Final Year Project Report

Full Unit – Interim Report

Building Mobile Application

Nway Nway Than Hlaing

A report submitted in part fulfilment of the degree of

BSc (Hons) in Computer Science

Supervisor: Prof. Khuong Nguyen

Department of Computer Science

Royal Holloway, University of London

December 13, 2024

Declaration

This report has been prepared on the basis of my own work. Where other published and unpublished source materials have been used, these have been acknowledged.

Word Count: 6391 words

Student Name: Nway Nway Than Hlaing

Date of Submission: December 13, 2024

Signature: Nway Nway Than Hlaing

Table of Contents

Abstract

The well-being of pets is closely linked to the consistency and quality of care they receive. While many pet owners believe they are providing their animals with quality care [1], a lack of proper care can still lead to significant pet welfare issues [2]. To address these problems, this project introduces **PURRNOTE**, a mobile application designed to streamline pet care management.

PURRNOTE combines essential tools into a single platform, including personalized pet profiles, diet and activity tracking, and an integrated calendar with reminders for events such as feeding, exercise, grooming, and veterinary visits. The app also features a reminder notification system, vet locator, and a social networking feature that allows users to share tips, advice, and memorable moments with their pets such as travelling and activities done together. While it is built using React Native for cross-platform compatibility and Firebase for secure backend support, PURRNOTE ensures a valuable and efficient user experience.

The development process followed a structured approach, beginning with research and user requirements gathering, followed by design prototyping, implementation, and testing. Key features, including user account management, pet profiles management, reminders, bookings, calendar integration, and notification inbox, have been implemented. The development of the vet locator, social networking feature, and enhancements to the overall user experience are still in progress.

By providing a centralized, user-friendly solution, PURRNOTE aims to enhance the quality of life for both pets and their owners, ensuring consistent care and improving the management of routines important for the pets. By having an organized pet care routine, and tracking activities, pet owners will understand more about their pets and will be able to make a stronger bond with their pets. Upon completion, the app promises to be a valuable tool for pet owners to efficiently manage their pets' wellbeing.

Chapter 1: **Introduction**

1.1 Problem Statement

Pet ownership involves significant responsibilities, particularly in ensuring that pets receive consistent and appropriate care. While most pet owners recognize the importance of feeding, exercise, grooming, and regular veterinary visits, balancing these tasks can be challenging, especially for those with busy lifestyles. This often leads to difficulties in staying on top of essential care activities, creating stress and confusion for pet owners, and ultimately affecting their pets' overall well-being. The lack of routine in managing feeding, exercise, grooming, and veterinary care can lead to disorganization, stress, and potential health risks for both pets and their owners [3].

Pet nutritional needs vary by species, breed, and health status, adding complexity to pet care. For instance, dogs, as omnivores, require a balanced diet of proteins, fats, and carbohydrates, while cats, as obligate carnivores, have specialized dietary needs [4]. PURRNOTE allows users to create multiple pet profiles, storing each pet's details and establishing individual or shared routines. It also features a calendar that shows reminders and allows users to update each reminder with a simple click on the event.

Additionally, regular veterinary care is vital for long-term pet health, and pets require vaccines, spaying or neutering, and check-ups. Given the importance of grooming and veterinary appointments, PURRNOTE offers a dedicated view for appointments, allowing users to see all appointments at once or filter them by pet.

While there are many pet care apps available, most focus on only specific aspects of care such as veterinary appointments, diet tracking, pet monitoring, calendar with event reminders, or vet locator. There is a clear gap in the market for a comprehensive solution that simplifies the management of all these tasks in one place. PURRNOTE addresses this gap by offering an integrated platform where pet owners can manage schedules, set reminders for appointment and tasks, social platform for all within a single, user-friendly app.

1.2 Literature Review

The increasing popularity of mobile apps has transformed many industries, with health and wellness apps experiencing significant growth. Pet care apps have become a valuable tool for pet owners, allowing them to manage and monitor their pets' health and activities. This literature review explores existing research and practices related to the development of mobile apps in the pet care sector, social networking features in apps, the implementation of notifications and reminders.

Existing Pet Care Apps

The number of mobile apps is increasing at a fast pace, and more people are downloading apps on various devices daily. As smartphone usage continues to rise, the mobile app industry is expected to keep expanding, with more users adopting mobile apps in the future [14]. Mobile app downloads have been steadily increasing each year, reaching billions globally between 2016 and 2020, and this trend shows no sign of slowing down [14].

Many existing pet care apps focus primarily on task management but miss the opportunity to build a community around pets. This gap offers a significant chance for innovation by combining pet care management with social features, which could enhance user retention and overall app success. For instance, apps like 11Pets [10], Pawmates [11], Doglog [12], and Mypet Reminder [9] provide separate features for task management, social interaction, and calendar integration, but do not combine them into a single cohesive experience. Additionally, while there are standalone **vet locator**

apps, there is limited research on integrating all these features into one unified app. This creates a clear opportunity to offer a more comprehensive solution that combines pet care, social networking, and location-based services in a single platform.

While each of these apps offers valuable features, they do not fully meet the needs of pet owners who are looking for an all-in-one solution. **11Pets** offers a range of useful features, including appointment booking, behavior monitoring, and activity tracking. However, it lacks social features, calendar integration for reminders, and a vet locator [10]. On the other hand, **Pawmates** provides social features, a vet locator, and a pet food store locator, but does not include task management capabilities [11]. Similarly, while **Doglog** offers social features and reminder functionality but lacks calendar integration and a vet locator [12], MyPet Reminders provides calendar-integrated reminders and supports multiple pet profiles, catering to cats and dogs [9]. **Purrnote** addresses these gaps by combining all these essential features such as task management, reminder notification, social interaction, calendar integration, and a vet locator, into a single app, offering a comprehensive solution for pet owners.

Chapter 2: **Project Specification**

2.1 Project Overview

The goal of this project is to develop PURRNOTE, a mobile application designed to simplify pet care management. The app addresses common challenges by offering a unified platform that manages routines, scheduled events, a vet locator, and social networking for pet owners. Unlike many pet care apps that focus on individual features, PURRNOTE integrates these functionalities into one app, enabling pet owners to provide organized and consistent care for their pets. It is available on both iOS and Android platforms, ensuring accessibility to a wide audience of pet owners.

2.2 Objectives and Aims

The primary objective of this project is to create a functional mobile pet care application that helps users easily track and manage their pets' data. Specific aims for the project include:

- Simplify Pet Care Management: Centralize pet care tasks in one app, making it easier for owners to stay organized.
- Enhance Organization and Reminders: Implement a reminder system for pet care tasks, ensuring timely actions.
- Encourage a social community: Develop a social networking section for pet owners to share their memories or experiences with their pets.
- Ensure Data Security: Safeguard user and pet data using secure backend systems.

2.3 Key Features

1. **User Authentication**: Secure login and account creation, enabling users to access personalized features and data.

2. **User Profiles**: Users can create and manage their profiles, including updating personal information, changing profile images, modifying passwords, logging out, and deleting their accounts.

3. **Pet Profiles**: Users can create personalized profiles for each pet, storing key details such as name, breed, age, color, gender, weight, and a description of the pet's personality.

4. **Calendar**: The calendar helps users set reminders for daily routines. Users can view and manage their schedule by selecting a specific date, or by choosing a particular month and year. Additionally, users can easily perform CRUD (Create, Read, Update, Delete) operations on events by simply clicking on them.

5. **Appointment Management**: Users can view and manage veterinary appointments, grooming sessions, and other important tasks. When creating an event, users can specify whether it's an appointment. Appointments or bookings can be viewed separately by selecting a specific date or filtered by pet to view individual appointments.

