Versions:

Date	Versiune	Autor	Comentarii
7.11.2024	01	Pop Alexandra	First version (at home, only the first 2 contents)
8.11.2024	02	Pop Alexandra	Second version (mostly the Use Cases and System Architecture)
24.01.2025	03	Pop Alexandra	Bugs and conclusion

Analisys&Design for FurryFriends

Content:

1. General Presentation

The project is a comprehensive mobile application designed to support pet owners, non-pet owners, companies (specifically animal shelters), and a special role known as "Owner" with advanced permissions. The app offers a community platform where users can engage, create profiles for their pets, schedule appointments, view nearby pet-friendly services, and access helpful information about pet care. Additionally, the app includes a robust management system for “Owners” to monitor user activity, see statistics, and even moderate user accounts.

The app's main objectives are to:

- Provide pet owners with a convenient way to manage their pet's profiles and set reminders for veterinary care and feeding schedules.
- Allow non-pet owners and animal shelter representatives to engage with the community and access specific features related to their needs.
- Enable “Owners” to see detailed statistics about app usage, review all registered users, and manage accounts if needed, including banning or deleting users.
- Offer animal shelters a platform to manage appointments and accept donations for shelter operations.
- Create a social space for all users to interact, share pet stories, and foster a sense of community.

2. Theoretical Fundamentals

The project is based on principles of user-centered design, role-based access control, and secure data management. Theoretical considerations include:

- **Role-Based Access Control (RBAC):** Each user role (Pet Owner, Non-Pet Owner, Company/Animal Shelter, Owner) has specific permissions and access rights. RBAC is used to ensure users only access features relevant to their role.
- **Data Security and Privacy:** User data, including pet profiles, user registration details, and account information, is securely stored. Only “Owners” can access sensitive data like account statistics and user lists.
- **Community Engagement and Moderation:** This theory supports fostering a safe and interactive community by allowing users to report inappropriate content. “Owners” have moderation privileges to manage flagged content and user accounts.
- **Location-Based Services (LBS):** Location services provide relevant recommendations to users based on proximity, like nearby pet-friendly parks or shelters.

3. IT Technology (which will be implemented)

The app is developed using:

- **React Native:** A cross-platform framework that enables efficient development for both iOS and Android devices.
- **Expo:** A toolchain for React Native that simplifies development, testing, and deployment.
- **AsyncStorage (React Native):** Used to store user data securely on the device. This library is critical for handling user data, pet profiles, and settings.
- **Node.js (Planned):** For backend services if future development requires a server.
- **Firebase or Similar Database (Planned):** For storing and managing user data more securely, especially if the app is to scale beyond local storage.

- **Cloud-Based Services (Planned):** For features like real-time messaging, image storage for pet profiles, and large-scale analytics.

4. Functionalities

The app includes a range of features tailored to different user roles:

- **Registration and Login:**
  - Users can register with a unique username and password.
  - Role selection is part of the onboarding process, with options for Pet Owner, Non-Pet Owner, Company (Animal Shelter), and Owner.
  - Only users selecting “Owner” must enter a secondary 8-digit code for added access to advanced features.
- **Role-Specific Features:**
  - **Pet Owners:** Can create and manage detailed pet profiles, set reminders, and receive notifications about upcoming vaccinations and checkups.
  - **Non-Pet Owners:** Can interact with the community by sharing and commenting on posts.
  - **Company (Animal Shelter):** Animal shelters can manage appointments, accept donations, and communicate with users.
  - **Owner (Special Role):** Has access to the app's statistics, including user data and pet profile statistics, with options to ban or delete users.
- **Pet Profile Management:**
  - Pet owners can add pet details like name, breed, medical history, and vaccination records.
  - Options for reminders and alerts related to pet care.
- **Community Interaction:**
  - Users can post and comment on stories, ask questions, and share experiences, creating a social platform.

  - Users can report inappropriate content, which is then reviewed by "Owners."

- **Statistics and Moderation (For Owners):**
  - Owners can view statistics on user activity, such as registration dates and trends.
  - Owners can access a "See Users" screen where they can view, ban, or delete users if needed.
- **Location-Based Recommendations:**
  - Users can find nearby animal shelters, veterinary clinics, and pet-friendly parks.
- **Appointment Scheduling (For Animal Shelters):**
  - Animal shelters can create and manage time slots for users to book consultations or services.
- **In-App Purchases and Donations (Planned):**
  - Users will have options to make donations to shelters or purchase pet-related items.

5. Actors and related access rights

- **Pet Owner:**

- **Access Rights:**
  - Register and log in.
  - Create, update, and delete pet profiles.
  - Set reminders for pet care (vaccinations, feeding schedules).
  - Engage with the community by posting stories or commenting.
  - View location-based recommendations (e.g., nearby vets and pet parks).
- **Future Access Rights:**

- Access additional pet care tips and product recommendations based on pet profile details.

Non-Pet Owner:

- **Access Rights:**
  - Register and log in.
  - Interact with the community by sharing posts and commenting.
  - Access location-based recommendations (e.g., nearby animal shelters and adoption events).
- **Future Access Rights:**
  - Participate in forums or Q&A related to pet care and adoption.

Company (Animal Shelter):

- **Access Rights:**
  - Register and log in.
  - Manage appointments and set availability for consultations or services.
  - Interact with users, answer inquiries, and manage bookings.
  - Accept donations from users.
- **Future Access Rights:**
  - Access analytics on bookings and user engagement for operational insights.

Owner (Special Role with Advanced Permissions):

- **Access Rights:**
  - Register, log in, and verify with a secondary 8-digit code.
  - Access “Statistics” to view user registration data, user roles, and engagement metrics.

- Access "See Users" screen with permissions to view, ban, or delete user accounts.
- Moderate flagged content within the community.

- **Future Access Rights:**
  - Access advanced data analytics, including user activity heatmaps, most active hours, and overall app health.

6. Use Case Diagrams

6.1. User Registration and Login

Goal:

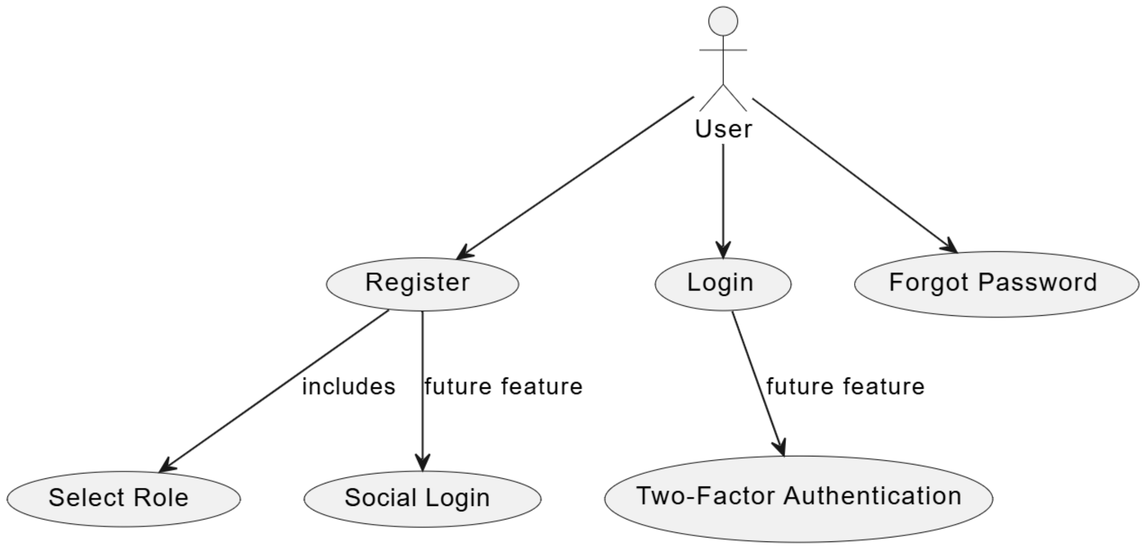
Allow users to register with a unique account, choose a role, and securely log in to the app with advanced security options.

Actors:

- **User** (can be a pet owner, non-pet owner, animal shelter representative, or owner with special permissions)

Description:

- **Register:** Users can create an account by entering a unique username and password. After registration, they must select their role (Pet Owner, Non-Pet Owner, Company/Animal Shelter, or Owner with access to statistics).
  - **Future Feature:** Support for social logins (Google, Facebook) and two-factor authentication (2FA) to enhance security.
- **Login:** Users can log in by entering their username and password.
  - **Forgot Password:** Users can reset their password via email if they forget it.
  - **Future Feature:** 2FA verification after password entry for additional security.



6.2 Role Selection and Access Control

Goal:

Provide role-based access control, allowing users to choose their role once after registration, with different permissions and functions based on the role.

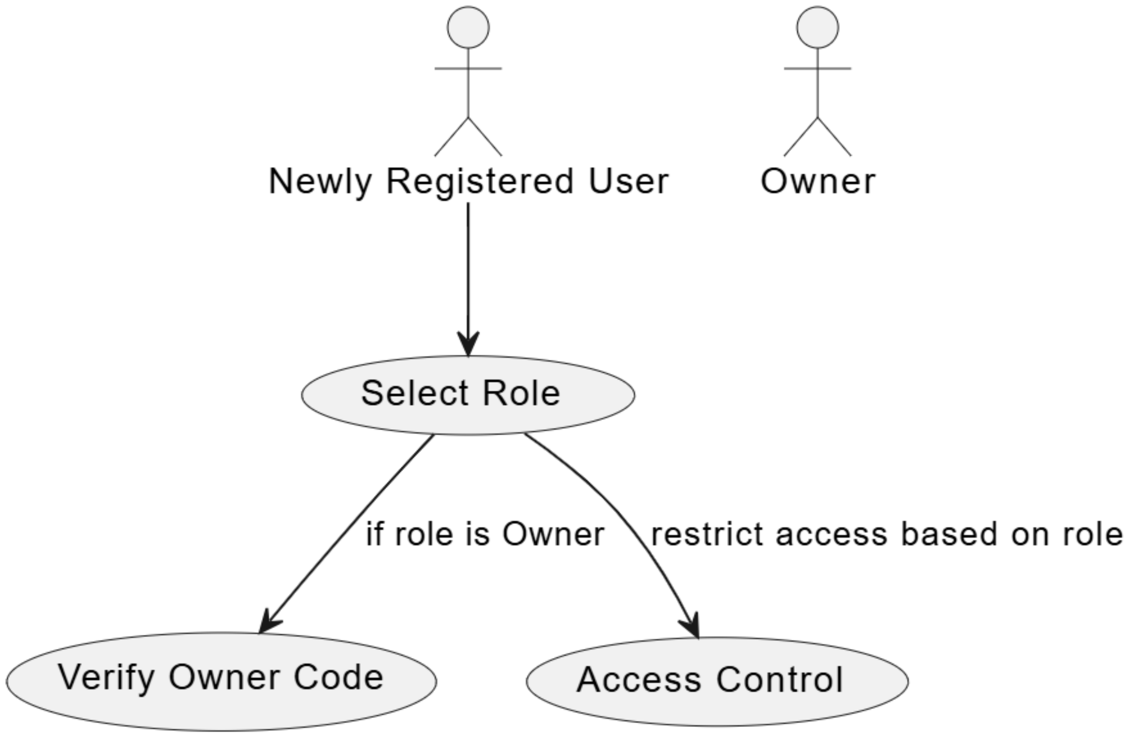
Actors:

- **Newly Registered User**
- **Owner**

Description:

- **Select Role:** New users select their role and receive access only to the features allowed for that role (Pet Owner, Non-Pet Owner, Company/Animal Shelter, or Owner).
- **Verify Owner Code:** If a user selects the role of "Owner," they must enter a specific 8-digit code to verify their identity.
    - **Future Feature:** Owners continue to use the code each time they access sensitive features, such as user management and statistics.

- **Restrictions:** Once a role is selected, it cannot be changed later to maintain the integrity of user access control.



6.3. Pet Profile Management (for Pet Owners)

Goal:

Enable pet owners to create and manage detailed profiles for each pet they register, including medical and scheduling information.

