

Deliverable 5

The Tech Sense

Smart Indoor Parking System

Group 10

Raghav Sharma	N01537255
Kunal Dhiman	N01540952
Nisargkumar Pareshbhai Joshi	N01545986
Rushi Manojkumar Patel	N01539144

Table of Contents

1. Brief Description of the Project	3
2. Signatures Table.....	3
3. GitHub Repo Link	3
4. Login Credentials	3
5. Sprint 5 Goals	3
6. Technical Debt Tasks.....	4
7. Completed Work by Each Team Member	5
8. C4 Model: Component Diagram RP	7
9. App Functionalities	7
10. FAB Button Implementation.....	9
11. Offline Mode Functionality.....	11
12. Exception Handling	13
13. Complete Scrum Dashboard	17
14. Post-Mortem/Project Review	20
15. Technical Debt Addressing	21
16. Code Refactoring.....	21
17. DevOps Impact.....	23
18. Coding Standards	24
19. Security Vulnerability	25
20. Progress Bar on Feedback Screen	26
21. Alert Dialog on Successful Form Submission	27
22. Data Stored in Database from Feedback.....	28
23. Test Case Strategy	29
24. JUnit Test Cases	29
25. Roblectric Test Cases RP	30
26. Espresso Test Cases	31
27. DB Data Screenshots	31
28. Suggestions for Future Projects.....	35

1. Brief Description of the Project

The "Parkit" app connects users with parking lot owners, simplifying the process of finding and booking parking spaces. Users can search for, reserve, and save parking spots, track their reservations, and customize settings such as theme and currency. Additional features like promotions, feedback collection, help support, and notifications enhance user experience and encourage repeat business.

For parking lot owners, the app offers tools to manage locations, available slots, and track their income. Owners can update location details, monitor slot availability, and view income reports, helping them optimize occupancy and attract more customers. This app benefits both users and owners by improving parking efficiency and maximizing revenue.

2. Signatures Table

Name	Student ID	Git Hub ID	Signature	Effort (%)
Raghav Sharma	N01537255	RaghavSharma7255	RS	100%
Kunal Dhiman	N01540952	KunalDhiman0952	KD	100%
Nisarg Kumar Paresh Bhai Joshi	N01545986	NisargJoshi5986	NJ	0%
Rushi Manojkumar Patel	N01539144	RushiPatel9144	RP	0%

3. GitHub Repo Link

<https://github.com/RushiPatel9144/SmartIndoorParkingSystem>

4. Login Credentials

User's Credentials	Owner's credentials:
Email: aaa@bbb.com Password: Admin101!	Email: ccc@ddd.com Password: Admin101!

5. Sprint 5 Goals

- ❖ Implement confirmed booking functionality.

- ❖ Add notifications for booking and payment updates.
- ❖ Integrate functionality for parking directions.
- ❖ Ensure proper UI/UX in dark mode and favorites section.
- ❖ Add booking reminder notifications and accurate location titles.
- ❖ Enhance network stability for offline features.
- ❖ Redesign interactive owner dashboard and favorites section.
- ❖ Implement a fully functional payment system with Stripe.
- ❖ Enable owner login/signup and dashboard functionality.
- ❖ Add swipe location deletion and implement FAB for adding locations.
- ❖ Start tracking car status and notify users in parking slots.
- ❖ Process refunds after booking cancellations.
- ❖ Display promotions and ensure the application of coupons during checkout.

6. Technical Debt Tasks

→ Enhance Owner Dashboard, Account Management, and Sign-Up Security

- **Description:** Develop and implement an owner dashboard that displays revenue and additional details.
Create and integrate account management features specifically for owners.
Enhance the sign-up process by implementing regex for password validation.
- **Reason for Technical Debt:** These tasks were backlogged in the previous sprint, leading to delays in providing essential features and security enhancements.
- **Value Delivered:** Provides owners with a comprehensive view of their revenue, enhancing decision-making and financial management.
Ensures owners have robust account management capabilities, improving user experience and security.
Strengthens password security, reducing the risk of unauthorized access and improving overall system security.

→ Refactor Classes and Remove Redundant Code

- **Description:** Refactor large classes like Booking Manager into smaller, more manageable classes and eliminate repeated code.
- **Reason for Technical Debt:**
 - The current implementation includes large, monolithic classes with redundant code, making maintenance and debugging challenging.
 - Separate the code for working with the favorite button, including all its functionality, into a new class called Favourite Manager.

- **Value Delivered:**

- Improves code maintainability and readability by breaking down large classes and eliminating redundant code.
- Ensures that each class has a single responsibility, adhering to design principles and making the codebase more efficient and easier to work with.
- Facilitates future development and debugging, making the codebase more modular and manageable.

7. Completed Work by Each Team Member

Kunal Dhiman (KD):

- ❖ **Implemented confirmed booking page** - Designed and implemented the page to confirm bookings with proper details.
- ❖ **Added parking directions** - Integrated directions into the confirmation screen and activity.
- ❖ **Set accurate titles for locations** - Updated location titles dynamically in activity fragments.
- ❖ **Added booking reminder notifications** - Incorporated reminders to notify users about their bookings.
- ❖ **Ensured proper dark theme functionality** - Enhanced color schemes and ensured the app UI functions seamlessly in dark mode (in progress).
- ❖ **Added Notifications for Booking and Payment Updates:** Started developing a system to notify users about their booking and payment statuses.
- ❖ **Improved Car Status Tracking in Parking Slots:** Added a system to notify users of their car's status within parking slots (in progress).

Nisarg Joshi (NJ):

- ❖ **Redesigned favorite UI and display images** - Improved UI and added empty-state visuals.
- ❖ **Updated privacy policy and terms of use** - Incorporated updated legal requirements into the app.
- ❖ **Implemented promotion utilization check for users** - Added backend logic to verify applied promotions.
- ❖ **Designed feedback form UI** - Created a new UI for user feedback.
- ❖ **Enhanced Promotion Display on Checkout:** Integrated a feature to display applied promotions during checkout for better user clarity and satisfaction.

- ❖ **Improved UI with Navigation Arrows and Redesigned Favorites Section:** Enhanced the onboarding experience by adding navigation arrows and updated the "Favorites" UI with placeholder images for empty states.

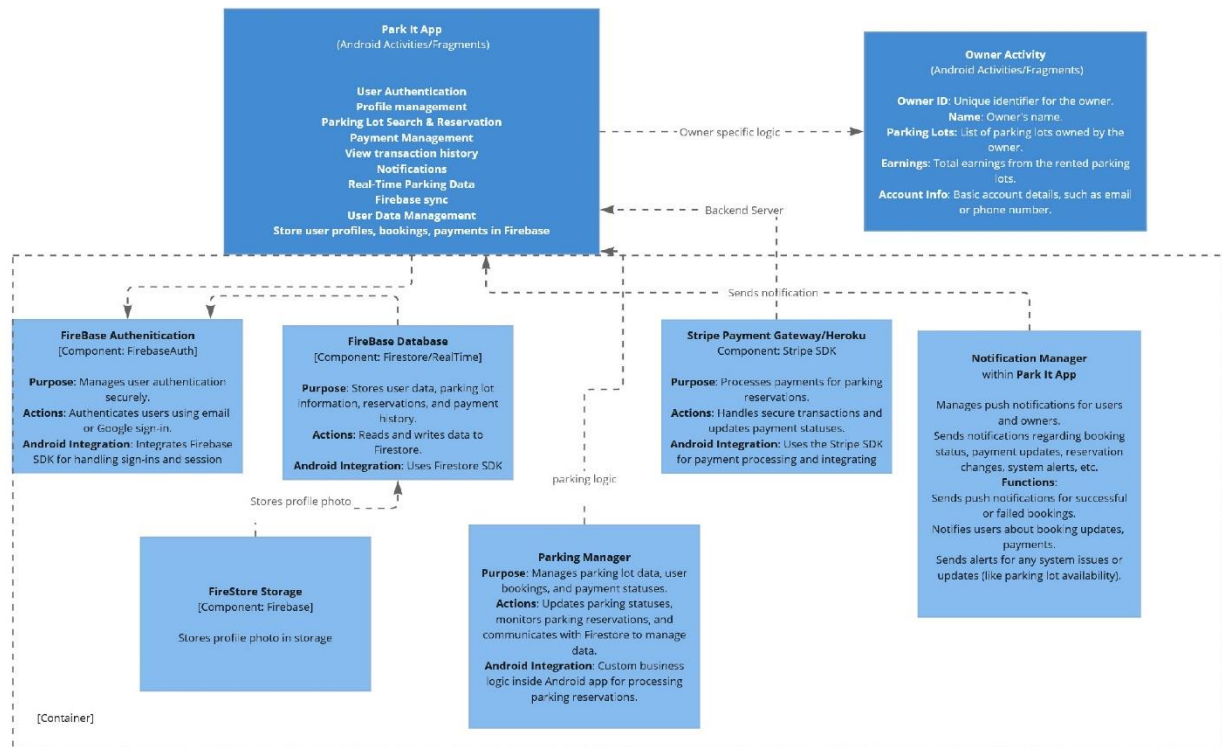
Raghav Sharma (RS):

- ❖ **Enhanced Network Stability and Offline Functionality:** Improved app behavior during network downtime and ensured critical features work offline seamlessly.
- ❖ **Implemented FAB for Adding Locations and Location Price Update:** Added a floating action button for owners to quickly add parking locations and allowed them to dynamically update parking spot pricing.
- ❖ **Developed Transaction History and Refund Processing for Owners:** Created a detailed transaction history dashboard for owners and integrated functionality to handle refunds for booking cancellations.
- ❖ **Implemented Progress Bars for Data Loading:** Added progress bars with delays for smoother data loading in features like maps and account details.
- ❖ **Enhanced Promotion Display on Checkout:** Integrated a feature to display applied promotions during checkout for better user clarity and satisfaction.
- ❖ **Developed Owner and User Authentication System:** Handled login and signup processes for both owners and users, including password validation using Regex and detailed error messages for failures.
- ❖ **Implemented swipe location deletion** - Allowed users to delete parking locations via swiping.

Rushi Patel (RP):

- ❖ **Design Owner Dashboard Buttons:** Create action buttons in Card View with clear descriptions for the Owner Dashboard.
- ❖ **Login and Signup Handling:** Implement login and signup processes for owners and users.
- ❖ **Add "Forgot Password" Option:** Integrate a password recovery option on the login page.
- ❖ **Password Validation:** Use regex for secure and strong password validation.
- ❖ **Error Messages for Login/Signup:** Display detailed error messages for authentication failures.
- ❖ **Session Manager:** Implement session management to display owner data across all fragments.
- ❖ **Navigation Arrows for Onboarding:** Add navigation arrows to improve onboarding screen usability.
- ❖ **Privacy Policy and Terms Links:** Include links to privacy policy and terms of use on the signup page.