6. **Push Notifications**: Real-time alerts for tasks will be sent to the user at the time of event, ensuring pet owners stay on top of their responsibilities.

7. **Vet Locator**: A feature to help users find nearby veterinary services, ensuring access to essential pet care.

8. **Community Engagement**: Connect with other pet owners, share advice, tips, and experiences through the app's social networking feature.

2.4 Scope and Deliverables

Scope:

- The project will focus on developing the mobile app, which includes features such as scheduling reminders for daily routines, and appointments, handling multiple pet profiles, calendar integration, vet locator, and community engagement as described in 2.3.
- The app will not include hardware integration, and it will focus on software-based management of pet care tasks in this project.
- The app will be compatible with iOS and Android devices, ensuring accessibility across a wide range of smartphones.

Deliverables:

1. Functional Mobile App: A fully working version of PURRNOTE that integrates the core features.
2. Documentation: Comprehensive technical documentation, including code descriptions and user guides.
3. User Manual Video Demonstration: A complete video guide explaining how to use the app's features.
4. Future Work Plan: A roadmap outlining future app features or improvements that could be added after the initial delivery.

2.5 Technology Stack

The PURRNOTE mobile application will utilize the following technologies to enhance performance, real-time data synchronization, simplified app testing during implementation.

1. **React Native for Frontend**

   React Native is a JavaScript framework that allows cross platform development for developers to write a single codebase to work on both iOS and Android platforms, making it easier and faster to develop mobile applications for both operating systems simultaneously [5].

   Key Features:

   - Hot reloading (live updates during development),
   - Component-based architecture (reusable code for UI elements),
   - Broad community support, and
   - Native module integration (ability to use native code for advanced features).

2. **Firebase for Backend**

   Firebase is a cloud-based platform that provides backend services such as data storage, user authentication, and real time database synchronization as described in [6]. Firebase authentication supports secure login system and firestore database offers a real time NoSQL database for managing data [6].

   Key Features:

   - Real-time database updates (instant updates across devices),
   - Automatic synchronization (data remains consistent across devices),

- Scalable backend system (can handle increased load as the app grows), and
- Secure user authentication (ensures safe login and access management).

3. Cloudinary (Cloud Storage for Images)

Cloudinary is a cloud storage service for uploading and managing images such as user and pet profile photos that provides tools to upload, store and optimize images [7]. It helps manage media files and ensures they load quickly without affecting app performance.

Key Features:

- Image optimization (automatically compresses and adjusts images for faster loading),
- Secure storage (ensures media is stored safely), and
- URL-based media management (enables easy linking and access to media).

4. Native Notify (Push Notifications)

Native Notify enables apps to send notifications to users even when the app is not actively being used. It also integrates smoothly with React Native, allowing for timely push notifications related to pet care reminders.

Key Features:

- Real-time notifications (immediate delivery of messages),
- Easy integration with mobile apps (simple setup with React Native), and
- Customizable notification options (personalize message content and delivery).

5. Expo Go (Testing on Physical devices)

Expo Go is a mobile app for running and testing React Native apps on physical devices or simulators. It supports real time interaction on both iOS and Android devices, making it easier to test during development without needing to compile the app manually [5]. It simplifies the testing process since it reflects on the device immediately without the need for separate builds for each platform [5].

Key Features:

- Instant device preview,
- Real-time code updates.

Chapter 3: **Software Engineering**

The development of PURRNOTE follows the Agile methodology, which supports flexibility, iterative progress, and continuous improvement. This approach is well-suited for app development, where regular feedback and adjustments are necessary.

3.1 Methodology

The development of PURRNOTE followed the **Agile methodology**, which emphasizes flexibility, iterative progress, and continuous improvement [8]. Agile is a project management framework that promotes collaboration, responsiveness to change, and delivering work in small, incremental phases known as sprints [8]. As an individual project, Agile principles were adopted to manage the development process, breaking the work into smaller, manageable phases. This approach allowed for regular testing, feedback, and refinement of features, ensuring the app evolved to meet user needs and project objectives.

Throughout the semester, meetings were held with the supervisor, to review the progress of the project and gather feedback. These meetings provided valuable guidance, ensuring that the app met the intended objectives and enabling adjustments to be made based on the feedback received.

Sprint Phases:

- Planning: Each sprint begins with a planning meeting to define the goals and features to be developed.
- Development: The development phase involves coding and implementing the discussed features for the sprint.
- Testing: Once planned features are developed, it undergoes thorough testing to ensure it works as expected.
- Review and Feedback: After testing, the feature is reviewed, and feedback is collected to refine or improve it in future sprints.

3.2 Design Approach

The design approach focuses on providing a user-centered experience, with an emphasis on usability, simplicity, and accessibility. The design follows a clean, simple, and intuitive layout, with easy navigation and clear instructions for the pet owner. The color scheme, typography, and icons were chosen to be friendly, engaging, and easy to read on mobile screens.

Wireframing and Prototyping:

- Initial wireframes and prototypes were created to visualize the app's layout, navigation, and user interface (UI). Designing tool, Figma, was used to design the app's UI.
- After creating the wireframes, prototypes were designed to visualize the final look of the app, including structure, colors, fonts, and overall design.

3.3 System Architecture

3.3.1 Design Pattern

To ensure maintainability and flexibility, **Model-View-Controller (MVC) design pattern** have been used. This pattern separates the app's data (Model), user interface (View), and business logic (Controller), making it easier to maintain and expand.

- **Model** manages the data used in the application. It handles the storage, retrieval, and update of data, ensuring data integrity and security.
- **View** is responsible for the presentation layer of the application. It displays data provided by the Model in a format suitable for user interaction, typically a user interface where users can perform actions.
- **Controller:** Acts as an intermediary between the Model and the View. It processes all user inputs, sends requests to the Model, and updates the View accordingly.

This structured separation not only enhances manageability and scalability but also improves the ease of maintenance and testing. The MVC pattern optimizes the development process, enabling more effective management of complex projects through a separation of concerns. This architectural approach provides clarity and structure throughout the application lifecycle.

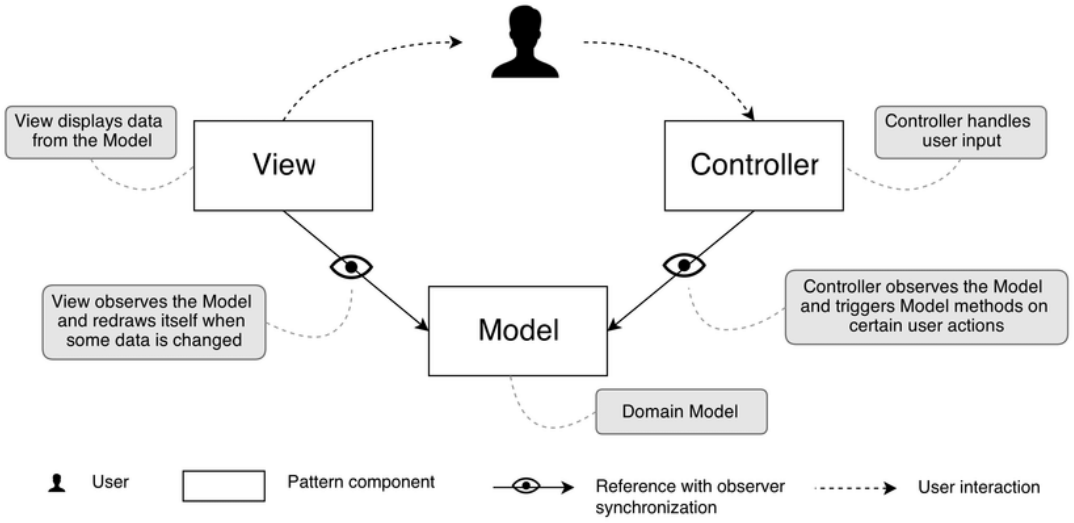

Figure 1. Structure of MVC Architectural Design Pattern [15]

3.3.2 System Design

System design is the overall process of defining the architecture, components and data flow of a system. The figure below illustrates the system design of the app. React Native acts as the core framework, handling user interactions and sending API requests to Firebase for backend operations such as authentication and data management. Cloudinary is used for media uploads, while Native Notify manages user notifications. Expo Go facilitates app testing and user interface display on devices.

