Software Requirements Specification

for

FetchMeHome

Version 1.6 approved

Prepared by Baskar Kayalvizhi Athithiya, Benjamin Yeoh Jun Jie, Zou Xuehao, Tin Jing Lun Javier

SCMA Team 3

20/04/2025

Table of Contents

Revision History

Name	Date	Reason For Changes	Version
Benjamin Yeoh	13/02/2025	Added Lab 1 Deliverables	1.0
Baskar Athithiya	26/02/2025	Added Lab 2 Deliverables	1.1
Zou Xue Hao	03/04/2025	Added Lab 3 Deliverables and Updated Diagrams	1.2
Benjamin Yeoh	16/04/2025	Added Lab 4 Deliverables and Updated Diagrams	1.3
Javier Tin	16/04/2025	Modified Section 5.3, Added 5.6,6.1,6.2, Add dialog diagram in appendix B	1.4
Javier Tin	19/04/2025	Updated Class Diagram	1.5
Benjamin Yeoh	20/04/2025	Added 3.1	1.6

1. Introduction

1.1 Purpose

This document specifies the software requirements for FetchMeHome, a web application developed to simplify the adoption and retrieval process of pets in Singapore. This Software Requirements Specification (SRS) document details the requirements for FetchMeHome, covering functional and non-functional specifications for the core components of the application. This SRS encompasses the entire system, describing both frontend and backend functionalities and integration with various external data APIs.

1.2 Document Conventions

This SRS follows the following documentation conventions to ensure clarity and consistency:

- Font Styles: ‘Arial’ for body text and ‘Times New Roman’ for headers.
- Date Format: Dates are to be written in the format of DD/MM/YYYY (e.g. 01/01/2024)
- Requirement Identification: Each requirement is uniquely identified for easy reference.

1.3 Intended Audience and Reading Suggestions

This Software Requirements Specification (SRS) document is intended for various stakeholders involved in the development and deployment of FetchMeHome:

- Developers: To gain a comprehensive understanding of the application's functionality and technical requirements for efficient implementation, developers should pay close attention to the System Features and External Interface Requirements to ensure seamless integration and proper implementation of features.

Readers are advised to start with the overview sections to get an overall understanding of the product's purpose and scope before diving into sections pertinent to their roles.

1.4 Product Scope

FetchMeHome is a digital platform designed to simplify and enhance the processes of pet adoption and pet recovery. It empowers pet owners to post detailed adoption listings with comprehensive pet profiles, while allowing interested users to submit structured applications directly through the platform. This streamlines the adoption journey by providing a centralized system for managing applications and communication — making the experience clearer, more organised, and stress-free for both parties.

In addition, **FetchMeHome** serves as a reliable channel for pet recovery. Pet owners can report lost pets through the app, and individuals who find a lost pet can submit photos to the owner for verification. This secure and integrated verification process accelerates reunions and reduces the friction typically experienced when coordinating through informal or scattered channels.

By addressing the current inefficiencies in pet adoption and lost pet recovery, **FetchMeHome** aligns with broader goals of improving community animal welfare, leveraging technology to solve real-world problems, and promoting responsible pet ownership through digital innovation.

1.5 References

React: https://reactjs.org/docs/getting-started.html
Express.js: https://expressjs.com/en/starter/installing.html
Node.js: https://nodejs.org/en/docs
MongoDB: https://www.mongodb.com/docs/

2. Overall Description

2.1 Product Perspective

2.1.1 System Overview Design

A system overview diagram (Figure 2.1.1) and a system architecture diagram (Figure 2.1.1) is provided to illustrate the interconnections and external interfaces.

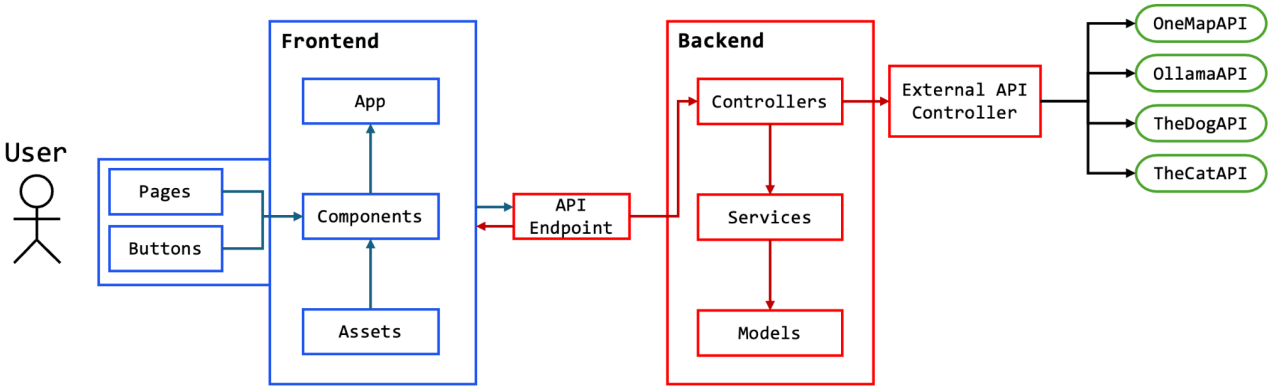


Figure 2.1.1 : System Overview Diagram

2.1.2 System Architecture Design

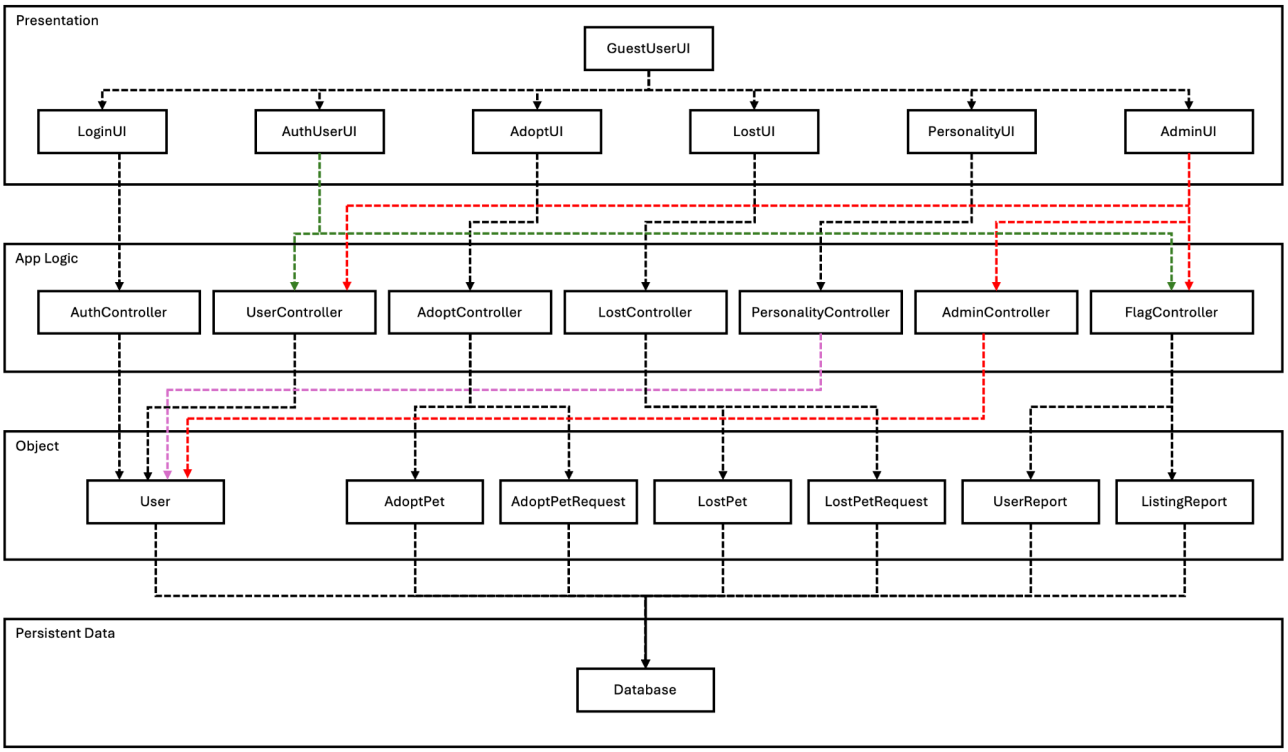


Figure 2.1.2: System Architecture Diagram

Presentation Layer

This layer is responsible for user interaction with the application. It consists of separate UIs for different user roles:

1. **GuestUserUI**
   - Serves as the primary interface for guest users with limited access to existing functions
   - Calls the appropriate UI – such as AdoptUI, LostUI, and LoginUI upon user's selection

2. **LoginUI**
   - Facilitates user authentication
   - Allows users to register for an account and log in using their email and password
   - Integrated with the **AuthController** for handling registration and login requests

3. **AuthUserUI**
   - Serves as the primary interface for authenticated (logged-in) users
   - Calls the appropriate UI – such as AdoptUI, LostUI, and PersonalityUI upon user's selection

4. **AdoptUI**
   - Displays published posts of pets that are put up for adoption

- **Authenticated Users** can indicate interest or report the listing, whereas **Guest Users** are only allowed to view the listings.
- Functionality is managed through the **AdoptPetController**

5. **LostUI**
   - Displays published posts of pets that are lost
   - **Authenticated Users** can submit image verification to the owner or report the listing, whereas **Guest Users** are only allowed to view the listings.
   - Functionality is managed through the **LostPetController**

6. **PersonalityUI**
   - Displays a personality questionnaire that only Authenticated Users can access
   - Functionality is managed through the **PersonalityController**

7. **AdminUI**
   - Accessible only to administrator accounts and consists of administrative functions
   - Admin operations rely on the **AdminController** and is integrated with **UserController**, **AdoptController**, **LostController**, and **FlagController**.