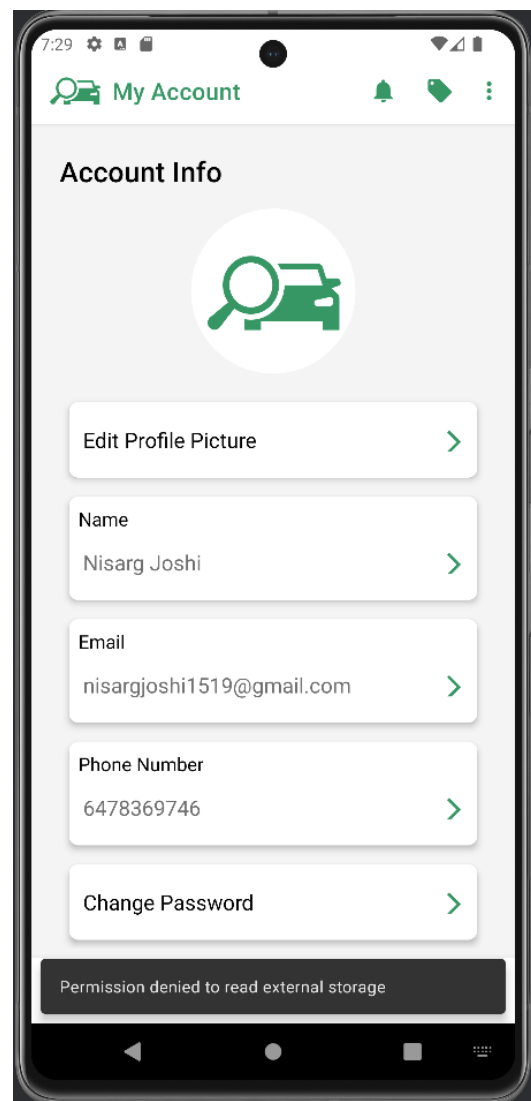
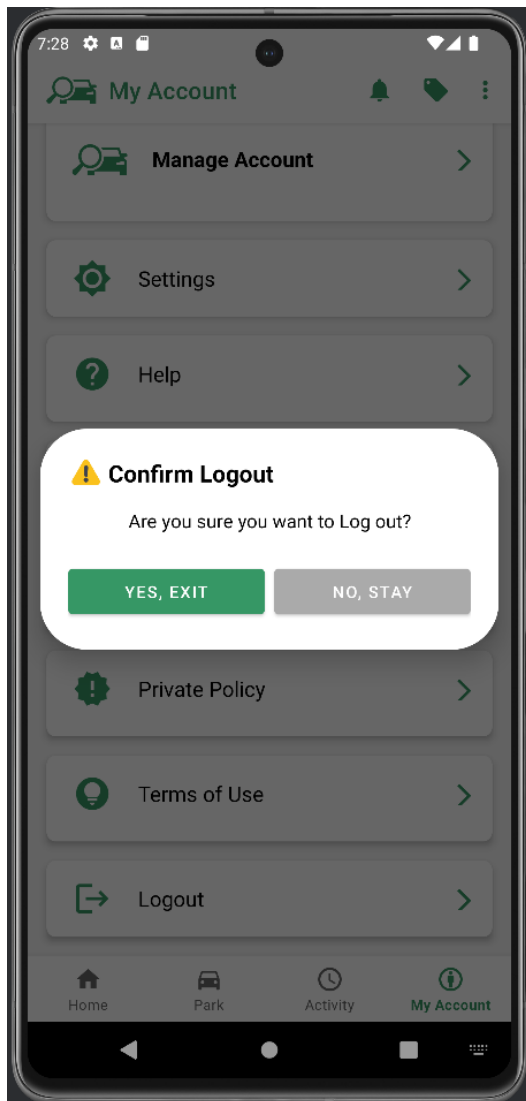
8. C4 Model: Component Diagram

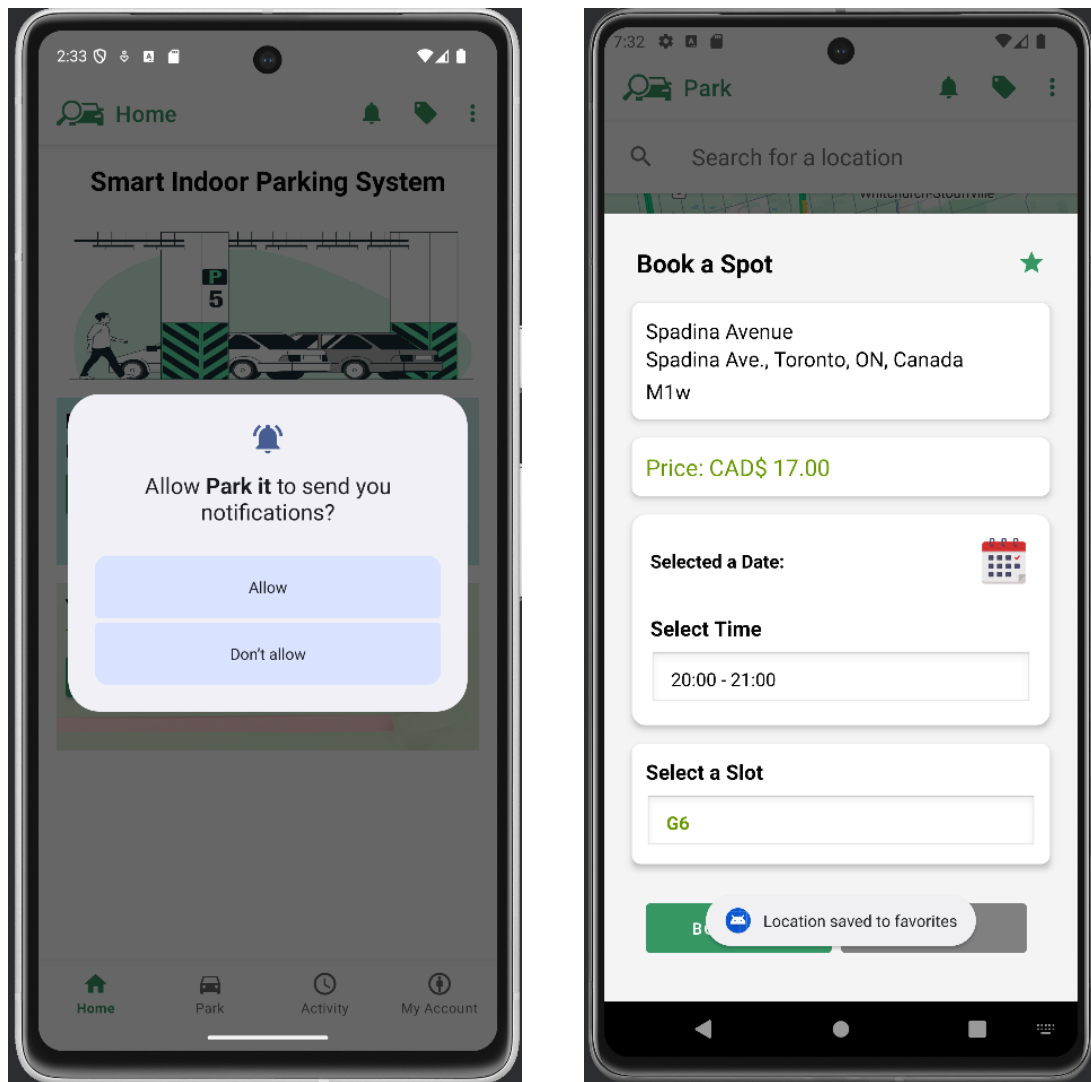


9. App Functionalities

Demonstrate the use of the following with screenshots: **AlertDialog**, **Snackbar**, **Toast**, **Notification** (runtime for API 33 and higher)

1. This **AlertDialog Box** asks the user to confirm before logging out.
2. This **snack bar** indicates that the app attempted to access external storage, but the necessary permissions were not granted by the user.
3. This **toast** indicates that this location is added to the favorites.
4. This is the **notification** shown to the user when they open the app for the first time after the installation.

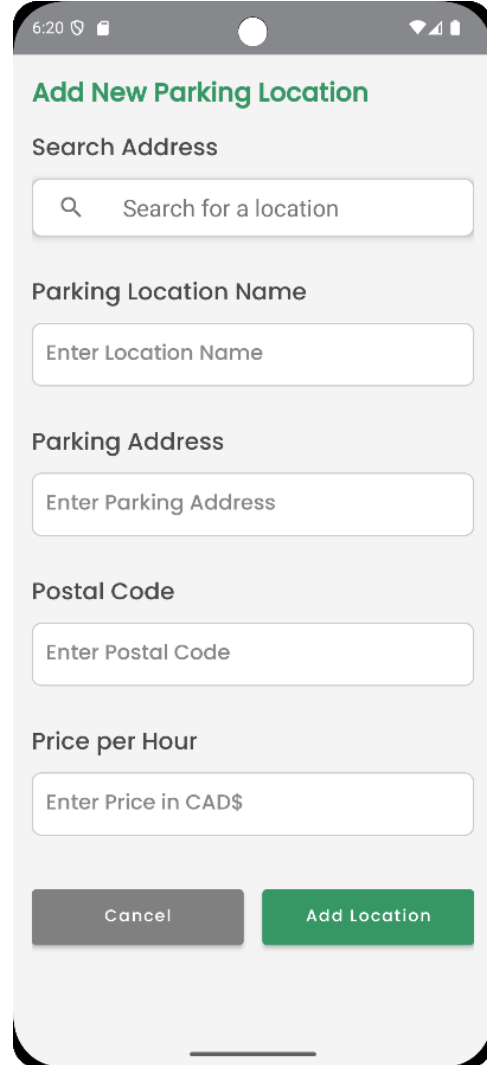
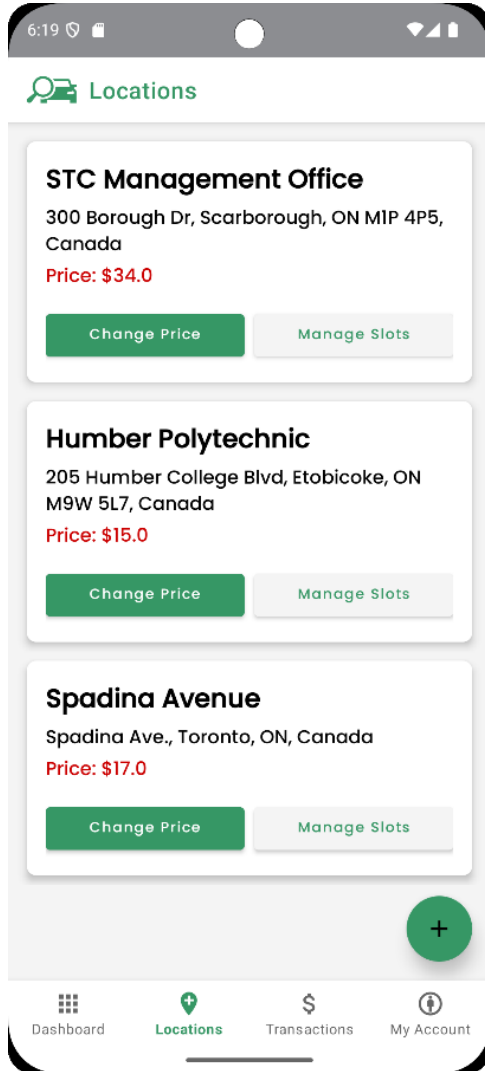