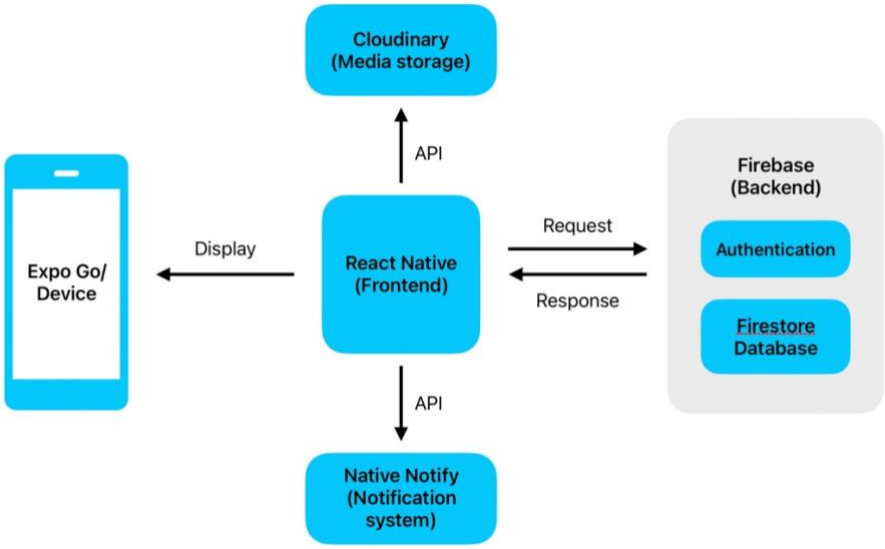

Figure 2. System Architecture Diagram of PURRNOTE

UML Diagram

Structure and behavior of a system is visualized by UML (Unified Modelling Language) diagram. The activity diagram below illustrates the user authentication workflow, detailing the sequence of steps for accessing protected and unprotected pages. It highlights decision points, actions, and outcomes in scenarios such as signing in, registering an account, or handling failed sign-ins.

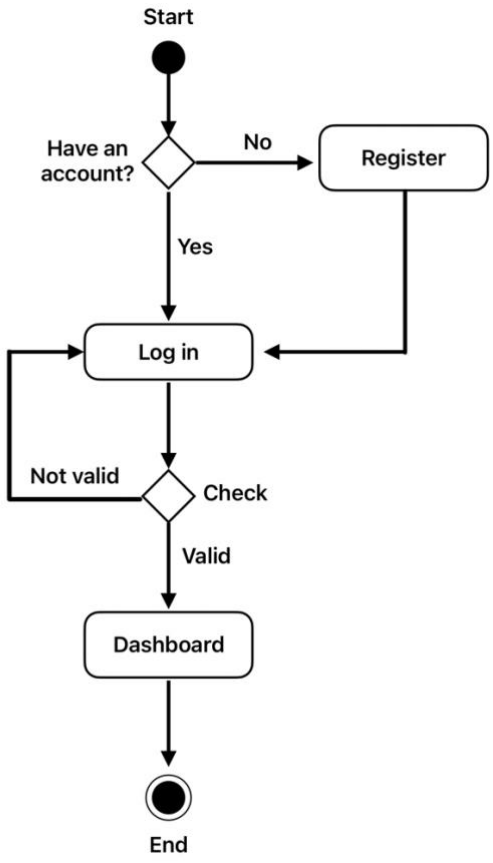

Figure 3. Activity Diagram of User Authentication Workflow

3.4 Code Quality

The project follows a well-organized structure, with the use of components for usability and long-term maintainability. This modular approach allows for easier updates and testing. Naming conventions and code style guidelines were followed throughout the development process. Variable and function names are taken care of to be descriptive and follow a consistent naming pattern, making it easier to understand their purpose. Indentation and spacing are consistent across the codebase, ensuring that the code is easy to read and navigate.

3.5 Version Control

Git, version control system, is used to manage the codebase, track the changes, and keep the project organized. Branching method is used to separate the development of different features. For each new feature, such as User Authentication, Pet Profiles, and Reminders, a separate branch was created. Naming convention is applied to all branches in the format of "feature/<name>" such as "feature/auth", "feature/calendar", and "feature/dashboard".

Frequent commits were made after significant updates, ensuring that progress was tracked in small, manageable steps. Each commit had a descriptive message to outline the change, such as "Implemented user authentication" or "Added pet profile management". Once the minimal viable version of the app was ready, the version was tagged and a release was created, marking the app's core functionality as complete. This version was tagged (e.g., **v1.0.0**) for future reference. The consistent use of Git not only facilitates version control but also contributes to a better understanding of best practices for keeping the project organized and improving workflow.

3.6 Testing

Testing is a critical aspect of ensuring that PURRNOTE functions as expected. The testing methodology used includes the following testing methods.

Functional Testing

The app's core features, such as adding pet profiles, setting reminders, and managing the calendar, have been tested and verified that they work as expected. This involved directly interacting with the app, performing actions like adding a pet profile and setting a reminder, and confirming that these changes were correctly reflected in the app. The goal of this testing was to ensure that all essential features worked smoothly and as expected, simulating user interactions.

Integration Testing

Integration testing focused on verifying that different module of the app work together seamlessly. For instance, interactions between the AddEventModal component and Firestore were tested to ensure that event data was correctly saved and retrieved. Additionally, the integration between the Pet Modals and Cloudinary was validated to confirm that pet images were uploaded successfully, and their URLs were stored accurately in Firestore. These tests ensured that the app's various components and external services functioned cohesively, maintaining data consistency.

User Acceptance Testing (UAT)

After the app was developed, UAT was conducted with a group of six users. Feedback was collected to evaluate the app's usability and functionality. This feedback was used to improve the app in future iterations. UAT helps ensure the app meets user needs and expectations.

