

CENG-322 Deliverable 5

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%

Brief description of the project - A smart and eco-friendly air purification system that uses phytoplankton to naturally convert CO₂ into oxygen. The system connects with an Android app for real-time monitoring, automated alerts, and sustainability-focused feedback.

What issue your product will solve -

Our system addresses poor indoor air quality caused by high CO₂ levels and low oxygen circulation. Unlike passive air purifiers, it uses live algae to naturally photosynthesize and purify the air while providing a clean energy cycle and an interactive dashboard for education and engagement.

Github repo link - <https://github.com/Algaerithms-Inc/PhytoplanktonAirSystems>

List number of commits for each member for this sprint only. Create a table:

Team member (Commits)	Julian Imperial	Dhairya Pal	Sanskriti Mansotra	Dharmik Shah
July 25	0	0	1	1
July 26	0	5	0	2
July 27	0	13	0	0
July 28	0	1	0	0
July 29	0	2	0	2
July 30	0	1	0	1
July 31	0	3	1	0

August 1	6	10	2	3
August 2	1	10	6	5
August 3	12	0	5	11
August 4	1	10	12	1
August 5	6	4	2	1
August 6	1	4	0	2

19. Describe 3 tasks from the Scrum dashboard that are related to addressing technical debt.

1. Mobile Theme Alignment

Task: Organized color folder, aligned light/dark mode, and ensured consistent theming across fragments.

Technical Debt Addressed:

Earlier, theme colors were inconsistently applied across the app. This task centralized the color system, reducing duplication and ensuring visual consistency, which improves long-term maintainability and scalability.

2. Dashboard Fully Functional

Task: Implemented live sensor data display, AQI calculation, database connectivity, and dynamic notifications.

Technical Debt Addressed:

Initially, the Dashboard screen was UI-only with no real functionality. This task completed the intended features and connected the UI to real data, removing placeholder components and making it production-ready.

3. Contact Support UI

Task: Designed and implemented a complete Contact Support screen with actionable call and email options using CardView.

Technical Debt Addressed:

The Contact Support screen was previously incomplete. This task finalized the screen with a clean, functional layout, ensuring users can now easily reach out — a critical usability fix that closes a long-standing gap.

20. Describe in detail, the work that has been completed by each team member in this sprint only. -

Julian - Junit testing of the Leaderboard fragment, by making 10 test cases and making sure that they all pass, and are meaningful to test. Completing the functionality of the Dashboard fragment, intended to inform users about the metrics their air system is producing. Aligning the overall bottom navigation screens to better match one another.

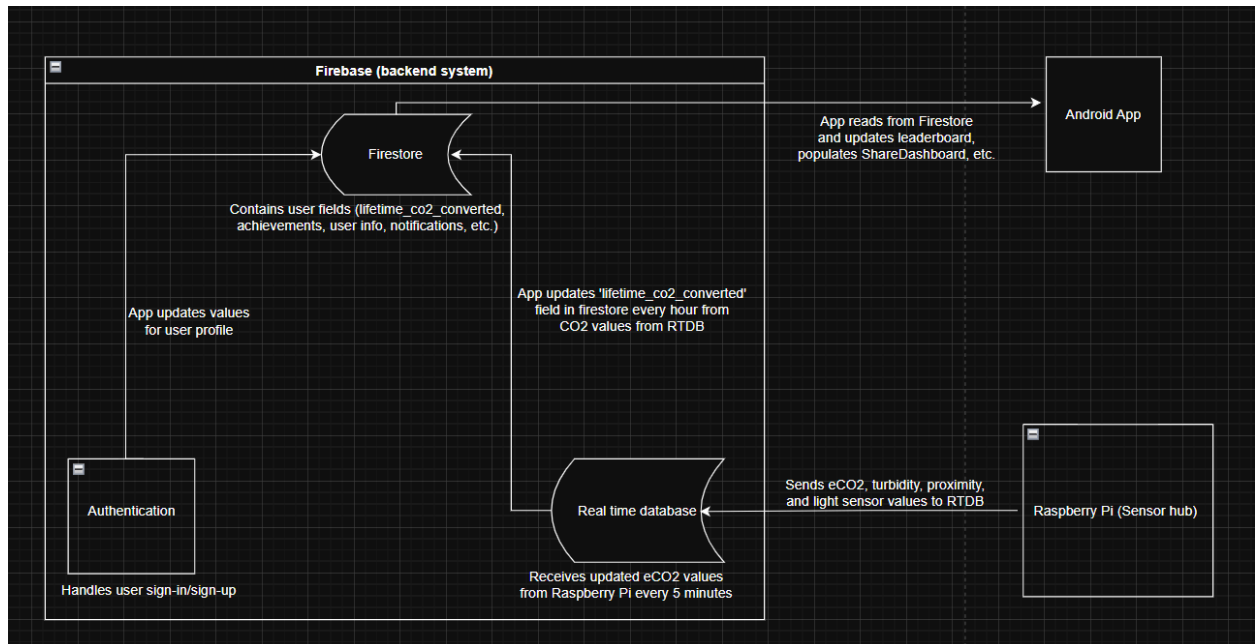
Dhairya - Creation of two C4 Model component diagrams, which are Firebase and Android App components. Roblectric (SettingsFragment) and Espresso (FeedbackFragment) testing. Revamping the color scheme of the app and making sure everything's visible and relevant to the algae-green app theme in both light and dark mode. Fixed three bugs that would crash the app during quick switching between screens and dark mode not staying on when the app is relaunched.

Sanskriti - Working on FAB button functionality. Also fixed Contact Support and Accountinfo screens UI. Also in charge of the Scrum dashboard creation with its tasks and stories this 5th sprint. Did junit testing on Userstat class by making 10 test cases and making sure that they all pass, and are meaningful to test.

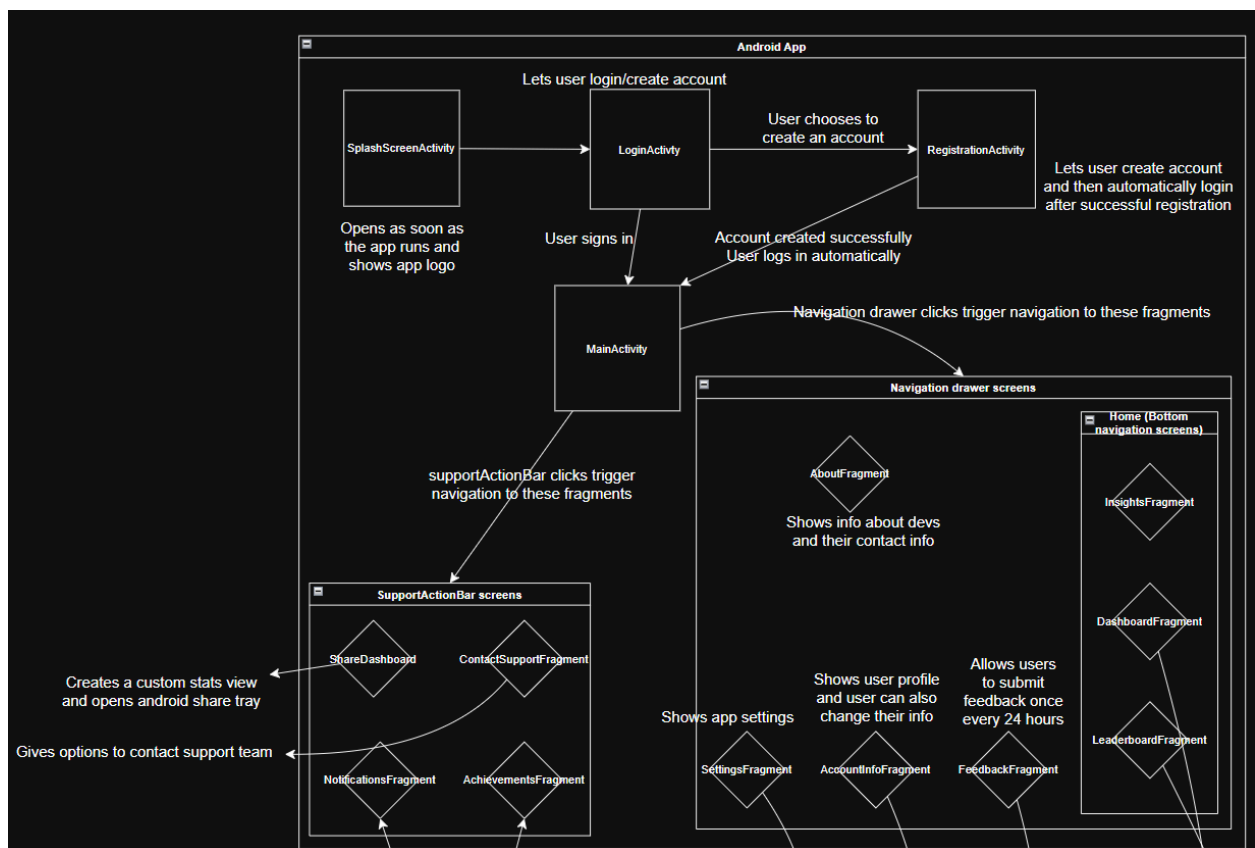
Dharmik - In charge of offline Mode features. Editing the UI for Insights, for better user viewing. Ensuring that all the DB connectivity and functions are working and compatible with the application. Splash screen animation, and a different splash screen theme for each light and dark mode.