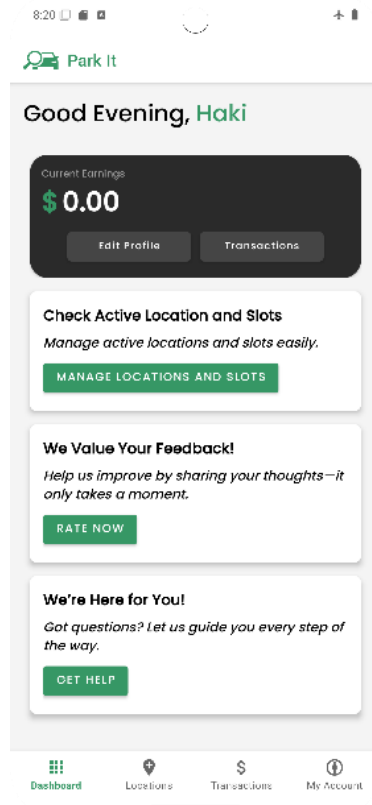


10. FAB Button Implementation

Show the functionality of the Floating Action Button (FAB) with a screenshot.

The Floating Action Button (FAB) is implemented on the locations screen in the owner interface. It allows owners to add a new parking location. When the owner clicks on the FAB, they can proceed to input the details for the new parking location.



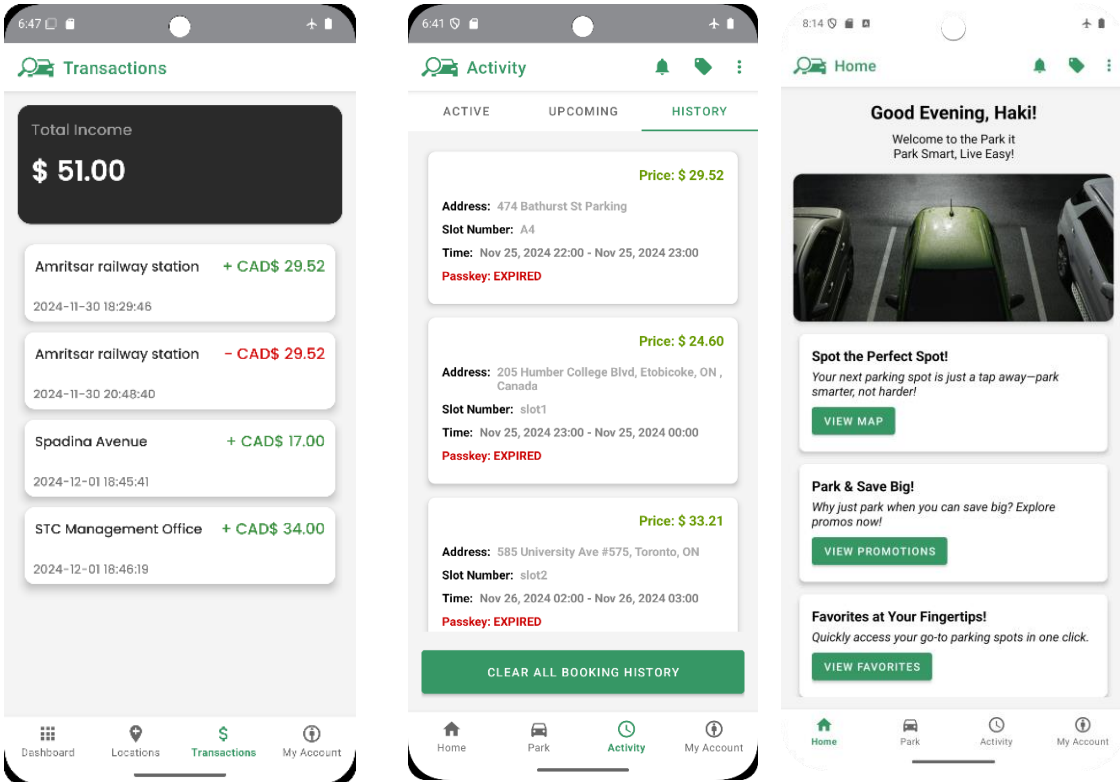
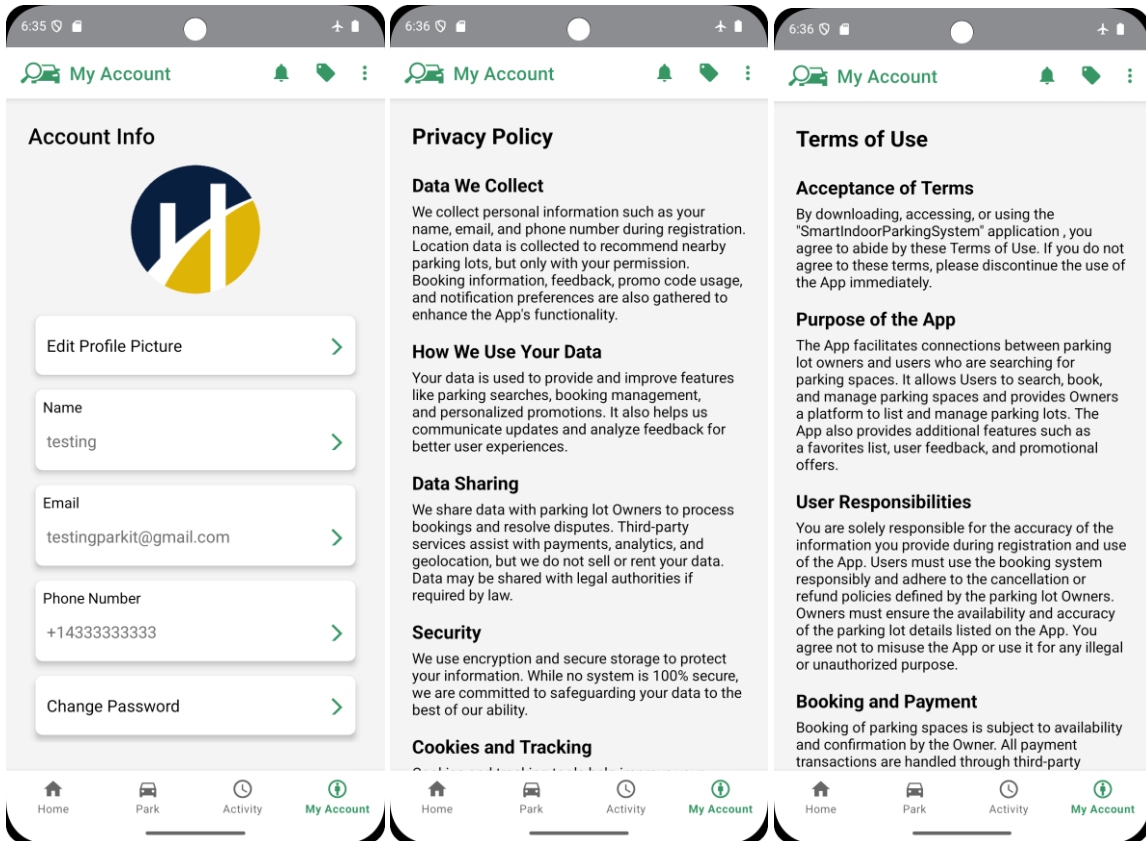


11. Offline Mode Functionality

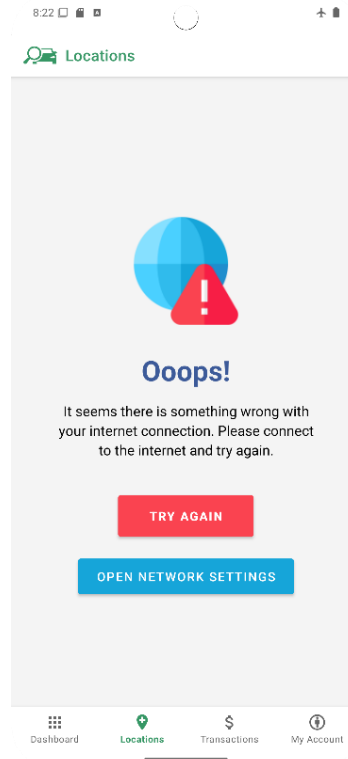
Document a meaningful offline feature, tested in Airplane mode.

When the app is used in Airplane mode, a network error page may appear on certain screens, but users can still navigate to all screens and access the following features without needing an internet connection:

1. **Manage Account:** Users can view their account details.
2. **Notifications:** Previously received notifications are accessible.
3. **Terms of Use and Privacy Policy:** Users can review the app's policies.
4. **History:** Users can view their past activities (User) and transactions (Owner interface).
5. **Owner Dashboard and User Home.**



And Other Screens that will look like this in offline mode –



12. Exception Handling

Describe how exceptions are handled in the app, including a list of exceptions.