Chapter 4: **Development Process**

4.1 Timeline

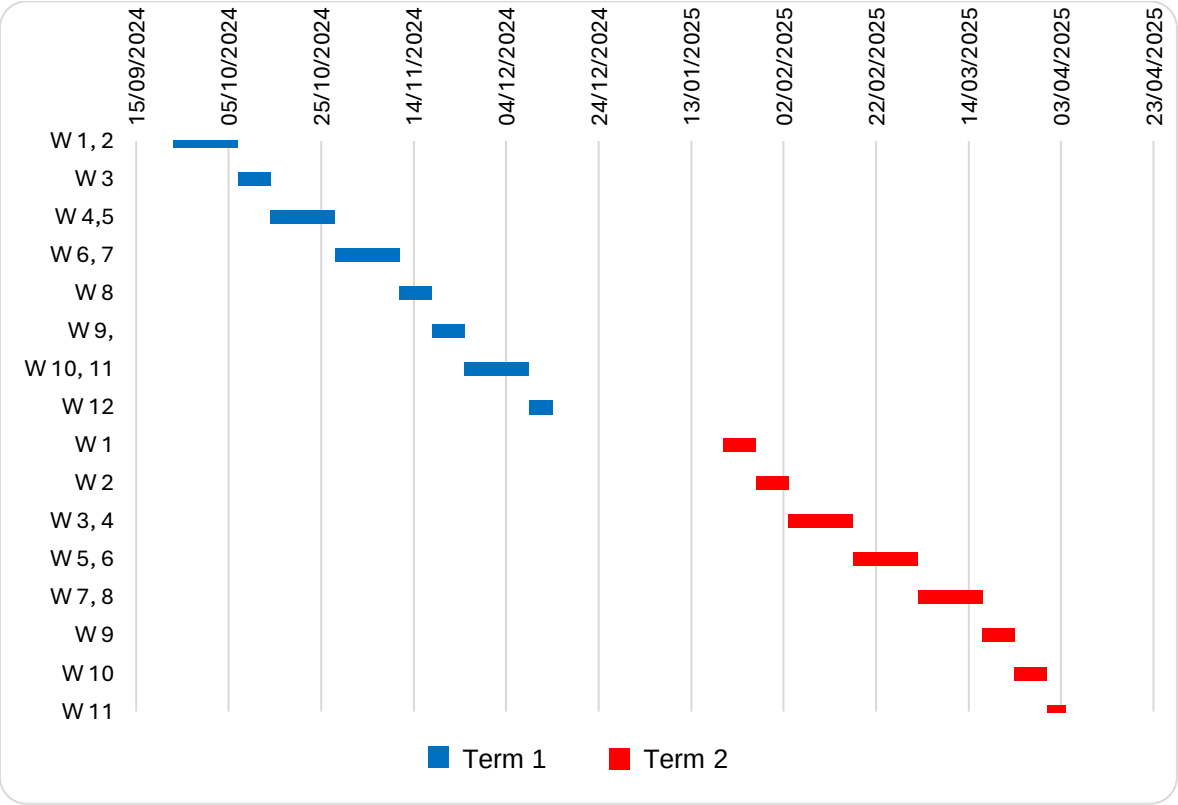

Figure 4. Gantt Chart Display of Project Timeline and Weekly Phases

Term 1

Week	Description	Date	Deliverables
1, 2	Research and analyze existing pet care apps to define features and project requirements	23/09/2024 - 6/10/2024	Finalized Project structure
3	Design user interface, create wireframes and prototypes for key features	07/10/2024 - 13/10/2024	UI design
4, 5	Build app layout for user authentication, user profile, pet profiles, and booking page	14/10/2024 - 27/10/2024	Basic features
6, 7	Implement scheduling reminders and CRUD functionality for user and pet data	28/10/2024 - 10/11/2024	Reminders page
8	Build calendar feature with events and different views for scheduling	11/11/2024 - 17/11/2024	MVP with main features
9	Test APIs and all MVP features (functional and UX testing), refactor if needed	18/11/2024 - 24/11/2024	Tested version of app
10, 11	Prepare interim report, create demonstration video and presentation slides	25/11/2024 - 08/12/2024	Presentation slides, demonstration video
12	Finalize code, documentation, and project report	09/12/2024 - 13/12/2024	Final code, documentations

Term 2

Week	Description	Date	Deliverables
1	Research Social networking in app Define Requirements	20/01/2025 - 26/01/2025	Defined project plan, feature list
2	Design the app's user interface, including wireframes and basic layouts for key features like profiles, content sharing, and navigation.	27/01/2025 - 02/02/2025	UI design
3, 4	Build the content sharing layout where users can create posts, share images, and interact with content.	03/02/2025 - 16/02/2025	Content sharing layout developed
5, 6	Develop the friendship feature allowing users to send and accept friend requests, view friends' posts, and interact within the app.	17/02/2025 - 02/03/2025	Friendship feature (frontend)
7, 8	Integrate the backend to manage features like user accounts, posts, and friend requests. Ensure proper data handling.	03/03/2025 - 16/03/2025	Integrated backend for social features
9	Test all features and APIs to make sure everything works correctly. Refactor code if needed.	17/03/2025 - 23/03/2025	Tested app with functional and UX testing
10	Prepare the final project report and presentation. Include key features, challenges, and a demo video.	24/03/2025 - 30/03/2025	Final code, presentation slides, demo video
11	Finalized project delivery	31/03/2025 – 03/04/2025	Final project and documentations

Project Diary (Completed tasks in term 1):

For a detailed log of daily and weekly activities, challenges faced, and the solutions implemented during the project, please refer to the project diary section. The diary includes comprehensive notes on development, testing, and iterations for each phase.
Link to diary: Project Diary Link

4.2 Progress so far

As the first term comes to an end, the features outlined in the project timeline have been successfully implemented. These features are now fully functional and provide a strong foundation for the remaining development of the PURRNOTE app.

Key features such as **user authentication**, **user profiles**, **pet profiles**, and **calendar functionality** have been integrated and tested. The **appointment management** system has also been implemented, allowing users to create, track, and manage appointments. Additionally, while real-time push notifications have been implemented to alert users about upcoming events and tasks, this feature is still in progress to ensure users have a seamless experience.

Testing of these features has been conducted to ensure their functionality, with some refinements and adjustments made to improve user experience based on feedback and testing results. The app is now ready for further development and feature enhancements in the coming term 2.

4.2.1 Screen Overview

The overview of the screens will be discussed in detail in this section. Visual representations of the implemented pages of the app have been included to illustrate its current functionality.

Sign Up Screen serves as the secure entry point for new users. It features a clean and intuitive layout, guiding users through a straightforward registration process.

Login Screen serves as the primary authentication gateway for existing users to access the mobile application. Users can enter their credentials through clearly defined input fields with intuitive navigation. Instant feedback on incorrect email formats or invalid password entries are supported, ensuring a smooth login process.

Home Screen is accessed once the user is authenticated. In this screen, easy access to the pets' page, calendar page and booking page have been integrated with the proper UI layout. Users can easily add events to the calendar by clicking the 'Add Event' button, which opens a modal to input event details.

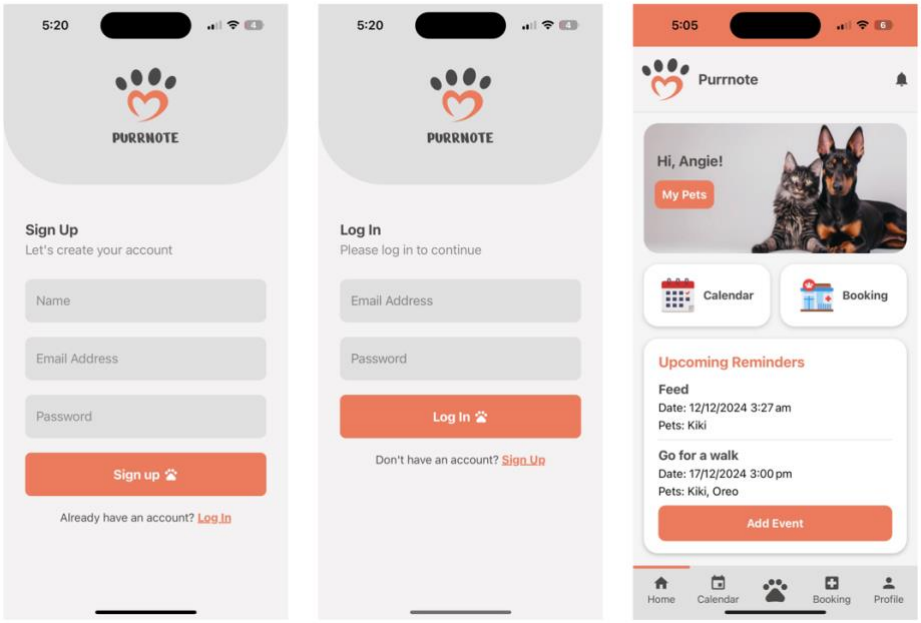

Figure 5. Sign up page, Log In page, and Home page of the app

Calendar Screen provides a dynamic calendar which allows month and year selections. The calendar is designed to distinguish between dates with events and those without, visually highlighting dates that have scheduled events. Performing CRUD functionalities is also straightforward. When an event is selected, it will display a modal with prefilled event details where user can update or delete the event. Additionally, the 'Add' button opens a modal for entering event details.