Application Logic Layer

This layer contains controller classes that process business logic and retrieve necessary entities from the Object Layer.

1. **AuthController**
   - Called by LoginUI to authenticate users
   - Contains the logic for user registration, login, password verification and token generation.

2. **UserController**
   - Called by LoginUI, AuthUserUI, and AdminUI.
   - Provides logic for user management

3. **AdoptController**
   - Called by AdoptUI, AuthUserUI, and AdminUI
   - Contains the logic for handling pet adoption

4. **LostController**
   - Called by LostUI, AuthUserUI, and AdminUI
   - Contains the logic for handling lost pet recovery

5. **PersonalityController**
   - Called by PersonalityUI
   - Contains the logic for managing calls to OllamaAPI, DogsAPI, and CatsAPI

6. **AdminController**
   - Called by AdminUI
   - Contains the logic to perform administrative tasks

7. **FlagController**

- Called by AuthUserUI and AdminUI
- Contains the logic for submitting and managing reports

App Object Layer

This layer contains entity classes representing core application data. These entities are stored in the database.

1. **User**
   Contains information about a user. Used by **AuthController**, **UserController**, **PersonalityController**, and **AdminController**

2. **AdoptPet**
   Represents a pet available for adoption. Used by **AdoptController**.

3. **AdoptPetRequest**
   Represents an application to adopt a specific pet. Used by **AdoptController**.

4. **LostPet**
   Represents a pet that has been reported lost by its owner. Used by **LostController**.

5. **LostPetRequest**
   Represents a submission made by a user who found a pet. Used by **LostController**.

6. **UserReport**
   Represents a report made against a user. Used by **AdminController** and **FlagController**.

7. **ListingReport**
   Represents a report made against a pet listing (either adoption or lost). Used by **AdminController** and **FlagController**.

Persistent Data Layer

This layer contains the MongoDB database storing all entities in a JSON-like format.

2.1.3 Class Diagram

The Class Diagrams (Figure 2.1.3 and 2.1.4) illustrates the interactions between various classes and highlights the essential functionalities that FetchMeHome must perform.

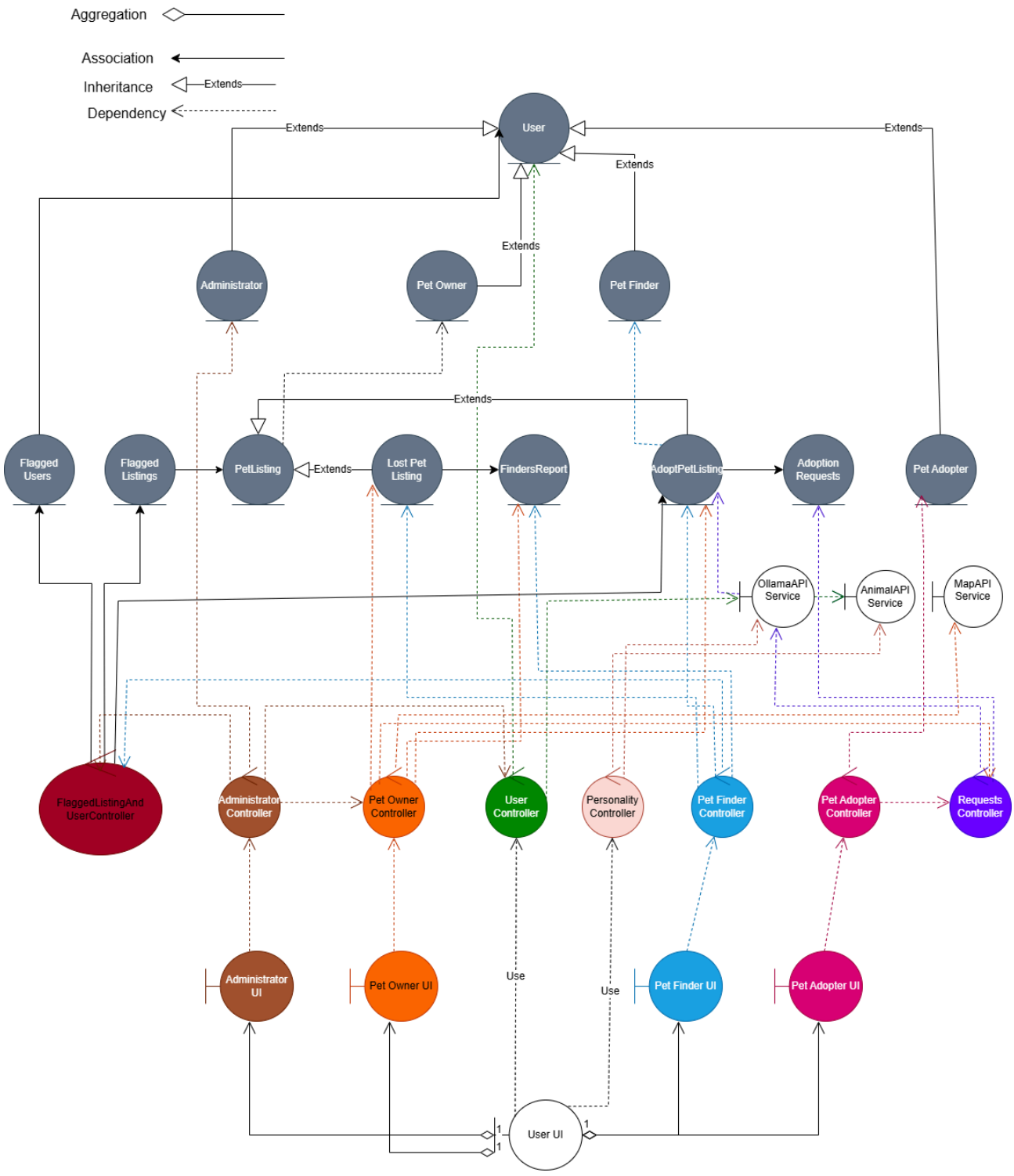


Figure 2.1.3 : Class Diagram

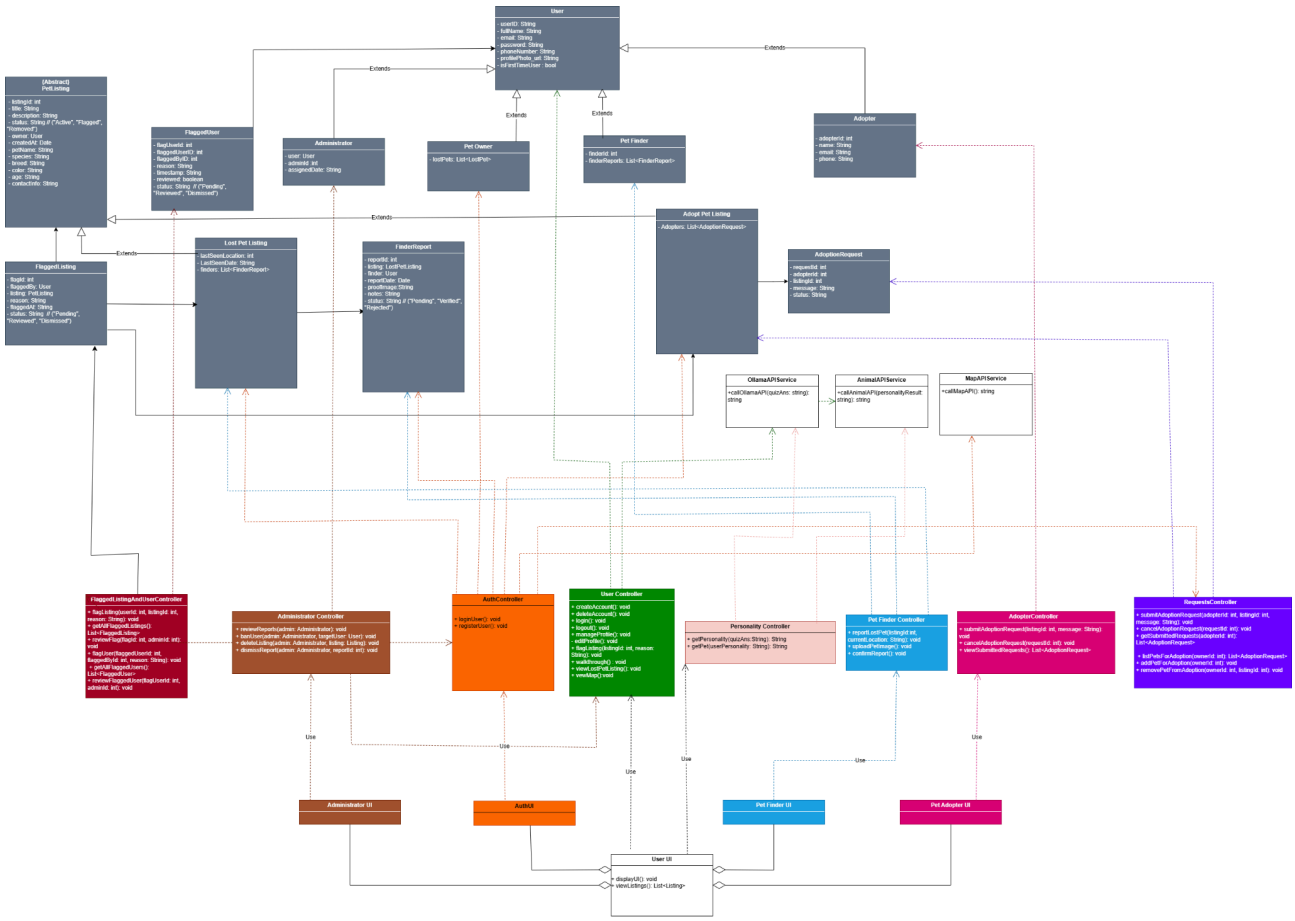


Figure 2.1.4 : Key Class Diagram

2.2 Product Functions

2.2.1 Use Case Diagram

The following use case diagram (Figure 2.2.1) depicts the key product functionalities that FetchMeHome includes. There are 4 primary users of FetchMeHome– Admin, Pet Owner, Pet Adopter, and Pet Finder. Each type of user interacts with FetchMeHome to use their role-specific functions

