding the red bar with the text "delete" was not easy. The gesture callback details the x and y coordinates of the movement which was used to calculate the boundaries of the red block. Using the same data, I was able to add the text "delete" so the user understands their action. The implementation for swipe to move was similar to swipe to delete. However, swiping to move opens up the AlertDialog containing projects. Once the user selects a project, the projectId of the task is updated and is reflected on the UI.



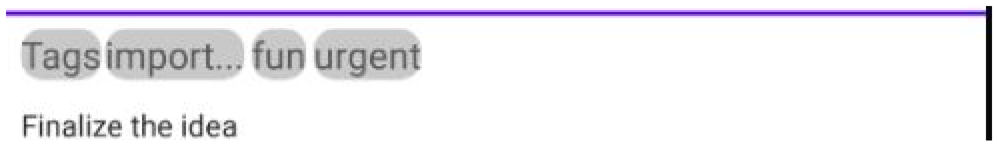


UI Decisions

Tags

Each project is allowed up to 10 tags describing it. When the user selects the “tag” button, an edittext field becomes visible and focused. Once the user hits enter, a tag is displayed on the screen and the user can continue to enter more tags. Also, the user can long press on a tag to delete it.

The major problem with freeform edittext is that the user can type off screen and the tag would not fit. In order to prevent this, each project is allowed up to 10 tags and if the tag’s character length is greater than 12, the UI would display the first 9 characters and append “...” to the end. The backend will contain the original tag string so searching is not impacted by this change.

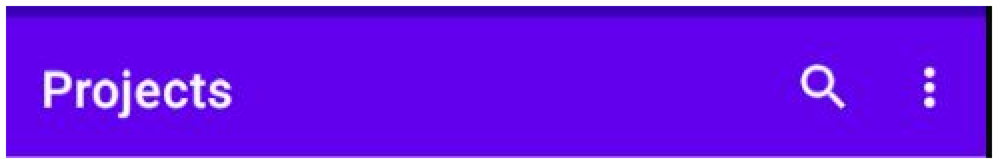


Floating Action Button (FAB)

Floating action buttons are prevalent design artifacts especially when there is a specific and frequent action. Both in the project list view and task list view use FABs to allow users to enter projects and enter task names respectively.

Action bar

Two features of the application is the ability to log out and search for results. The action bar allows these options to be always available to the user but also tucked away. Given the search feature should be used more often, the icon is visible for the users.

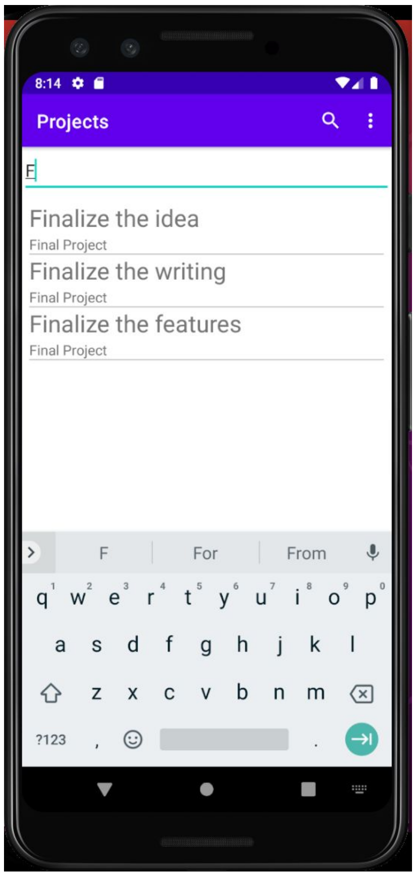


Inbox

Given that some tasks do not require a huge project, a default project is created called Inbox. Inbox is automatically synced to the firebase server and cannot be deleted. User can create a project called “Inbox” which can cause conflicts when interacting with the database. In order to solve this issue, a new boolean field called “isDefault” was added so it is easy to identify which project is the true inbox project.

Search

The search UI consists of an edittext and a recycler view. As the user enters their search term, the model searches all the projects’ tags and then searches the tasks. If it finds a tag that matches, then all the tasks under that project will show up. If it finds that a task’s title contains the search term, then only that task will appear in the results. Furthermore, for each matched task, the corresponding project is retrieved. See Database Schema about design decisions of relational schema.



Failed Features - War stories

Per-Task Tags

While adding tags to each project is nice, adding tags to items is useful. I created the layout to look exactly the same as the project's tags. However, having multiple edittext created an inconsistent experience. When typing, another edittext would claim focus and ruin the experience. I tried disabling the edittexts when not used but then the UI didn't seem lively. Rather than trying to fix the styling of disabled, I tabled the feature for later.

Android Wear

I wanted to create a companion app that showed a list of items from one project. Rather than the watch communicating with the phone, I decided that the phone will sync a token and the watch will make a firebase-related request using the token. In other words, the phone was only needed once. After coding the UI and functionality, the watch didn't sync with the phone. After hours of debugging, it turns out that Watch VM cannot connect with Phone VM and I needed a physical phone. So this feature ended up being cancelled as I don't have an android phone.

The code exists in a separate branch.