Actors:

- **Pet Owner**

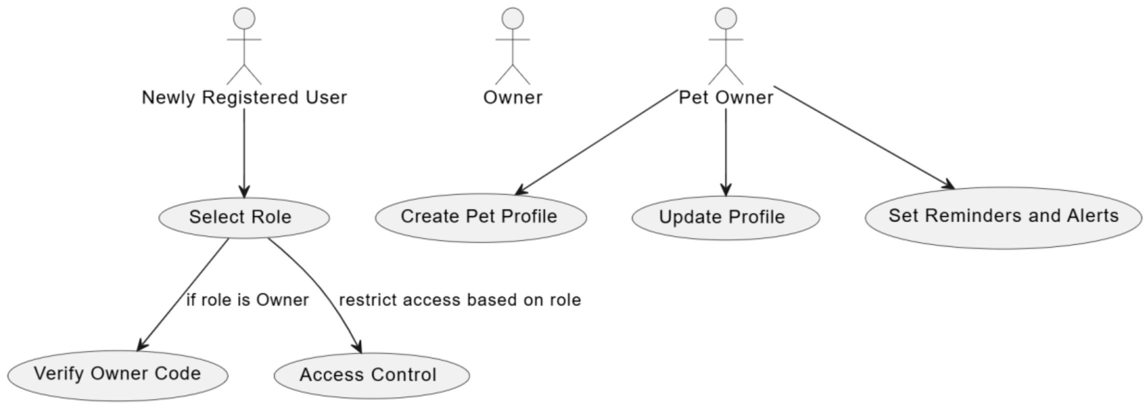
Description:

- **Create Pet Profile:** Pet owners can enter information such as the pet's name, breed, age, vaccination records, medical history, feeding schedules, and reminders.

   - **Future Feature:** Allow users to upload photos to personalize pet profiles.

- **Update Profile:** Owners can modify or delete pet information as needed, keeping the pet's profile up-to-date.

- **Reminders and Alerts:** Pet owners can set alerts for vaccinations, checkups, and feeding schedules, helping them manage pet care.



6.4. Community Interaction

Goal:

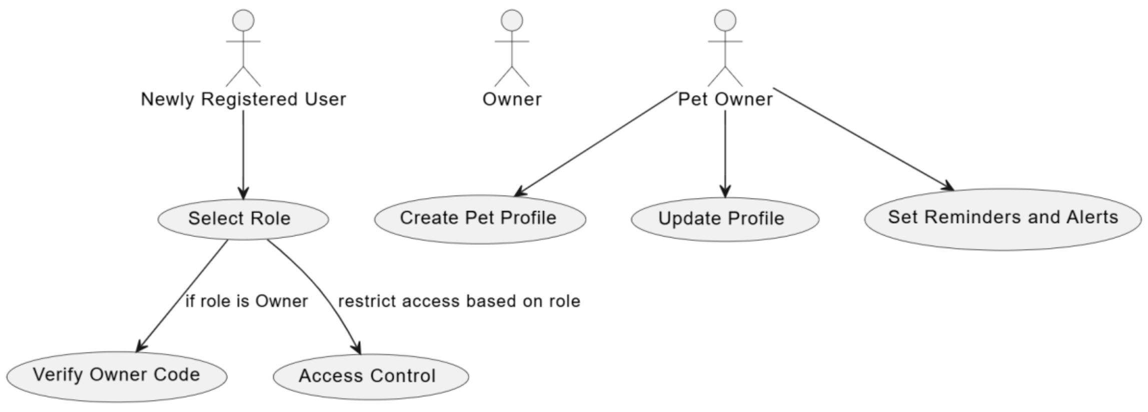
Create a space where users can engage, ask questions, share pet-related stories, and offer advice, promoting a sense of community among pet owners and enthusiasts.

Actors:

- **Pet Owner**
- **Non-Pet Owner**
- **Company/Animal Shelter**

Description:

- **Post and Share:** Users can create posts to share pet stories, photos, or seek advice from other users.
- **Comment and Like:** Users can engage with posts through comments and likes.
  - **Future Feature:** Allow users to follow each other to build connections within the community.
- **Content Moderation:** Users can report inappropriate posts, which are flagged for review by the owner.
  - **Owner's Access:** Owners have the ability to moderate reported content and ban users if necessary.



6.5 Appointment Scheduling (for Animal Shelters)

Goal:

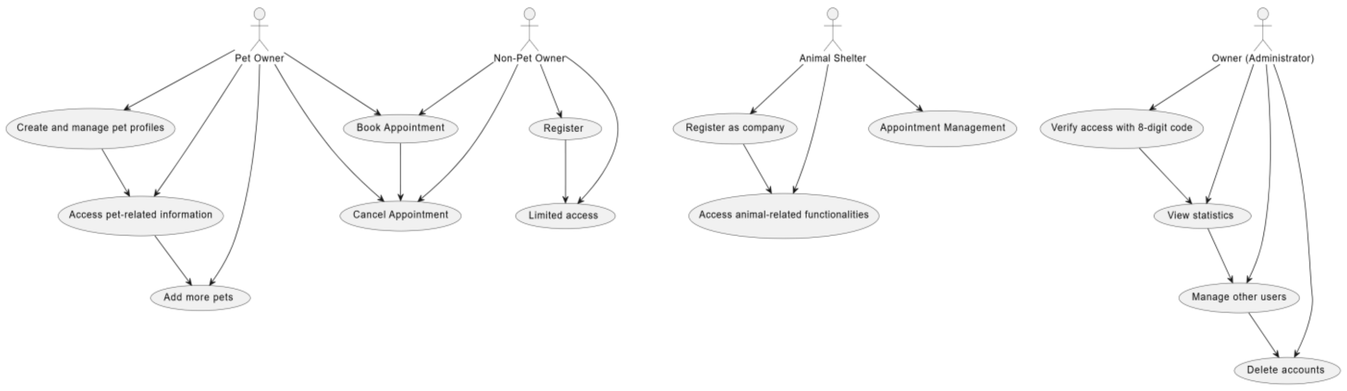
Allow animal shelters to set available time slots for adoption consultations or veterinary services, which users can book.

Actors:

- **Pet Owner**
- **Non-Pet Owner**
- **Company/Animal Shelter (Company)**

Description:

- **Book Appointment:** Users can view available times and book appointments for consultations or services.
- **Cancel Appointment:** Users can cancel appointments if needed.
- **Appointment Management (for Shelters):** Shelters can manage their appointment slots, confirm bookings, and set available times.



6.6. Owner Analytics Dashboard

Goal:

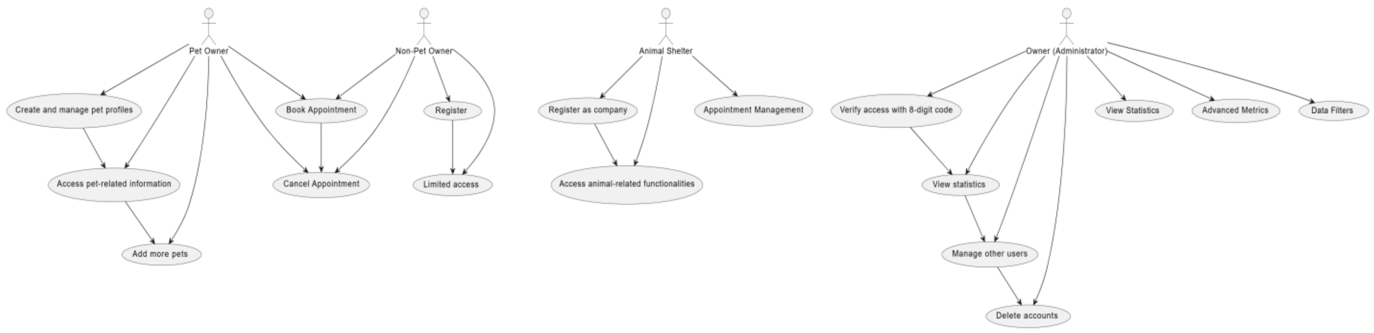
Provide a detailed analytics dashboard with insights into user activity, pet registrations, and general app engagement metrics.

Actors:

- **Owner**

Description:

- **View Statistics:** Owners can view data about the app's usage, including the number of registered users, types of pets registered, and trends in user activity.
- **Advanced Metrics:** The dashboard includes graphs and charts showing metrics such as the most popular pet breeds, new registrations by date, and active users.
- **Data Filters:** Owners can filter data by date, user type, or activity type for deeper insights.



6.7. Location-Based Recommendations

Goal:

Use location data to recommend nearby services and pet-friendly spaces, making it easier for users to access resources.

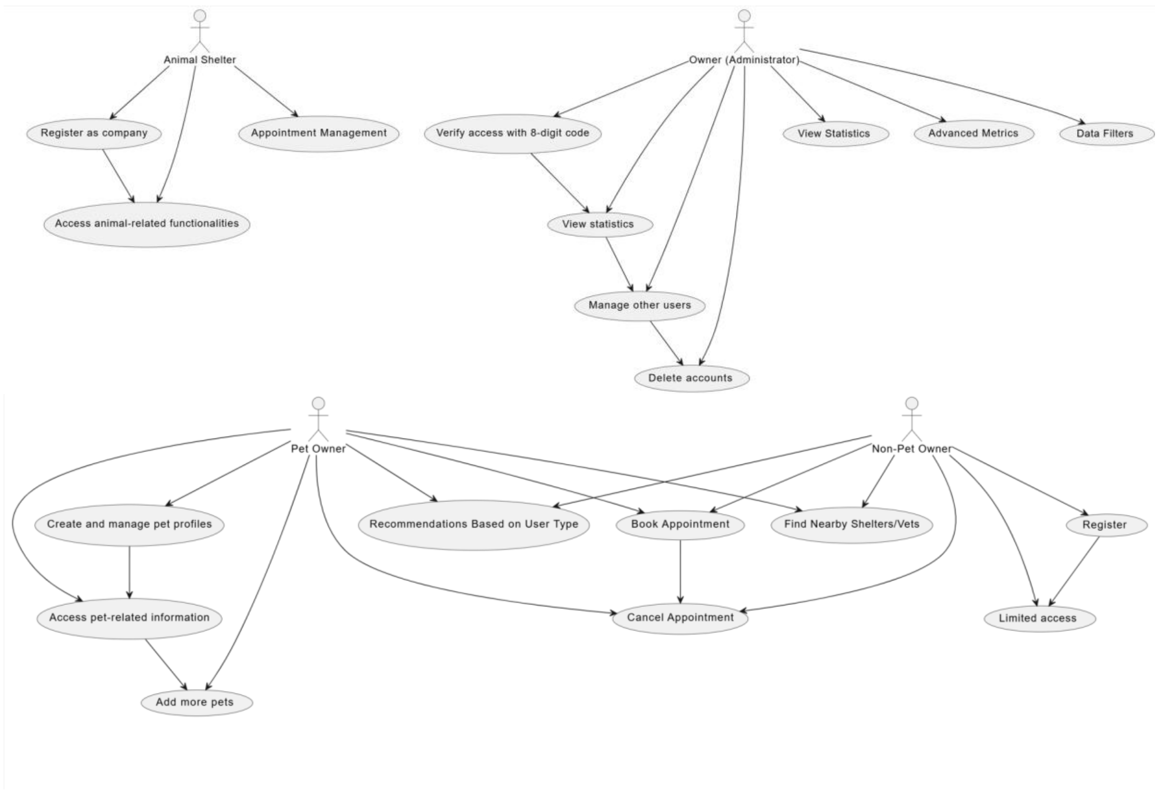
Actors:

- **Pet Owner**
- **Non-Pet Owner**

Description:

- **Find Nearby Shelters/Vets:** Users can view nearby animal shelters, veterinary clinics, and pet-friendly parks based on their location.

- **Recommendations Based on User Type:** Pet owners may see different recommendations than non-pet owners, such as tips on pet care and adoption.



6.8. Messaging and Notifications

Goal:

Enable direct communication between users and animal shelters, improving interactions and providing timely reminders.