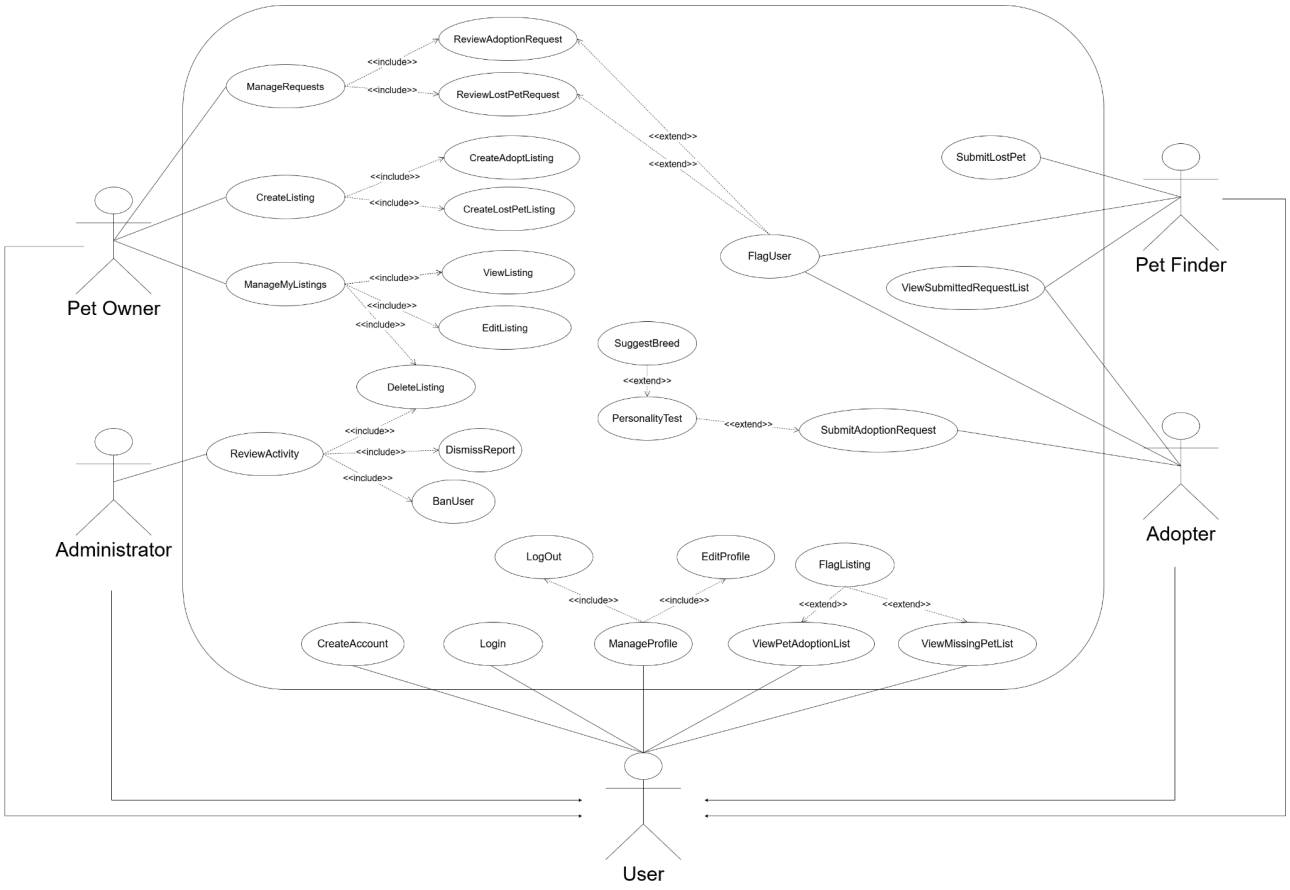


Figure 2.2.1 : Use Case Diagram

2.2.2 Key Product Functions

- For Users:
    - Login and Account Creation
    - Edit Profile
    - View Adopt and Lost Pet Listings
    - Report Listings
    - Pet Personality Quiz and Pet Recommendations
    - View Adoption and Lost Pet History

- For Owners:
  - Create, Edit and Delete Adopt and Lost Pet Listings
  - Report Users
  - Review Adopt and Lost Pet Requests
- For PetFinders:
  - Upload images of the Lost Pet for the Owner's verification
- For Adopters:
  - Request Adoption of Pets
- For Admin:
  - View Flagged Listings and Users
  - Delete Listings
  - Take Follow-Up actions:
    - Ban User
    - Delete Listing
    - Dismiss Report

2.2.3 Dialog Map

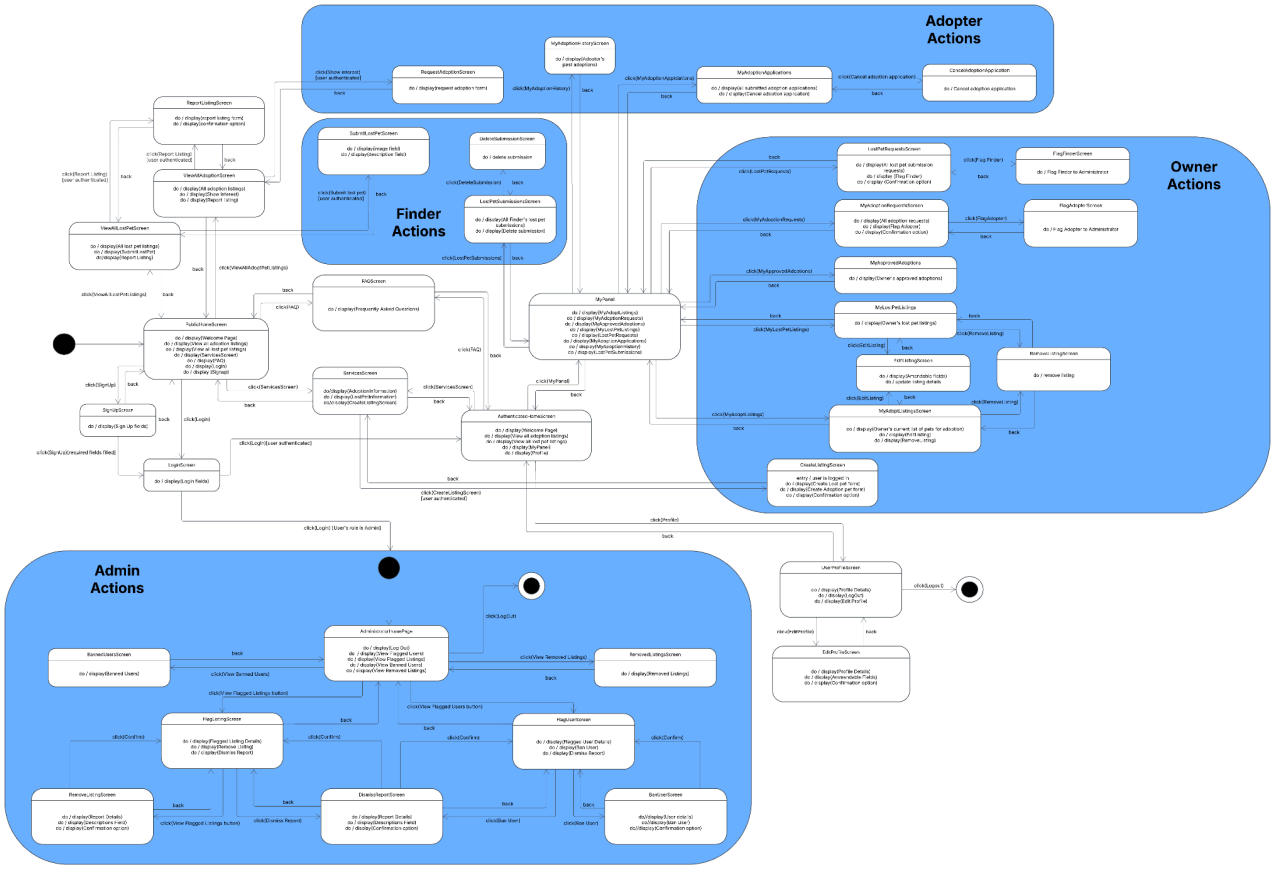


Figure 2.2.3 : Dialog Map

2.3 User Classes and Characteristics

Users of the platform are categorized into

- **Pet Owners**: Users who either have pets they would like to put up for adoption or pets that they have lost and need help locating. They need to be able to create postings with all the necessary details and approve or reject incoming requests/submissions for their pets.
- **Pet Finders**: Community individuals who will assist in finding lost pets. They should be able to upload photos and personal particulars for Pet Owners to verify and for contact purposes.
- **Pet Adopters**: Eager individuals who wish to adopt pets. They should be able to see the list of available pets and express interest in adopting them.
- **Administrator**: Staff who oversee platform operations. They should be able to carry out admin duties which includes regulating listings and users.

2.4 Operating Environment

FetchMeHome is a web application designed for laptops and desktops.