→ **convertToMillis Method (BookingUtils Class)**

- **Exception Handled:** ParseException
- **Purpose:**
 - Ensures the app handles invalid date-time strings gracefully by catching parsing errors.
 - If an exception occurs, it logs the stack trace and returns 0, avoiding app crashes.

```

public class BookingUtils { 18 usages

    /**
     * Converts a date and time string to milliseconds.
     *
     * @param dateTime The date and time string in the format "yyyy-MM-dd HH:mm".
     * @return The corresponding time in milliseconds.
     */
    public static long convertToMillis(String dateTime) { 9 usages
        SimpleDateFormat sdf = new SimpleDateFormat(pattern: "yyyy-MM-dd HH:mm", Locale.getDefault());
        try {
            Date date = sdf.parse(dateTime);
            return date != null ? date.getTime() : 0;
        } catch (ParseException e) {
            e.printStackTrace();
            return 0;
        }
    }
}

```

→ loadCroppedImage Method (ManageAccountFragment Class)

- **Exceptions Handled:**
 - **IOException**
 - **Purpose:**
 - Handles potential errors when decoding an image from a URI or setting the image bitmap.
 - If an exception is encountered, the stack trace is printed, ensuring the app remains functional without crashing.

```

ManageAccountFragment.java x LocationsFragment.java SwipeToDeleteCallback.java AddLocationActivity.java NoNetworkFrac
42 public class ManageAccountFragment extends Fragment {
188
189     // Load the cropped image into the profile picture view
190     private void loadCroppedImage(Uri imageUri) {
191         try {
192             Bitmap bitmap;
193             ImageDecoder.Source source = ImageDecoder.createSource(requireActivity().getContentResolver(), imageUri);
194             bitmap = ImageDecoder.decodeBitmap(source);
195             profilePicture.setImageBitmap(bitmap);
196         } catch (IOException e) {
197             e.printStackTrace();
198             profilePicture.setImageResource(R.mipmap.ic_launcher);
199         }
200     }
}

```

→ validatePrice Method

- **Exception Handled:** NumberFormatException

- **Purpose:**
 - Manages invalid numeric inputs gracefully by displaying a toast message when the user enters a non-numeric price.
 - Ensures the app does not crash due to invalid inputs by returning -1.

```
public static double validatePrice(Context context, String input) { 2 usages
    String priceStr = input.trim();

    if (priceStr.isEmpty()) {
        Toast.makeText(context, "Price cannot be empty" , Toast.LENGTH_SHORT).show();
        return -1;
    }
    try {
        double price = Double.parseDouble(priceStr);
        if (price < 0) {
            Toast.makeText(context, "Price cannot be negative" , Toast.LENGTH_SHORT).show();
            return -1;
        }
        return price;
    } catch (NumberFormatException e) {
        Toast.makeText(context, "Invalid price format. Please enter a valid number." , Toast.LENGTH_SHORT).show();
        return -1;
    }
}
```

→ checkNetworkAvailability Method

- **Exception Handled:** General Exception
- **Purpose:**
 - Handles errors during network availability checks.
 - Logs the exception instead of displaying stack traces in the UI.
 - Ensures the scheduler shuts down in case of an error to prevent resource leakage.

```

private void checkNetworkAvailability() { 1 usage
    ScheduledExecutorService scheduler = Executors.newSingleThreadScheduledExecutor();
    scheduler.scheduleWithFixedDelay(() -> {
        try {
            if (NetworkUtils.isNetworkAvailable(context: NoNetworkActivity.this)) {
                runOnUiThread(this::finish); // Network is back, close the activity
                scheduler.shutdown(); // Shutdown the scheduler when the network is back
            }
        } catch (Exception e) {
            // Log the exception instead of printing the stack trace
            Log.e(TAG, String.valueOf("Error checking network availability"), e);
            scheduler.shutdown(); // Shutdown the scheduler in case of an error
        }
    }, 1, 5, 1, TimeUnit.SECONDS); // Start immediately, check every 1 second
}

```

→ onResponse Method

- **Exception Handled: JSONException:**
- **When:** If there's an issue parsing the JSON response (for example, if the response is not valid JSON or the expected fields are missing).
- **Handled By:** The catch (JSONException e) block catches this exception.
- **Action Taken:** If the JSON parsing fails, it runs runOnUiThread() to show a toast message indicating that the parsing of the server response failed, ensuring that the user is informed in a friendly manner without crashing the app.

```

public void onResponse(Response response) throws IOException {
    if (response.isSuccessful()) {
        try {
            JSONObject jsonResponse = new JSONObject(response.body().string());
            String clientSecret = jsonResponse.getString(name: "clientSecret");
            transactionId = jsonResponse.getString(name: "transactionId");
            runOnUiThread(() -> startPaymentFlow(clientSecret));
        } catch (JSONException e) {
            runOnUiThread(() -> showToast("Failed to parse server response"));
        }
    } else {
        runOnUiThread(() -> showToast("Server error: " + response.message()));
    }
}

```


13. Complete Scrum Dashboard

Include screenshots of the last Sprint and completed stories and tasks.

Sprint 5

Task name	Assignee ↕	Due date	Priority	Status	Size
▼ Sprint 5 - 12 Days					
✔ Display Booking Details on Confirmation Page	KD Kunal Dhiman	Nov 20 – Dec 3	High	Done	M
✔ Implement Booking Confirmation Notifications	KD Kunal Dhiman	Nov 20 – Dec 3	Critical	Done	M
✔ Implement Payment Status Notifications	KD Kunal Dhiman	Nov 20 – Dec 3	Medium	Done	M
✔ Integrate Parking Directions in Confirmation Page	KD Kunal Dhiman	Nov 20 – Dec 3	High	Done	M
✔ Ensure Consistency in Location Titles Across Screens in Activities	KD Kunal Dhiman	Nov 20 – Dec 3	High	Done	M
✔ Add Booking Reminder Notifications	KD Kunal Dhiman	Nov 20 – Dec 3	High	Done	M
✔ Review and Update UI for Dark Theme Compatibility	KD Kunal Dhiman	Nov 20 – Dec 3	Critical	Done	M
✔ Implement Car Tracking in Parking Slots	KD Kunal Dhiman	Nov 20 – Dec 3	High	Done	L
✔ Send Notifications for Car Status Updates	KD Kunal Dhiman	Nov 20 – Dec 3	Medium	Done	L

✔ Fetch and Display Applied Promotions on Checkout	ni nisargjoshi1...	Nov 20 – Dec 3	Medium	Done	S
✔ Add Placeholder Image and Message for Empty Favorites	ni nisargjoshi1...	Nov 20 – Dec 3	High	Done	M
✔ Review and Update Privacy Policy and Display	ni nisargjoshi1...	Nov 20 – Dec 3	Critical	Done	M
✔ Update Terms of Use Document and Display	ni nisargjoshi1...	Nov 20 – Dec 3	Medium	Done	M
✔ Track User Promotion Utilization in Database and Display Promotion Usage	ni nisargjoshi1...	Nov 20 – Dec 3	High	Done	M
✔ Create UI for Feedback Form with proper button name and proper progress bar	ni nisargjoshi1...	Nov 20 – Dec 3	Critical	Done	M
✔ Implement and Design Quick Buttons on Home Screen with descriptions	ni nisargjoshi1...	Nov 20 – Dec 3	Critical	Done	L
✔ Develop Logic for Promotions in Payment	ni nisargjoshi1...	Nov 20 – Dec 3	Critical	Done	L

✔ Transaction Logic for Calculating Owner Income	RS Raghav Shar...	Nov 20 – Dec 3	Medium	Done	M
✔ Implement Swipe-to-Delete Function for Locations	RS Raghav Shar...	Nov 20 – Dec 3	High	Done	M
✔ Display Transaction History for Owners with Sorting	RS Raghav Shar...	Nov 20 – Dec 3	High	Done	M
✔ Implement Refund Logic for Booking Cancellations	RS Raghav Shar...	Nov 20 – Dec 3	Low	Done	L
✔ Optimize Data Sync for Offline Functionality	RS Raghav Shar...	Nov 20 – Dec 3	Critical	Done	M
✔ Handle Errors in Offline Mode Using Error Page	RS Raghav Shar...	Nov 20 – Dec 3	Medium	Done	M
✔ Add Floating Action Button for Location Addition in Owner Interface	RS Raghav Shar...	Nov 20 – Dec 3	High	Done	M
✔ Implement Real-Time Updates for Location Prices	RS Raghav Shar...	Nov 20 – Dec 3	High	Done	L
✔ Handle Map Fetching Delays and Implement Progress Bar	RS Raghav Shar...	Nov 20 – Dec 3	Medium	Done	M
✔ Optimize User Account Data Fetching and Implement Swipe-to-Refresh	RS Raghav Shar...	Nov 20 – Dec 3	Medium	Done	M

✔ Add Income Display to Owner Dashboard Home Screen	ru rushipatel22...	Nov 20 – Dec 3	Critical	Done	M
✔ Design Action Buttons with Descriptions for Owner Dashboard in Card view for	ru rushipatel22...	Nov 20 – Dec 3	Medium	Done	M
✔ Handle Owner and User Login and Signup	ru rushipatel22...	Nov 20 – Dec 3	Critical	Done	M
✔ Add Forgot Password Option to Login Page	ru rushipatel22...	Nov 20 – Dec 3	High	Done	M
✔ Develop Regex-Based Password Validation	ru rushipatel22...	Nov 20 – Dec 3	Critical	Done	M
✔ Handle Detailed Error Messages for Login/Signup Failures	ru rushipatel22...	Nov 20 – Dec 3	Critical	Done	M
✔ Use Session Manager to Display Owner Data in All Fragments	ru rushipatel22...	Nov 20 – Dec 3	Critical	Done	M
✔ Add Navigation Arrows to Onboarding Screens	ru rushipatel22...	Nov 20 – Dec 3	Medium	Done	S
✔ Add Links to Privacy Policy and Terms of Use on Signup	ru rushipatel22...	Nov 20 – Dec 3	Medium	Done	M

Stories and Tasks

Story 1: Booking Confirmation and Notifications

✓ Display Booking Details on Confirmation Page	KD Kunal Dhiman
✓ Implement Booking Confirmation Notifications	KD Kunal Dhiman
✓ Implement Payment Status Notifications	KD Kunal Dhiman
✓ Implement Car Tracking in Parking Slots	KD Kunal Dhiman
✓ Send Notifications for Car Status Updates	KD Kunal Dhiman
✓ Integrate Parking Directions in Confirmation Page	KD Kunal Dhiman
✓ Add Booking Reminder Notifications	KD Kunal Dhiman







Story 2: UI Consistency and User Experience + ...

✓ Ensure Consistency in Location Titles Across Screens in Activities	KD Kunal Dhiman
✓ Review and Update UI for Dark Theme Compatibility	KD Kunal Dhiman
✓ Create UI for Feedback Form with proper button name and proper progress bar	ni nisargjoshi1...
✓ Implement and Design Quick Buttons on Home Screen with descriptions	ni nisargjoshi1...
✓ Optimize User Account Data Fetching and Implement Swipe-to-Refresh	RS Raghav Shar...
✓ Add Navigation Arrows to Onboarding Screens	ru rushipatel22...
✓ Add Placeholder Image and Message for Empty Favorites	ni nisargjoshi1...
✓ Handle Map Fetching Delays and Implement Progress Bar	RS Raghav Shar...



Story 3: Legal and Privacy Management

✓ Review and Update Privacy Policy and Display	ni nisargjoshi1...
✓ Update Terms of Use Document and Display	ni nisargjoshi1...
✓ Add Links to Privacy Policy and Terms of Use on Signup	ru rushipatel22...







