CENG-322 Deliverable 2

Team #1 - **Algaerithms Inc.**

Project name - Phytoplankton-based air purifiers

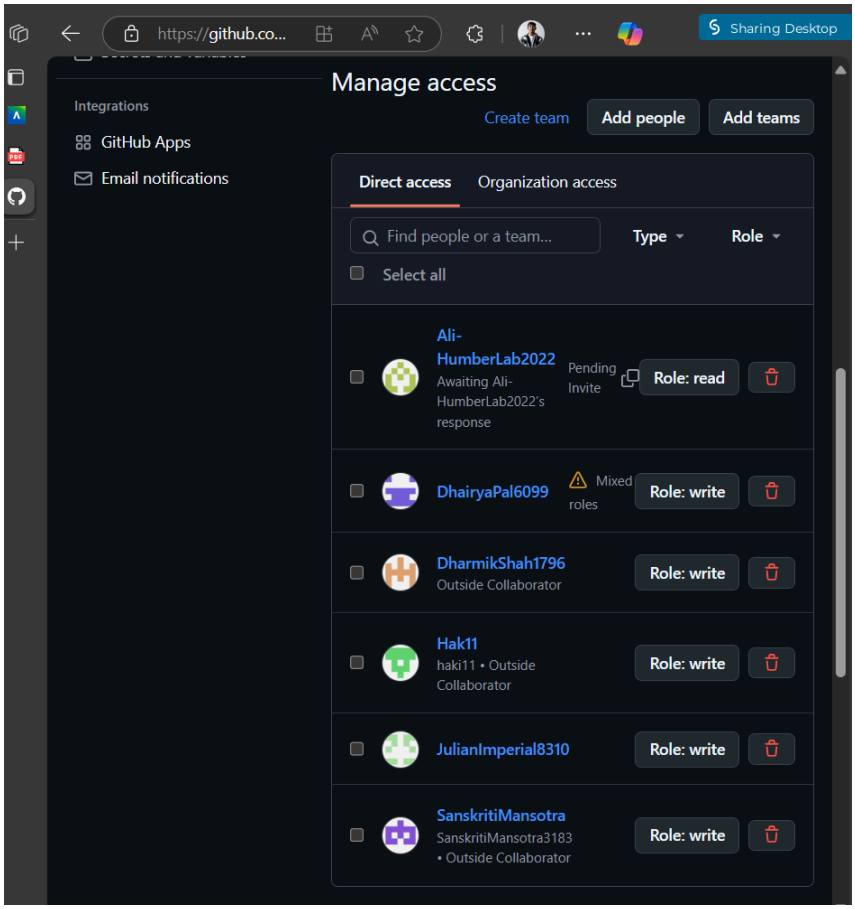
Name	Student ID	Github ID	Signature	Effort
Julian Imperial	N01638310	JulianImperial8310	J.I	100%
Dhairya Pal	N01576099	DhairyaPal6099	D.P	100%
Sanskriti Mansotra	N01523183	SanskritiMansotra3183	S.M	100%
Dharmik Shah	N01581796	DharmikShah1796	D.S	100%