Booking Screen is implemented to separately view all the appointments, which can also be filtered by pet. This feature enables users to view each pet's appointment schedule, ensuring no important appointments are missed.

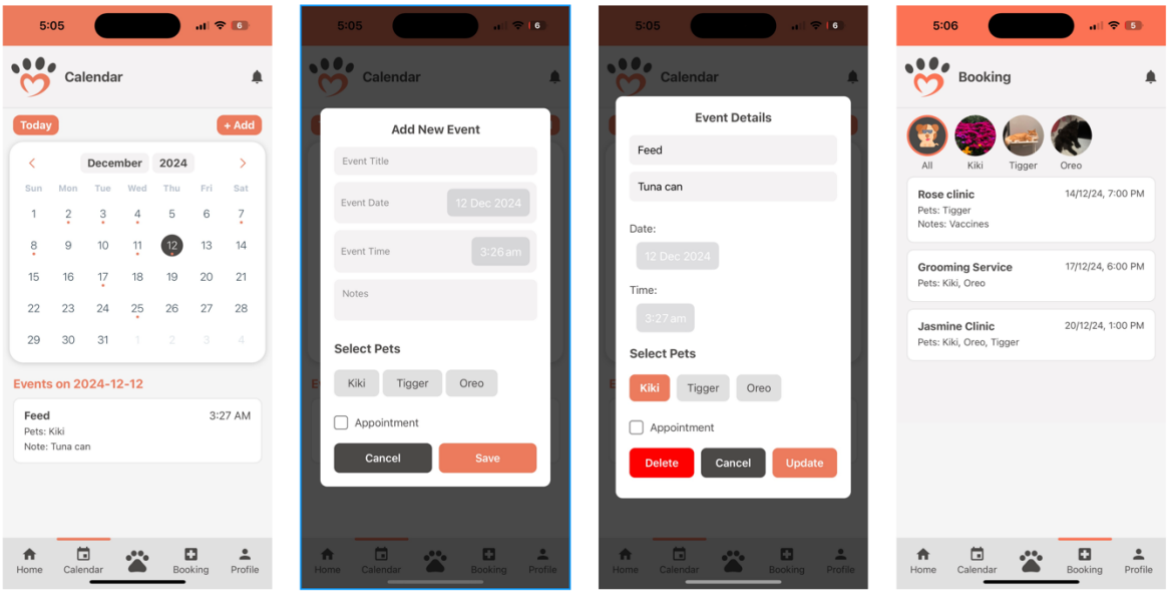

Figure 6. Calendar page, booking page and event features

Pets Screen will be displaying all the pet profiles, set up by the user. The 'Add' button opens a modal for entering pet details, including the option to upload a profile image. Each profile redirects to a detailed view where users can update or delete pet information.

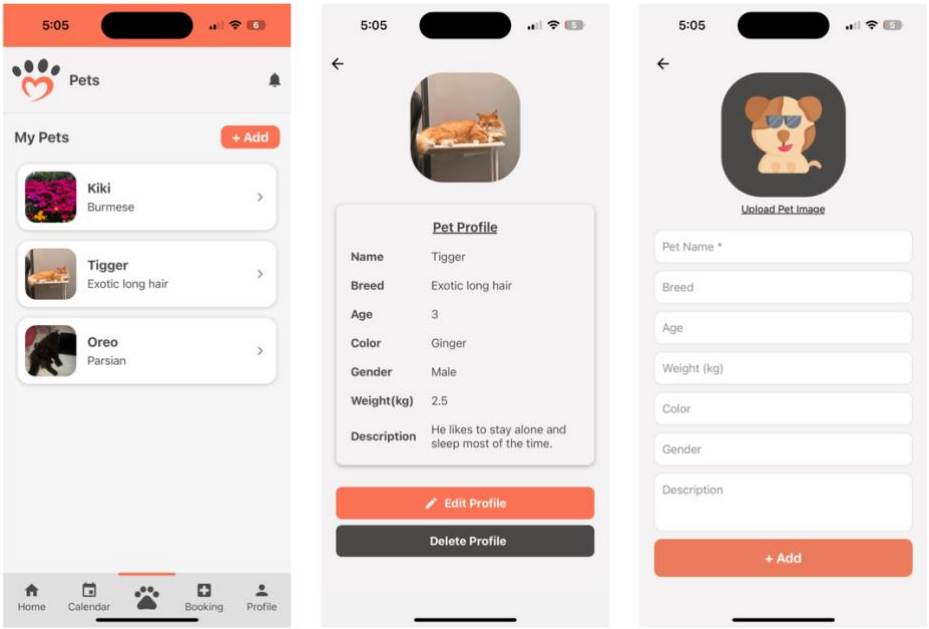

Figure 7. Pets profile page, creating and deleting pet profiles

User Profile Screen includes settings to update the details, change password, log out, or delete the account. The 'Edit Profile' and 'Change Password' buttons provides user-friendly modals for updating account details or credentials.

Notification Inbox Screen displays all notifications sent to the user in a single location. Sending push notification feature is still being refined to ensure users receive notifications reliably and without any interruptions.

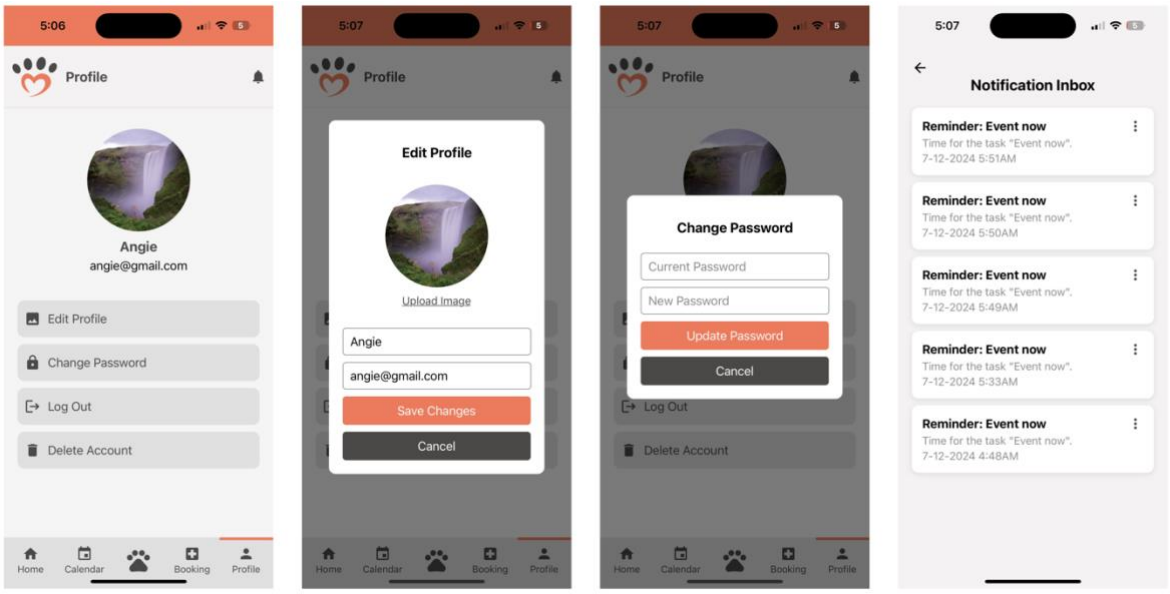

Figure 8. Profile page, user settings and notification inbox

4.3 Challenges and Solutions

1. **Framework Switch:** Initially, Flutter was selected for the project; however, due to the time required to familiarize with Flutter and the challenges related to integrating UI features with Firebase, the timeline was not met during the initial stages. To ensure all planned features could be completed within the proposed timeframe, React Native was chosen as an alternative. Although switching to React Native required redoing parts of the project, this change ultimately accelerated the development process, allowing me to get back on track with the project timeline after two weeks.