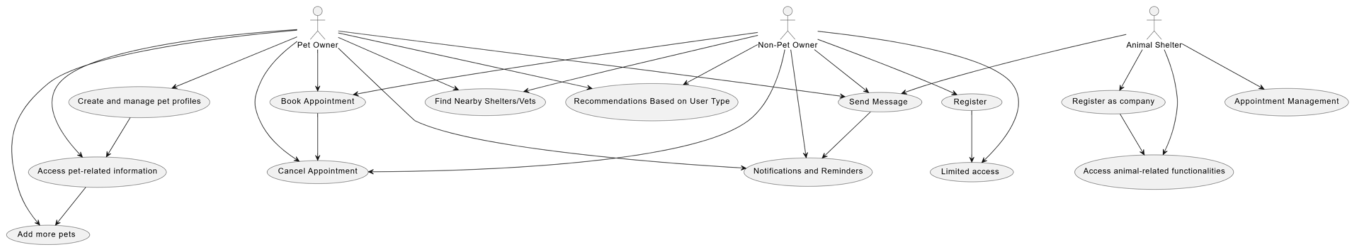
Actors:

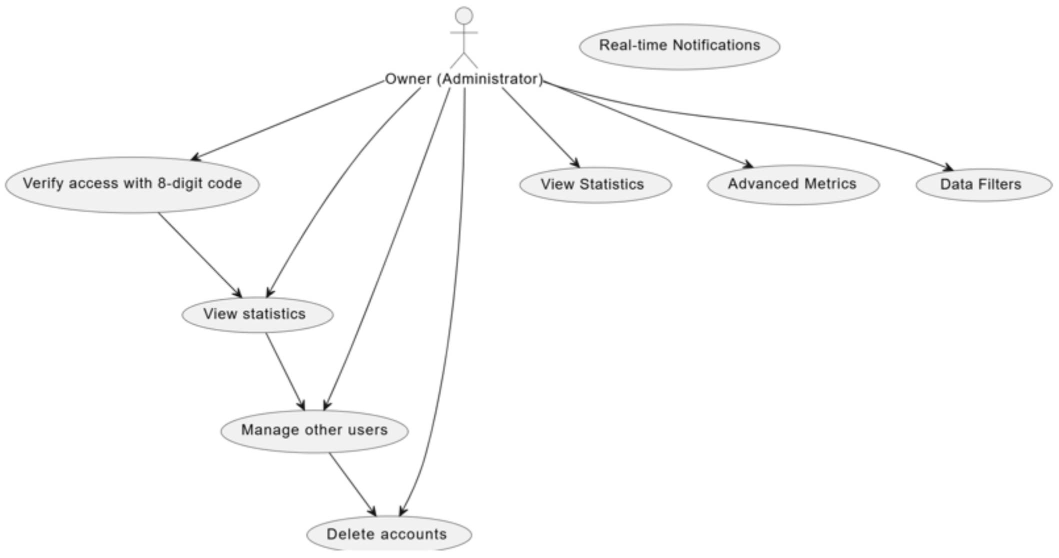
- **Pet Owner**
- **Non-Pet Owner**

- **Company/Animal Shelter**

Description:

- **Send Message:** Users can message animal shelters for inquiries, consultations, or assistance with adoptions.
- **Notifications and Reminders:** Users receive notifications for new messages, upcoming appointments, and reminders for pet care.
    - **Future Feature:** Real-time notifications about community updates, appointment confirmations, and pet-related alerts.


Pet Owner
Non-Pet Owner
Animal Shelter
Create and manage pet profiles
Book Appointment
Find Nearby Shelters/Vets
Recommendations Based on User Type
Send Message
Register
Register as company
Appointment Management
Access pet-related information
Cancel Appointment
Notifications and Reminders
Limited access
Access animal-related functionalities
Add more pets



6.9. User Account Management and Moderation (for Owners)

Goal:

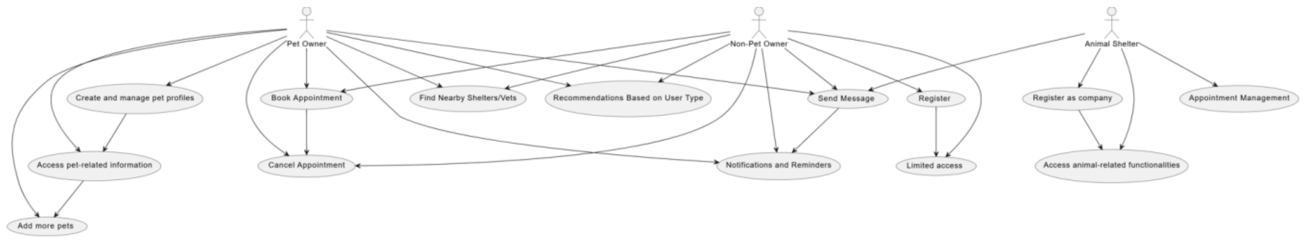
Allow owners to manage and moderate user accounts, giving them control over banned users and flagged content.

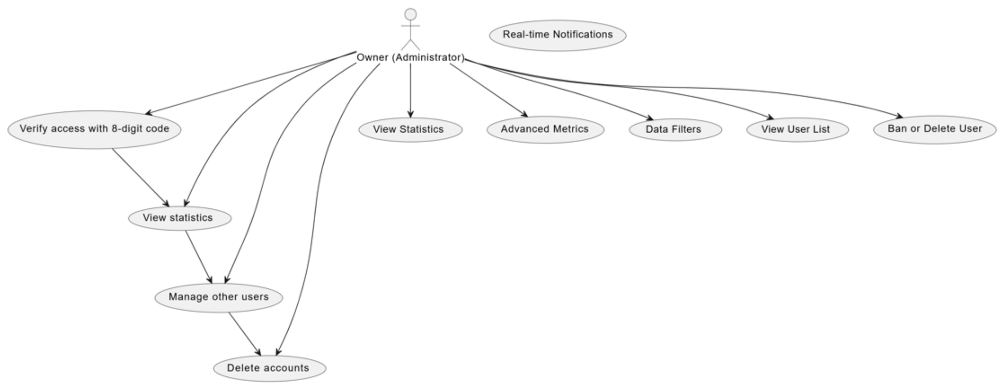
Actors:

- **Owner**

Description:

- **View User List:** Owners can view all registered users, sorted by registration date, including details on registered pets and any flagged content.
- **Ban or Delete User:** Owners have the authority to ban users based on policy violations or delete accounts upon user request.
  - **Future Feature:** Owners can view detailed reports for each banned user and monitor compliance with community guidelines.


Pet Owner
Non-Pet Owner
Animal Shelter
Create and manage pet profiles
Book Appointment
Find Nearby Shelters/Vets
Recommendations Based on User Type
Send Message
Register
Register as company
Appointment Management
Access pet-related information
Cancel Appointment
Notifications and Reminders
Limited access
Access animal-related functionalities
Add more pets



6.10. In-App Purchases and Donations

Goal:

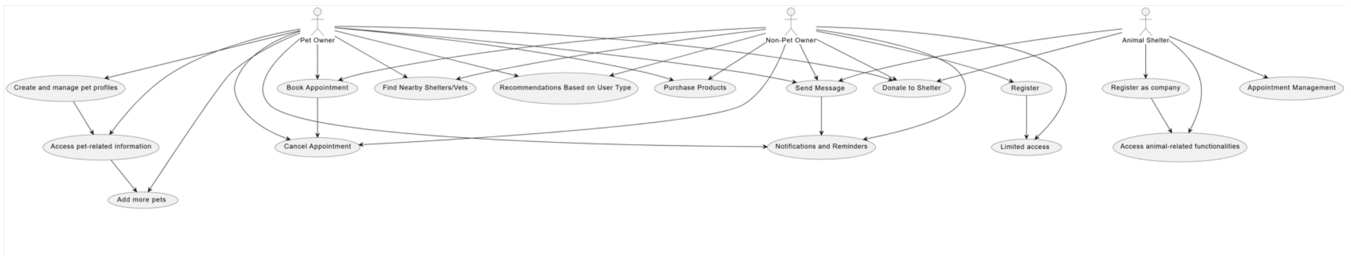
Allow animal shelters to accept donations and let users purchase pet-related products directly through the app.

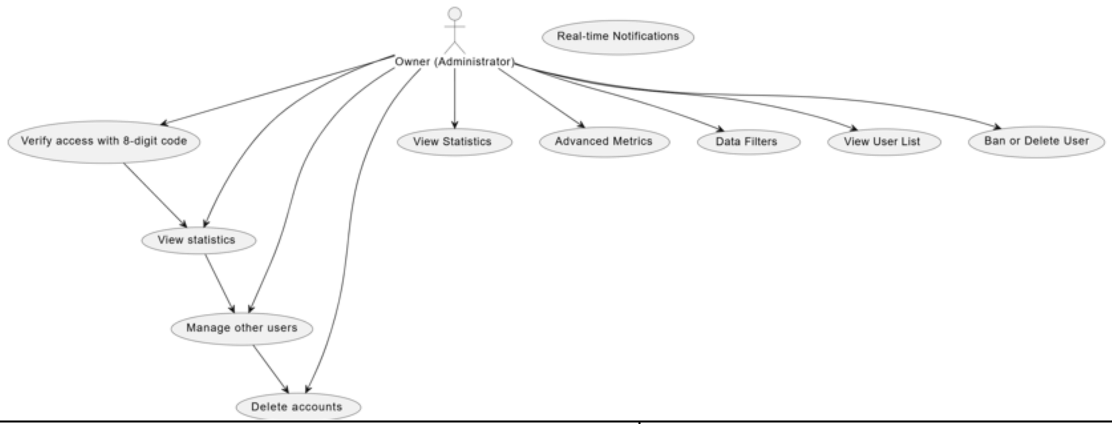
Actors:

- **Pet Owner**
- **Non-Pet Owner**
- **Company/Animal Shelter**

Description:

- **Donate to Shelter:** Users can donate funds to animal shelters to support their activities and services.
- **Purchase Products:** Users can buy pet supplies directly from the app, benefiting shelters through affiliate sales or in-app promotions.





7. System Architecture

The system architecture of this project is based on a client-server model, with the current implementation using **React Native** for the client-side application and **AsyncStorage** for local data persistence. In future developments, a backend server and a cloud database can be integrated to enhance functionality, scalability, and security. This architecture allows for modular development, role-based access control, and smooth data handling across different user roles.

Current Architecture

1. **Client-Side (Front-End)**

    - **Framework:** The mobile app is built using **React Native**, which provides cross-platform support for both Android and iOS, allowing a single codebase to be deployed across devices.
    - **User Interface (UI):** The UI is designed using React Native components and styled components. It supports dynamic role-based screens and allows each user role (Pet Owner, Non-Pet Owner, Company, and Owner) to see only relevant features and navigation paths.
    - **Navigation:** The app uses **React Navigation** to manage screen transitions, allowing different flows for different user roles, such as:
        - Pet owners managing pet profiles.
        - Non-pet owners engaging with community content.
        - Companies managing appointments and donations.
        - Owners accessing statistics and moderation tools.
    - **Data Management: AsyncStorage** is used to store user information and pet profiles locally on the device. It holds data such as:
        - User details (username, password, role, pets, registration date).

- Pet profiles (name, breed, vaccination records).
- Additional settings like reminders for pet care.

2. **Data Layer**
   - **Local Storage: AsyncStorage** serves as the local database to manage data locally on the device, providing a lightweight solution for data persistence. It supports key functionalities without requiring an internet connection.
   - **Data Synchronization (Future Implementation):** In the future, a cloud database (e.g., Firebase or a custom server) could be added to sync local data with a centralized backend, enabling:
     - Real-time data synchronization across devices.
     - User access to data even after reinstalling the app.
     - Centralized control and better data security.
   - **Security Considerations:** Currently, the system uses simple password-based authentication stored locally. Future developments will prioritize enhanced security measures, including encrypted storage and secure user authentication.
3. **Server-Side (Future Development)**
   - **Backend Server (Planned):** A backend server (Node.js or similar) can be implemented to handle requests, store data securely, and process statistics. The server can also be responsible for handling sensitive administrative tasks, such as banning users or managing user reports.
   - **Cloud Database (Planned):** A cloud-hosted database like Firebase, MongoDB, or SQL could replace AsyncStorage for improved data persistence and security, providing:
     - Centralized storage of user accounts, pet profiles, and user activity logs.
     - Backup and recovery options.

- Scalable storage for large volumes of user-generated data and community posts.

- **Role-Based API Access:** The server will implement APIs that enforce role-based access. Owners, for instance, can access statistics and user data, whereas Pet Owners and Companies have limited access to their own data.