Story 4: Promotion and Transaction Management

✓ Transaction Logic for Calculating Owner Income	 Raghav Shar...
✓ Fetch and Display Applied Promotions on Checkout	 nisargjoshi1...
✓ Track User Promotion Utilization in Database and Display Promotion Usage	 nisargjoshi1...
✓ Develop Logic for Promotions in Payment	 nisargjoshi1...
✓ Display Transaction History for Owners with Sorting	 Raghav Shar...
✓ Implement Refund Logic for Booking Cancellations	 Raghav Shar...





Story 5: Offline Functionality and Error Handling

✓ Optimize Data Sync for Offline Functionality	 Raghav Shar...
✓ Handle Errors in Offline Mode Using Error Page	 Raghav Shar...

Story 6: Owner Interface and Management Features

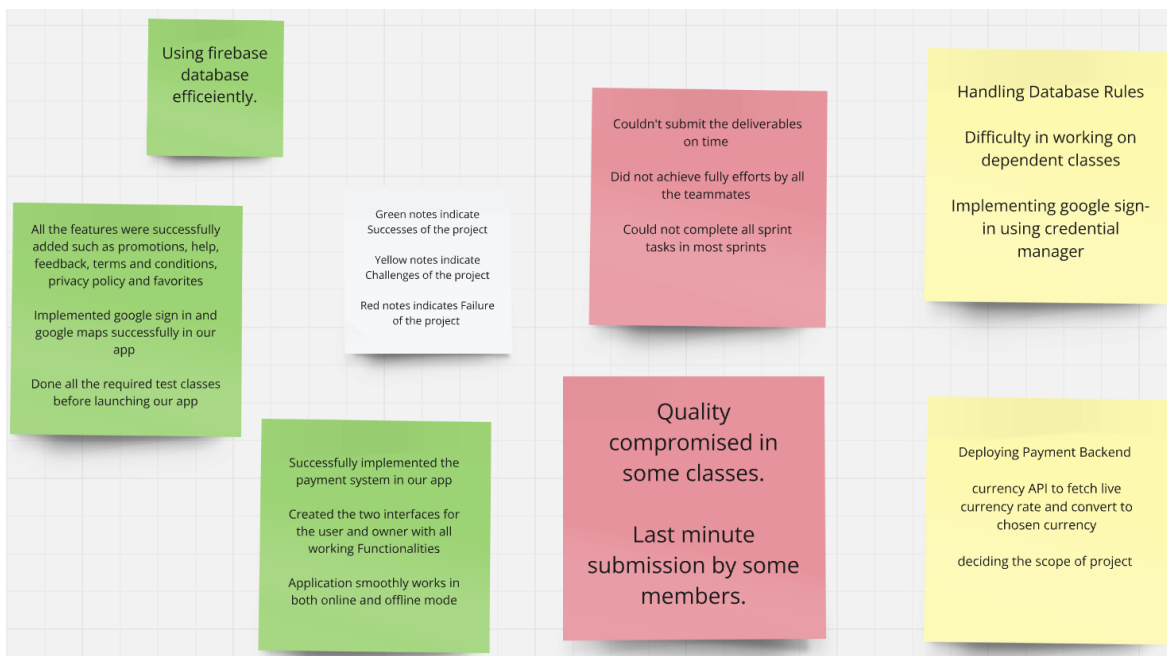
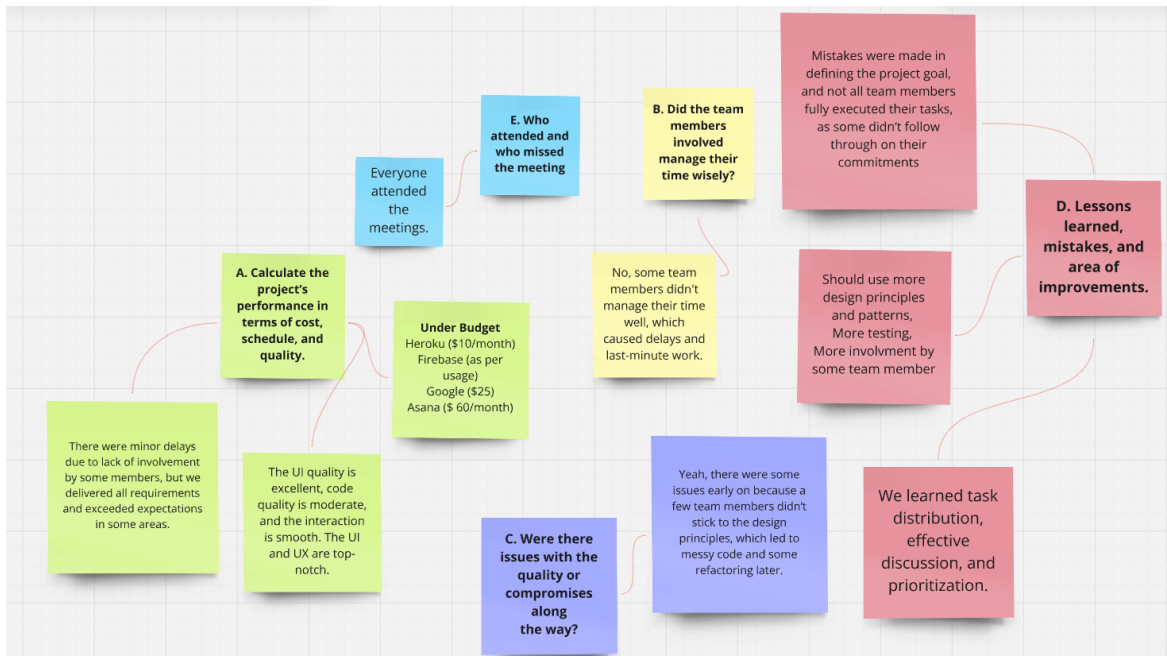
✓ Add Floating Action Button for Location Addition in Owner Interface	 Raghav Shar...
✓ Implement Swipe-to-Delete Function for Locations	 Raghav Shar...
✓ Add Income Display to Owner Dashboard Home Screen	 rushipatel22...
✓ Implement Real-Time Updates for Location Prices	 Raghav Shar...
✓ Design Action Buttons with Descriptions for Owner Dashboard in Card view for	 rushipatel22...
✓ Use Session Manager to Display Owner Data in All Fragments	 rushipatel22...

Story 7: Account Management and Authentication

✓ Handle Owner and User Login and Signup	 rushipatel22...
✓ Add Forgot Password Option to Login Page	 rushipatel22...
✓ Develop Regex-Based Password Validation	 rushipatel22...
✓ Handle Detailed Error Messages for Login/Signup Failures	 rushipatel22...

14. Post-Mortem/Project Review

Document a project review with performance evaluation, issues, and lessons learned



15. Technical Debt Addressing

Technical Debt Addressing:

- **Identifying Debt:** Reviewed code for inefficiencies and inconsistencies (e.g., SharedPreferences, Firebase queries).
- **Prioritization:** Used Scrum to focus on tasks impacting UX and scalability.

Key Actions:

1. **Centralized SharedPreferences:** Created SessionManager for cleaner, more maintainable data handling.
2. **Owner Dashboard:** Refactored data management, optimized queries, and integrated Stripe for payments.
3. **Unit Tests:** Added tests for core features like parking management and Stripe integration.

Value Delivered:

- Improved code quality, scalability, and performance with streamlined management and robust testing.

16. Code Refactoring

Document two areas where refactoring occurred, with examples and explanations.

a.) Refactoring Area: Validation Logic Extraction

- **Before Refactoring:** The validation logic was directly implemented within the `onConfirmButtonClicked ()` method, making the method long, difficult to read, and harder to maintain.

- **After Refactoring:** The validation logic was extracted into a separate class, **AddLocationValidator**. Each validation rule (for location name, postal code, price, and location address) was moved into its own static method in the AddLocationValidator class. This helps in Separation of Concerns, Modularization, Improved Readability, and Ease of Maintenance.

```
private void onConfirmButtonClicked() {
    if (!AddLocationValidator.isLocationNameValid(locationName, "Please enter the location name. "))
        return;
    if (!AddLocationValidator.isPostalCodeValid(postalCode, "Please enter the postal code. "))
        return;
    if (!AddLocationValidator.isPriceValid(
        price,
        "Please enter the price.",
        "Invalid price format.",
        "Price must be a positive value. "))
        return;
    if (!AddLocationValidator.isLocationAddressValid(locationAddress)) {
        Toast.makeText(context, this, "Please select a location using the search bar.", Toast.LENGTH_SHORT).show();
        return;
    }

    addParkingLocationToDatabase();
}
```

b.) Refactoring Area : **fetchTermsOfUse()** : Optimizing Data Handling with HashMap for Terms

Before Refactoring:

The code repeatedly checked for specific terms in documentSnapshot using multiple if statements and manually set values to corresponding TextViews. This was repetitive, inefficient, and hard to maintain.

```
1 usage
private void fetchTermsOfUse() {
    db.collection(COLLECTION_LEGAL).document("terms_of_use").get().addOnSuccessListener(new OnSuccessListener<DocumentSnapshot>() {
        @Override
        public void onSuccess(DocumentSnapshot documentSnapshot) {
            if (documentSnapshot.exists()) {
                try {
                    JSONObject jsonContent = new JSONObject();
                    if (documentSnapshot.contains("acceptance_of_terms")) {
                        jsonContent.put("acceptance_of_terms", documentSnapshot.getString("acceptance_of_terms"));
                    }
                    if (documentSnapshot.contains("purpose_of_the_app")) {
                        jsonContent.put("purpose_of_the_app", documentSnapshot.getString("purpose_of_the_app"));
                    }
                    if (documentSnapshot.contains("user_responsibilities")) {
                        jsonContent.put("user_responsibilities", documentSnapshot.getString("user_responsibilities"));
                    }
                    if (documentSnapshot.contains("booking_and_payment")) {
                        jsonContent.put("booking_and_payment", documentSnapshot.getString("booking_and_payment"));
                    }
                } catch (JSONException e) {
                    e.printStackTrace();
                }
            }
        }
    });
}
```

After Refactoring:

A HashMap was introduced to map terms to their respective TextViews. This streamlined the logic, reduced redundancy, and improved scalability for adding or managing terms.

```
1 usage
private void fetchTermsOfUse() {
    db.collection(COLLECTION_LEGAL).document( documentPath: "terms_of_use") DocumentReference
        .get() Task<DocumentSnapshot>
        .addOnSuccessListener(documentSnapshot -> {
            if (documentSnapshot.exists()) {
                updateTermsContent(documentSnapshot);
            }
        })
        .addOnFailureListener(e -> {
            if (getContext() != null) {
                Toast.makeText(getContext(), getString(R.string.default_content), Toast.LENGTH_SHORT).show();
            }
        });
}

1 usage
private void updateTermsContent(@NonNull DocumentSnapshot documentSnapshot) {
    for (String term : termsTextViews.keySet()) {
        String content = documentSnapshot.getString(term);
        updateTextView(term, content);
    }
}
```

17. DevOps Impact

If we had implemented DevOps practices in our parking app project, it would have significantly improved the development, deployment, and overall management of the app. Here's how these practices would have made a difference:

1. Continuous Integration (CI):

Regularly integrating source code changes, including Firebase, Google sign-in, and Stripe integrations, would have enabled early issue detection. Automated tests would have ensured new features didn't break existing functionality.