Table of contents -

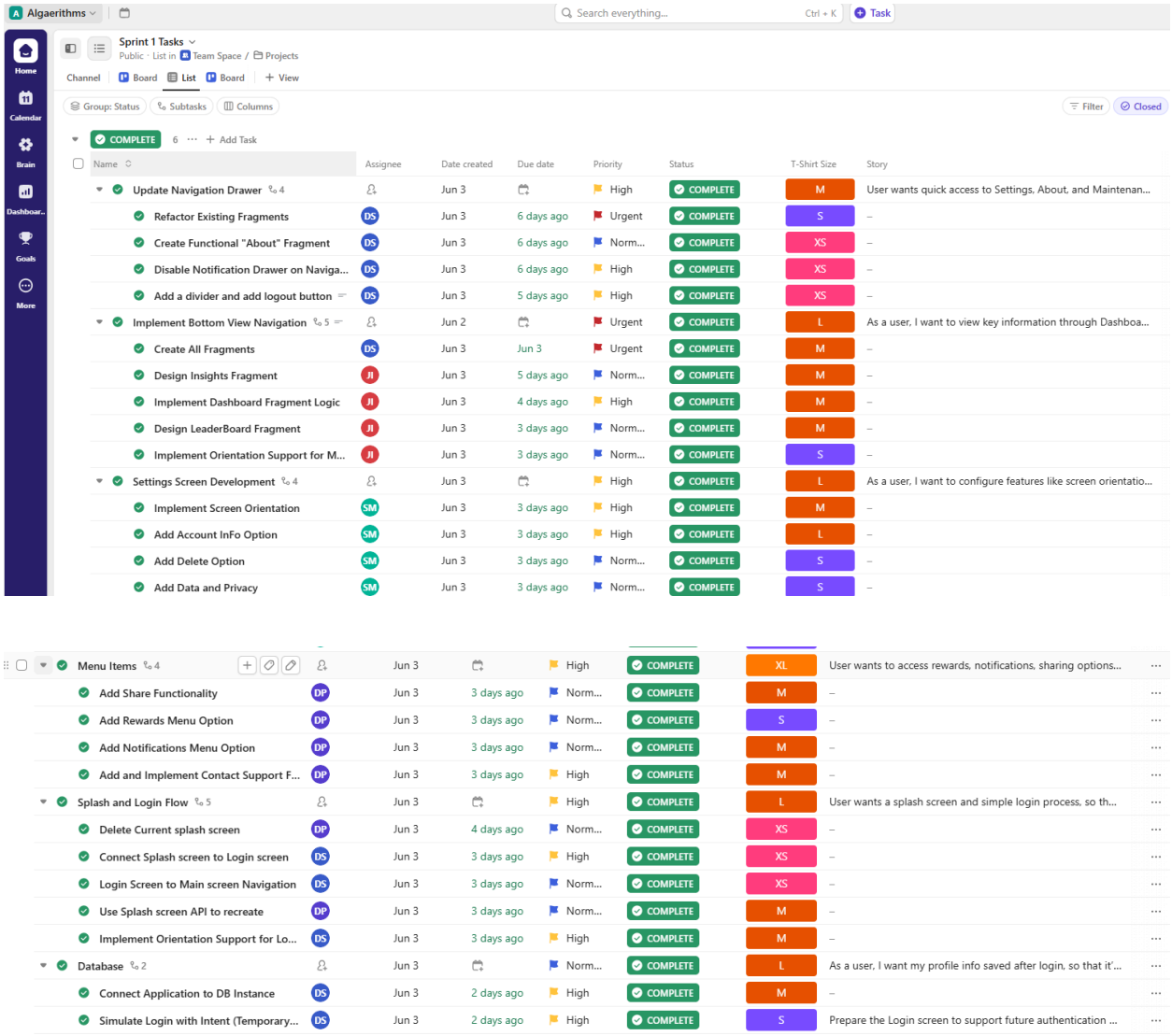
1. **Project Scope and goals that are targeted in this course.**
   The goal of our project is to create a smart, algae-based air purification system that helps improve indoor air quality in a sustainable and self-sufficient way. We are combining phytoplankton, which naturally convert CO_2 into oxygen, with real-time monitoring through an Android app. The final vision is to give users a visually engaging and interactive way to monitor air quality, while also promoting eco-conscious behavior. The system not only improves indoor environments by reducing pollution and increasing oxygen levels, but also raises awareness about climate-friendly technologies. Through features like gamified achievements, alerts, and live data tracking, we aim to make clean air more accessible, measurable, and meaningful for everyday users.