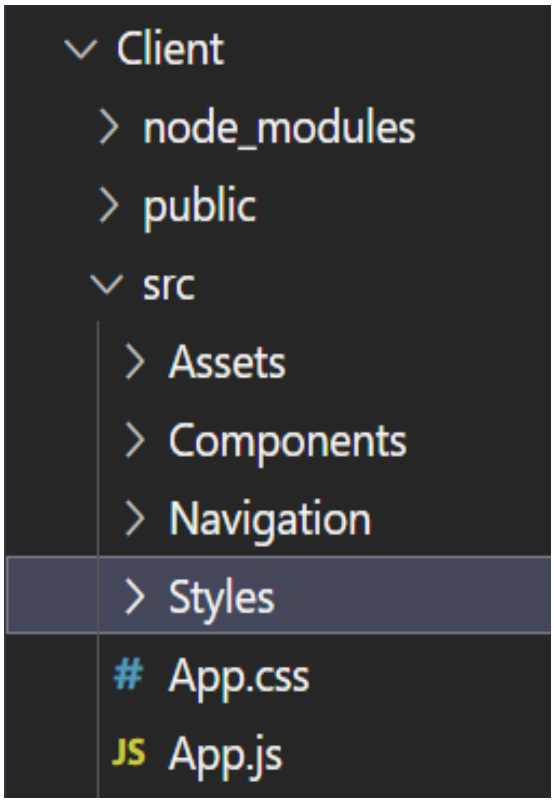
2.5 Design and Implementation Constraints

Overview of Tech Stack Used:

- Frontend**:**
  - React
- Backend:
  - Express.js
  - Node.js
- Database:
  - MongoDB

2.5.1 Frontend

A. Built with React.js



Client: (React) mainly consists of the frontend application code, which includes various parts of the app such as components, styling, images, and routing. These are organized into folders to keep the project modular, readable, and maintainable.

Assets is a folder which contains all of the images used in the app.

Components is a folder where we have our more commonly used items/reusable elements such as pages.

Navigation is a folder where we have our frontend routes configurations.This is where we manage how users navigate between different pages or components in the App.

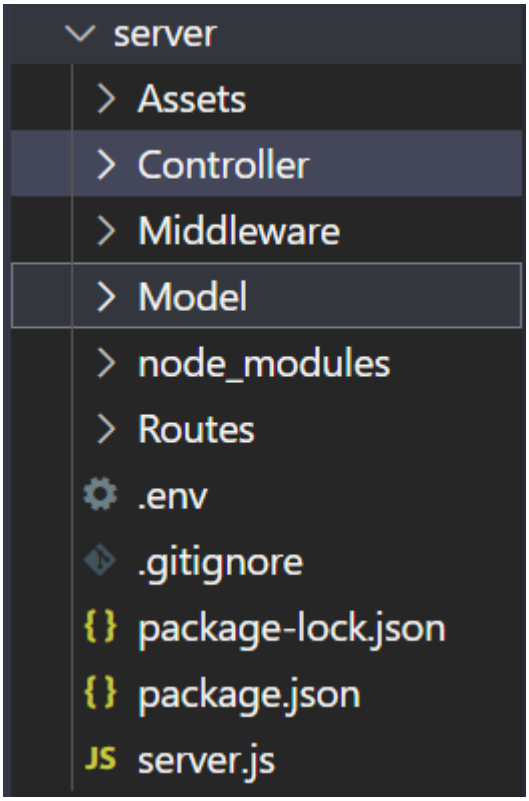
Styles is a folder that contains all the styling-related files for the application.

App.js is the entry point for the Client.

2.5.2 Backend

A. Server

a. Built with Express.js and Node.js
b. Database used: MongoDB



Assets contain the images uploaded by Users.

Controller contains the logic for handling incoming requests and sending responses.

Middleware is used for tasks like authentication, logging, validation, or error handling.

Model is used for defining schemas (MongoDB) and setting up methods for querying and updating data.

Routes are responsible for defining the application's endpoints and connecting each endpoint to the appropriate controller function. They map HTTP methods (GET, POST, etc.) to controller actions.

.env contains all our live API_KEYS used.

App.js is the entry point for the Client.

server.js is the entry point for Server.

2.5.3 Database

The database for FetchMeHome is built using **MongoDB**, which efficiently handles non-relational data.


```
_id: ObjectId('67fcde226d3eef966e182ef1')
name : "testadopt2"
age : "2"
area : "21 CLUB STREET, NIL, Postal: 069410"
justification : "tt"
email : "dei@gmail.com"
phone : "111"
type : "Dog"
filename : "1744625186330-436871751.png"
status : "Pending"
postedBy : ObjectId('67f782ed3af6182f11b3a3f6')
createdAt : 2025-04-14T10:06:26.337+00:00
updatedAt : 2025-04-14T10:06:26.337+00:00
__v : 0
```

2.6 User Documentation

FetchMeHome includes a comprehensive set of documentation to guide developers. The documentation ensures a clear understanding of the setup, usage, and integration of the different components of the application.

- **README File**
  The main repository features a README file that provides an overview of FetchMeHome, including setup instructions, architecture and repository structure.
- **INDEX File**
  The main repository features an INDEX file that provides links to all lab deliverables and the code.

2.7 Assumptions and Dependencies

FetchMeHome is dependent on the following Government/External APIs:

- Live Map of Singapore (OneMap API):
  - To display a live map of Singapore under Post Adopt and Lost Pets
  - https://www.onemap.gov.sg/apidocs/

- Dog API:
  - To display a list of dogs where its Personality matches the User
  - https://www.thedogapi.com/

- Cat API
  - To display a list of cats where its Personality matches the User
  - https://thecatapi.com/

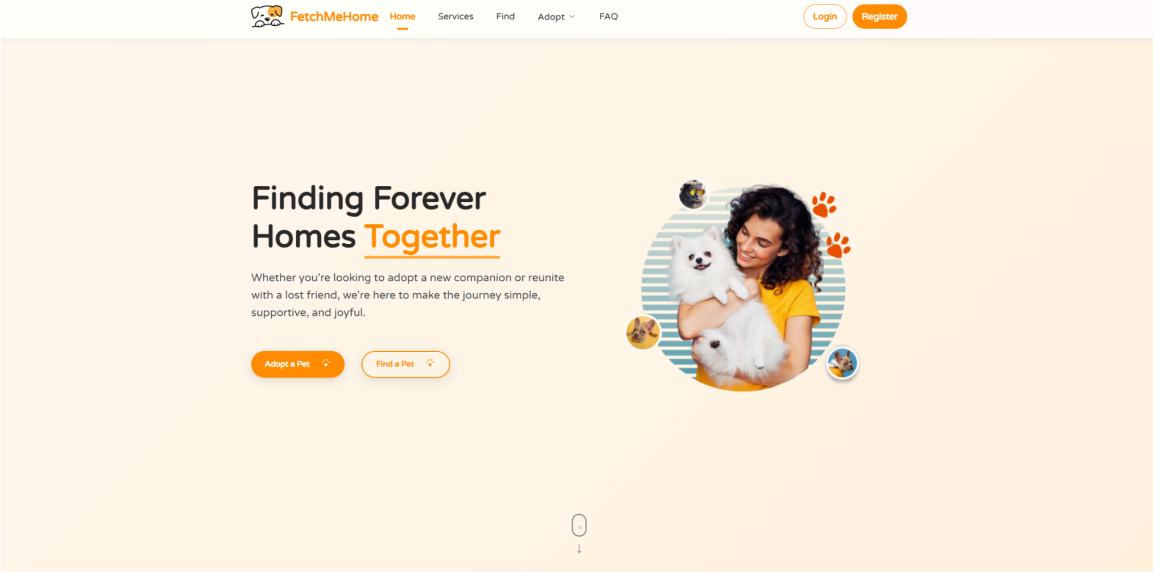
3. External Interface Requirements

3.1 User Interfaces

3.1.1 Functional Requirement 1: User

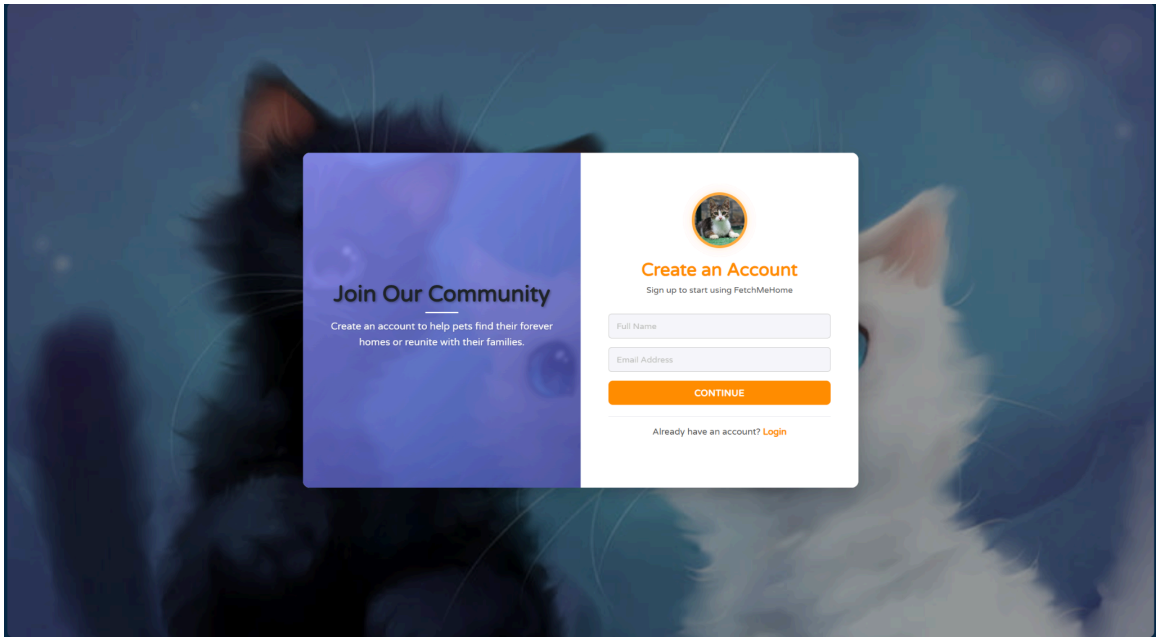
3.1.1.1 Landing Page / Home Screen (Guest View)

- Description: The initial page view for non-authenticated users, providing an overview and navigation to login, register, and browse available listings.