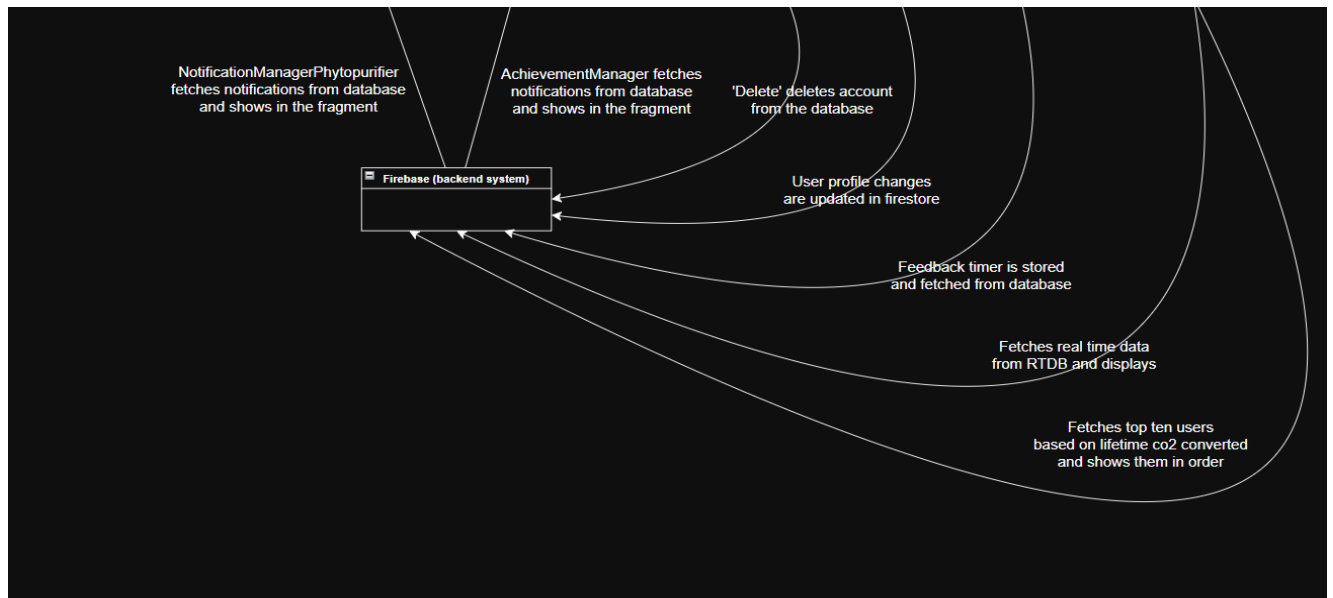
21. Using C4 Model, draw "Component Diagram". Choose two of the containers in your system. Draw two detailed components diagram.