4. **Role-Based Access Control (RBAC)**

- The system architecture includes **Role-Based Access Control** to restrict access to specific features based on the user's role.
  - **Pet Owners** can manage pet profiles and access reminders.
  - **Non-Pet Owners** can interact with community content.
  - **Companies** have access to features like appointment scheduling and donations.
  - **Owners** have advanced permissions, including accessing statistics and user management (e.g., banning users).
- This modular approach ensures each user type experiences a tailored interaction with the app, enhancing security and usability.

5. **Future Enhancements for System Architecture**

- **User Authentication and Authorization:** Migrating to a secure authentication system (such as Firebase Auth or OAuth) would enhance login security, prevent unauthorized access, and facilitate role-based access control on the server side.
- **Analytics and Monitoring:** Adding tools like Google Analytics or Firebase Analytics will provide insights into user behavior and app usage, allowing the system to optimize and adapt to user needs.
- **Push Notifications and Real-Time Updates:** To improve user engagement, push notifications for pet care reminders, appointment confirmations, or community interactions could be implemented.

- **Reporting and Moderation Tools:** Building a reporting system for flagged content and a moderation dashboard for Owners will foster a safer community.

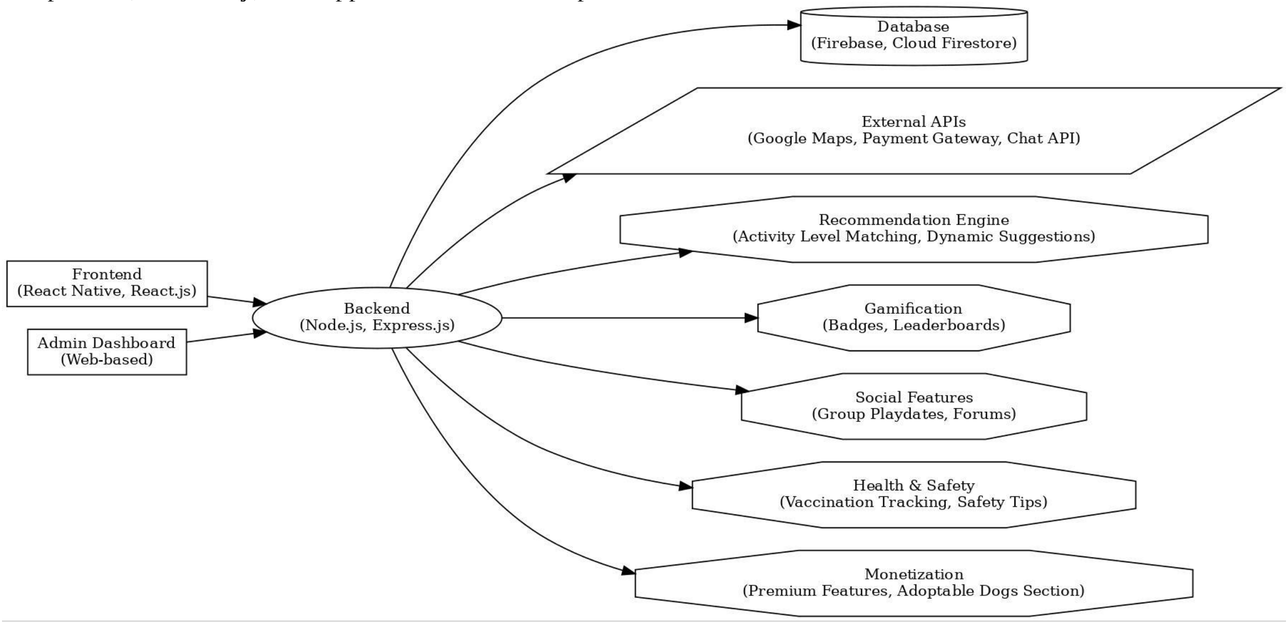
Workflow and Data Flow

1. **User Workflow:**
   - Users register and select a role during onboarding.
   - Based on the role, users are presented with specific screens and features.
   - Data like pet profiles or appointment schedules are stored locally and can be synced to a central server in future development.
2. **Data Flow:**
   - **Registration/Login:** User data (username, password, role) is stored locally in AsyncStorage.
   - **Pet Profile Management:** Pet data, including reminders, is stored in AsyncStorage for Pet Owners, with plans to sync data with a central server.
   - **Statistics for Owners:** User activity data is collected and stored locally, but will eventually be managed by a server to provide real-time statistics, user management, and secure access control.
3. **Modular Components:**
   - **Role-Based Navigation Module:** Manages screen flows based on user roles.
   - **Data Storage Module:** AsyncStorage handles local storage; future cloud integration will enable centralized data access.
   - **Community Interaction Module (Future):** Enables social interactions among users with moderation tools for Owners.

8. Design (UML Diagrams)

- General Architecture

The general architecture diagram for the **Furry Friends** application highlights its modular and scalable structure. At the core lies the **Backend** powered by **Node.js and Express.js**, which acts as the central hub connecting various components. The **Frontend** includes mobile and web applications developed using **React Native** and **React.js**, providing user interfaces for pet owners, non-pet owners, and shelters. An **Admin Dashboard**, also web-based, interacts with the backend for administrative controls. The backend integrates with a **Firebase/Cloud Firestore Database** for secure storage of user, dog, and event data. Additionally, it communicates with **External APIs** such as Google Maps for location-based services, a payment gateway for monetization, and a chat API for real-time interactions. Future enhancements include a **Recommendation Engine** for intelligent matching, **Gamification Features** like badges and leaderboards, and **Social Features** such as group playdates and forums. **Health & Safety** services will ensure verification of vaccination and provide safety tips, while **Monetization** includes premium features and a dedicated adoptable dogs section. The architecture ensures smooth communication between components, scalability, and support for future developments.


Database (Firebase, Cloud Firestore)
External APIs (Google Maps, Payment Gateway, Chat API)
Recommendation Engine (Activity Level Matching, Dynamic Suggestions)
Frontend (React Native, React.js)
Backend (Node.js, Express.js)
Admin Dashboard (Web-based)
Gamification (Badges, Leaderboards)
Social Features (Group Playdates, Forums)
Health & Safety (Vaccination Tracking, Safety Tips)
Monetization (Premium Features, Adoptable Dogs Section)

- FlowChart/Site Map Diagram

The **flowchart/site map diagram** for the **Furry Friends** application outlines the hierarchical navigation and features accessible to different user types. The entry point is the **Home Screen**, where users can choose to log in or register. Upon successful login, users are directed to the **User Dashboard**, which branches into features based on their roles:

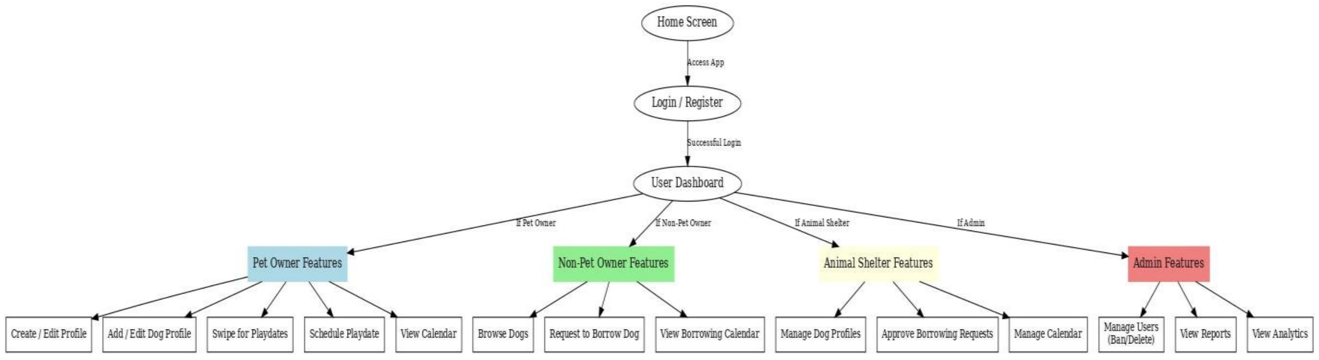
1. **Pet Owners**: Can create or edit profiles for themselves and their dogs, swipe for playdates, schedule playdates, and view their calendar to manage scheduled activities.

2. **Non-Pet Owners**: Can browse dog profiles, request to borrow a dog from a shelter, and view their borrowing schedule on a dedicated calendar.

3. **Animal Shelters**: Have access to manage dog profiles, approve borrowing requests from non-pet owners, and organize borrowing schedules using the calendar.

4. **Admins**: Possess administrative controls, including managing user profiles, handling reports, and accessing analytics and dashboards to oversee platform activity.

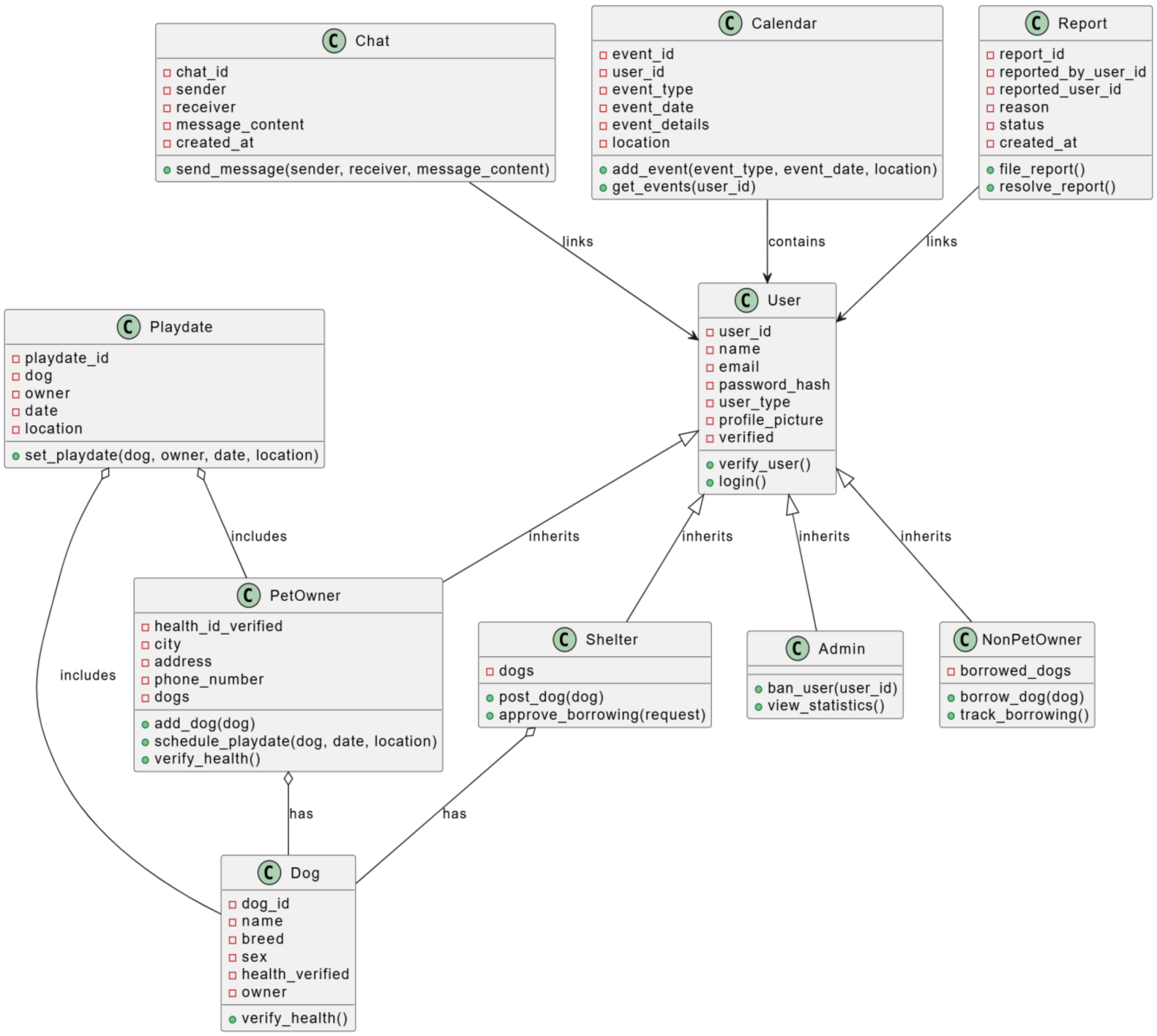
The diagram effectively illustrates the distinct pathways and exclusive features for each user type, ensuring a clear understanding of their navigation and functionality within the application. It also emphasizes role-based customization and the structured flow of activities from general access points to specific feature sets.