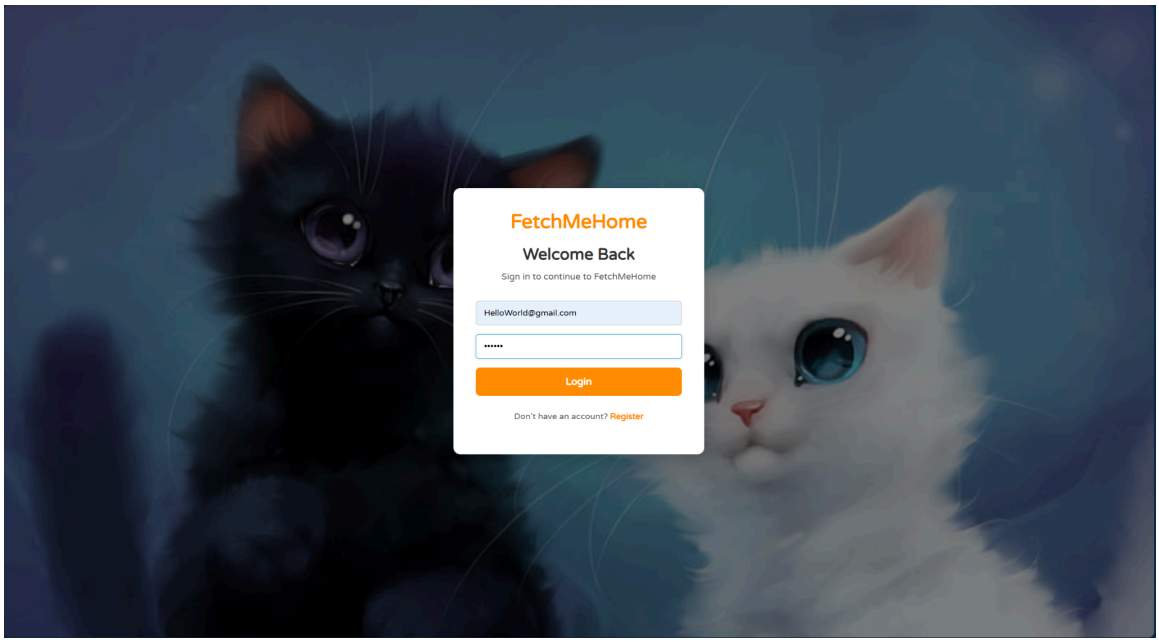
3.1.1.2 Create Account (Use Case #1-1)

- Description: Allows first-time users to create an account.



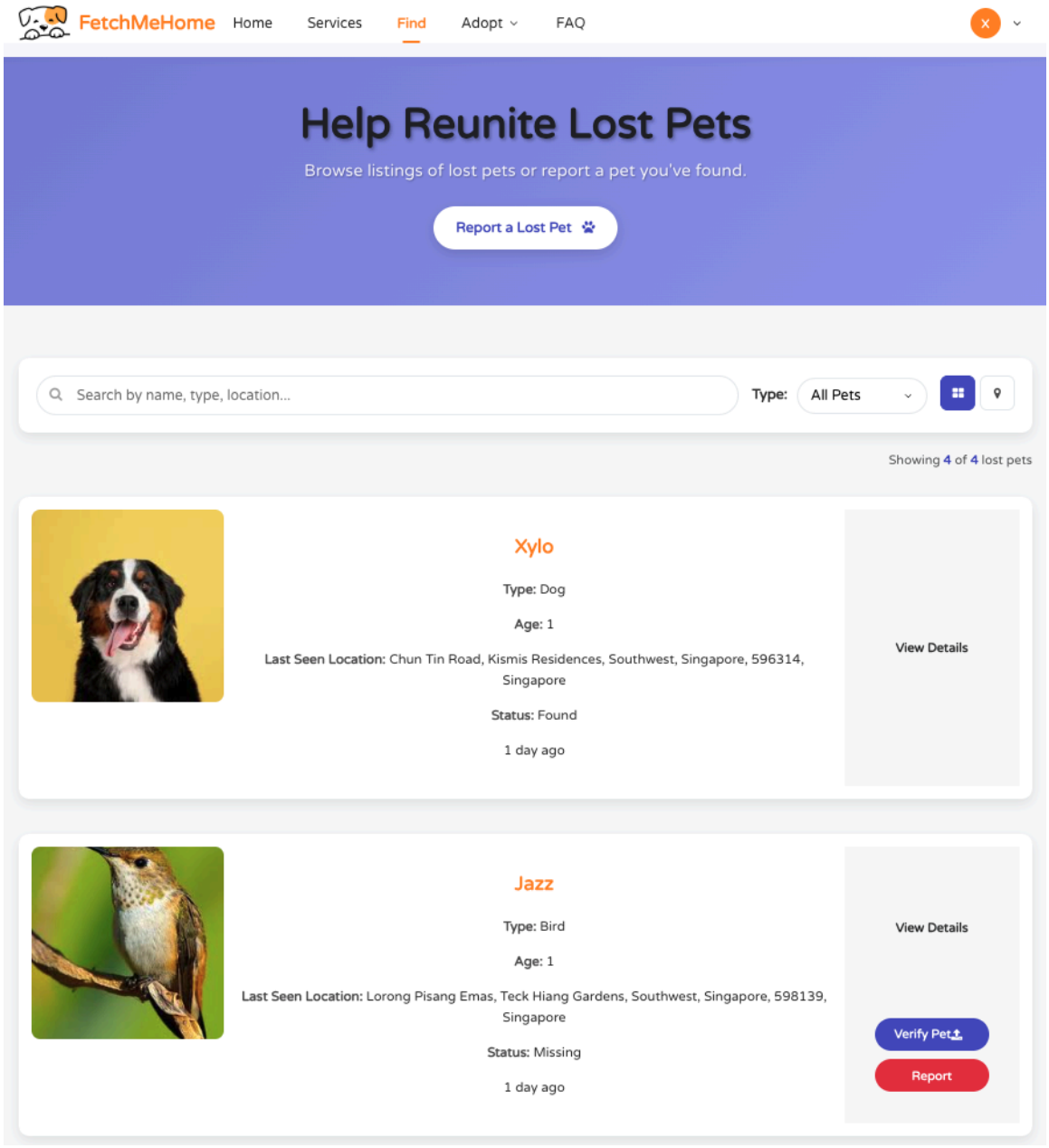
3.1.1.3 Login (Use Case #1-2)

- Description: Allows the user to login using their email and password.



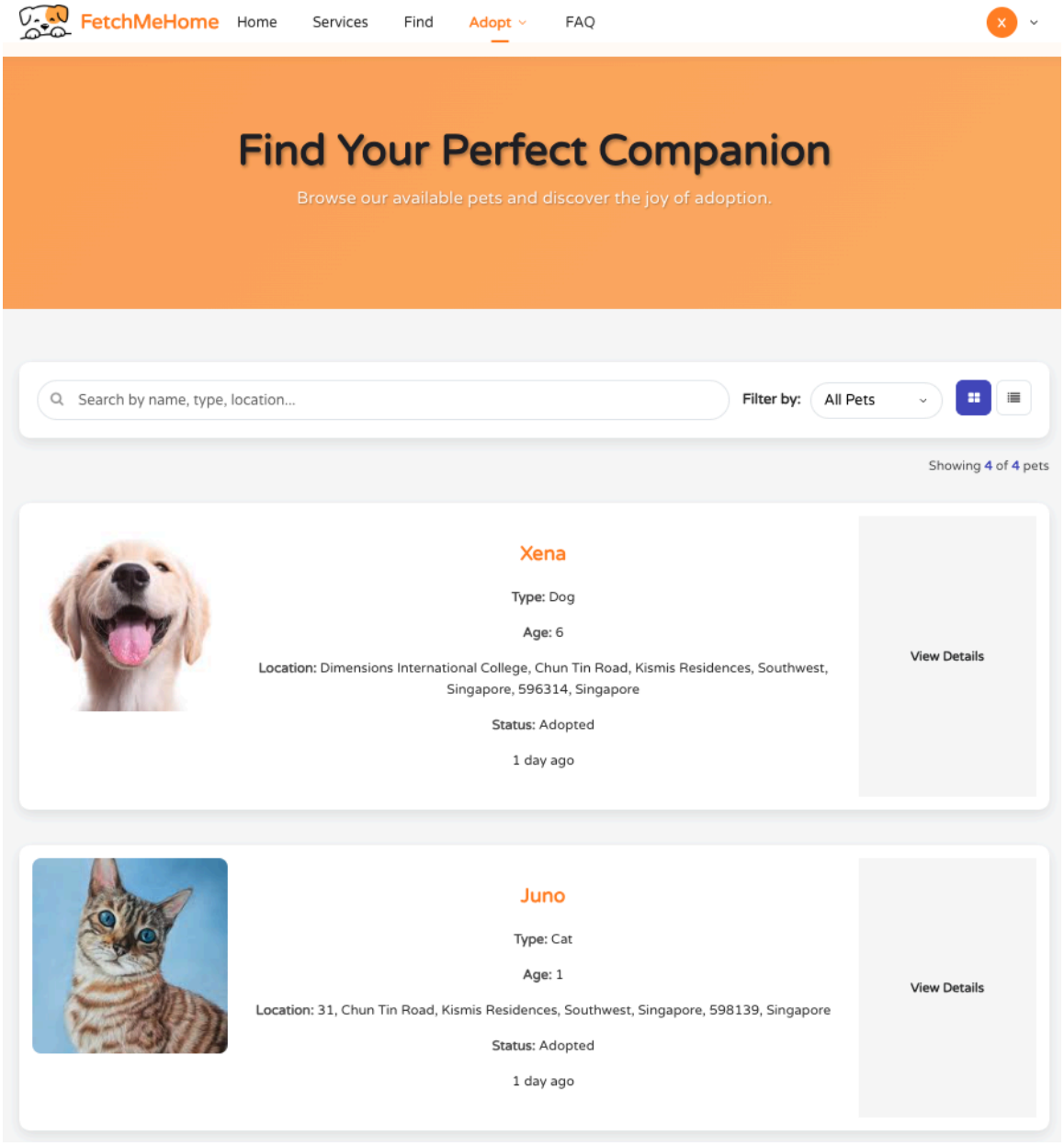
3.1.1.4 View Missing Pet List (Use Case #1-3)

- Description: Users can view all available lost pet listings.