Container #1 - Firebase



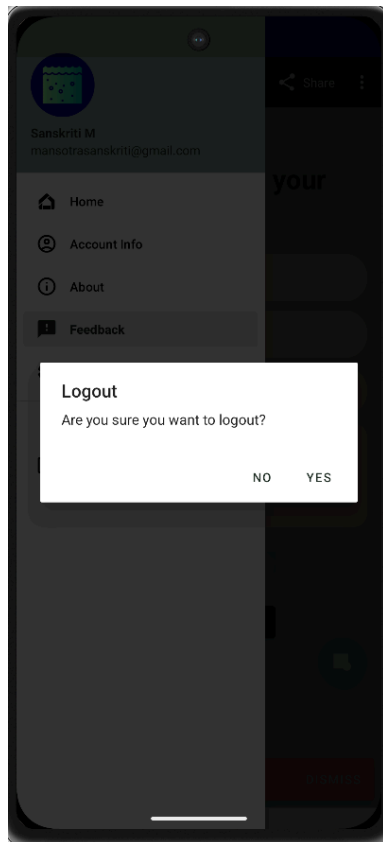
Container #2 - Android App



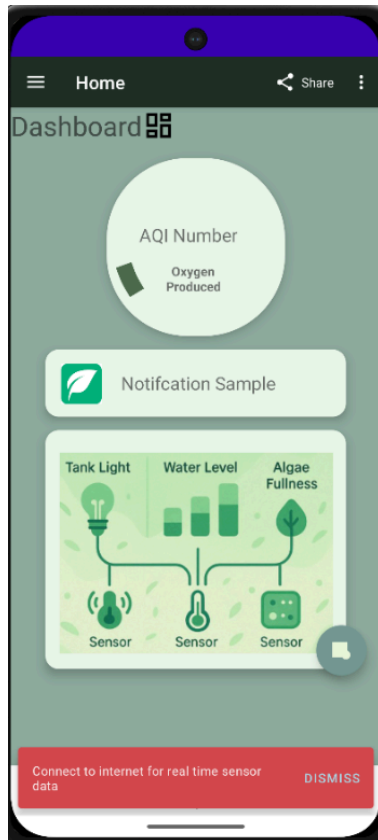


23. Demonstrate the use of all of the following in your app, in the appropriate places: Include screenshots to show.

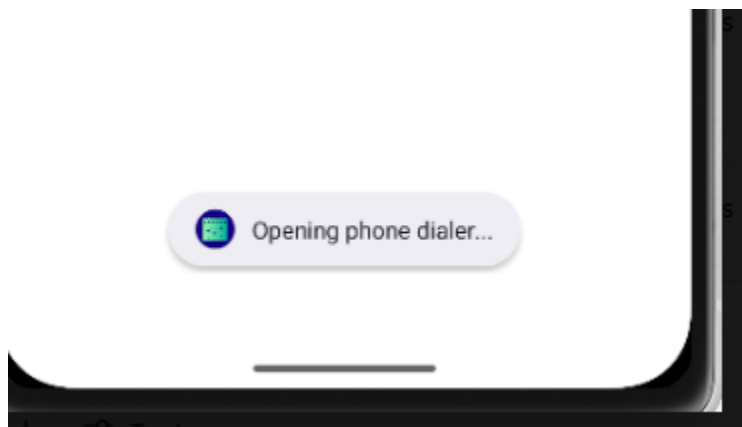
A. AlertDialog



B. Snackbar

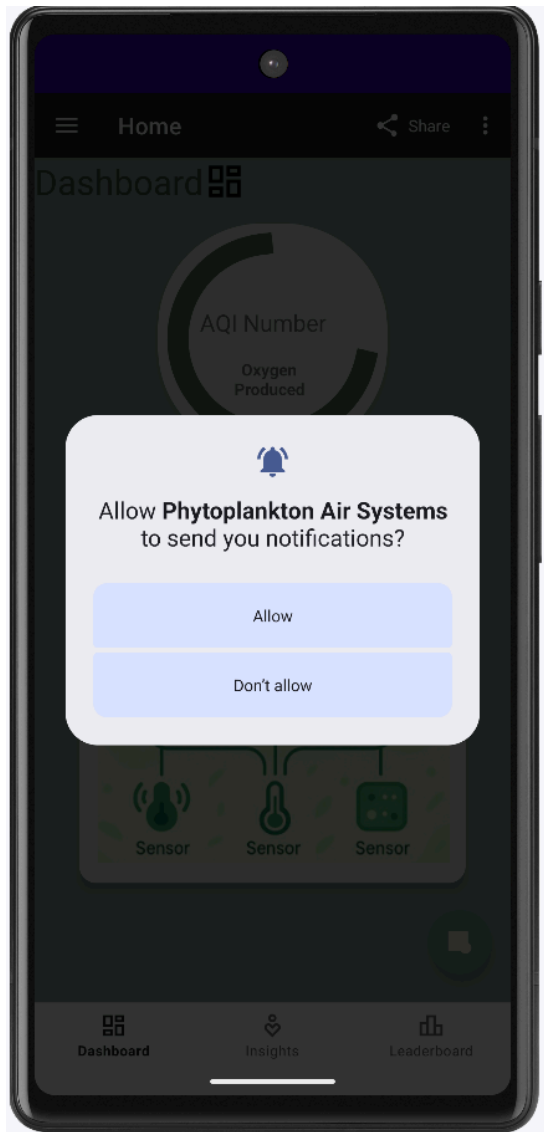


C. Toast

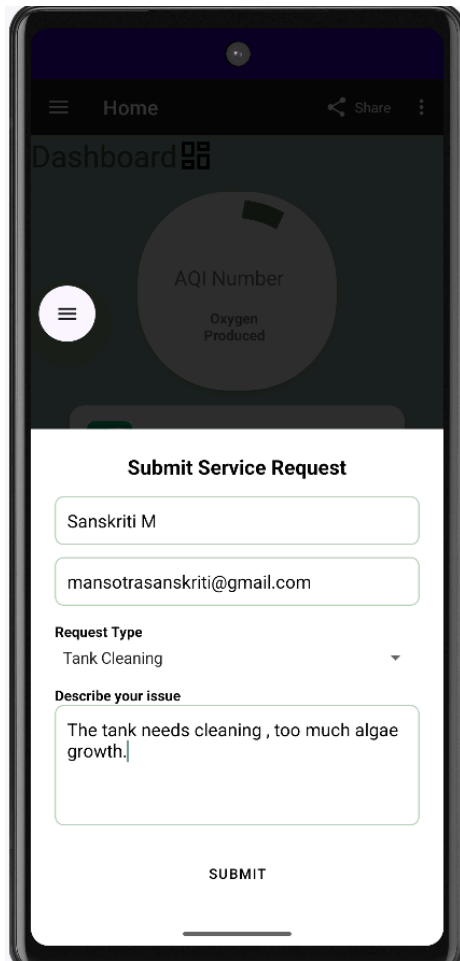


D. Notification, handle runtime for API 33 and higher.

Notification, handle runtime for API 33 and higher.



24. Screenshot showing the fab button and how the functionality fits into your app.



- **FAB opens a BottomSheet:** A service request form slides up from the bottom.
- **Form features:**
 - Auto-filled name and email using Firebase user data
 - Dropdown to choose request type (e.g., algae pod refill, tank maintenance)
 - Notes field for optional details
- On clicking **Submit**, the data is saved to Firestore as a **ServiceRequest** object.

Why this fits the app:

This feature supports the app's core goal: **maintaining algae air purification systems efficiently**. It enables users to:

- Request services instantly
- Stay within the app (no external emails or calls)
- Ensure faster issue resolution by our team

It improves user experience, streamlines communication, and makes the app more self-service-oriented.

25. Should implement some functionality for off-line mode. Document what feature will work off-line. Should be a meaningful feature. Will be tested, when in Airplane mode. Provide Screenshot. - A message in the app saying "Offline".

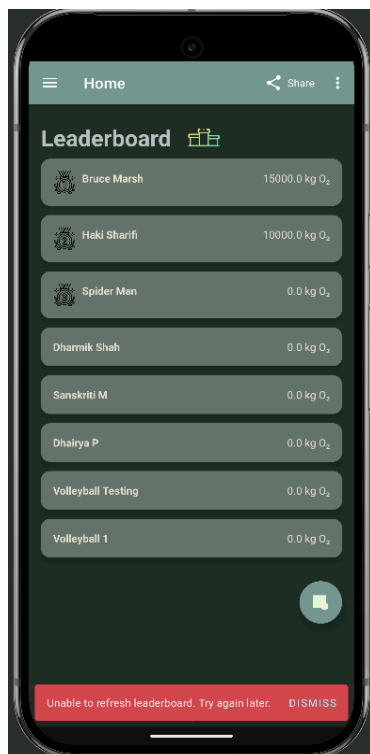
- Dashboard, Contact support, legal documents, about us page

Users can see the last fetched sensor data on Dashboard, the last fetched leaderboard on Leaderboard screen and the last fetched list of achievements from when the phone was last online. About us and legal documents (privacy policy and terms of service) are fully functional regardless of the internet connection. There are snackbars that will show when the user moves to these screens that will mention the fact that the app is offline and that the user needs to connect to the internet to see the live sensor data and achievements. The user can also see their account details if they've moved to that screen once when they were online and they move back because of cache storage.

User can see their account details but can't change it:



User is able to see the last fetched leaderboard:



26. Describe how did you handle exceptions, and list the exceptions that you had to handle.






































FirestoreException - which occurs due to permission errors, missing documents, or exceeded quotas. We handled this by checking the exception code, logging the error for debugging, and providing user-friendly feedback.

NullPointerException - Prevented by validating input before accessing user data.

NetworkException - typically caused by lack of internet connectivity. We addressed this by detecting the error and prompting the user to reconnect, ensuring the app didn't crash due to network instability.

IllegalArgumentException - Which can be triggered when invalid document IDs or parameters are passed into Firestore calls.

28. Complete Scrum dashboard, with all stories and tasks. Take screenshots of the last sprint only.

Name	Assignee	User Story	Priority	Start date	T-shirt Size 	Status 
▶  FAB Button Functionality 		As a user, I want to subm...	 High	3 days ago	M	 CLOSED
▶  Dashboard Fully Function 		As a user, I want to view ...	 High	2 days ago	L	 CLOSED
▶  Mobile Theme alignment 		As a user , I want to see ...	 High	6 days ago	M	 CLOSED
▶  Offline Mode 		As a user I wanna have a...	 Urgent	6 days ago	M	 CLOSED
▶  UI Changes 		As a user I want interactiv...	 Norm...	5 days ago	S	 CLOSED
▶  Insights Fragment 		As a user I wanna learn ...	 High	3 days ago	M	 CLOSED
▶  Testing Classes 		As a developer I wanna e...	 Urgent	Jul 29	M	 CLOSED
+ Add Task						

29. Post-Mortem, Project Review Meeting, document the below:

A. Begin your post-mortem, conduct a performance review of the project. In other words, calculate the project's performance in terms of cost, schedule, and quality.

In terms of performance schedule, our project overall took 4 months. With, 2 week sprints in the form of deliverables to be submitted, which we then take the feedback from the deliverables

Key Schedule Delays:

- Integration of sensor data with live phytoplankton responsiveness
- The UI and UX fully functionality by the last Sprint
- Fragment navigation and emulator/device testing in Android

- Real-time performance bugs and UI crashes (e.g., "device offline", fragment unknown errors)

Quality Metrics Used:

- UI Usability (based on app testing and user feedback)
- UX Usability, by testing the app flow ourselves
- Code coverage from JUnit and Espresso tests

Costs: The total estimated cost of the project was \$150. Of this, our team managed \$100 directly in the form of hardware purchases, mainly for the sensors, and the rest was towards any software licenses for the development and future launch of our application.

B. Did the team members involved manage their time wisely? Or everything was done last minute.

We would have meetings right after the sprint start, then we use the first week of that sprint to research on the most optimal ways that we would do our delegated tasks, which is decided upon in the meeting. Then towards the end of the sprint is when we would start completing the majority of our tasks to be done, leaving a day or two usually to finish up all the deliverable tasks to be done.