- Class Diagram

The **class diagram** for the **Furry Friends** application showcases the main entities, their attributes, methods, and relationships. At its core is the `User` class, inherited by specific roles: `PetOwner`, `NonPetOwner`, `Shelter`, and `Admin`. Each role has unique attributes and methods, such as `PetOwner` managing dogs and scheduling playdates, `NonPetOwner` borrowing dogs, and `Admin` overseeing platform management. Supporting classes like `Dog`, `Playdate`, `Chat`, `Calendar`, and `Report` handle interactions such as playdate scheduling, user communication, event management, and issue reporting. Relationships are well-defined, with users linked to their dogs and playdates, while a centralized system tracks chats, reports, and calendar events, ensuring modularity and clear role-specific responsibilities.

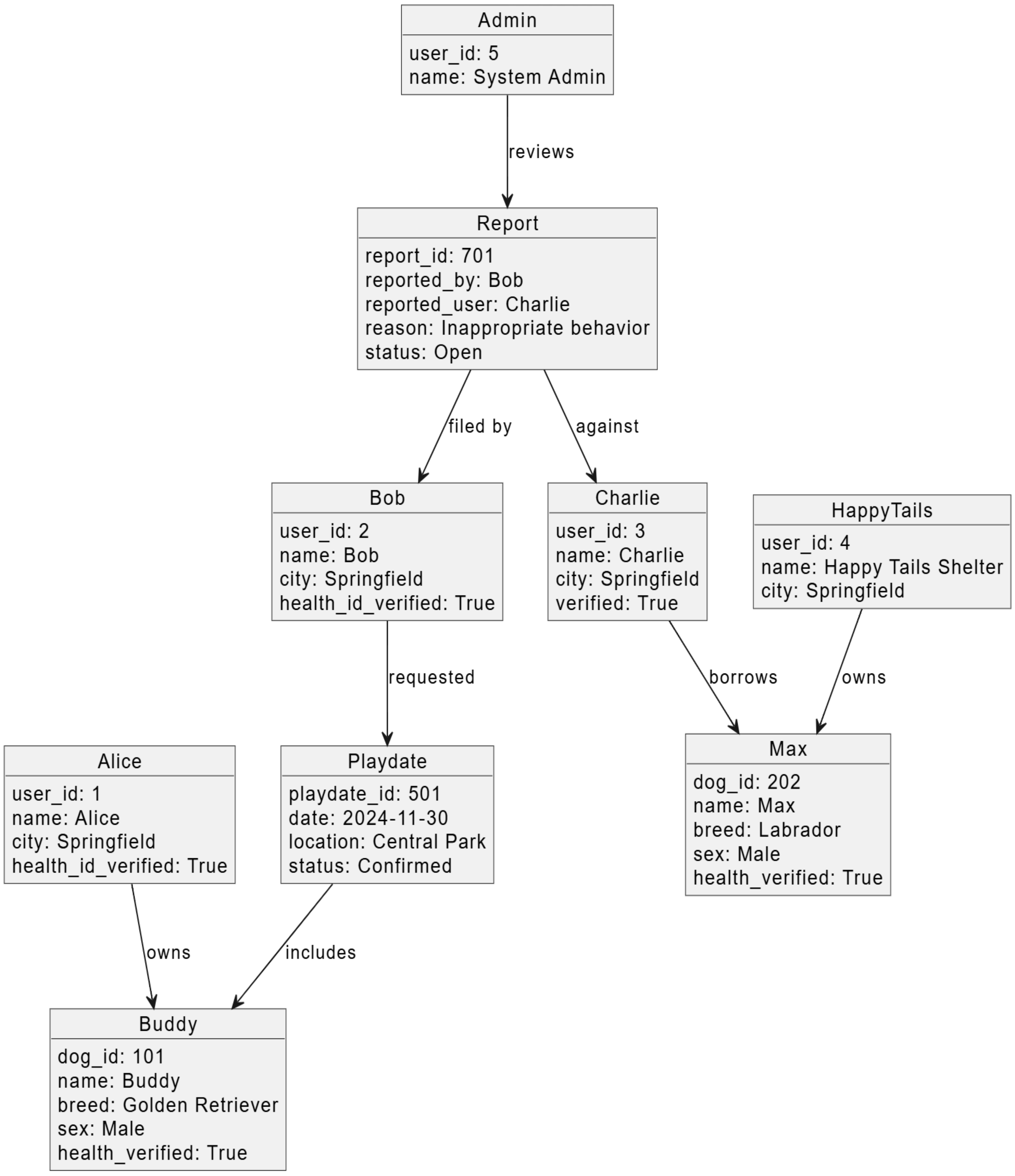
Recommended: 20.11.2017	**Approved:**
Company/University Name: Address: Universitatea Tenhica Cluj-Napoca Romania	



- Object Diagram

The **object diagram** for the **Furry Friends** application provides a snapshot of specific instances of users, dogs, playdates, and reports within the system. It showcases interactions between entities, such as **Alice** (a pet owner) owning a dog named **Buddy**, who is scheduled for a playdate at Central Park with **Bob**. Meanwhile, **Bob** files a report against **Charlie** for inappropriate behavior, which is reviewed by the **Admin**. **Charlie**, a verified user, borrows a dog named **Max** from the **Happy Tails Shelter**, demonstrating how shelters manage dogs available for borrowing. This diagram emphasizes the relationships between objects (e.g., ownership, borrowing, reporting) and the flow of responsibilities, illustrating the system's real-world interactions and role-specific functionalities.

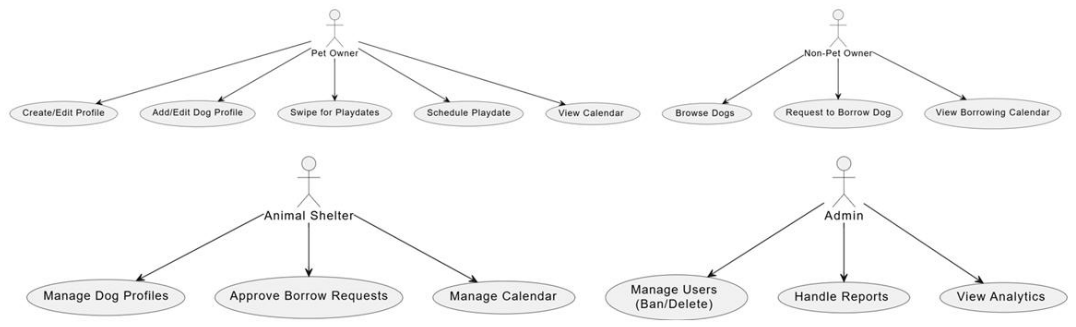
Recommended: 20.11.2017	**Approved:**
Company/University Name: Address: Universitatea Tenhica Cluj-Napoca Romania	



Admin
user_id: 5
name: System Admin
reviews
Report
report_id: 701
reported_by: Bob
reported_user: Charlie
reason: Inappropriate behavior
status: Open
filed by
against
Bob
user_id: 2
name: Bob
city: Springfield
health_id_verified: True
Charlie
user_id: 3
name: Charlie
city: Springfield
verified: True
HappyTails
user_id: 4
name: Happy Tails Shelter
city: Springfield
requested
borrows
owns
Alice
user_id: 1
name: Alice
city: Springfield
health_id_verified: True
Playdate
playdate_id: 501
date: 2024-11-30
location: Central Park
status: Confirmed
Max
dog_id: 202
name: Max
breed: Labrador
sex: Male
health_verified: True
owns
includes
Buddy
dog_id: 101
name: Buddy
breed: Golden Retriever
sex: Male
health_verified: True

- Use Cases Diagram

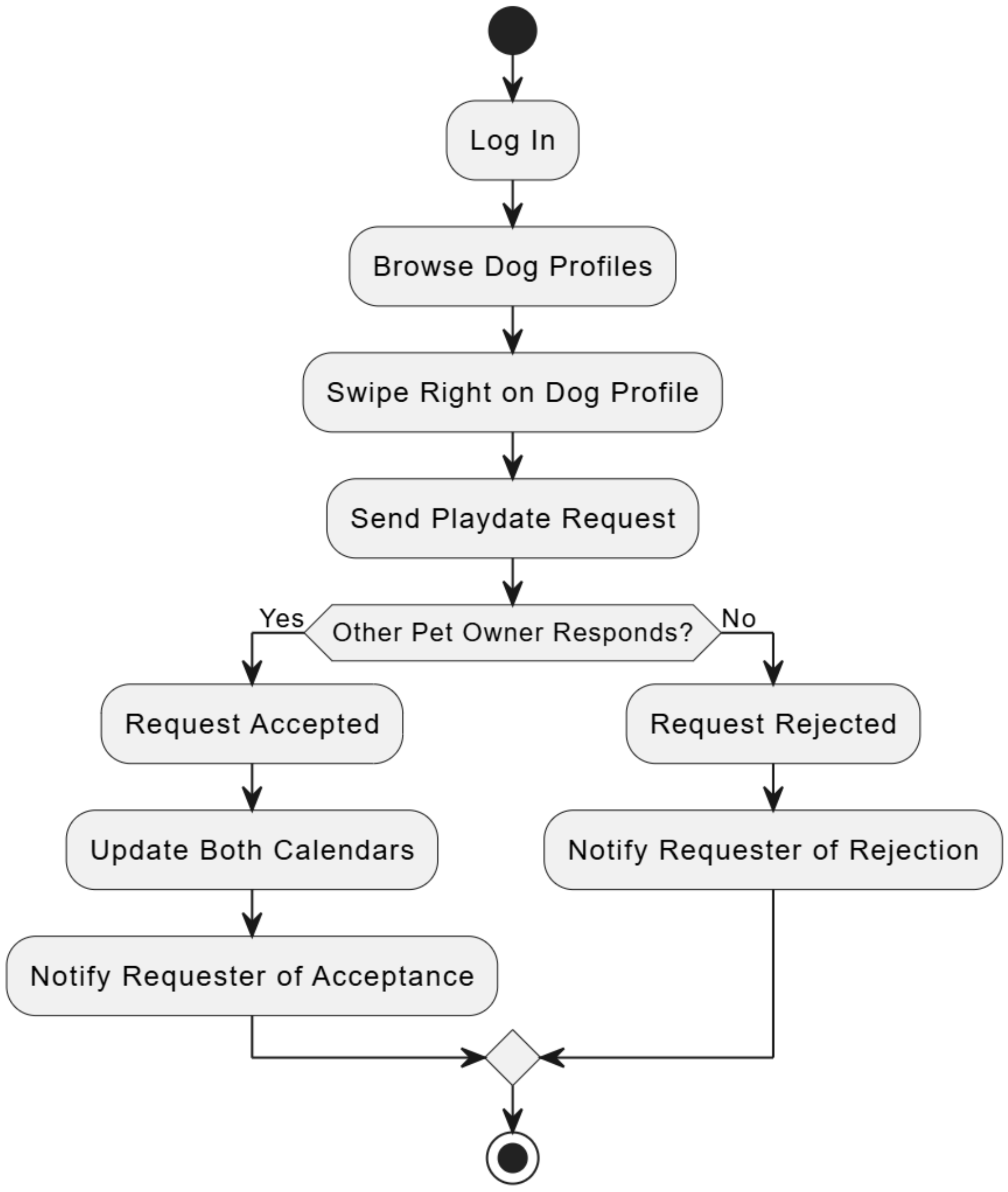
The **use case diagram** for the **Furry Friends** application illustrates the core functionalities available to each user type: **Pet Owner**, **Non-Pet Owner**, **Animal Shelter**, and **Admin**. Pet owners can create and edit their profiles, manage dog profiles, swipe for playdates, schedule playdates, and view their calendar. Non-pet owners can browse dog profiles, request to borrow a dog, and track borrowing schedules using their calendar. Animal shelters are responsible for managing dog profiles, approving borrowing requests, and organizing schedules in their calendar. Admins have overarching control, including managing user accounts (ban/delete), handling reports, and viewing platform analytics. This diagram emphasizes the role-specific interactions and the system's modular structure, ensuring all user needs are efficiently addressed.