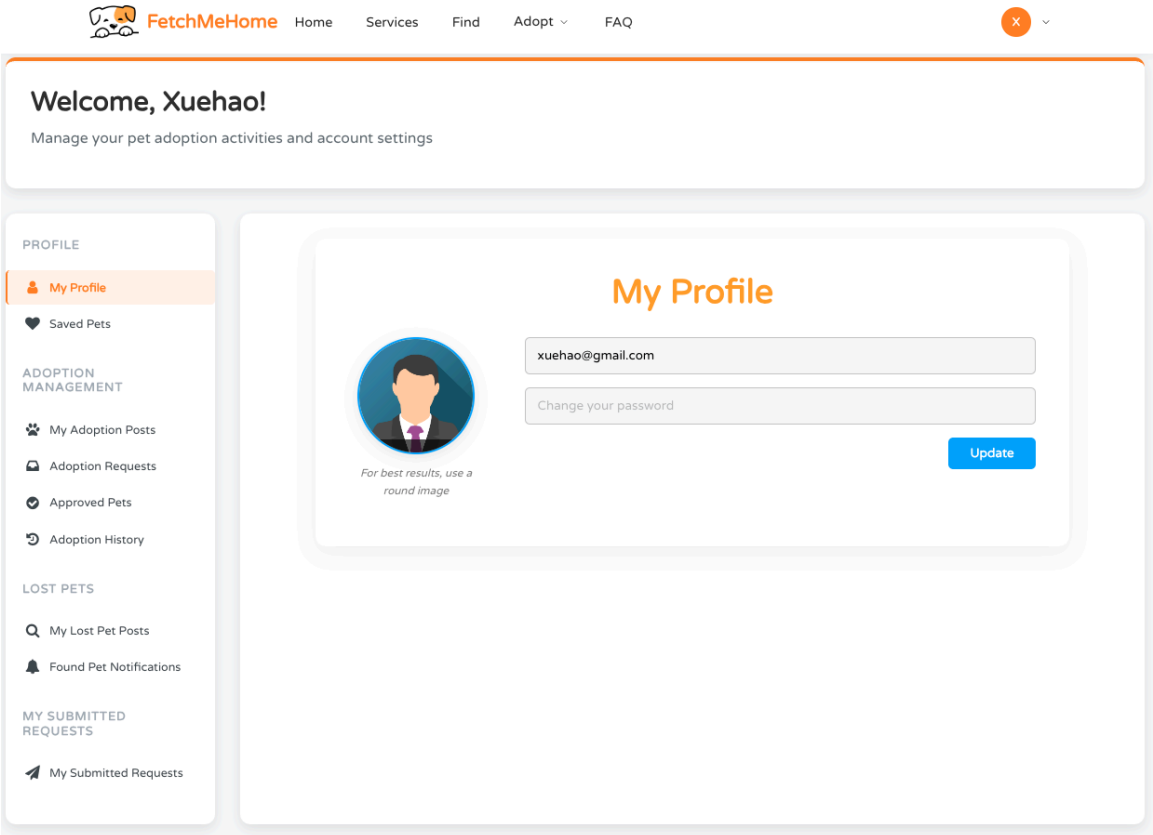
3.1.1.5 View Pet Adoption List (Use Case #1-4)

- Description: Users can view all available adoption listings.



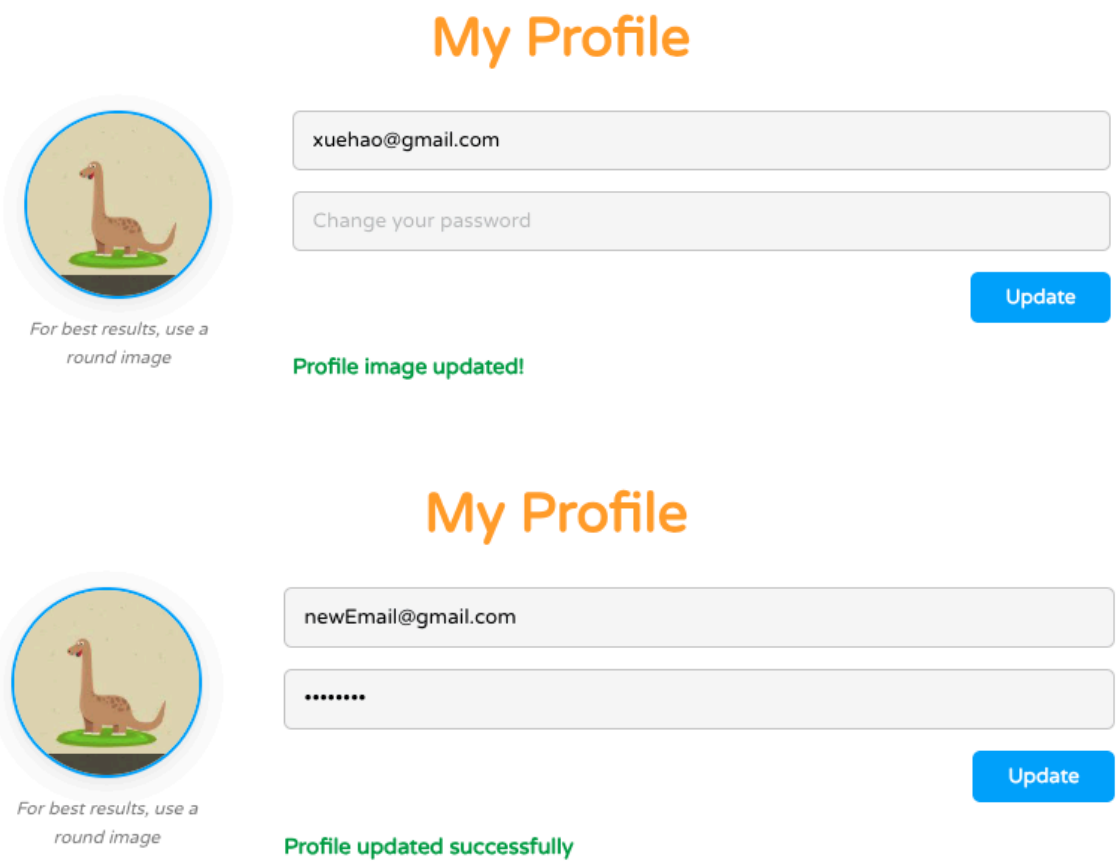
3.1.1.6 Manage Profile (Use Case #1-5)

- Description: Authenticated users can manage/edit the status of their profile.



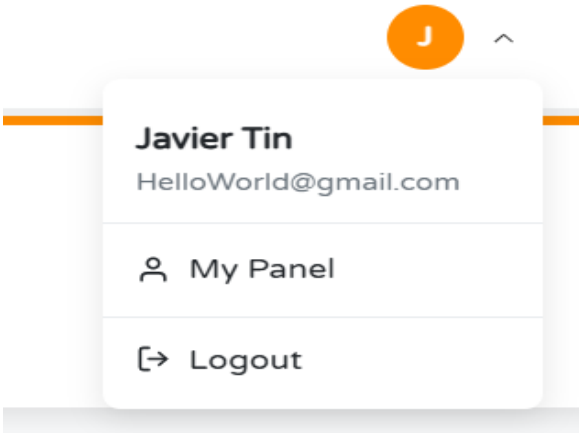
3.1.1.7 Edit Profile (Use Case #1-6)

- Description: Authenticated users can edit the details of their profile such as email address, password and add/modify their profile photo.



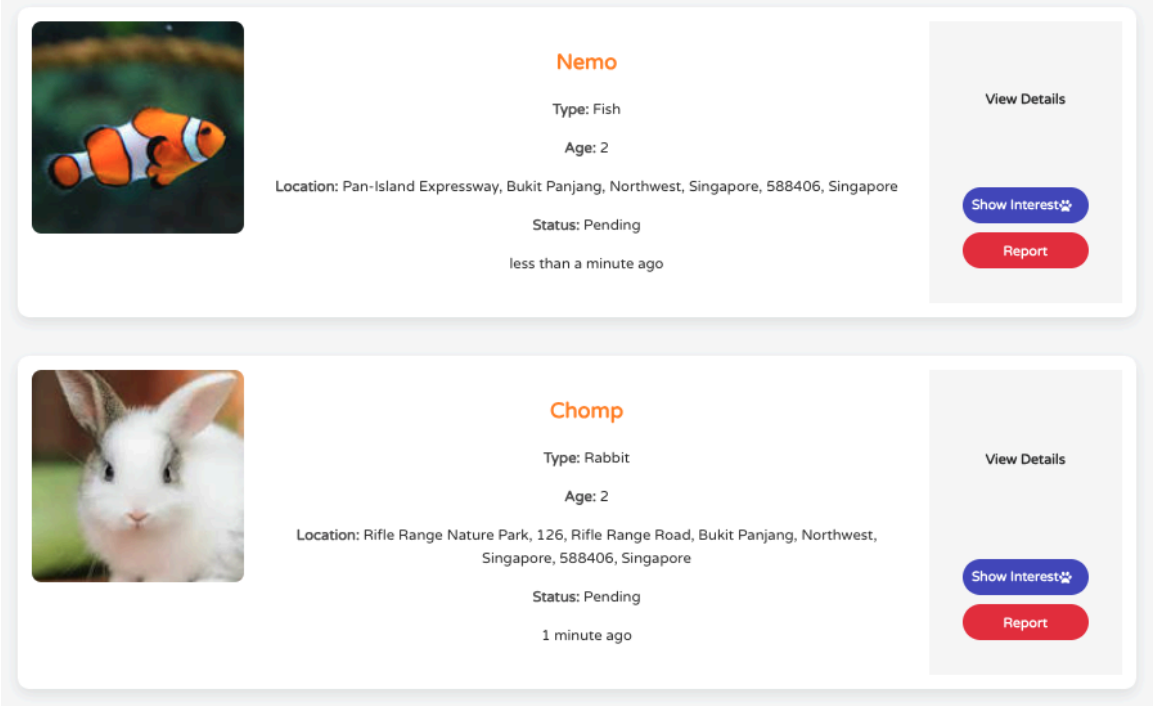
3.1.1.8 Log Out (Use Case #1-7)

- Description: Authenticated users can log out of their FetchMeHome account.