C. Were there issues with the quality or compromises along the way?

It was hard to overall visualize the end project as each team member had very unique ideas that they all wanted to have implemented, and so the compromise would be that we collectively agreed on a decision that would help us move onto the next thing. As for the quality, there were times where the UI of some screens would be lackluster, and not be aligned with the mobile theme that we all agreed upon and so that was something that we had to keep up with and stay consistent with.

D. Lessons learned, mistakes, and area of improvements.

Lessons learned - We learned about how to test the mobile app's certain features and functionalities, where we can debug and fix any errors that come up. A thing we also ended up getting a better grasp on, was the frontend and backend of a mobile application system, and getting better familiarised with that. Especially since not all members were particularly skilled at all aspects, we had to divide and conquer.

Mistakes - Sometimes when we work on some tasks, we tend to underestimate how long that task will actually take and end up working on it for much longer than intended. Also lacking the technical knowledge beforehand, thinking that a task is much simpler. We would have these complex ideas that would be too time consuming for a certain sprint, and not focusing on the main deliverable tasks to be done, leaving us with less time.

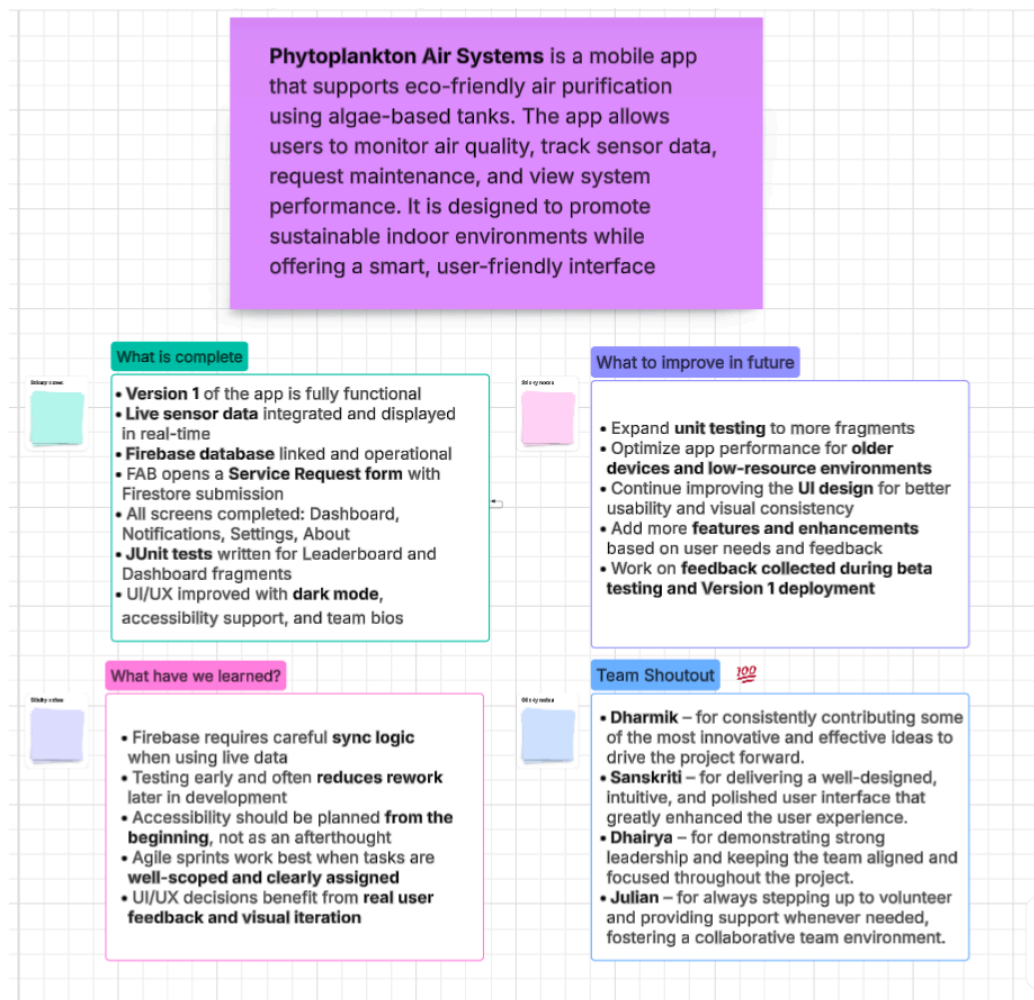
AOI - We can definitely improve on finishing our tasks much earlier, to avoid any sudden errors or crashes coming up in the application, at the last minute, as this was something we experienced in one of the earlier sprints.

E. Who attended and who missed the meeting. -

Everyone would attend the meeting on time and more on the time management in the post-mortem project review.

30. Use a tool to record your Project Review Meeting (i.e.

<https://lucidspark.com/landing/create/online-sticky-notes>, <https://miro.com/>)



31. How did you address technical debt in your project?

By the end of the last sprint, we had completed most of the core functionality of our app. In this sprint, our focus shifted toward refining the user experience and improving code structure. We worked on two of the ten main screens, making significant enhancements to visual consistency and app responsiveness from the user's perspective.

Initially, our goal was to ensure functionality - for example, we created basic cardviews in the AchievementsFragment and NotificationsFragment just to get them working. In this sprint, we refined their appearance by updating colors and layouts to align with the overall app theme. A more substantial improvement was in the code architecture. Previously, classes like LoginActivity were responsible for handling both Google Sign-In and regular login logic, resulting in tightly coupled code. During this sprint, we refactored these components to follow the Model-View-Controller (MVC) pattern across all fragments, reducing the amount of logic in the UI classes and significantly improving maintainability and scalability. Another good example of us addressing technical debt is the Share option in our app that used to just take the screenshot of the screen and open it in the Android share tray. But now using the ShareDashboard java class, we make a custom view that includes some important information that the user would actually want to share that gets shared in the Android share tray.

32. Document two areas of refactoring and why you did it!. Real examples from your code and not just statements. Copy the code from your project and comment on it.

```
private void startCountdownTimer(long durationMillis) {
    if (countDownTimer != null) {
        countDownTimer.cancel();
    }
    countDownTimer = new CountDownTimer(durationMillis, 60 * 1000) {
        @Override
        public void onTick(long millisUntilFinished) {
            long hours = millisUntilFinished / (1000 * 60 * 60);
            long minutes = (millisUntilFinished / (1000 * 60)) % 60;
            String countdownText = String.format(Locale.getDefault(), "Available
in %d hrs %d mins", hours, minutes);
            view.updateCountdownText(countdownText);
        }

        @Override
        public void onFinish() {
            view.setSubmitButtonEnabled(true);
            view.updateCountdownText("");
        }
    };
    countDownTimer.start();
}
```

This screenshot houses most of the countdown timer logic. This used to be a part of FeedbackFragment, but this sprint following MVC, we moved this code to FeedbackController.java class. In the code, view is an object of type FeedbackView (an interface implemented in FeedbackFragment). FeedbackFragment, FeedbackController, and FeedbackModel classes work together as per the MVC design principle. This refactoring enables us to manage all the controls that happen in the Feedback Fragment all in one java class.

```

public class ValidationUtils {

    public static boolean isValidName(String name) {
        return !TextUtils.isEmpty(name);
    }

    public static boolean isEmptyEmail(String email) {
        return TextUtils.isEmpty(email);
    }

    public static boolean isValidEmailFormat(String email) {
        return Patterns.EMAIL_ADDRESS.matcher(email).matches();
    }

    public static boolean isValidPhoneNumber(String number) {
        PhoneNumberUtil phoneUtil = PhoneNumberUtil.getInstance();
        try {
            Phonenumber.PhoneNumber parsedNumber = phoneUtil.parse(number,
null);
            return phoneUtil.isValidNumber(parsedNumber);
        } catch (NumberParseException e) {
            return false;
        }
    }

    public static boolean isValidPassword(String password) {
        return password != null &&
password.matches("(?=.*[a-z]) (?=.*[A-Z]) (?=.*[\\d\\W]).{6,}$");
    }

    public static boolean isValidConfirmPassword(String password, String
confirmPassword) {
        return password != null && password.equals(confirmPassword);
    }

}

```

This class is a factory for Validation utilities. So all of our classes can call these static methods from this class so we don't have to repeat validation/regex code in every class. This is based on the factory design pattern and helps us make sure that validation logic is the same for all our classes - this helps reduce error.