2. **GitHub Repo link and strategy. All members must contribute to the repo. Explain the github strategy.**
   https://github.com/Algaerithms-Inc/PhytoplanktonAirSystems
   Tasks distribution - Every sprint, tasks are divided among the team initially based on everybody's strengths and roles in the team. Tasks can be redistributed should a team member feel the need to request it.
   Branches - Team members create a branch (clone from master) for every feature they are assigned. Multiple tasks can be worked on on a feature branch. How many tasks are worked on in a branch is left to the team member's discretion. The branch naming convention we're following is teamMemberName_featureName.
   Merging to master - After the team member finishes working on their branch, and make sure they meet the initial DoD criteria (no accidental git tracked files committed, strings.xml resolved, etc.), they merge master into their own branch (so all the changes in master are added to this pull request) and then create a pull request and ask for a formal review from team member(s). Only Dhairya merges the pull request. If Dhairya isn't available, he asks someone else to volunteer for the time he's unavailable. This helps the team work in an orderly fashion.
   Before creating a pull request we make sure there's no accidental git tracked files being pushed to the repo and we work together in case a team member needs help understanding how different git commands work.

3. **Screenshot showing you have invited your hardware prof to the repo.**



4. **Minimum of 5 stories and each story must be split to a minimum of 4 tasks.**
5. **Take a screenshot showing clearly the stories and breakdown of tasks with owner, status, start/end date, size, and priority.**