2. **UI Consistency Across Devices:** Ensuring a consistent and responsive UI across different mobile devices and screen sizes was challenging. However, by using React Native's responsive design features, such as using SafeAreaView, KeyboardAvoidingView, this challenge was successfully addressed.

3. **Data Synchronization:** Synchronizing data across devices, especially for pet profiles and reminders, required careful implementation. After researching state management and utilizing Firebase's real-time capabilities, the issue was resolved with the proper use of state management techniques.

4.4 Future Work

In **Term 2**, the focus will shift towards enhancing the existing version of the app, implementing the vet locator, social networking features, and integrating the backend, followed by extensive testing. The first step will be to improve features such as notification system, password recovery, and recurring events with daily, weekly, monthly, or annually options. Additionally, advancements in UI/UX consistency will be made to align with upcoming tasks throughout the development process. Another step will be defining the key features required and designing user interface for the vet locator and social networking features of the app.

The next phase is to implement the vet locator followed by development of the content-sharing layout, enabling users to create posts, share media, and interact with others' content. This feature will

significantly optimize user engagement and interaction within the app's social environment. The backend integration will then be addressed, focusing on supporting the social features.
Once the core features are implemented, we will move into the testing phase, including unit testing, functional, integration, and user experience testing of the APIs. This will ensure that the social features are working as expected, with any identified bugs being fixed. Simultaneously, the final report and presentation will be prepared. This will include documentation of the development process, challenges faced, solutions implemented, an overview of the app's capabilities, and the potential future extensions. A demonstration video will also be created to showcase the app's key features and functionality. Finally, Term 2 will conclude with finalizing all documentations, addressing any remaining issues, and ensuring the app is fully polished and ready for delivery.

Chapter 5: **Technical Achievements**

The development of PURRNOTE has involved several key technical achievements that form the core functionality of the app. Significant progress was made in creating the Minimum Viable Product of the app. The key technical achievements will be discussed below.

1. User Authentication

One of the first challenges was implementing a secure and reliable user authentication system. To address this, **Firebase Authentication** was integrated into the app, enabling users to securely sign up, log in, and manage their accounts. Firebase Authentication was chosen for its robust and easy-to-use features, including email/password login and secure token generation. Additionally, Firebase ensured safe password storage by hashing passwords before saving them, providing an extra layer of security for user data.

Code Overview:

To integrate Firebase services such as Authentication, Firestore, and Analytics into the app, the following initialization code was used. The firebaseConfig object, containing project-specific keys and identifiers, is used for initializing Firebase.

```
// Initialize Firebase
const app = getApps().length === 0 ? initializeApp(firebaseConfig) :
getApp();

export const auth = initializeAuth(app, {
  persistence: getReactNativePersistence(AsyncStorage),
});
export const firestore = getFirestore(app);

let analytics;
isSupported().then((supported) => {
  if (supported) {
    analytics = getAnalytics(app);
  }
});
```

Backend functions are implemented separately from the frontend code, using callable functions to handle specific tasks. Below are the login and sign-out functions, demonstrating how authentication features are managed in the backend.

```
// Log In Function
const handleLogin = async () => {
    setLoading(true);
    try {
      await signInWithEmailAndPassword(auth, email, password);
      Alert.alert("Success", "You have successfully logged in!");
      await registerIndieID(
        auth.currentUser.uid,
        25248,
        "wtOK6Mg9wWTJpjgjr1qH0v"
      );

      navigation.reset({
        index: 0,
        routes: [{ name: "Dashboard" }],
      });
    } catch (error) {
      Alert.alert("Login Failed", error.message);
```

```
    } finally {
      setLoading(false);
    }
};

// Sign Out Function
export const handleSignOut = async (navigation, subId) => {
  try {
    if (subId) {
      await unregisterIndieDevice(subId, 25235,
"rBDIqttve0mXsFAkpSkis7");
      console.log("Device unregistered from Native Notify with Sub ID:",
subId);
    }
    await signOut(auth);
    axios.delete(

`https://app.nativenotify.com/api/app/indie/sub/25248/wtOK6Mg9wWTJpjgjr1q
H0v/${subId}`
    );
    Alert.alert("Success", "You have been logged out.");
    navigation.navigate("Login");
  } catch (error) {
    console.error("Error signing out: ", error);
    Alert.alert("Logout Error", error.message);
  }
};
```

In the Login and Sign-Out functions above, registration of the user's device is included to enable or disable the notification service. Upon logging in, the device is registered to enable notifications, and it is opted out of this service when the user logs out.

2. Data Management

Effective data management is crucial for the app's functionality, particularly as it handles various types of events, user, and pet-related data that must be stored and updated reliably. To manage this data, **Firebase's Firestore** was chosen for real-time NoSQL database storage. The firestore database is structured by defining rules to ensure scalability and performance as the amount of data increased.

Code Overview:

Data is organized into collections such as users, pets, appointments, and events to ensure efficient querying and scalability. These rules ensure that only authenticated users can access their data while preventing unauthorized access to other users' information. Below is an overview of the Firestore security rules implemented for the app:

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /users/{userId} {
      allow read, write: if request.auth != null && request.auth.uid ==
userId;

      match /pets/{petId} {
        allow read, write: if request.auth != null && request.auth.uid ==
userId;
      }

      match /appointments/{appointmentId} {
        allow read, write: if request.auth != null && request.auth.uid ==
userId;
      }
```

```
      match /events/{eventId} {
        allow read, write: if request.auth != null && request.auth.uid ==
userId;
      }
    }
  }
}
```

The Firestore security rules shown above ensure that each user's data is isolated. This security is achieved by checking the request.auth.uid against the userId in the Firestore path, as seen in the rules. Below is an example of how events data is fetched for a user. The fetchUserEvents function retrieves all events for the authenticated user by accessing the events subcollection. It uses Firestore's getDocs function to fetch the data and maps it into a clear format. If no events are found, an empty array is returned.

```
export const fetchUserEvents = async (userId) => {
  try {
    const eventsCollection = collection(firestore,
`users/${userId}/events`);
    const eventsSnapshot = await getDocs(eventsCollection);

    if (eventsSnapshot.empty) {
      return [];
    }

    const events = eventsSnapshot.docs.map((doc) => ({
      id: doc.id,
      ...doc.data(),
    }));
    return events;
  } catch (error) {
    console.error("Error fetching user events: ", error);
    Alert.alert("Error fetching user events", error.message);
    return [];
  }
};
```

To improve user experience during data fetching, a loading indicator is displayed to inform users that the data is still being loaded, especially when dealing with large datasets.

```
const [loading, setLoading] = useState(true);

useEffect(() => {
    const fetchPets = async () => {
      try {
        const userId = auth.currentUser.uid;
        const petsCollectionRef = collection(firestore,
`users/${userId}/pets`);
        const petsSnapshot = await getDocs(petsCollectionRef);

        const petsData = petsSnapshot.docs.map((doc) => ({
          id: doc.id,
          ...doc.data(),
        }));

        setPets(petsData);
      } catch (error) {
        console.error("Error fetching pets:", error);
```

```
        Alert.alert("Error", "Failed to load pets. Please try
again.");
      } finally {
        setLoading(false);
      }
    };
    if (isFocused) {
      fetchPets();
    }
  }, [isFocused]
);
```

In this code, a state variable is used to monitor the data fetching process. Initially set to true, it switches to false once data fetching is complete. During this process, when the state is true, the if condition triggers the display of a loading indicator on the frontend and it is implemented as below.