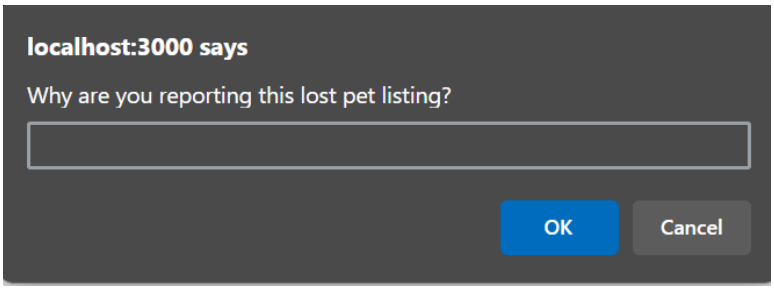


3.1.1.9 Flag Listing (Use Case #1-8)

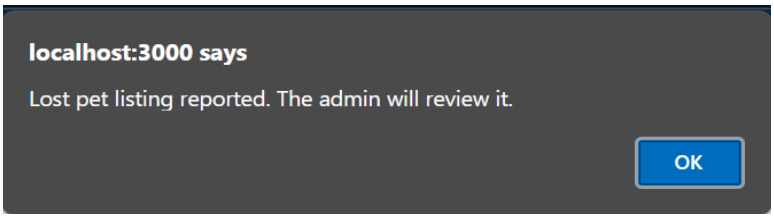
- Description: Authenticated users can flag listings deemed inappropriate.



- Description: Users are required to enter a short justification of the report.

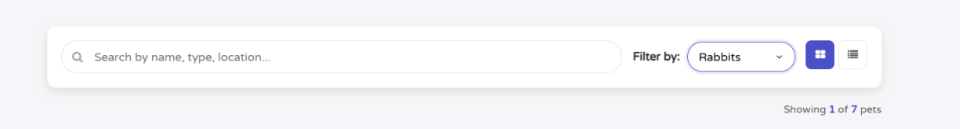


- Description: Report is sent to the admin for review upon submitting.

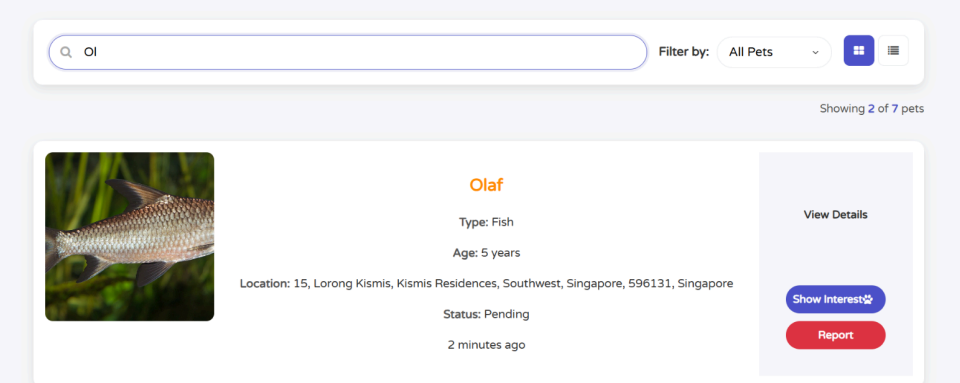


3.1.1.10 Search/Filter Listings (Implied Functionality)

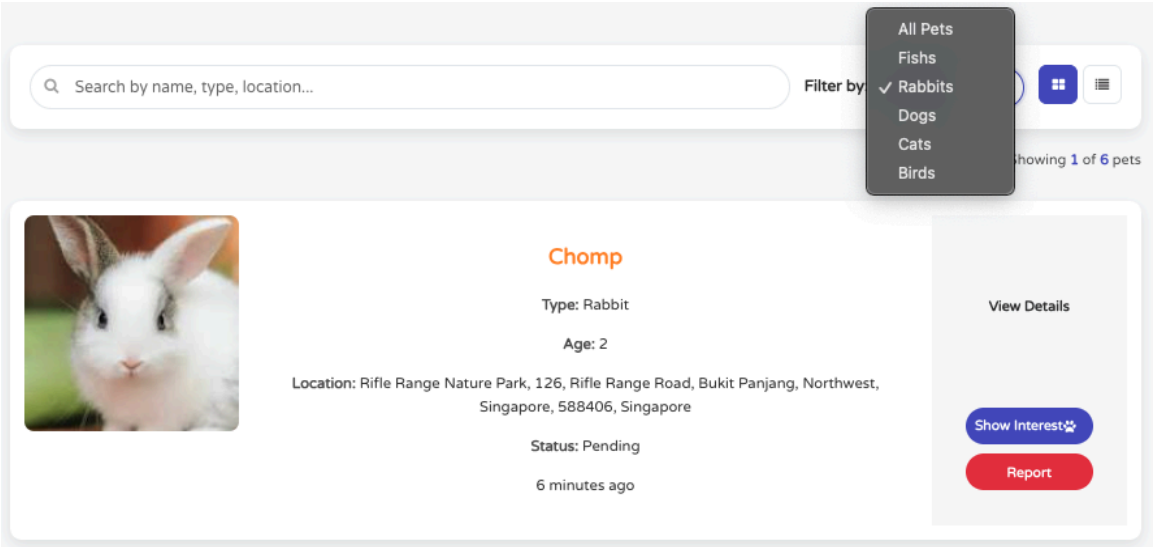
- Description: Users can filter by searching adoption and lost pet listings based on criterias such as pet type, location, etc.



- Description: For example, searching for Olaf in the filter bar would only show the listing associated with Olaf.

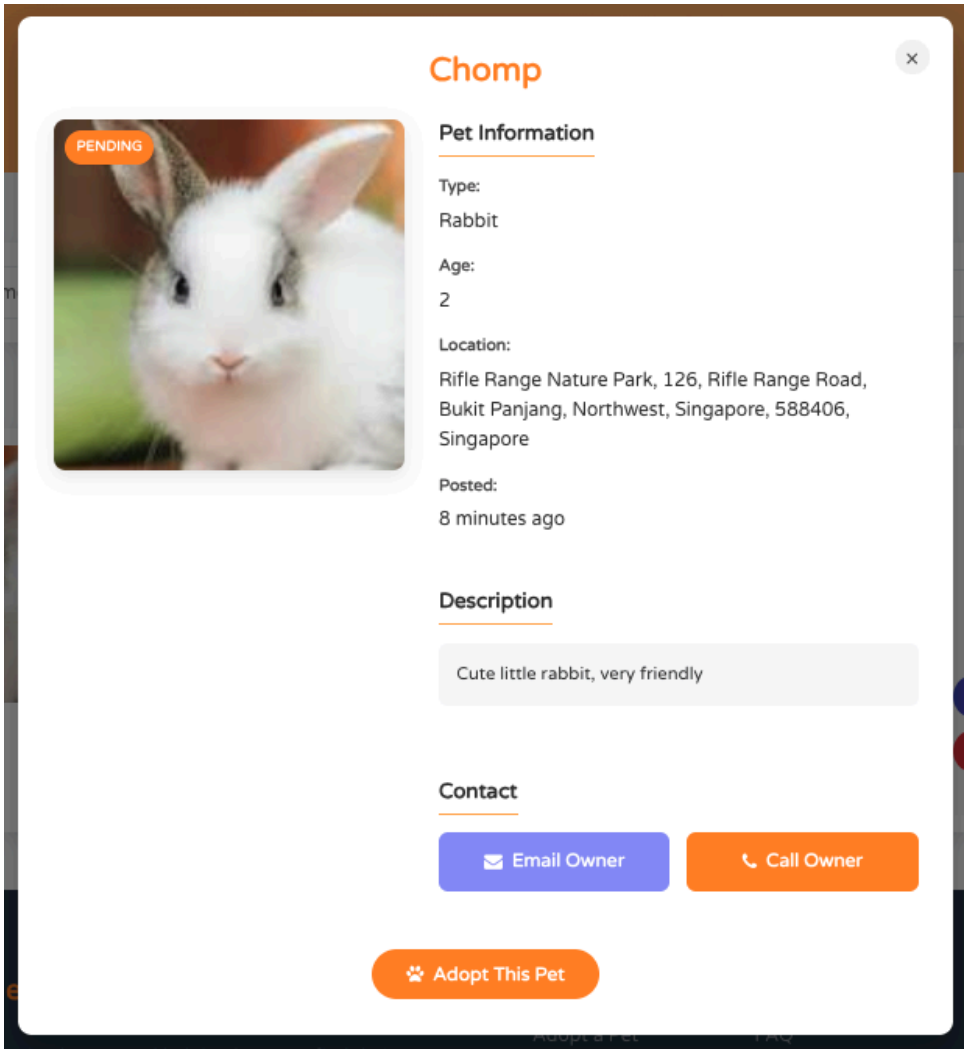


- Description: Users can also use the dropdown menu to filter posts. (Eg. Rabbits)