- Sequence Diagram

The **sequence diagram** for the **Furry Friends** application illustrates the process of scheduling a playdate between two pet owners. The sequence begins with the user logging in and browsing available dog profiles. Upon swiping right on a desired profile, a playdate request is sent to the other pet owner. If the other pet owner responds positively, the request is accepted, and both calendars are updated to reflect the scheduled playdate, followed by a confirmation notification to the requester. If the request is rejected, the requester is notified of the rejection. The flow ensures a smooth interaction, with clear handling of both acceptance and rejection scenarios, emphasizing the app's role in facilitating communication and organization between users.

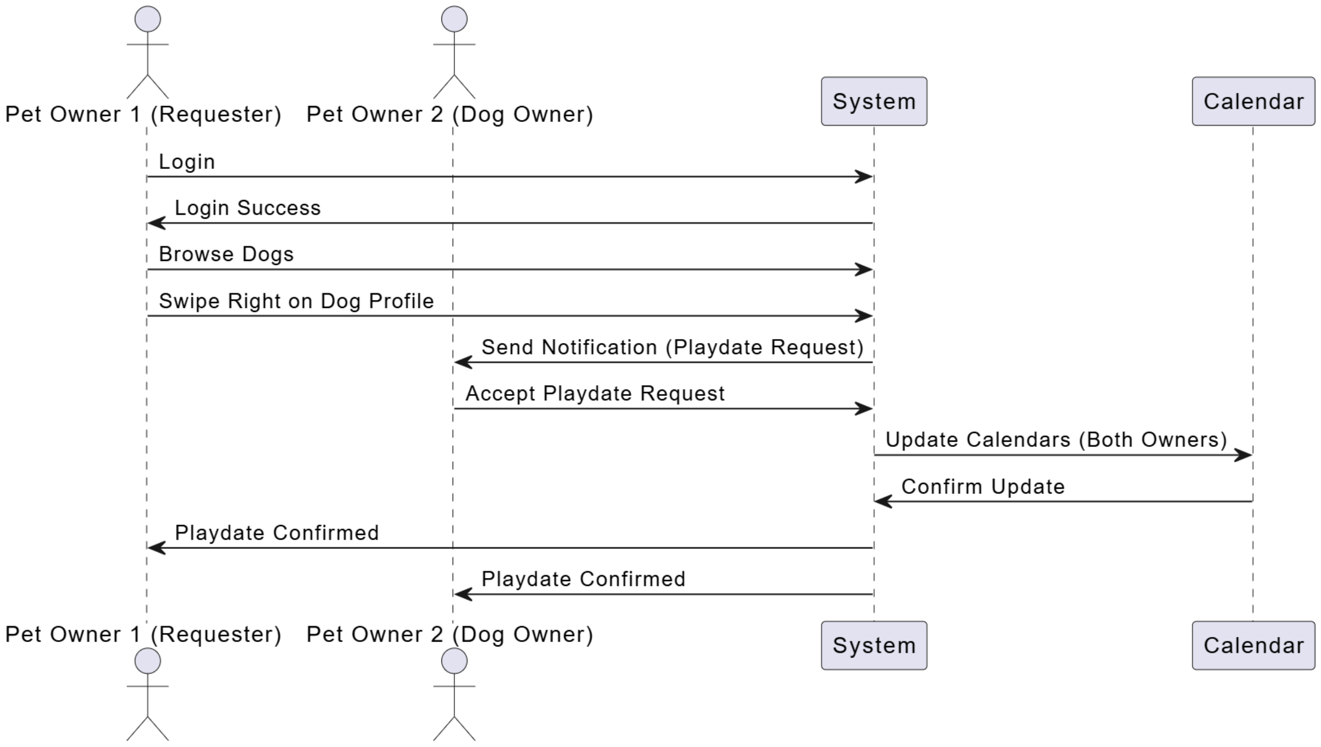
Recommended: 20.11.2017	**Approved:**
Company/University Name: Address: Universitatea Tenhica Cluj-Napoca Romania	



Log In
Browse Dog Profiles
Swipe Right on Dog Profile
Send Playdate Request
Other Pet Owner Responds?
Yes
No
Request Accepted
Update Both Calendars
Notify Requester of Acceptance
Request Rejected
Notify Requester of Rejection

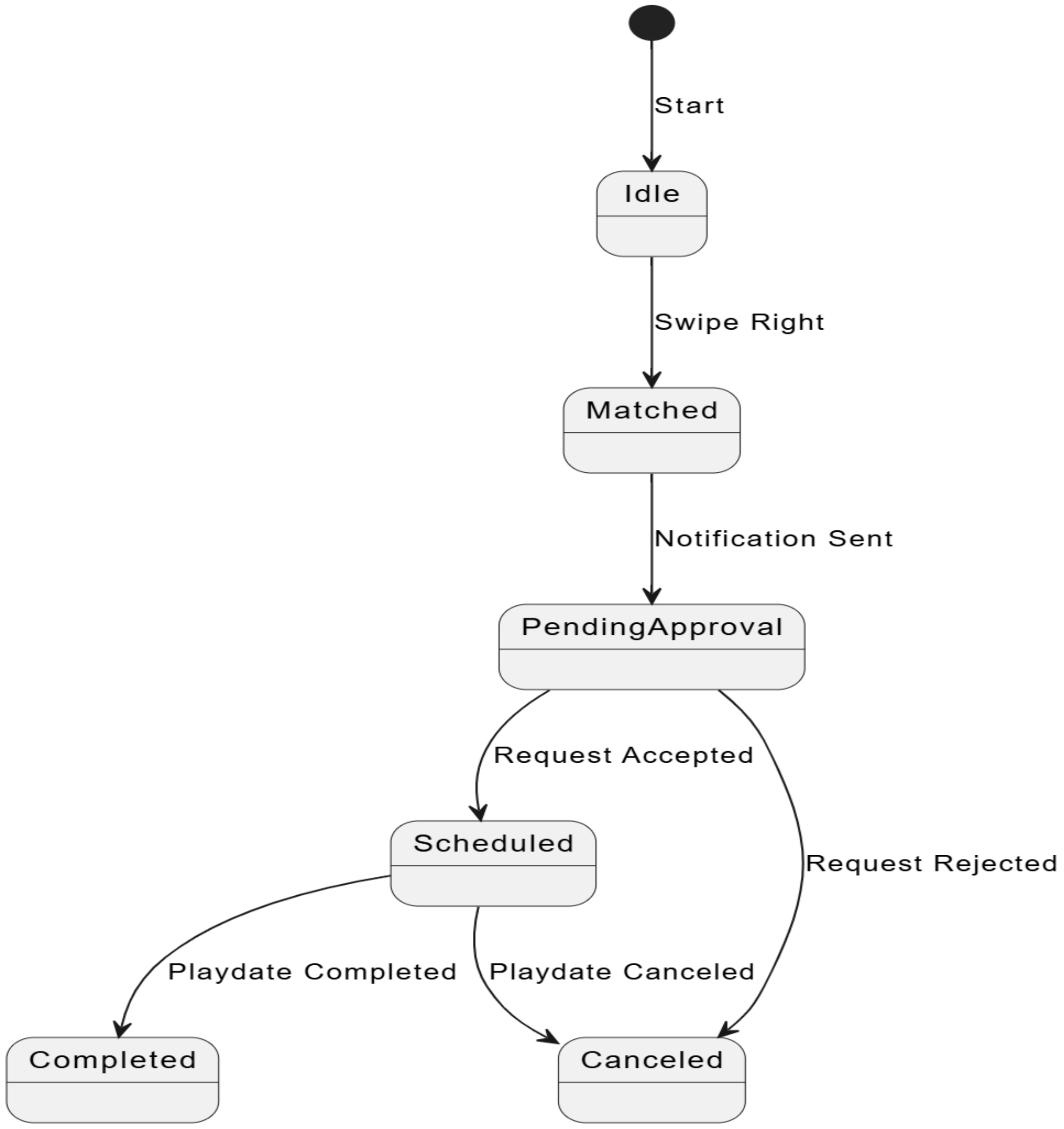
- Activity Diagram

The **activity diagram** for the **Furry Friends** application demonstrates the process of arranging a playdate between two pet owners. The sequence begins with **Pet Owner 1 (Requester)** logging in and successfully accessing the system. After browsing dog profiles, they swipe right on a desired dog, triggering the system to send a playdate request notification to **Pet Owner 2 (Dog Owner)**. Upon receiving and accepting the request, the system updates the calendars for both owners with the scheduled playdate details. A confirmation of the update is sent back to both users, finalizing the playdate arrangement. This diagram highlights the smooth interaction between the users, the system, and the calendar, ensuring efficient scheduling and clear communication throughout the process.



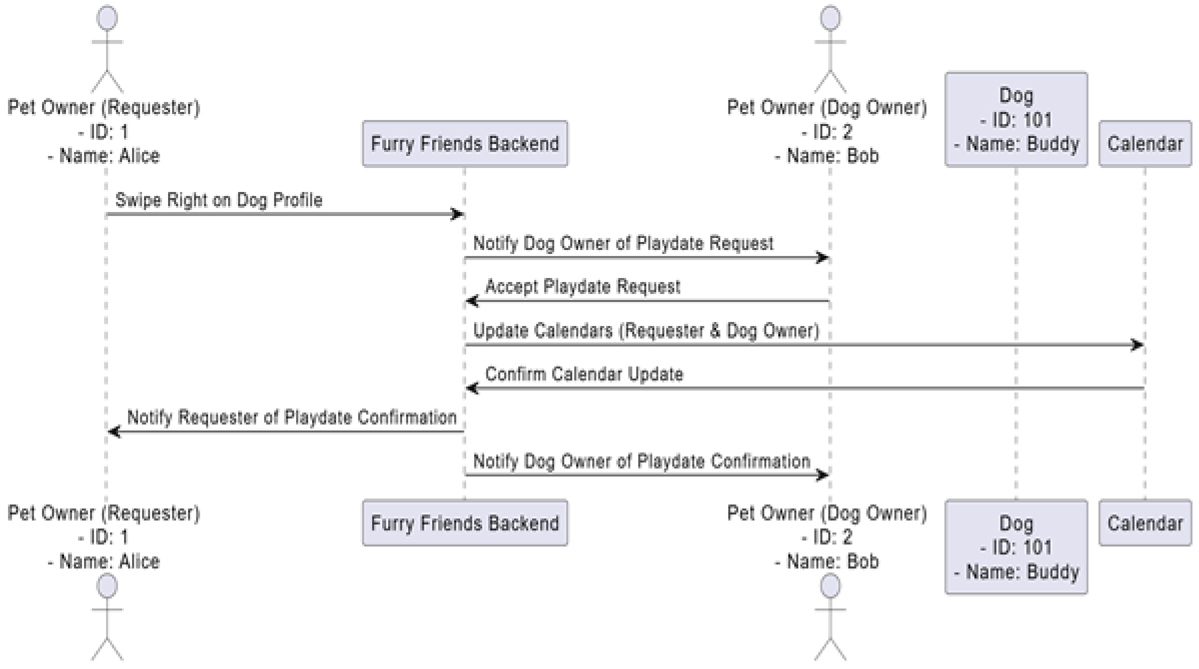
- State Transition Diagram

The **state transition diagram** for the **Furry Friends** application outlines the lifecycle of a playdate request. The process begins in the **Idle** state, where a user swipes right on a dog profile, transitioning the system to the **Matched** state. A notification is sent to the other pet owner, moving the request into the **PendingApproval** state. From here, two outcomes are possible: if the request is accepted, the system transitions to the **Scheduled** state; if rejected, the request moves to the **Canceled** state. Once scheduled, the playdate progresses to either **Completed** (after a successful event) or **Canceled** (if the playdate is called off). This diagram effectively captures the different states of a playdate request and the transitions driven by user actions or outcomes.