```
if (loading) {
    return (
      <View style={styles.loadingContainer}>
        <ActivityIndicator size="large" color={colors.primary} />
      </View>
    );
}
```

3. Data Rendering

Data fetched from the Firestore database is rendered on the frontend. To enhance usability and maintain clean code structure, the rendering function is defined separately from the main frontend logic.

Code Overview:

The code below shows how pet profiles are rendered to display on the Booking page.

```
// Render pet profile
const renderPetProfile = (pet) => {
    const imageSource = pet.imageUrl ? { uri: pet.imageUrl } : dog;

    return (
      <TouchableOpacity
        key={pet.id}
        onPress={() => setSelectedPetId(pet.id)}
        style={styles.petProfileContainer}
      >
        <Image
          source={imageSource}
          style={[
            styles.petImage,
            selectedPetId === pet.id && styles.selectedPet,
          ]}
        />
        <Text style={styles.petName}>{pet.name}</Text>
      </TouchableOpacity>
    );
};
```

4. Calendar Integration

Another essential feature of the app is the calendar integration, which enables users to schedule and manage important events for their pets. The React Native Calendar component was integrated to display events in a calendar view, allowing users to easily create, view, and manage. This calendar is synchronized with Firebase, ensuring real-time updates across devices. One of the challenges faced during development was integrating the database with the frontend modal components for adding new events and updating existing ones. After extensive testing and several rounds of refactoring, the outcome is a fully functional calendar that supports CRUD (Create, Read, Update, Delete) operations.

Additionally, keeping the event modals as separate components improved both the usability of the app and the organization of the code. To manage events efficiently, separate modal components were created for adding new events and editing existing events.

Code Overview:

```
<AddEventModal
      isVisible={isAddingEvent}
      setIsVisible={setIsAddingEvent}
      newEvent={newEvent}
      setNewEvent={setNewEvent}
      selectedPets={selectedPets}
      setSelectedPets={setSelectedPets}
      petNames={petNames}
      addEvent={handleAddEvent}
      loading={loading}
/>

<EventModal
      isVisible={isEventModalVisible}
      setIsVisible={setIsEventModalVisible}
      selectedEvent={selectedEvent}
      setSelectedEvent={setSelectedEvent}
      petNames={petNames}
      updateEvent={handleUpdateEvent}
      deleteEvent={handleDeleteEvent}
      updateLoading={updateLoading}
      deleteLoading={deleteLoading}
/>
```

The two components in the above code are invoked from the Calendar page. Both components use props like isVisible to control their visibility and specific functions such as addEvent, and updateEvent. The use of having separate components help encapsulate functionality and maintain a clean code structure.

4. Profile Image Upload

To implement this feature, Cloudinary is used to store images uploaded by user into cloud. The pickImage function uses ImagePicker to request permissions for accessing the device's media library. If granted, it allows the user to select an image and sets the selected image's URI as the profile image. The uploadProfileImage function takes the image URI, converts it into FormData, and uploads it to Cloudinary using their API. It ensures the uploaded image URL is returned for further use. Error handling is included in both functions to manage permission denials, canceled selections, or upload failures, ensuring a smooth and user-friendly experience.

Code Overview:

```
const pickImage = async () => {
    try {
      const permissionResult =
```

```
        await ImagePicker.requestMediaLibraryPermissionsAsync();

      if (permissionResult.status !== "granted") {
        Alert.alert(
          "Permission Denied",
          "You need to allow permission to access your photos."
        );
        return;
      }

      const result = await ImagePicker.launchImageLibraryAsync({
        mediaTypes: ImagePicker.MediaTypeOptions.Images,
        quality: 1,
      });

      if (!result.canceled && result.assets?.length > 0) {
        const uri = result.assets[0].uri;
        setProfileImage(uri);
      } else {
        console.log("Image selection canceled or invalid structure");
      }
    } catch (error) {
      console.error("Error selecting image: ", error);
      Alert.alert("Error", "An error occurred while selecting the
image.");
    }
};

const uploadProfileImage = async (imageUri) => {
    if (!imageUri) return null;

    const formData = new FormData();
    formData.append("file", {
      uri: imageUri,
      type: "image/jpeg",
      name: "profile_image.jpg",
    });
    formData.append("upload_preset", "purr_note");

    try {
      const response = await fetch(
        "https://api.cloudinary.com/v1_1/dunbwugns/image/upload",
        {
          method: "POST",
          body: formData,
        }
      );
      const data = await response.json();

      if (data.secure_url) {
        return data.secure_url;
      } else {
        console.error("Upload failed, no secure_url returned:", data);
        return null;
      }
    } catch (error) {
      console.error("Image upload failed:", error);
      return null;
    }
};
```

If there is no image uploaded by user, a fallback image will be displayed. The code below ensures that the profile image is replaced with a default fallback image when the user has not provided one. The getProfileImageSource function checks if a valid imageUri exists and is not set to "default". If valid, it returns the provided image URI, otherwise, it falls back to a predefined default image.

```
const getProfileImageSource = (imageUri) => {
    return imageUri && imageUri !== "default"
      ? { uri: imageUri }
      : defaultProfileImage;
};
```

5. Navigation Stack

Navigation stack manages redirections between screens, starting with Login as the initial screen. The Stack.Navigator allows users to transition between screens in a stack-based manner, with customizable options like hiding headers (headerShown: false) and enabling gestures (gestureEnabled: true). Each screen is registered with a unique name and a corresponding component. The MyStack component is provided to visualize the app's primary navigation structure.

Code Overview:

```
const Stack = createNativeStackNavigator();

const MyStack = () => {
  return (
    <NavigationContainer>
      <Stack.Navigator
        initialRouteName="Login"
        screenOptions={{
          headerMode: "screen",
          headerShown: false,
          gestureEnabled: true,
        }}
      >
        <Stack.Screen
          name="Login"
          component={Login}
          options={{
            gestureEnabled: false,
            headerShown: false,
          }}
        />
        <Stack.Screen name="SignUp" component={Signup} />
        <Stack.Screen name="Home" component={Home} />
        <Stack.Screen name="ReminderPage" component={ReminderPage} />
        <Stack.Screen name="Dashboard" component={Dashboard} />
        <Stack.Screen name="Notification" component={Notification} />
        <Stack.Screen name="Booking" component={Booking} />
        <Stack.Screen name="Profile" component={Profile} />
        <Stack.Screen name="CalendarPage" component={CalendarPage} />
        <Stack.Screen name="Pets" component={Pets} />
        <Stack.Screen name="AddPet" component={AddPet} />
        <Stack.Screen name="PetProfile" component={PetProfile} />
      </Stack.Navigator>
    </NavigationContainer>
  );
};