2. Continuous Delivery (CD):

Automating deployments would have reduced human error, streamlined feature rollouts, and allowed for efficient rollbacks during issues, ensuring faster and more reliable updates.

3. Microservices Architecture:

Separating components like user authentication, parking lot management, and

payment processing would have enabled independent scaling, faster development, and better fault isolation.

4. **Monitoring and Logging:**

Integrating monitoring tools would have helped track user logins, payments, and parking slot updates, enabling quick issue identification and troubleshooting.

5. **Collaboration and Communication:**

Using tools like Slack or Jira would have enhanced coordination during feature additions and problem-solving, ensuring efficient integration of third-party services like Firebase and Stripe.

6. **Automated Testing:**

Automated tests would have validated critical flows, including sign-ins, payments, and parking lot updates, saving time and reducing errors.

7. **Deployment Automation:**

Automating deployment processes with tools like Docker would ensure consistent configurations across environments, enabling quick rollbacks and minimizing downtime.

8. **Conclusion:**

DevOps practices would have improved scalability, reliability, and development efficiency, allowing faster releases and enhancing the user experience for both app users and parking lot owners.

18. Coding Standards

Describe the coding standards followed and how they were applied.

Coding Standards Followed:

1. **Camel Casing:** Variables and method names follow camelCase (e.g., progressBar, generatePassKey()).
2. **Capital Letters for Classes:** Class and interface names use PascalCase (e.g., BookingManger, SlotService).
3. **Consistent Naming:** Descriptive and meaningful names for classes, methods, and variables. Constants are written in uppercase (e.g., USER_TYPE_OWNER).

4. **Error Handling:** Proper exception handling with clear error messages and using callback functions where necessary.
5. **Avoidance of Deprecated Methods:** Deprecated methods are avoided to ensure the codebase remains up to date with current standards.
6. **Documentation:** Clear inline comments and Javadoc are provided for complex logic and classes to improve code readability.
7. **SonarLint Integration:** SonarLint is used to detect code quality issues and ensure compliance with best practices in real-time.

19. Security Vulnerability

Identify at least one security vulnerability in your code and propose future solutions.

Hardcoded API Keys and Client IDs:

APP_CLIENT_ID, API_PLACE_SEARCH, and publishable Key in PaymentConfiguration.init() are visible in the source code.

This exposes them to **potential attackers, risking unauthorized** access or misuse.

```
public static final String COLLECTION_LOCATION_OWNER = "parkingLocationIds"; 5 usages
public static final String APP_CLIENT_ID = "361561575523-vh9n3c4q3brm58ejsqgj8s9henk08515.apps.googleusercontent.com";
public static final int SPLASH_SCREEN_TIME_OUT = 3000; // used in Splash Screen 2 usages
public static final String API_PLACE_SEARCH = "AIzaSyCBb9Vvk3FUhAz6Tf7ixMIk5xqu3IGLZRd0"; 2 usages
```

Proposed Future Solutions

A. Use Environment Variables (for development):

Store sensitive keys and configuration values in environment variables, which can be injected into the app at build time or runtime.

B. Use a Secure Backend (Server-Side API Key Management):

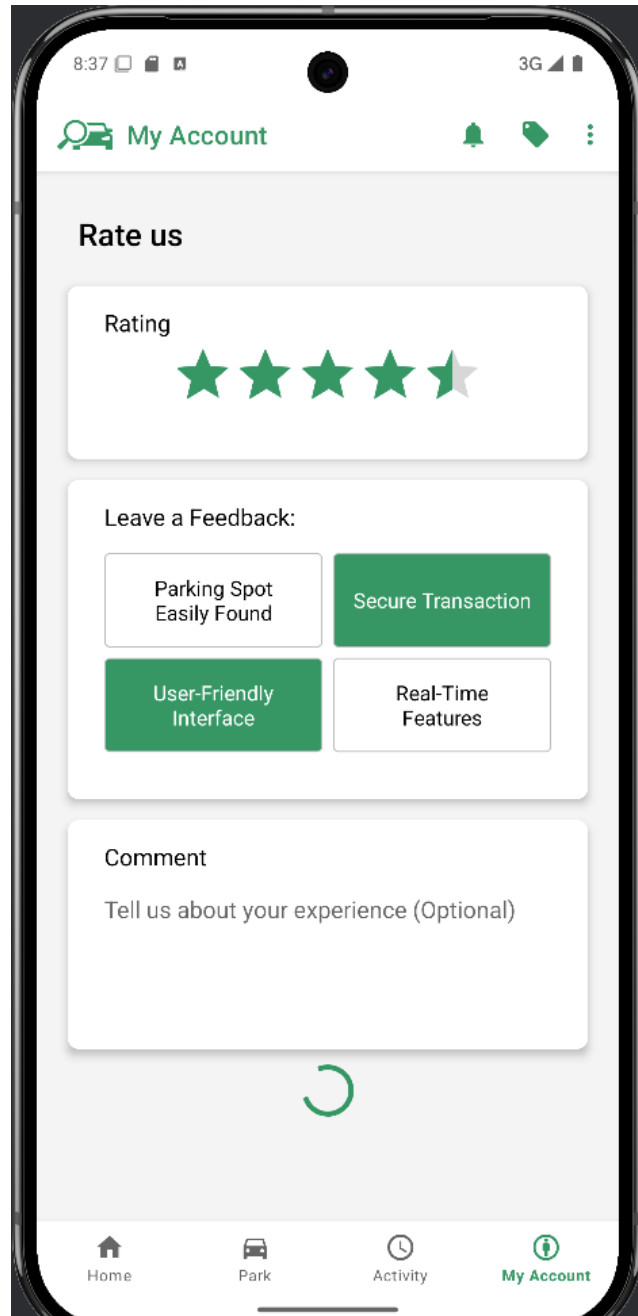
Move sensitive keys, like API client IDs and publishable keys, to your backend and only expose the necessary keys to the app at runtime.

C. Use Secrets Management Services:

Consider using dedicated secrets management services like **AWS Secrets Manager**, **Azure Key Vault**, or **Google Cloud Secret Manager** to securely store and access API keys and other sensitive information.

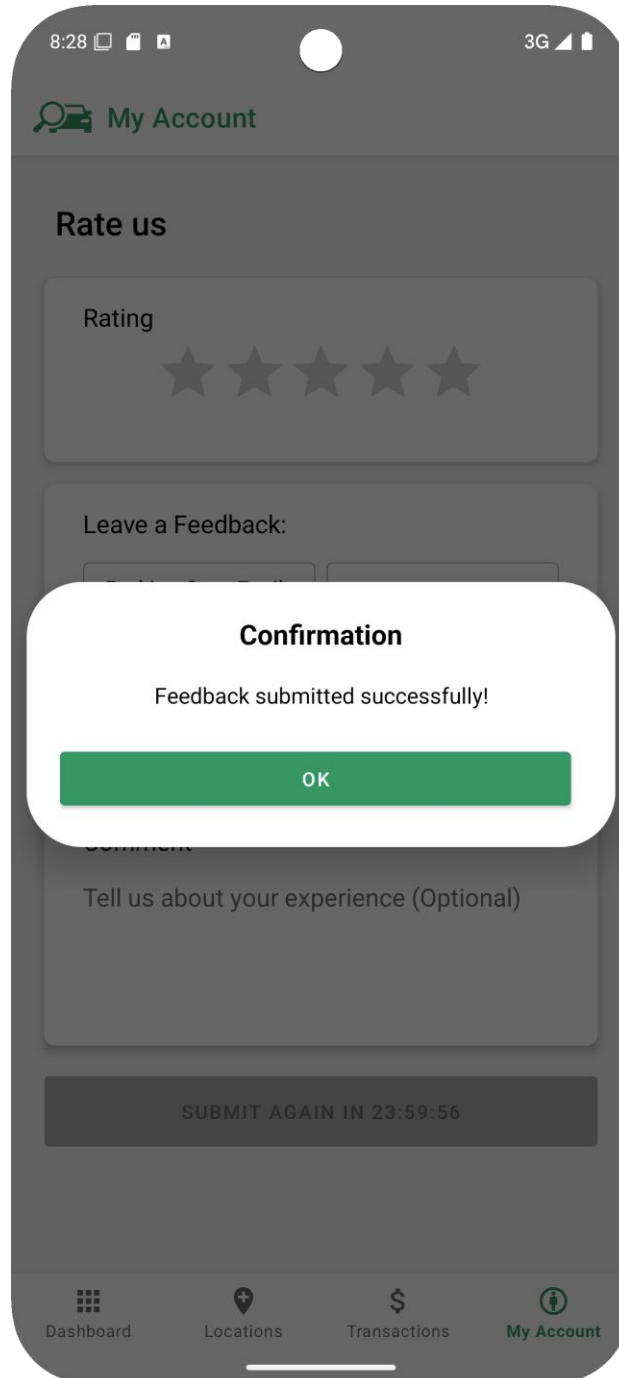
20. Progress Bar on Feedback Screen

Provide a screenshot showing the progress bar during feedback form submission.



21. Alert Dialog on Successful Form Submission

Provide a screenshot of the Alert Dialog displayed after form submission.



22. Data Stored in Database from Feedback

Show screenshots of at least 3 entries stored in the database from the feedback form.

This screenshot shows the Google Cloud Firestore console for the 'feedback' collection. The document ID is 'Q0LyDPupR6CHMY1gEygd'. The document contains the following fields:

- comment: "Really Helpful application, specially for the busy areas"
- deviceModel: "Google sdk_gphone64_x86_64"
- rating: 5
- selectedOptions: [0 "Real-Time Features", 1 "Parking Spot Easily Found", 2 "User-Friendly Interface"]
- userEmail: "nisargjoshi1519@gmail.com"
- userName: "Nisarg Joshi"
- userPhone: "6478369746"
- userType: "user"

This screenshot shows the Google Cloud Firestore console for the 'feedback' collection. The document ID is 'ZpPu4XSqLL6Fjcppne6'. The document contains the following fields:

- comment: ""
- deviceModel: "Xiaomi Redmi K20 Pro"
- rating: 3
- selectedOptions: [0 "Secure Transaction", 1 "User-Friendly Interface", 2 "Parking Spot Easily Found"]
- userEmail: "testingpark@gmail.com"
- userName: "Owner"
- userPhone: "4342342342"
- userType: "owner"

This screenshot shows the Google Cloud Firestore console for the 'feedback' collection. The document ID is 'e0AHhb1Y4LvsWp7CCGvz'. The document contains the following fields:

- comment: ""
- deviceModel: "Xiaomi Redmi K20 Pro"
- rating: 3.5
- selectedOptions: [0 "Secure Transaction", 1 "User-Friendly Interface"]
- userEmail: "raghav1453sharma@gmail.com"
- userName: "RAGHAV SHARMA"
- userPhone: null
- userType: "user"

23. Test Case Strategy

Our strategy ensures comprehensive and maintainable test coverage by:

1. Separation of Concerns:

- a. Keeping logic separate from activities and fragments to make code testable.