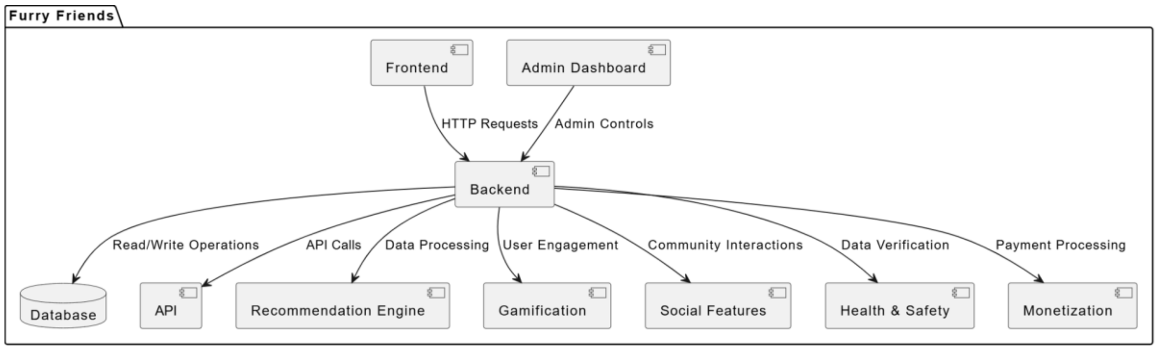
- Communication Diagram

The **communication diagram** for the **Furry Friends** application illustrates the interactions and message exchanges between key entities during the process of scheduling a playdate. It starts with **Pet Owner 1 (Alice)** swiping right on a dog profile, triggering the **Furry Friends Backend** to notify **Pet Owner 2 (Bob)** of the playdate request. Upon Bob accepting the request, the backend updates the calendars for both owners and the involved dog (**Buddy**) through the **Calendar** system. A confirmation is sent back to both Alice and Bob, completing the playdate scheduling process. This diagram emphasizes the flow of messages between users, the backend, and the calendar system, ensuring synchronized updates and clear communication at every step of the process.


Pet Owner (Requester)
- ID: 1
- Name: Alice
Furry Friends Backend
Pet Owner (Dog Owner)
- ID: 2
- Name: Bob
Dog
- ID: 101
- Name: Buddy
Calendar
Swipe Right on Dog Profile
Notify Dog Owner of Playdate Request
Accept Playdate Request
Update Calendars (Requester & Dog Owner)
Confirm Calendar Update
Notify Requester of Playdate Confirmation
Notify Dog Owner of Playdate Confirmation

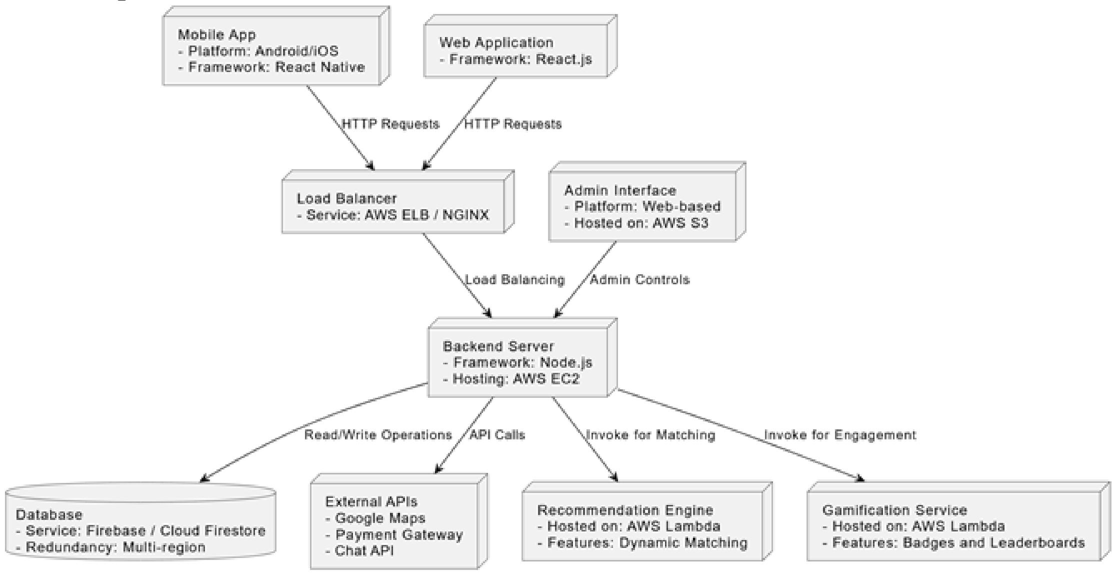
- Package Diagram

The **package diagram** for the **Furry Friends** application highlights the modular organization of its system components. At the core lies the **Backend**, which serves as the central hub, managing HTTP requests from the **Frontend** (user-facing mobile and web interfaces) and administrative controls from the **Admin Dashboard**. The backend interacts with multiple supporting packages, including the **Database** for data storage and retrieval, **APIs** for external services (e.g., Google Maps, Chat, and Payments), and advanced modules such as the **Recommendation Engine** for dynamic dog matching, **Gamification** for user engagement (e.g., badges and leaderboards), and **Social Features** for community interactions like forums and group playdates. Additional packages handle **Health & Safety** for vaccination tracking and **Monetization** for premium features and payment processing. This diagram emphasizes the system's scalability and modularity, with distinct packages ensuring seamless communication and efficient functionality.



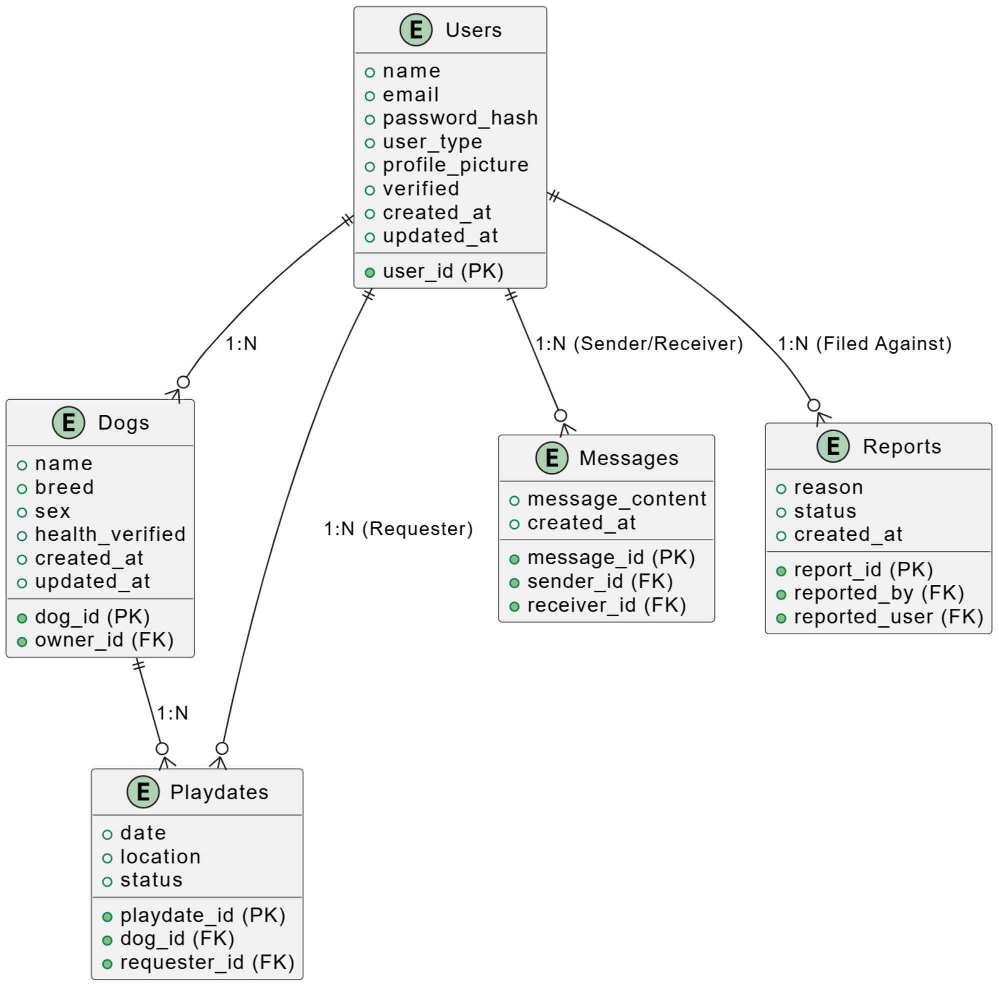
- Deployment Diagram

The **deployment diagram** for the **Furry Friends** application demonstrates the infrastructure and components supporting its operations. The system is accessed via a **Mobile App** (developed with React Native) and a **Web Application** (built with React.js), both of which communicate with a **Load Balancer** (AWS ELB/NGINX) that ensures efficient traffic distribution. Administrative tasks are handled through a **Web-based Admin Interface**, hosted on AWS S3. At the core, the **Backend Server** (Node.js, hosted on AWS EC2) processes requests and manages interactions with other components. It connects to a **Database** (Firebase/Cloud Firestore) for data storage with multi-region redundancy and integrates with **External APIs** (Google Maps, Payment Gateway, Chat API) for location services, payments, and real-time messaging. Advanced modules like the **Recommendation Engine** (for dynamic matching) and **Gamification Service** (for badges and leaderboards) are hosted on AWS Lambda, ensuring scalability and performance. This diagram highlights the robust, scalable architecture designed for seamless functionality and future growth.



- DataBase Diagram

The **database diagram** for the **Furry Friends** application illustrates the structured relationships between key entities to support the platform's functionality. The central **Users** table stores user information, including their `user_id` (primary key), email, and user type, and links to other entities. The **Dogs** table associates dogs with their owners (`owner_id` as a foreign key) and includes attributes like breed, sex, and health verification. Playdates are managed through the **Playdates** table, which tracks `dog_id`, `requester_id`, and details like date, location, and status. Communication between users is handled by the **Messages** table, storing `sender_id`, `receiver_id`, and message content. Lastly, the **Reports** table facilitates issue reporting, linking reported and reporting users, along with reasons and status. The diagram showcases one-to-many relationships (e.g., one user can own multiple dogs, send multiple messages, or schedule multiple playdates) to ensure efficient data management and seamless integration of platform features.



9. Operation Mode (Operation Guide, including screen-shots)

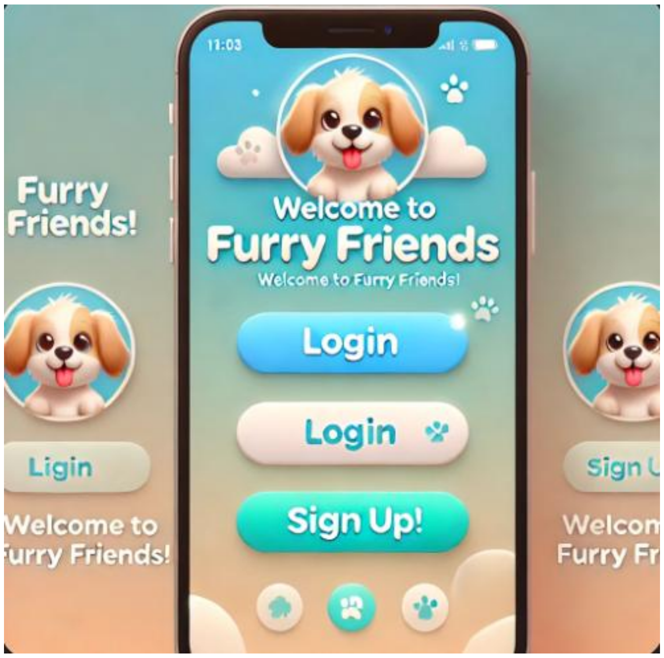
The Furry Friends application is a platform designed to connect pet owners, non-pet owners, and animal shelters for arranging playdates, borrowing dogs, and fostering community interactions. The app focuses on providing a seamless and intuitive user experience tailored to the specific roles of its users. Each user type—Pet Owner, Non-Pet Owner, Animal Shelter, and Admin—has access to distinct functionalities to ensure the platform meets their individual needs.

Key features of the application include profile creation for users and their dogs, a swiping interface for finding playmates, a chat function for coordination, and an integrated calendar to manage events. The app also incorporates verification measures to maintain trust and safety, ensuring users and their pets meet required standards. Its scalable and modular design supports smooth communication and future enhancements, such as gamification and advanced recommendation engines.

Operation Guide

Login and Registration

New users can register by entering their details, uploading a profile picture, and verifying their identity. Existing users can log in using their credentials to access a personalized dashboard. The app ensures a secure process to protect user data and establish trust.