export default MyStack;
```

Conclusion

In conclusion, the development of PURRNOTE has successfully addressed the core needs of pet owners by implementing key features such as user authentication, data management, calendar integration, and notifications inbox. These features provide a solid foundation for efficient pet care management, ensuring a seamless and user-friendly experience. Through careful planning, iterative development, and overcoming challenges related to integration and functionality, significant progress has been made towards creating a comprehensive minimum viable version of pet assistant app.

Looking ahead, PURRNOTE is well-positioned for future growth, with planned features such as the vet locator, social networking capabilities, and advancements to existing functionalities. The app's strong foundation and ongoing development will continue to enhance its capabilities, making it an invaluable tool for pet owners who seek an organized and reliable way to manage their pets' care.

Diary

Week 12 (9 December 2024)

Summary:
- Refactored the app for improvements and fixed bugs.
- Completed the report, demonstration video, diary, and documentation.
- Conducted testing on the app to ensure stability.
- Finalized the presentation for the project.

Challenges:
- No issues encountered.

Next Step:
- Continue work on Term 2, focusing on social networking features and backend integration.

Week 11 (2 December 2024)

Summary:
- Tested NativeNotify in a different project for testing purposes.
- Implemented the reminder notification feature.
- Created the notification inbox to manage reminders.
- Added month and year selectors for the calendar.
- Added a flexible date selector for event creation in the calendar.
- Completed preparations for the presentation and demonstration video.
- Made a release and tagged the first version of the Minimum Viable Product.

Challenges:
- No issues encountered.

Next Step:
- Continue with the report and refine the presentation.

Week 10 (24 November 2024)

Summary:
- Ensured all features had a consistent UI design.
- Improved the calendar feature by displaying appointments for better user experience.
- Modified vet appointment database models for easier access.
- Refactored the home page for better navigation.
- Tested each feature for functionality and user experience.

Challenges:
- Encountered challenges with implementing notifications, as Firebase stopped providing certain features for free.

Next Step:
- Test the notification system with NativeNotify.
- Refactor layouts and UI designs.
- Begin working on the report and presentation.

Week 9 (18 November 2024)

Summary:
- Implemented the reminder page to manage pet care reminders.
- Fixed layout issues and performed UI touch-ups for all pages.
- Researched notification systems to find a suitable solution.

Challenges:
- No challenges encountered.

Next Step:
- Conduct testing and refactor as needed.

Week 8 (11 November 2024)

Summary:
- Fixed bugs from previously implemented pages (calendar and events) to align with the updated code.
- Implemented calendar events with features like Date Picker and Event Picker.
- Designed event CRUD modals for easier management.
- Updated calendar events to change instantly when updated in the database.
- Created the reminder page to display data from the database.

Challenges:
- Everything worked as expected.

Next Step:
- Finalize the reminder system and implement the notification page.
- Test all existing pages.

Week 7 (4 November 2024)

Summary:
- Implemented the pet profile feature, allowing for multiple profiles.
- Connected to cloud image upload for user and pet profile photos.
- Designed the calendar with basic CRUD functionalities for events.
- Displayed events on the calendar for better user interaction.
- Developed CRUD functionality for events on the calendar.
- Developed the vet appointment page and connected it to the database.

Challenges:
- Initially implemented a yearly view for the calendar, but it was less effective than the monthly view.
- Opted to implement a monthly view with the ability to scroll left and right, which proved more user-friendly.

Next Step:
- Implement reminders for the events in the calendar.

Week 6 (28 October 2024)

Summary:
- Implemented user profile features including password change and account deletion using Flutter.
- Switched the framework to React Native, as Flutter was not compatible with the project's requirements.
- Implemented authentication features: login and signup.

Challenges:
- Encountered difficulties with Firebase integration in Flutter, which led to the switch to React Native.
- Switching required redoing the setup and pages, but it improved the development process.

Next Step:
- Continue developing pet profiles and calendar functionality.

Week 5 (21 October 2024)

Summary:
- Created a structured layout, aligned with GitLab guidelines.
- Implemented login and signup pages with authentication.
- Created the dashboard or landing page for user interaction.

Challenges:
- Implementing authentication took a significant amount of time due to complexity.

Next Step:
- Continue with the dashboard implementation.

Week 4 (14 October 2024)

Summary:
- Initiated small projects to test the most suitable technologies.

- Tested Flutter and Firebase by creating mock projects like a to-do list and counter.
- Researched Flutter Riverpod for state management.

Challenges:
- Lack of familiarity with Flutter.

Next Step:
- Start structuring the project layout and implement authentication.

Week 3 (7 October 2024)

Summary:
- Designed the UI/UX of the app.

Next step:
- Test the intended technology stack for the project.

Week 2 (1 October 2024)

Summary:
- Listed all user requirements.
- Drafted the project plan document.

Next Step:
- Begin structuring the project and researching technologies.

Week 1 (23 September 2024)

Summary:
- Conducted research on similar apps to understand the key features and competitive landscape.

Next Step:
- Continue refining the project plan based on findings.

Bibliography

[1] Glanville, C., Hemsworth, P. and Coleman, G. (2020) 'Conceptualising dog owner motivations: The Pet Care Competency model and role of "duty of care", Animal Welfare, 29(3), pp. 271–284. doi:10.7120/09627286.29.3.271.

[2] Hubrecht R, Wickens S and Kirkwood JK2017 The welfare of dogs in human care. In: Serpell JA (ed) The Domestic Dog: Its Evolution, Behavior and Interactions with People, Second Edition pp 272-299. Cambridge University Press: Cambridge, UK. https://doi.org/10.1017/9781139161800.014

[3] August, K.D., 2011. The responsibilities of pet ownership. Caring for family pets: Choosing and keeping our companion animals healthy, pp.17-28.

[4] Maguire, G., 2024. Pet care and training for beginners. Garreth Maguire.

[5] Van, H., 2020. Building a universal application with React and React Native.

[6] Chougale, P., Yadav, V., Gaikwad, A. and Vidyapeeth, B., 2021. Firebase-overview and usage. International Research Journal of Modernization in Engineering Technology and Science, 3(12), pp.1178-1183.

[7] Noor, J., Salim, S.I. and Al Islam, A.A., 2021. Strategizing secured image storing and efficient image retrieval through a new cloud framework. Journal of Network and Computer Applications, 192, p.103167.

[8] Al-Saqqa, S., Sawalha, S. and AbdelNabi, H., 2020. Agile software development: Methodologies and trends. International Journal of Interactive Mobile Technologies, 14(11).

[9] Serrano, I., 2022. Mind Your Pet (Master's thesis, Mills College).

[10] 11Pets: Pet care management. Retrieved December 10, 2024, from https://www.11pets.com

[11] Pawmates: Pet care management app. Retrieved December 10, 2024, from https://www.pawmatesapp.com

[12] DogLog: Pet care and health management app. Retrieved December 10, 2024, from https://www.doglogapp.com

[13] Tan, C.Y. and Murli, N., 2022. Development of Android-based Petmily Application. Applied Information Technology And Computer Science, 3(1), pp.75-93.

[14] Thomas, C.G. and Devi, J., 2021. A study and overview of the mobile app development industry. International Journal of Applied Engineering and Management Letters (IJAEML), 5(1), pp.115-130.

[15] Syromiatnikov, A. and Weyns, D., 2014, April. A journey through the land of model-view-design patterns. In 2014 IEEE/IFIP Conference on Software Architecture (pp. 21-30). IEEE.

User Manual

Prerequisites

Before running the project, ensure you have the following installed:

Node.js

1. Download Node.js from the Node.js official website.
2. Install the latest stable version by following the installation instructions for your operating system.
3. Verify the installation by running the following in your terminal:
   ```
   node -v
   npm -v
   npx --version
   ```
   This will display the installed versions of Node.js, npm and npx.

Expo Go

1. Install the Expo Go app on your device from the App Store or Google Play Store.
2. Ensure you have a compatible device to scan the QR code displayed during development.

Note: Firebase, Cloudinary, and Native Notify credentials are already configured in the project. No additional setup for these services is required.

Running the Project

1. Clone the repository:
   ```
   git clone https://gitlab.cim.rhul.ac.uk/wlis130/PROJECT.git
   ```
2. Navigate to the project directory:
   ```
   cd PROJECT/product
   ```
3. Install dependencies:
   ```
   npm install
   ```
4. Start the expo development server:
   ```
   npx expo start
   ```
5. Use the Expo Go app on your device to scan the QR code displayed in the terminal or browser to launch the app.

Demonstration Video

To learn how to use the app, check out this demonstration video. You can watch it by following the link below: https://youtu.be/x8M9JkyAKUI?si=1RhhLKrOEX7M3AKg