33. Describe how DevOps would have helped your project. -

DevOps would bring automation, speed, reliability, and collaboration to our eco-friendly air purifier project — allowing us to ship features faster, catch bugs earlier, and maintain a stable experience for users relying on real-time CO₂ and algae health monitoring.

34. What coding standards did you use, and how did you use them. -

We followed Java best practices, clean architecture principles, and structured testing standards to write readable, maintainable, and testable code throughout the project.

Logcat shows a NullPointerException at SettingsFragment.java:55, saying usernameTextView is null. - This is an example of effective logging and exception handling.

```
@Override
public void onBindViewHolder(LeaderboardAdapter.ViewHolder holder, int position) {
    UserStat user = userStats.get(position);
    holder.nameText.setText(user.name);
    holder.carbonDioxideText.setText(user.carbonDioxideConvertedKg + " kg CO2");

    if(position == 0){
        holder.medalIcon.setVisibility(View.VISIBLE);
        holder.medalIcon.setImageResource(R.drawable.medal1st);
        holder.nameText.setTextColor(ContextCompat.getColor(context, R.color.gold));
    } else if(position == 1){
        holder.medalIcon.setVisibility(View.VISIBLE);
        holder.medalIcon.setImageResource(R.drawable.medal2nd);
        holder.nameText.setTextColor(ContextCompat.getColor(context, R.color.silver));
    } else if(position == 2){
        holder.medalIcon.setVisibility(View.VISIBLE);
        holder.medalIcon.setImageResource(R.drawable.medal3rd);
        holder.nameText.setTextColor(ContextCompat.getColor(context, R.color.bronze));
    }
}
```

This old LeaderboardAdapter.java code violates the DRY principle.


```

@Override
public void onBindViewHolder(LeaderboardAdapter.ViewHolder holder, int position) {
    UserStat user = userStats.get(position);
    holder.nameText.setText(user.name);
    holder.carbonDioxideText.setText(user.carbonDioxideConvertedKg + " kg O2");

    if (position >= 0 && position <= 2) {
        int[] medalDrawables = { R.drawable.medal1st, R.drawable.medal2nd, R.drawable.medal3rd };
        int[] nameColors = { R.color.gold, R.color.silver, R.color.bronze };

        holder.medalIcon.setVisibility(View.VISIBLE);
        holder.medalIcon.setImageResource(medalDrawables[position]);
        holder.nameText.setTextColor(ContextCompat.getColor(context, nameColors[position]));
    }
}
}

```

This new code does not violate the DRY principle, because logic is used in arrays, removing repetitive code, and becomes easier to maintain and extend.

35. List at least one security vulnerability in your code, copy the code and comment on it. How to address it in the future.

During the user registration process in our app, we observed that if a user begins entering their details and then navigates away from the registration screen or goes offline before completing the process, an incomplete user account may still be created in Firebase Authentication. This behavior can potentially allow users to create multiple accounts unintentionally or maliciously, leading to account duplication and possible abuse of the system.

To address this, we plan to implement a more robust user creation flow that separates the authentication step from the final registration. Specifically, we will:

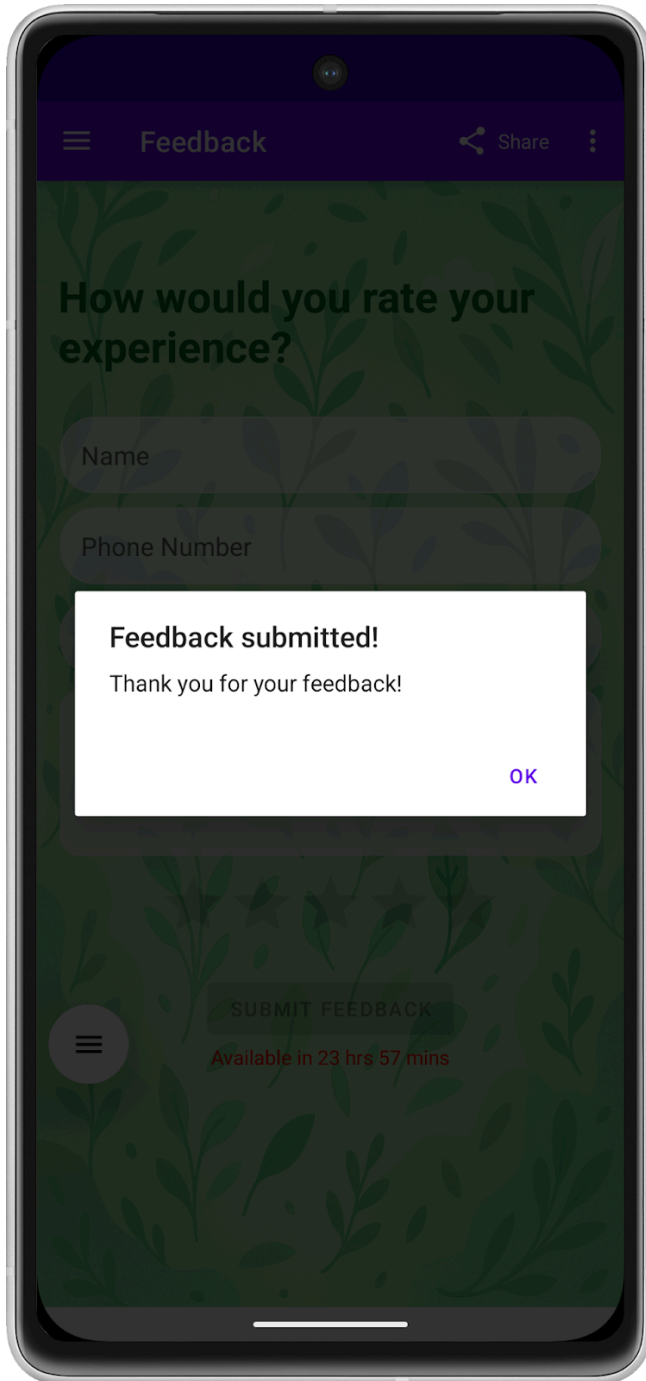
- Postpone creating the Firebase Authentication user until all required fields (e.g., username, email, password) are validated and confirmed.
- Periodically clean up inactive or incomplete accounts via backend scripts or Firebase functions to maintain data hygiene and prevent misuse.

This will help ensure that user accounts are only created once all necessary data is properly submitted and verified, improving both security and user experience.