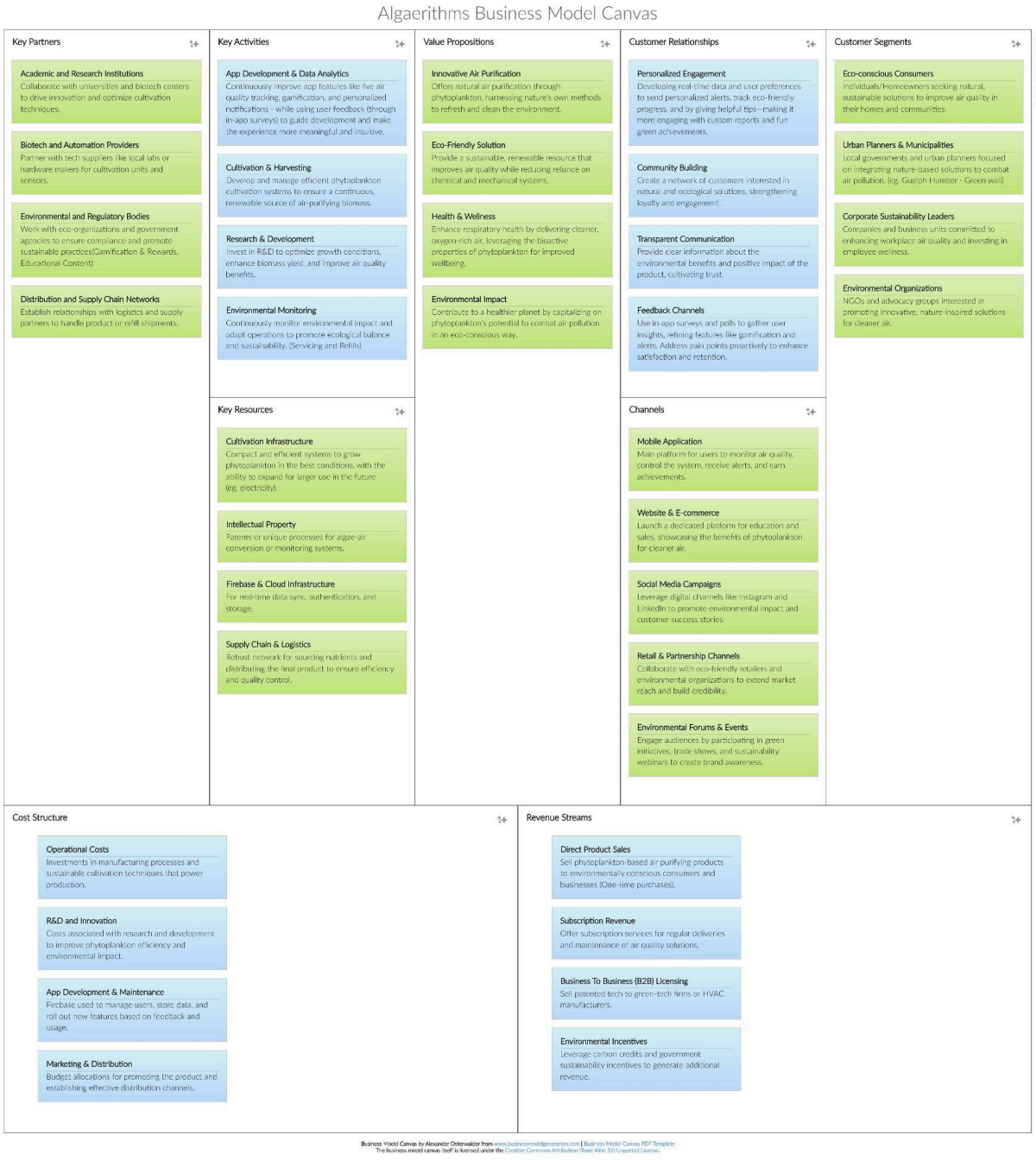


6. **Tasks which are marked done, explain how you met the DoD criteria.**
   All the tasks and subtasks were usually part of one feature of the app and were included in one branch. When the team member finishes working on them, they make sure they do not commit and push any unnecessary files/folders to their remote branch like .gradle/, build/, .idea/, etc. They also make sure to merge master into their branch before they create a pull request so their branch has all the changes made in master after they cloned it to make their branch. At this point, they also usually have to resolve any strings.xml, which can usually be done by adding both sides' (master branch and feature branch) strings to the final strings.xml. Now they are ready to create a pull request and when they do that, they add a team member (Dhairya) as the reviewer of the branch. If Dhairya is unavailable, someone else from the team volunteers to look at the branch, review it, and merge it to master. Changes are requested formally should there be a

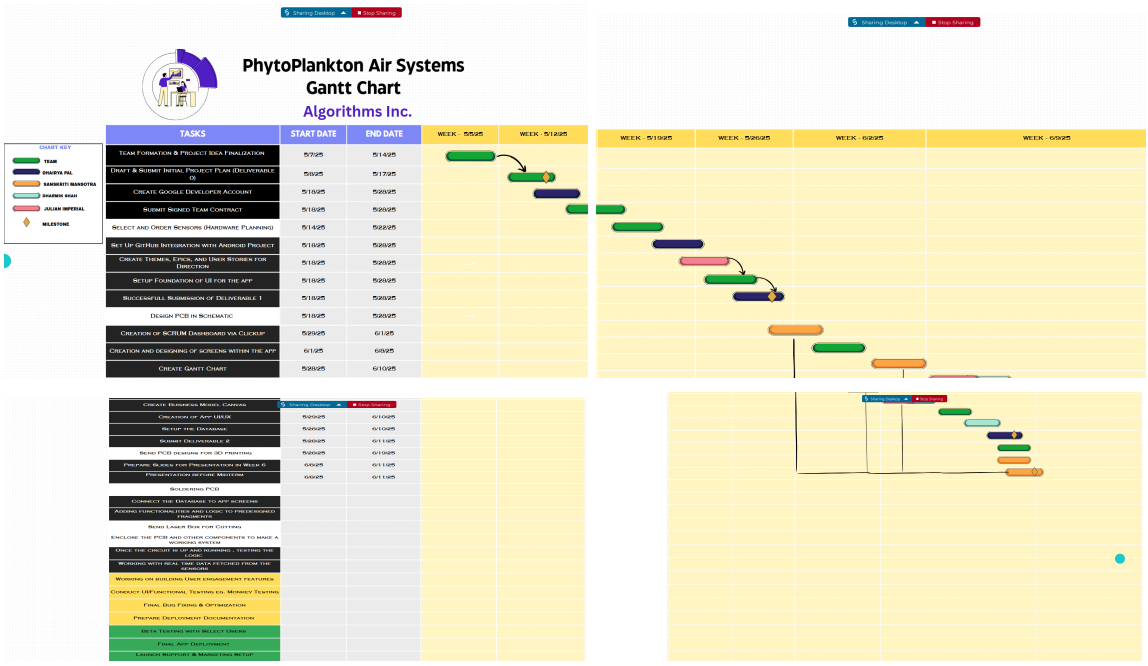
need, and after all the changes have been implemented, the branch is successfully merged, and the task is done.