2. Prioritization:

- a. Writing test cases for critical workflows, such as bookings, currencyManager, and network.
- b. Covering both positive and negative scenarios.

3. Test Types:

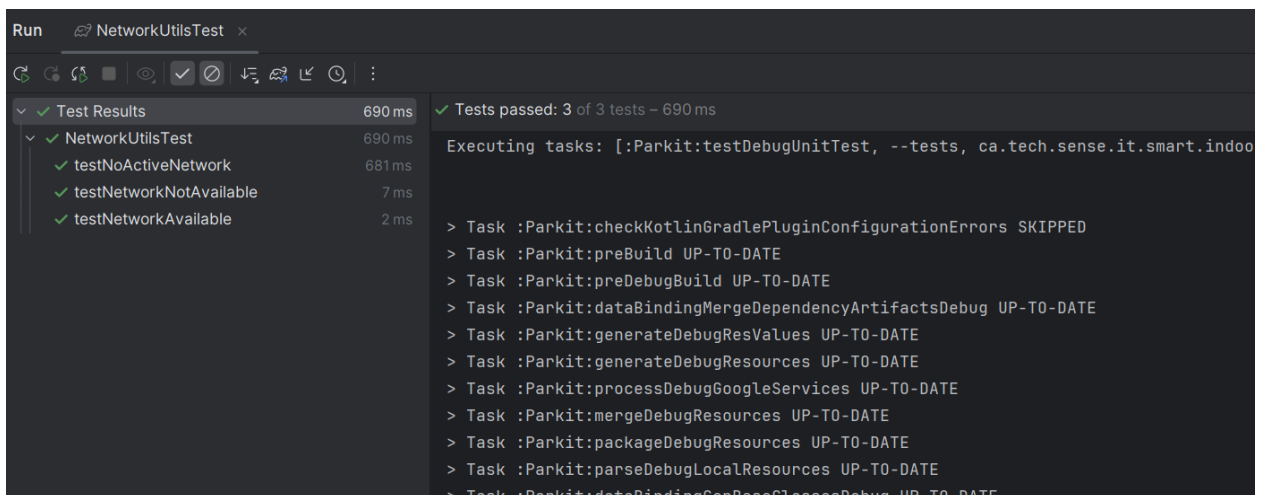
- a. **Unit Tests (JUnit):** For testing business logic in isolation.
- b. **Integration Tests (Robolectric):** To validate interactions between UI components and View Models.
- c. **UI Tests (Espresso):** To test app navigation and user interactions.

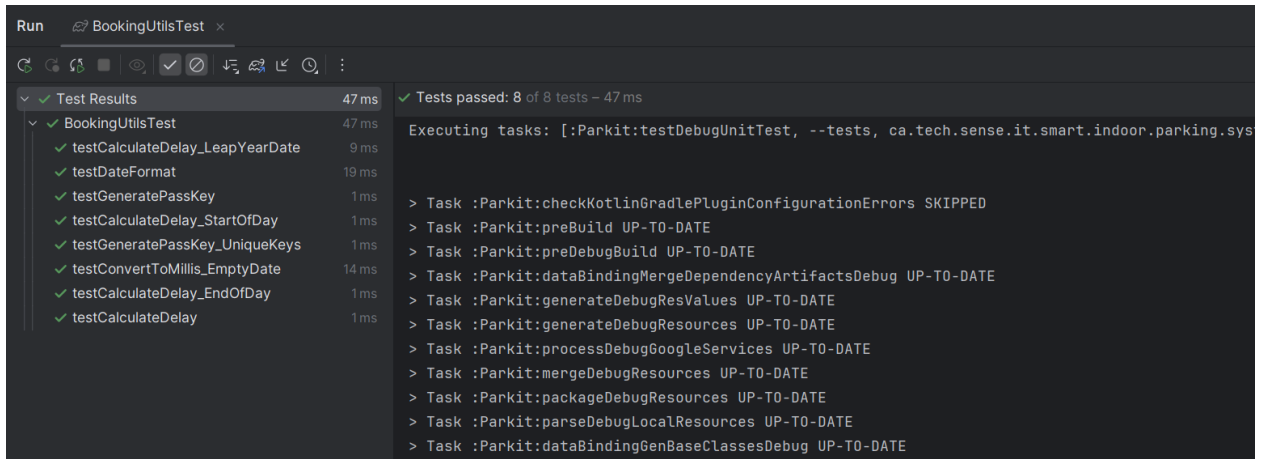
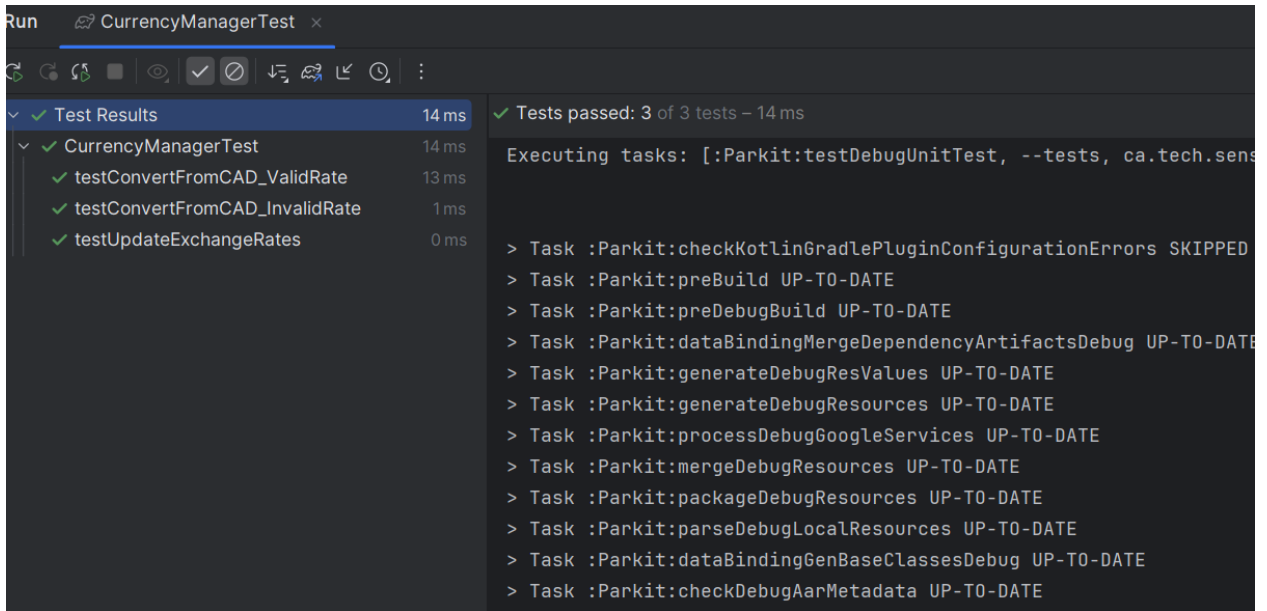
4. Reusability:

- a. Creating reusable test setups to reduce redundancy.

24. JUnit Test Cases

Provide screenshots showing successful JUnit 3 test cases.



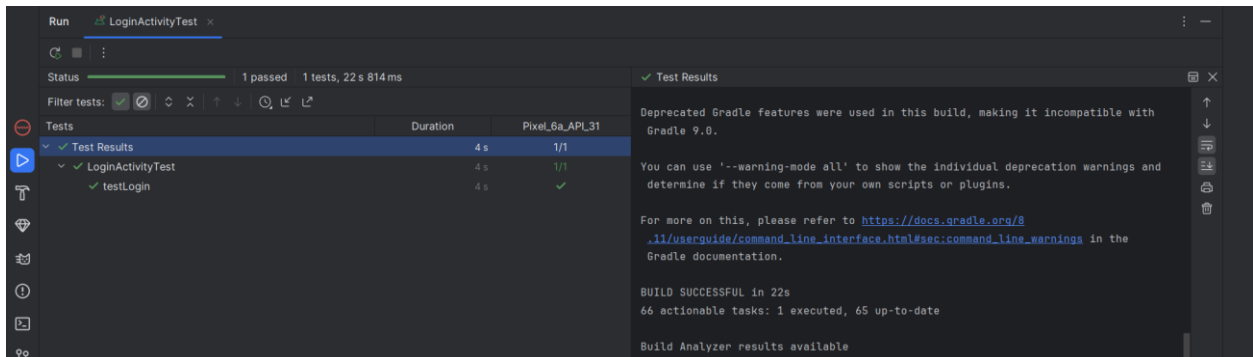
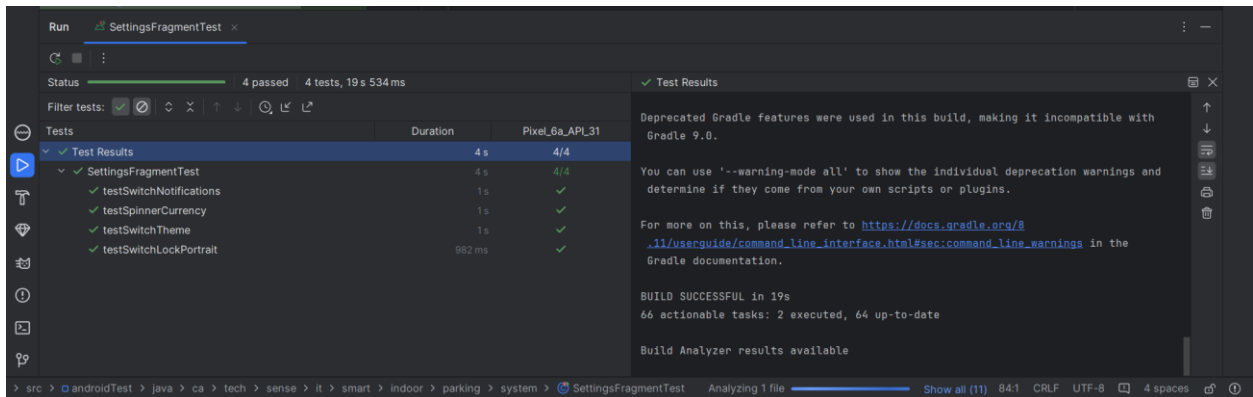
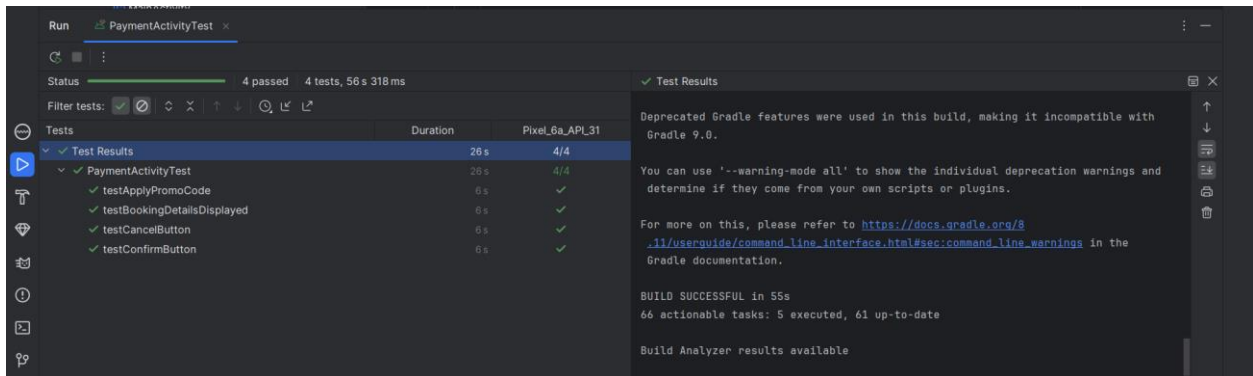