36. Add into pdf file screenshot showing the progress bar while the form is getting submitted on the feedback screen.



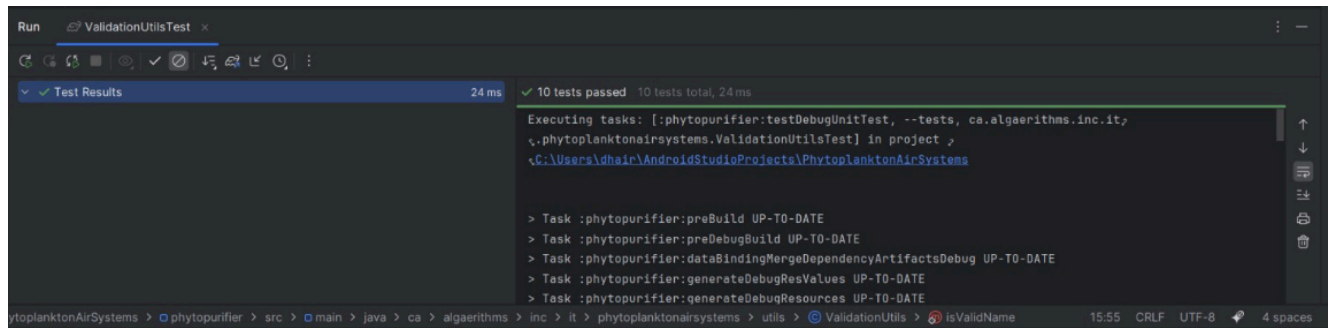
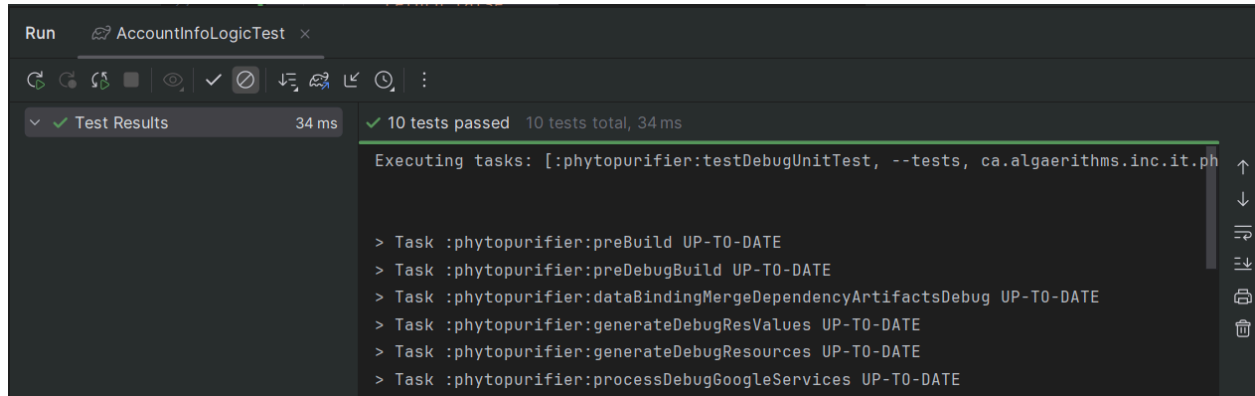
37. Add into pdf file screenshot showing the AlertDialog once the form is submitted successfully on the feedback screen. -

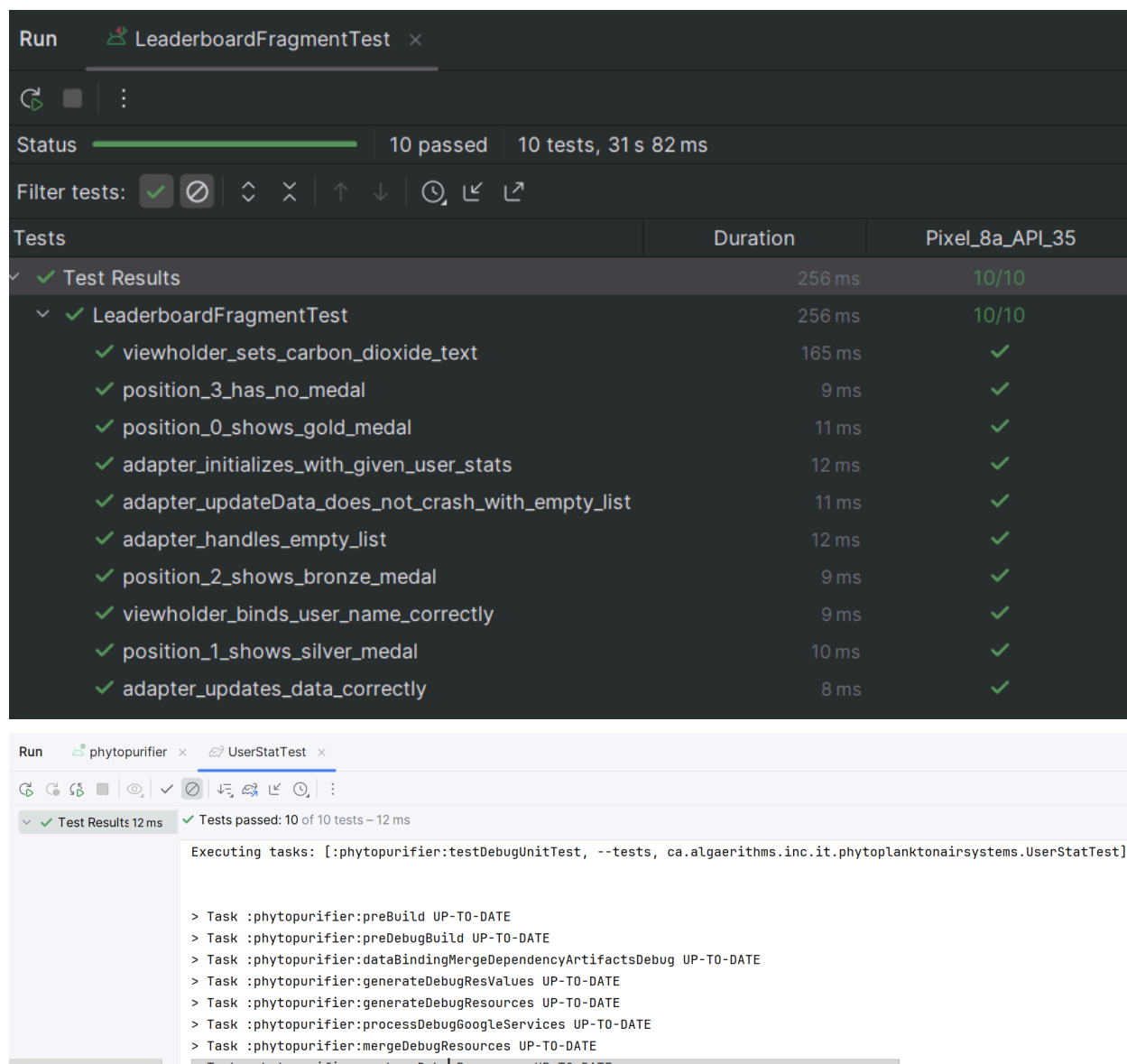


38. Describe your strategy for writing the test cases. -

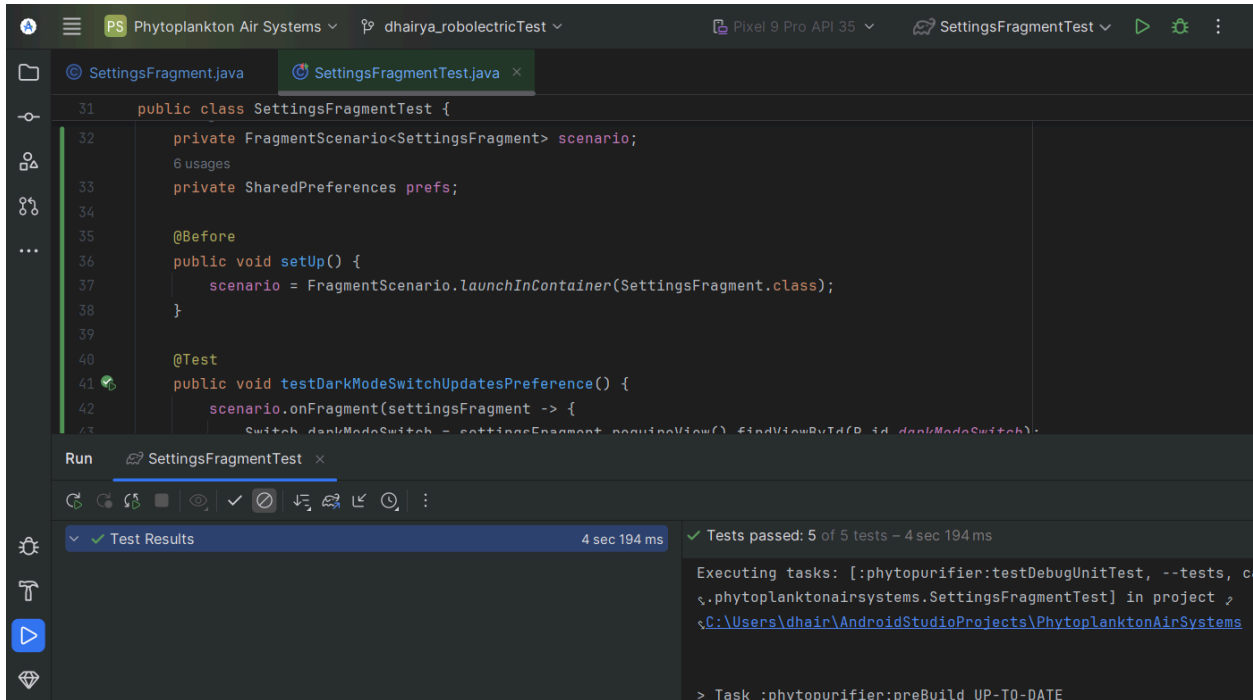
Having a testing goal with each test function, by breaking down what needs to be verified. Testing edge and negative test cases as well. Also covering at least one of each testable behavior such as the Rendering logic (what should appear), Conditional logic (positional coding etc.), and lastly the data handling (like list size, etc.)