Dashboard Navigation

Once logged in, users are directed to their role-specific dashboard:

- **Pet Owners** can manage their profiles, add or edit dog profiles, browse nearby dogs, schedule playdates, and view their calendar.
- **Non-Pet Owners** can browse shelter dog profiles, request to borrow dogs, and track their borrowing schedules.
- **Animal Shelters** manage dog availability, approve or decline borrowing requests, and update schedules.
- **Admins** monitor platform activity, manage user accounts, and resolve issues such as user reports.

Swiping for Playdates

Pet owners can swipe through profiles of nearby dogs using a Tinder-style interface. Profiles display key information such as the dog's name, breed, and age. Filters are available to refine the search based on distance, breed, and sex. Swiping right sends a playdate request, which is sent to the other dog owner for approval.

Furry Friends
Dog'
Age
Breed
Age
Breed
Ckip
Skip
Dogie
Connect
Seye

Borrowing Dogs

Non-pet owners can browse dog profiles posted by shelters and submit borrowing requests. Approved requests are added to their schedule, which is accessible through the integrated calendar. This functionality enables non-pet owners to enjoy the companionship of a dog for a day.

Scheduling and Tracking Playdates

Playdates are scheduled automatically when requests are accepted. Both users' calendars are updated to reflect the event, ensuring all parties have an organized view of upcoming activities. The app also sends reminders to users before the scheduled time.

Chat for Coordination

The chat feature allows users to communicate seamlessly to arrange details such as location and time for playdates or borrowing. Pre-saved message templates are available to make coordination faster. Users can also share photos or additional information about their dogs during conversations.

Administrative Functions

Admins have access to powerful tools to manage the platform. They can ban or delete problematic users, view detailed analytics such as active users and match success rates, and investigate reports filed by users. Admins ensure the platform remains safe and trustworthy for all member

10. Portability

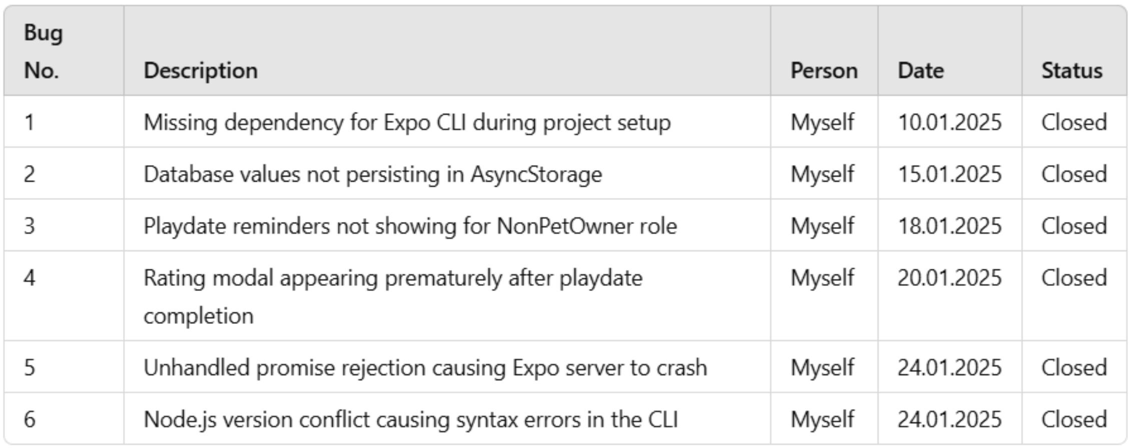
Portability

The **Furry Friends** application is designed to be highly portable, ensuring accessibility and usability across multiple platforms and devices. Developed using **React Native** for the mobile app and **React.js** for the web application, the app supports a wide range of operating systems, including **Android**, **iOS**, **Windows**, **macOS**, and **Linux**. This cross-platform compatibility ensures that users can access the application from their preferred devices, whether it is a smartphone, tablet, or desktop.

The backend, built on **Node.js** and hosted on cloud-based infrastructure such as **AWS EC2**, further enhances portability by enabling scalable deployment in diverse environments. With the integration of cloud services like **Firebase/Cloud Firestore** for the database and **AWS Lambda** for serverless components, the app can adapt to varying resource demands without compromising performance.

Additionally, the application's modular architecture allows for seamless updates and integration of new features, ensuring its adaptability to future technological advancements. By leveraging APIs and cloud-based technologies, **Furry Friends** can function reliably regardless of the underlying hardware, making it accessible to a broad user base. Whether users are in urban areas or remote locations, as long as they have internet access, they can enjoy the full functionality of the app without performance constraints. This focus on portability underscores the app's commitment to inclusivity and flexibility for all users.

11. BUGS



Bug No.	Description	Person	Date	Status
1	Missing dependency for Expo CLI during project setup	Myself	10.01.2025	Closed
2	Database values not persisting in AsyncStorage	Myself	15.01.2025	Closed
3	Playdate reminders not showing for NonPetOwner role	Myself	18.01.2025	Closed
4	Rating modal appearing prematurely after playdate completion	Myself	20.01.2025	Closed
5	Unhandled promise rejection causing Expo server to crash	Myself	24.01.2025	Closed
6	Node.js version conflict causing syntax errors in the CLI	Myself	24.01.2025	Closed

Conclusions

1. **Functional Stability**:
    - After addressing the critical bugs, the project has achieved a stable and functional state.
    - All primary features, including reminders for different user roles, playdate scheduling, chat functionality, and rating modals, work as intended.
2. **Resolved Challenges**:
    - Issues like missing dependencies, version mismatches, and AsyncStorage handling were successfully resolved, ensuring seamless integration of all components.
    - Handling asynchronous state updates in React was streamlined to avoid race conditions or unexpected behavior.
3. **User Role Specificity**:
    - Features are now tailored to each user role (PetOwner, NonPetOwner, AnimalShelter, and Owner), ensuring clarity and usability across different workflows.
    - The addition of reminders and role-specific functionalities enhanced user experience and satisfaction.
4. **Codebase Improvements**:
    - Refactoring of redundant code and optimization of asynchronous calls has resulted in better readability and maintainability of the codebase.
    - Implementation of a unified `initialize` function streamlined data fetching and reduced redundancy.
5. **System Environment**:
    - Updating the Node.js version to match project requirements resolved compatibility issues, allowing the development environment to function correctly.
    - This highlights the importance of ensuring proper environment setup for React Native projects using Expo.

6. **Project Success**:
   - The project successfully met its goals of creating a functional app with robust features for pet owners, non-pet owners, and animal shelters.
   - Issues encountered during development provided valuable learning opportunities and reinforced best practices in debugging and handling dependencies.

Final Reflection

Throughout this project, I've gained valuable insights into both technical and problem-solving skills. I've learned the importance of carefully managing dependencies and ensuring the development environment is properly configured, as even small mismatches can cause significant delays. Working with React Native and AsyncStorage taught me how to handle asynchronous operations effectively, and I now have a better understanding of managing state updates to avoid race conditions. Debugging issues like reminders not appearing for specific roles or handling unexpected behavior in the app also highlighted the value of systematic troubleshooting and the importance of testing in real-world scenarios.

Additionally, this project helped me appreciate the importance of user-specific functionality and delivering an intuitive experience tailored to different roles. From a personal perspective, it reinforced my adaptability and persistence when tackling challenges, like updating Node.js or resolving cross-platform compatibility issues. Overall, this experience has been incredibly rewarding, and I feel more confident in my ability to build scalable and user-friendly applications.

1. General Presentation

2. Theoretical Fundamentals

3. IT Technology

4. Functionalities

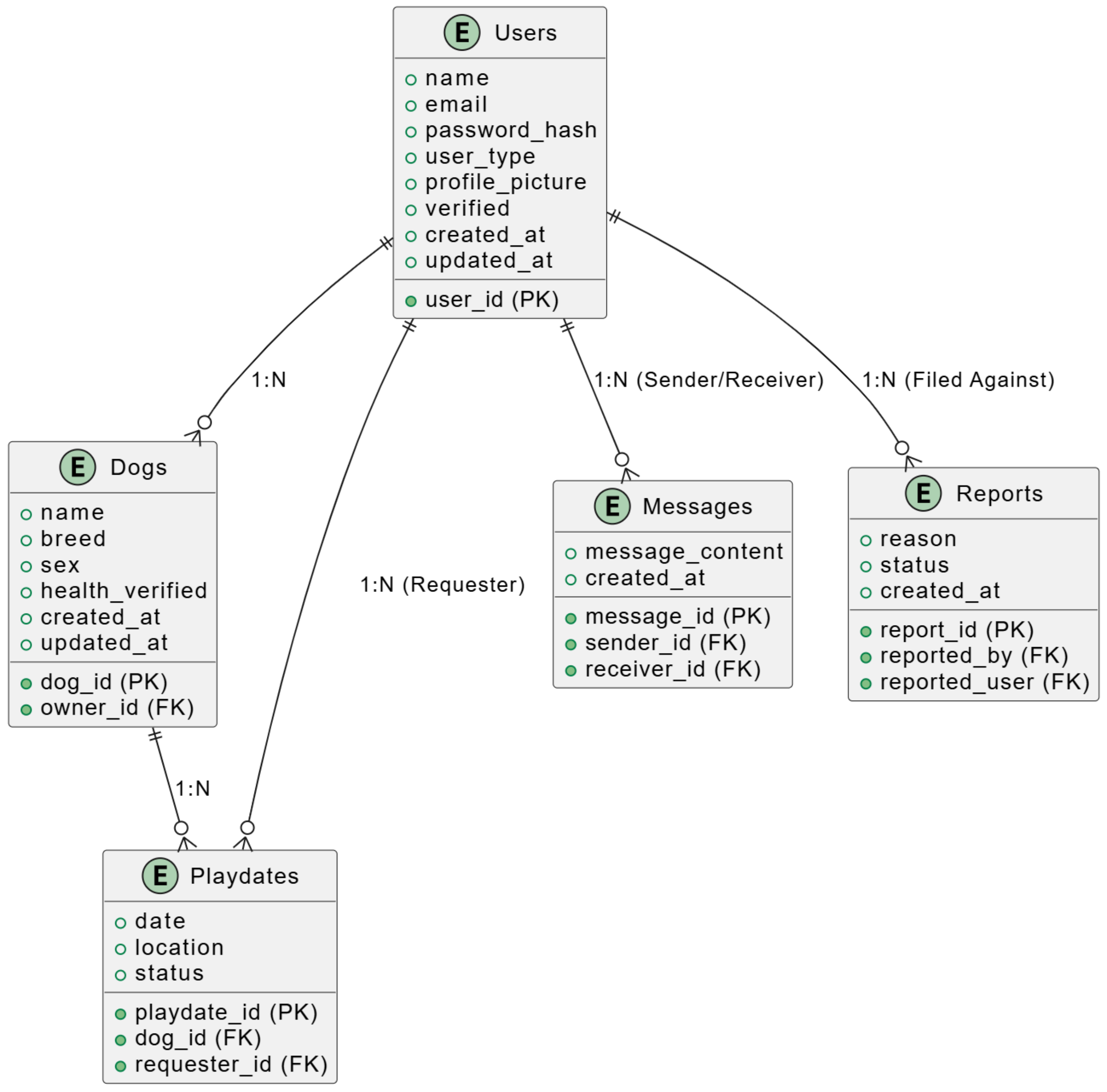
5. Actors and related access rights

6. Use Case Diagrams

7. System Architecture

8. Design (approx. 12 UML Diagrams!!!)

--next lab



Users
name
email
password_hash
user_type
profile_picture
verified
created_at
updated_at
user_id (PK)
1:N
1:N (Requester)
1:N (Sender/Receiver)
1:N (Filed Against)
Dogs
name
breed
sex
health_verified
created_at
updated_at
dog_id (PK)
owner_id (FK)
Messages
message_content
created_at
message_id (PK)
sender_id (FK)
receiver_id (FK)
Reports
reason
status
created_at
report_id (PK)
reported_by (FK)
reported_user (FK)
1:N
Playdates
date
location
status
playdate_id (PK)
dog_id (FK)
requester_id (FK)

9. Operation Mode (Interactivity presentation) + screen shots (UI design)

--next lab

10. Portability

--next lab

11. Competing software

--next lab