25. Robletric Test Cases

Provide screenshots showing successful Robletric test case.

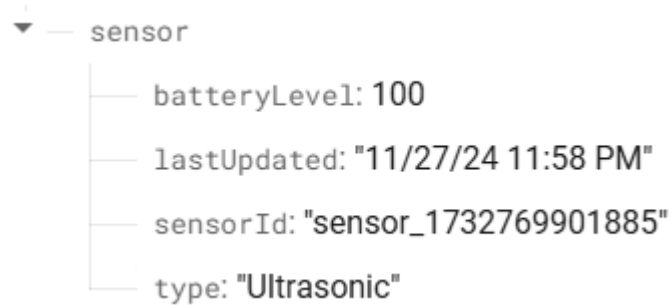
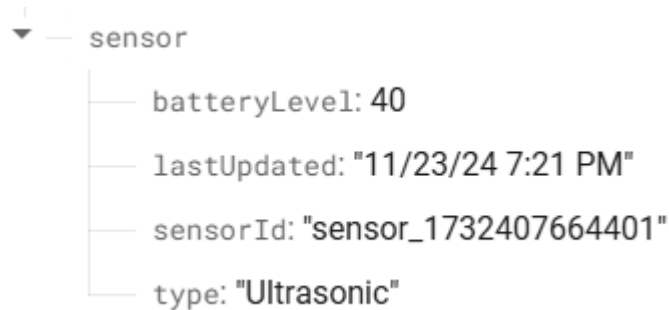
26. Espresso Test Cases
















Provide screenshots showing successful Espresso test cases.



27. DB Data Screenshots

Provide screenshots showing data from DB, sensors, login data (including Google login), and feedback screen.



testingpark@gmail.com		Nov 18, 2024	Dec 2, 2024	e65Gme5dxdXtf2DrNkMxEp...
xrdc@dtmf.com		Nov 17, 2024	Nov 17, 2024	zCkDSykOcHZ83cfvWQEdxV2...
rushipatel22112003@g...		Nov 17, 2024	Nov 17, 2024	b5zzfIPWe7XkMxdwC3GRDtk...
r@gmail.com		Nov 17, 2024	Nov 17, 2024	8xC4cLXfshdGzZCnY7Y7Tizm...
testingparkit@gmail.com	 	Nov 16, 2024	Dec 1, 2024	ILjIQhteW6b2NHNB8vlyv8GN...
sachinmirok02@gmail...		Nov 15, 2024	Nov 15, 2024	VNWpqnEPcYXqsemiWe0OnX...
dhimankunalkd.kd@gm...		Nov 14, 2024	Nov 14, 2024	koOhSsPTBGOEI8mYMOKHu7...
rushii@gmail.com		Nov 13, 2024	Dec 2, 2024	i156lNdoxZcGLckcVeFDvrMe...
rushi@gmail.com		Nov 13, 2024	Dec 2, 2024	3rZxVSuW6wSqyiwWEAyfd9q...
aditipatel@gmail.com		Nov 11, 2024	Nov 11, 2024	K0bJfdkpxzfxD1JOss7F3AXa...
nisargjoshi1519@gmail...		Nov 11, 2024	Dec 2, 2024	g7ZnJ1Av77gO4ZFaAd48qf04...
kunal.dhiman991@gma...		Nov 10, 2024	Nov 14, 2024	3MRU1DGzZNTwMcDVXuZcp...
kunaldhimankd.kd@gm...		Nov 10, 2024	Dec 3, 2024	GyqWi1biHLQG9eZHZuskxHC...
raghav1453sharma@g...		Nov 10, 2024	Dec 2, 2024	0ys6pwOYYjN9FCEhUFYL9mU...



sensor

batteryLevel: 90

lastUpdated: "01/12/2024 6:44 pm"

sensorId: "sensor_1733096647036"

type: "Infra"

<div> <div> <div></div> <div>feedback > Q0LyDPupR6CH.</div> </div> <div>More in Google Cloud</div> </div>		
<div> <div>(default)</div> <div> <div>+ Start collection</div> <div>feedback ></div> <div> <div>fees</div> <div>help</div> <div>legal</div> <div>owners</div> <div>users</div> </div> </div> </div>	<div> <div>feedback</div> <div> <div>+ Add document</div> <div> <div>6hfkqG7cmtOKTQbkFpYu</div> <div>Q0LyDPupR6CHMY1gEygd ></div> <div>R9gUypQHwUE1hC8uoVKV</div> <div>ZpPu4XSqxLL6Fjcppne6</div> <div>e0AHHbiY4LvsWp7CCGvz</div> <div>nqqvrB9SFnHNCVNpRd61</div> </div> </div> </div>	<div> <div>Q0LyDPupR6CHMY1gEygd</div> <div> <div>+ Start collection</div> <div>+ Add field</div> <div> <div>comment: "Really Helpful application, specially for the busy areas"</div> <div>deviceModel: "Google sdk_gphone64_x86_64"</div> <div>rating: 5</div> <div>selectedOptions <div>0 "Real-Time Features"</div> <div>1 "Parking Spot Easily Found"</div> <div>2 "User-Friendly Interface"</div> </div> <div>userEmail: "nisargjoshi1519@gmail.com"</div> <div>userName: "Nisarg Joshi"</div> <div>userPhone: "6478369746"</div> <div>userType: "user"</div> </div> </div> </div>

<div> <div> <div></div> <div>feedback > ZpPu4XSqxLL6Fjcppne6</div> </div> <div>More in Google Cloud</div> </div>		
<div> <div>(default)</div> <div> <div>+ Start collection</div> <div>feedback ></div> <div> <div>fees</div> <div>help</div> <div>legal</div> <div>owners</div> <div>users</div> </div> </div> </div>	<div> <div>feedback</div> <div> <div>+ Add document</div> <div> <div>6hfkqG7cmtOKTQbkFpYu</div> <div>Q0LyDPupR6CHMY1gEygd</div> <div>R9gUypQHwUE1hC8uoVKV</div> <div>ZpPu4XSqxLL6Fjcppne6 ></div> <div>e0AHHbiY4LvsWp7CCGvz</div> <div>nqqvrB9SFnHNCVNpRd61</div> </div> </div> </div>	<div> <div>ZpPu4XSqxLL6Fjcppne6</div> <div> <div>+ Start collection</div> <div>+ Add field</div> <div> <div>comment: ""</div> <div>deviceModel: "Xiaomi Redmi K20 Pro"</div> <div>rating: 3</div> <div>selectedOptions <div>0 "Secure Transaction"</div> <div>1 "User-Friendly Interface"</div> <div>2 "Parking Spot Easily Found"</div> </div> <div>userEmail: "testingpark@gmail.com"</div> <div>userName: "Owner"</div> <div>userPhone: "4342342342"</div> <div>userType: "owner"</div> </div> </div> </div>

<div> <div> <div></div> <div>feedback > e0AHHbiY4LvsWp7CCGvz</div> </div> <div>More in Google Cloud</div> </div>		
<div> <div>(default)</div> <div> <div>+ Start collection</div> <div>feedback ></div> <div> <div>fees</div> <div>help</div> <div>legal</div> <div>owners</div> <div>users</div> </div> </div> </div>	<div> <div>feedback</div> <div> <div>+ Add document</div> <div> <div>6hfkqG7cmtOKTQbkFpYu</div> <div>Q0LyDPupR6CHMY1gEygd</div> <div>R9gUypQHwUE1hC8uoVKV</div> <div>ZpPu4XSqxLL6Fjcppne6</div> <div>e0AHHbiY4LvsWp7CCGvz ></div> <div>nqqvrB9SFnHNCVNpRd61</div> </div> </div> </div>	<div> <div>e0AHHbiY4LvsWp7CCGvz</div> <div> <div>+ Start collection</div> <div>+ Add field</div> <div> <div>comment: ""</div> <div>deviceModel: "Xiaomi Redmi K20 Pro"</div> <div>rating: 3.5</div> <div>selectedOptions <div>0 "Secure Transaction"</div> <div>1 "User-Friendly Interface"</div> </div> <div>userEmail: "raghav1453sharma@gmail.com"</div> <div>userName: "RAGHAV SHARMA"</div> <div>userPhone: null</div> <div>userType: "user"</div> </div> </div> </div>

28. Suggestions for Future Projects

Offer suggestions for improvement and things you liked about the project.

Suggestion for Offline Classes

It would be helpful to have more opportunities for in-person feedback or meetings during the course. Scheduled check-ins or face-to-face discussions with the instructor could provide more direct communication and be more effective than relying solely on online interactions.

Regular Quizzes for Knowledge Reinforcement:

Offering at least two quizzes per quarter would be a great addition. This would help in reinforcing the concepts and reviewing the material more frequently, ensuring that students retain information better and stay engaged with the learning process. Quizzes could be designed to cover a mix of topics, ensuring that all key areas are reviewed regularly.