39. Screenshots showing 3 classes junit test cases are passing. -

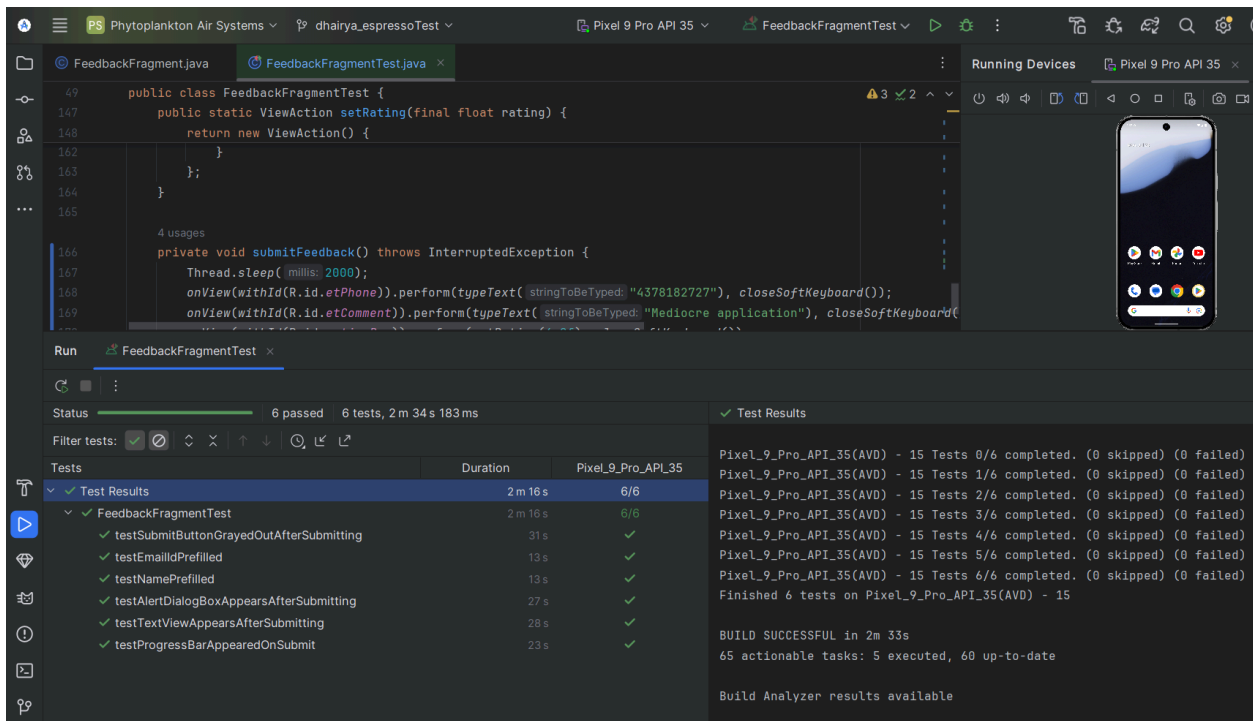




40. Screenshot showing Robleteric test cases are passing.

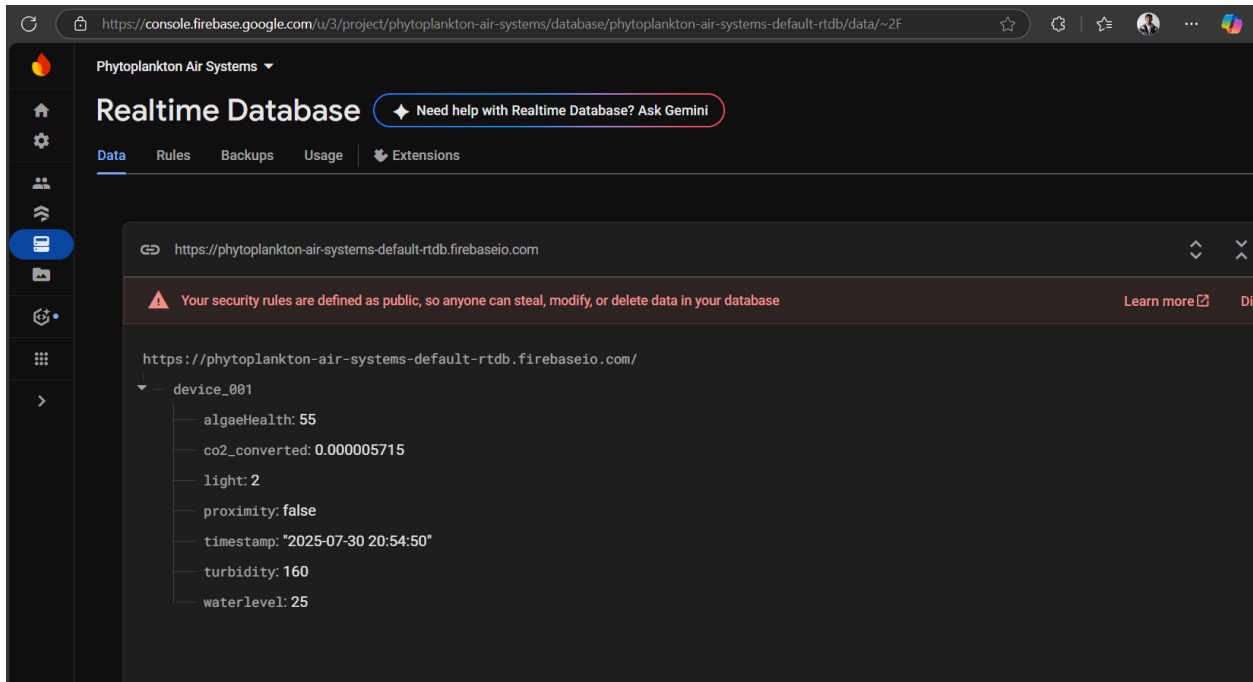


41. Screenshot showing Espresso test cases are passing.

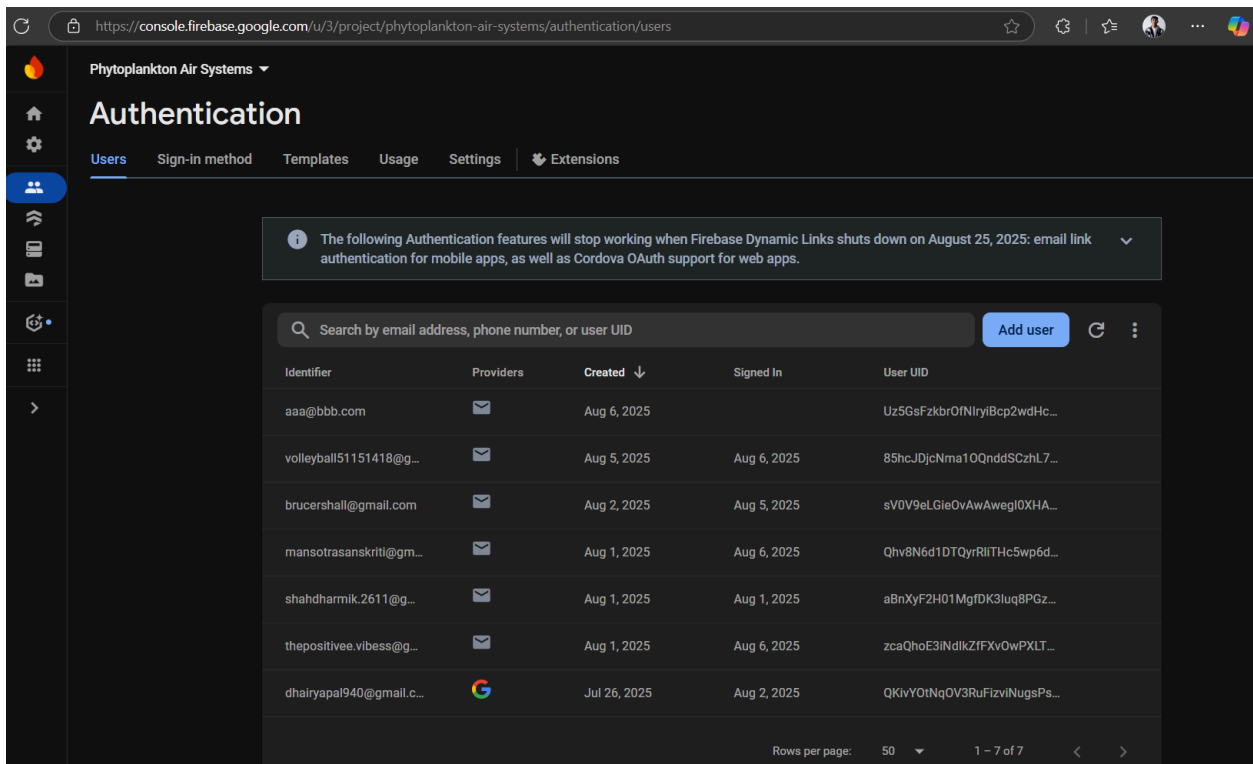


42. Screenshots showing the data in your DB, sensors data (at least 3 different sensors), and login data (including Google login).

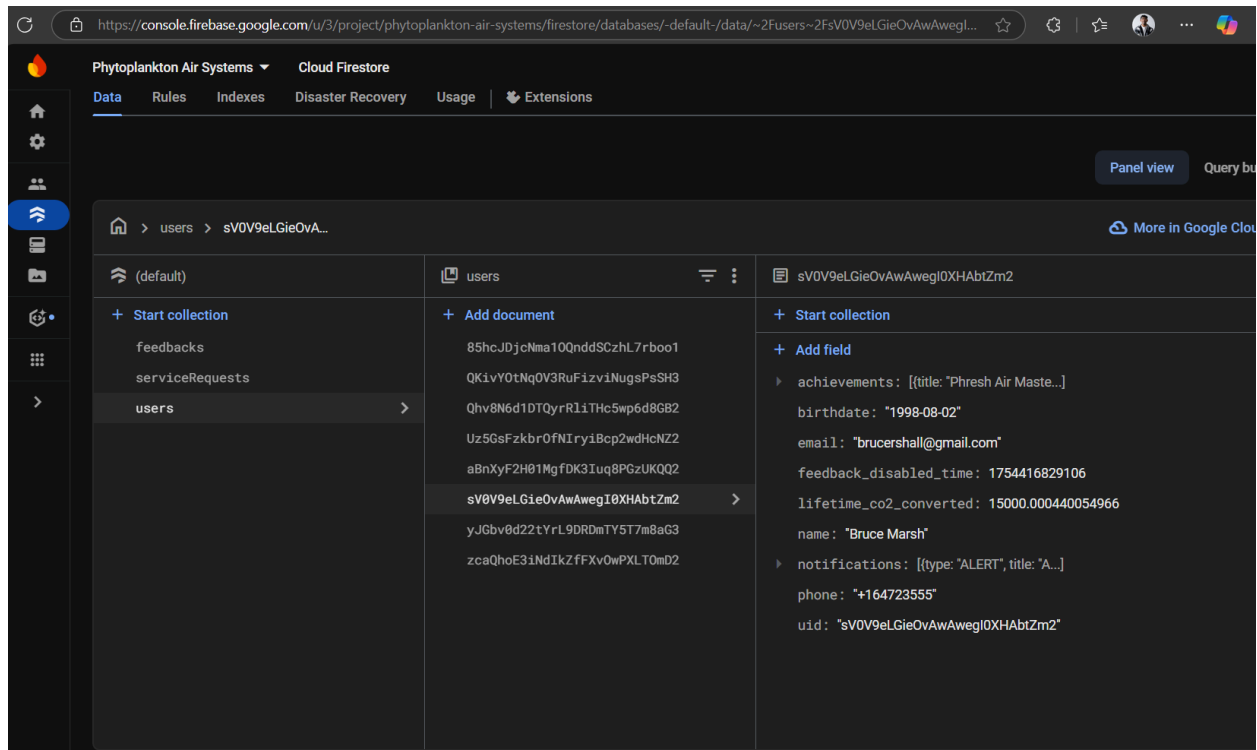
Sensor data (Realtime database):



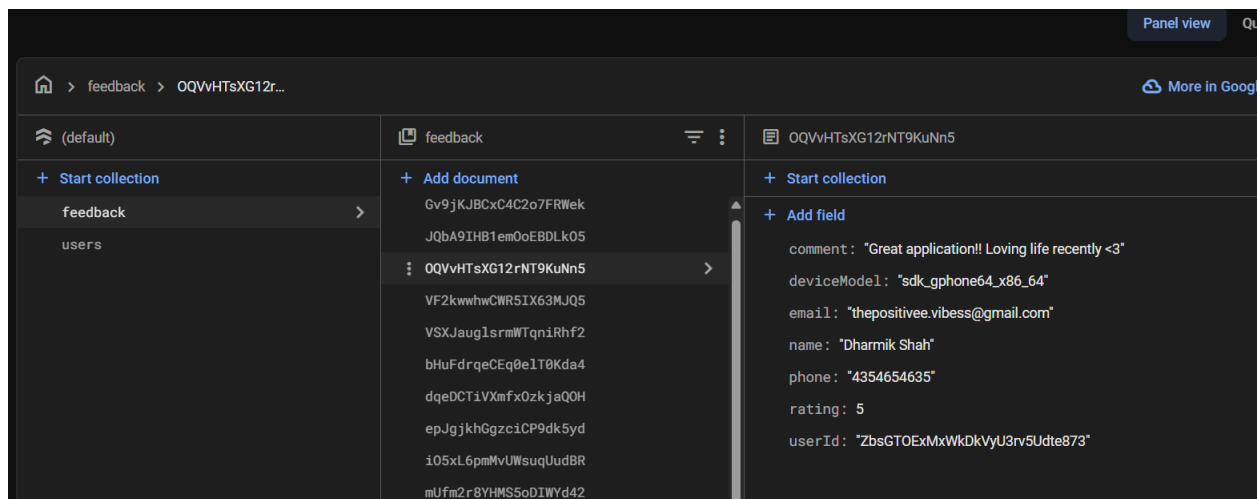
Authentication (including Google sign in):



Firestore user data:



43. Add into pdf file screenshot showing the data stored into the DB from Feedback. Must have at least 3 different entries.



44. Implement animation for the Splash screen, and two different splash screens (Dark and Light). Take screenshots showing the two different splash screens.



45. Suggestions to the instructor for future projects, things you liked, and things you suggest to be done differently and how.

Things we liked: We liked that we covered so many aspects of application building and how it is done in the industry. This course gave us hands-on experience, and at the very least knowledge about what tools are used and how they are used, like Scrum dashboard tools ([Monday.com](https://monday.com/), ClickUp, etc.), JFrog, Artifactory, GitHub, etc. We also covered a few things that we hadn't covered in the Mobile programming course - like adding both the text and icon to the support action bar, using the Countdown timer and disabling a button based on that, combining two different navigation layouts, etc.

Things that could've been done differently: The course was great overall and there is no real criticism to the delivery and the content, but if we had to mention something, then the user of DevOps early on in the course could have given us better insights into how this new field works, since this is becoming more famous lately. Maybe we could add some automation/pipeline creations into the course.