7. **List of items you included in the DoD.**
    - All the hardcoded strings go into strings.xml and are translated.
    - No accidental git tracked files must be pushed to the branch requesting a pull from master.
    - All pull requests must ask for review from team members, get feedback incorporated and then only be merged to master.
    - For the task to be complete the task should be merged with master without conflicts (except strings.xml) and without breaking anything.

8. Business Model Canvas, all the 9 fields are clearly shown and explained.



9. **Multiple screenshots of the Gantt charts showing the details.**



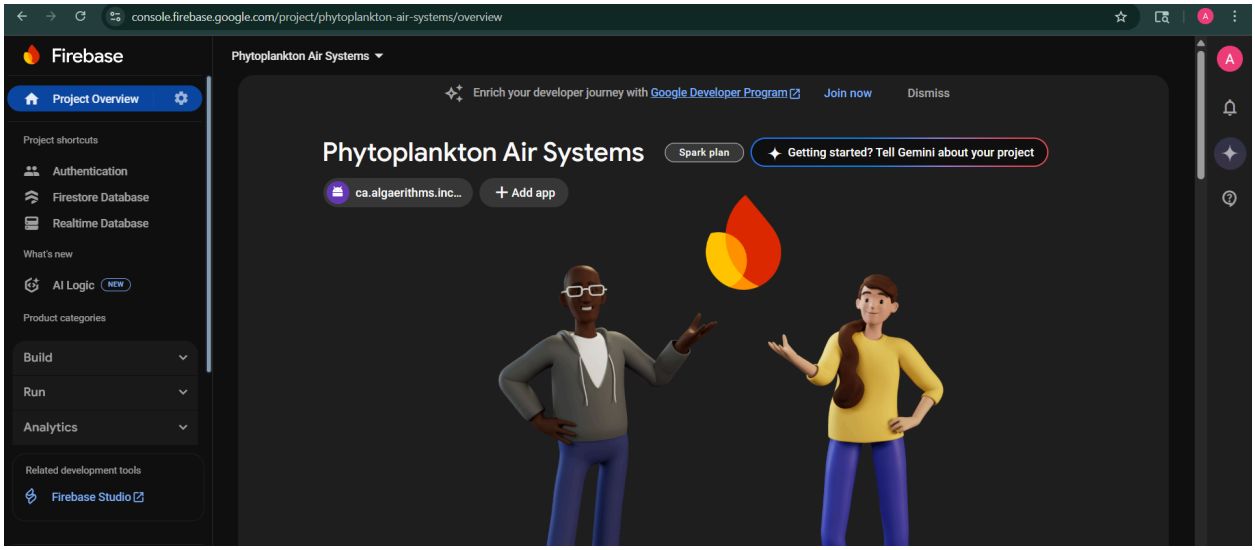
Canva link to view the Gantt chart:
https://www.canva.com/design/DAGp7wKGJI4/GT_IfpERySe2x3GXvtIkMA/edit?utm_content=DAGp7wKGJI4&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton

10. **Screenshot showing you have created the DB on the cloud, with some details and type of DB used.**

We are using Firebase Realtime Database as our cloud-hosted database for the phytoplankton app. Firebase was chosen for its real-time data syncing, easy integration with Firebase Authentication, and strong support for mobile apps.

Below is a screenshot showing that the database has been created on the Firebase Console, along with its configuration details:

- Type of DB: NoSQL (Firebase Realtime Database)
- Location: [region, e.g., "us-central1"]
- Usage: Storing user profiles, including name, email, phone number, and other app-specific data
- Linked services: Firebase Authentication (for Google and custom email/password login)



11. **Explain how you are planning to use the DB.**
    We plan to use the database primarily for user authentication and profile management. Users will be able to sign in using either their Google account or, if they choose to create an account manually, with their email and a custom password. To enhance security, we will also implement two-step verification during login.

    Additionally, we will store each user's basic information—such as their name, email, and contact details—in the database. This data will help us personalize their experience and manage their profiles effectively within the phytoplankton app.

12. **Coding work progress since deliverable 1. What additional features/functionality added since deliverable 1.**
    We added fragments as screens for our app so when the bottom navigation drawer options are clicked, you move to different screens - Dashboard, Insights, and Leaderboard. We renamed out navigation drawer items as well from Home and Gallery to Home, About, Settings, and Logout. We made our bottom navigation drawer to only appear in the home fragment and not the other fragments because of our use case (Those fragments do not seem appropriate in other navigation drawer fragments). Settings fragment has a few settings options now - Lock screen to portrait, Account Info (where the user can add information for their profile including name, email address, phone number, and profile photo), Delete account, and Data and privacy (where the user will be able to choose if they'd like to let the app use all the algae health data it has gathered since the user installed the product to show the trend in algae health, and whether they'd like the app to keep their preferences saved on their account, etc.).
    We also added menu items - Contact Support (which if the user clicks on, it takes them to their default phone app and dials the company's support number in), Share (This is not functional yet - but when the user clicks on this, it lets the user share their algae health dashboard to others using social media apps), Achievements (This is not

functional yet - The user can see their badges on a different screen), and Notifications (This is not functional yet - This will move to a different screen where the user can see their notification history). The notifications will usually be about any alerts to algae health, end-of-day updates on algae health, and weekly/monthly reports on their product health, etc.

13. **Document what Design Principle you used of which at least 2 is used, copy the code and comment on it. (Examples: Don't Repeat yourself, generalization, Single Responsibility Principle, Open Closed Principles Abstract/Interface classes. Overall SOLID)**

DashboardFragment.java

```
20  public class DashboardFragment extends Fragment {
        2 usages
24      private ToggleButton toggleButton;
25
        no usages
26      private FragmentDashboardBinding binding;
27
28      @Override
29      public View onCreateView(LayoutInflater inflater, ViewGroup container,
30                               Bundle savedInstanceState) {
31          // Inflate the layout for this fragment
32          View view = inflater.inflate(R.layout.fragment_dashboard, container, attachToRoot: false);
33
34          dashboardView = view.findViewById(R.id.dashboard_view);
35          graphView = view.findViewById(R.id.graph_view);
36          toggleButton = view.findViewById(R.id.toggle_view_button);
37
38          toggleButton.setOnCheckedChangeListener((buttonView, isChecked) -> {
39              if(isChecked){
40                  dashboardView.setVisibility(View.GONE);
41                  graphView.setVisibility(View.VISIBLE);
42              } else {
43                  dashboardView.setVisibility(View.VISIBLE);
44                  graphView.setVisibility(View.GONE);
45              }
46          });
47
48          return view;
49      }
50  }
```

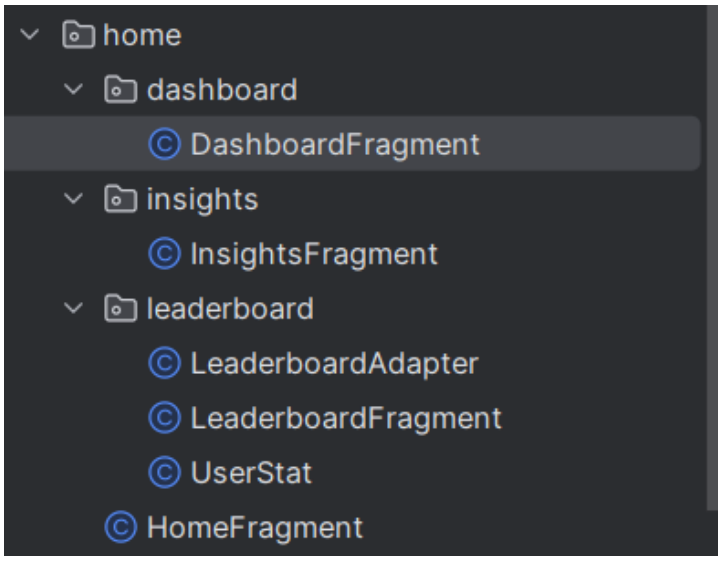
The Code above is an application of DRY, Don't Repeat yourself. Handles the logic of switching between views in a single lambda expression, otherwise we would have to use Abstract classes.

```
View view = inflater.inflate(R.layout.fragment_dashboard, container, false);

dashboardView = view.findViewById(R.id.dashboard_view);
graphView = view.findViewById(R.id.graph_view);
toggleButton = view.findViewById(R.id.toggle_view_button);
```

This code Above is also another example of DRY, by inflating the layout once, then reusing the view object rather than this which does not follow the DRY Principle, because its the creation of multiple instances rather than one which is what view does:

```
dashboardView = inflater.inflate(...).findViewById(R.id.dashboard_view);
graphView = inflater.inflate(...).findViewById(R.id.graph_view);
toggleButton = inflater.inflate(...).findViewById(R.id.toggle_view_button);
```



The screenshot above of the Android Studio Application folder structure is an example of SRP, Single responsibility principle. This is because each class sticks to its main functionality without having multiple functionalities for one class that should otherwise be in more than one class, such as the Leaderboard subfolder is showcasing.

Daily Standup Meeting :

Date	Team Member	Yesterday's Work	Today's Plan	Blockers
2025-06-05	Julian	Created a branch for land layout for main screens.	Continue to work on branch and start with designing main screens	Had to wait till the end of the day for Dharmik to finish up bottom navigation, and for it to be merged.
	Dhairya	Merged, the pull requests from Dharmik regarding the bottom and drawer navigation.	Started Menu Items branch (No Functionality yet)	Receiving feedback from team on what menu items to create and their relevance to the project
	Sanskriti	Editing of our click up dashboard, assigning tasks as Scrum Master	Ensuring that each team member knows how to use the scrum dashboard tool in this case clickup	Had to wait for the Navigation drawer to be cleaned up from Dharmik, and for it to be merged as well.
	Dharmik	Finished up, the bottom navigation and the cleanup of drawer navigation.	Checking the best database to use, for our application	Team consideration and editing of Fragment models to be used (ViewModel etc.) because of passing over data
2025-06-06	Julian	Decided on the theme and color styles we want for the application bottom Navigation screens	Started designing the Dashboard fragment, and also implementing custom fonts	Learning how to implement the custom font

	Dhairya	Creation of Menu Item branch, and figured out the menu items to be implemented	Created Support, Achievement, Share menu items with no functionality	Figuring out how to implement a menu item that shows both the icon and text for that menu item
	Sanskriti	Teaching members how to work with Click Up Scrum dashboard	Assigning T-shirt sizing to each and every task by asking each team member	Finding and deciding a collective duration for each T-shirt size
	Dharmik	Cleaned up Bottom and Drawer Navigation issues.	Creation of Login Screen, with no functionality.	Figuring out what theme the Login screen should be as it is the first thing a user will see, after the Splash Screen.
2025-06-07	Julian	Finishing up with the Dashboard fragment and using the custom Font.	Started working on the Insights and Leaderboard Fragment.	Figuring out the best way to implement a leaderboard screen.
	Dhairya	Creation of Menu Items, with no functionality only design	Editing existing branch for Splash to be used with the API	Ensuring that the new Splash Screen, is clear and eye-catching
	Sanskriti	Assigning the T-shirt sizing per task, and collectively coming to an agreement for the duration of t-shirt sizing	Created a branch for settings screen, and designed a basic UI.	Implementing the lock orientation feature to the Settings screen.
	Dharmik	The creation of the Login Screen with no functionality	Giving the login screen a theme that is suitable with the App and the Creation of our Firebase DB	Collective decision on what the theme of the login screen should be.