3.1.1.11 View Individual Listing Details (Implied Functionality)

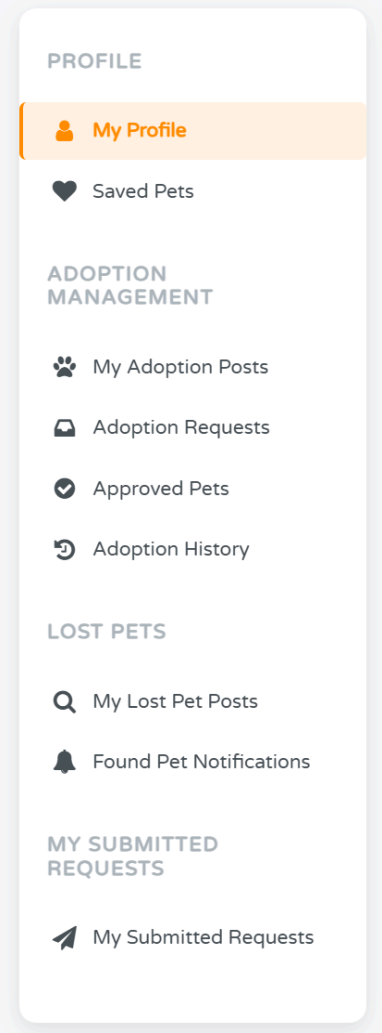
- Description: Users can click on an individual pet listing (lost or adoption) to view more details.



3.1.2 Functional Requirement 2: Owner Listing & Request Management

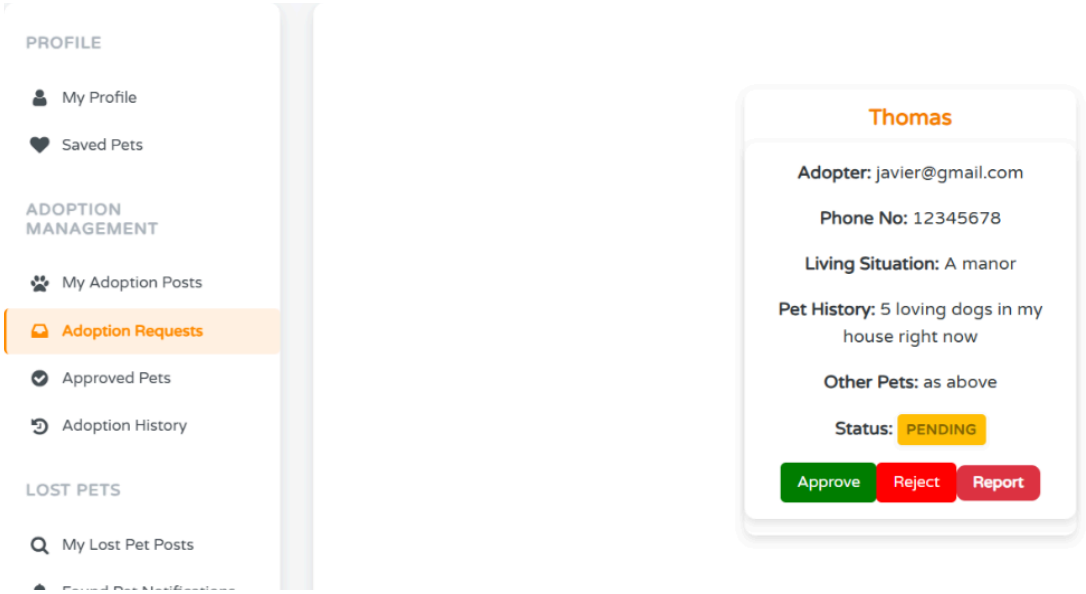
3.1.2.1 Manage Requests (Use Case #2-1)

- Description: Allows owners to manage requests made by pet finders and/or adopters.



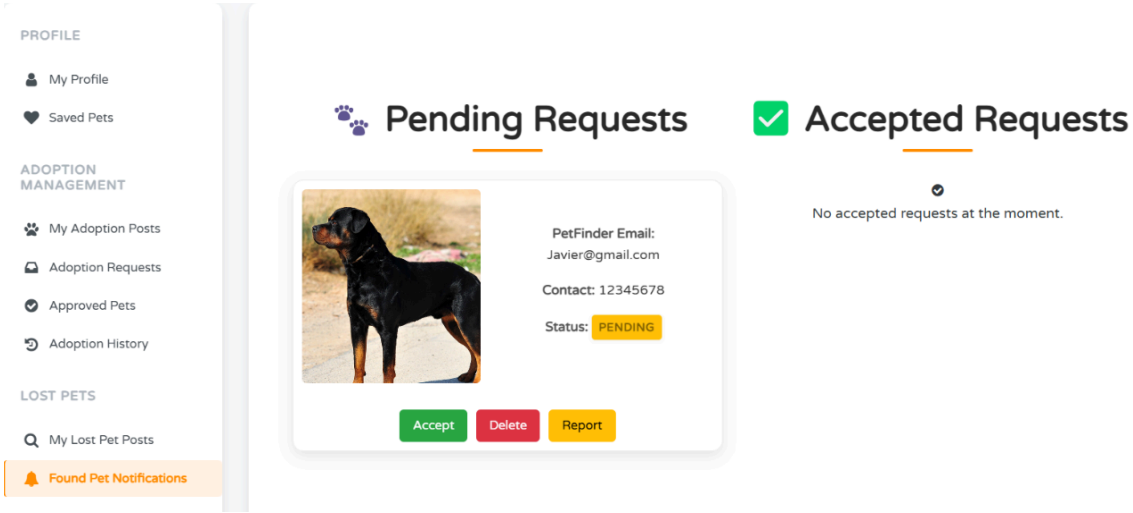
3.1.2.2 Review Adoption Request (Use Case #2-2)

- Description: Allows owners to view all available adoption requests made by adopters for their pets.



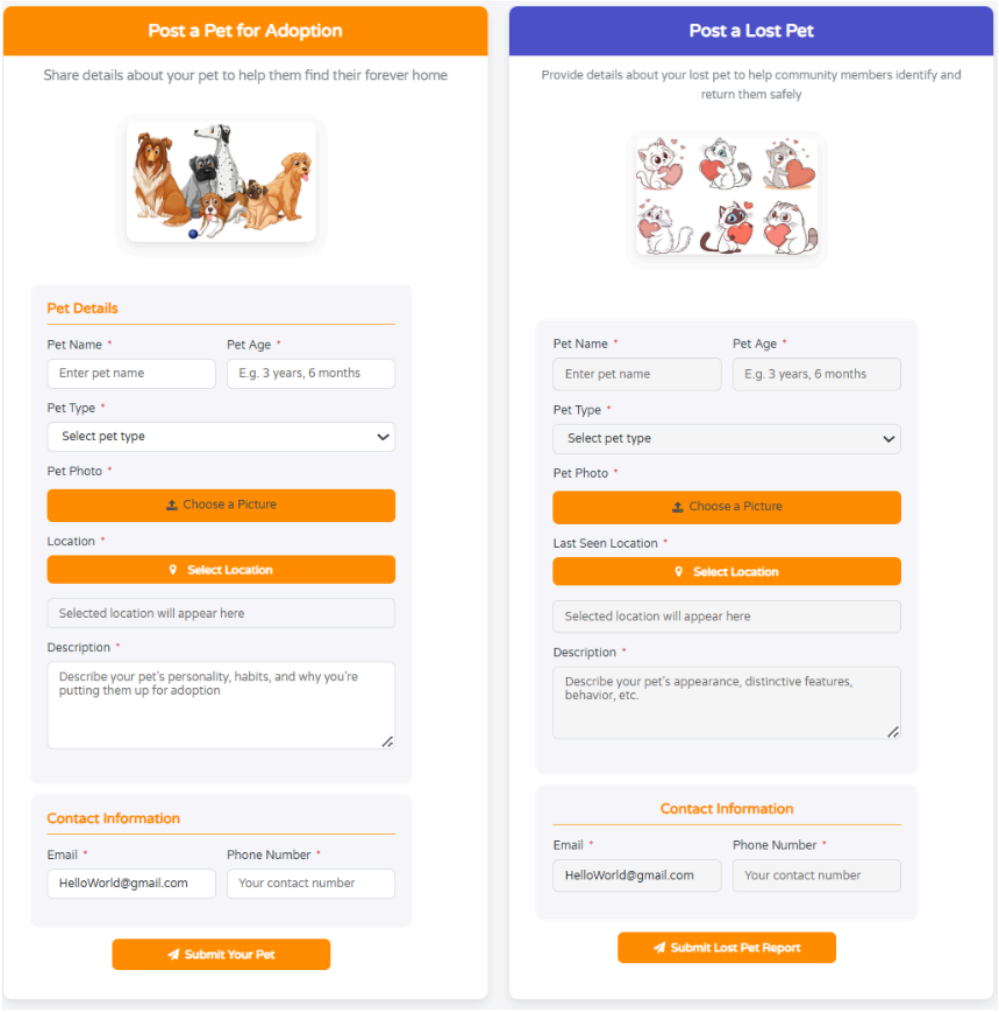
3.1.2.3 Review Lost Pet Request (Use Case #2-3)

- Description: Allows owners to view all lost pet reports made by pet finders for their lost pets.



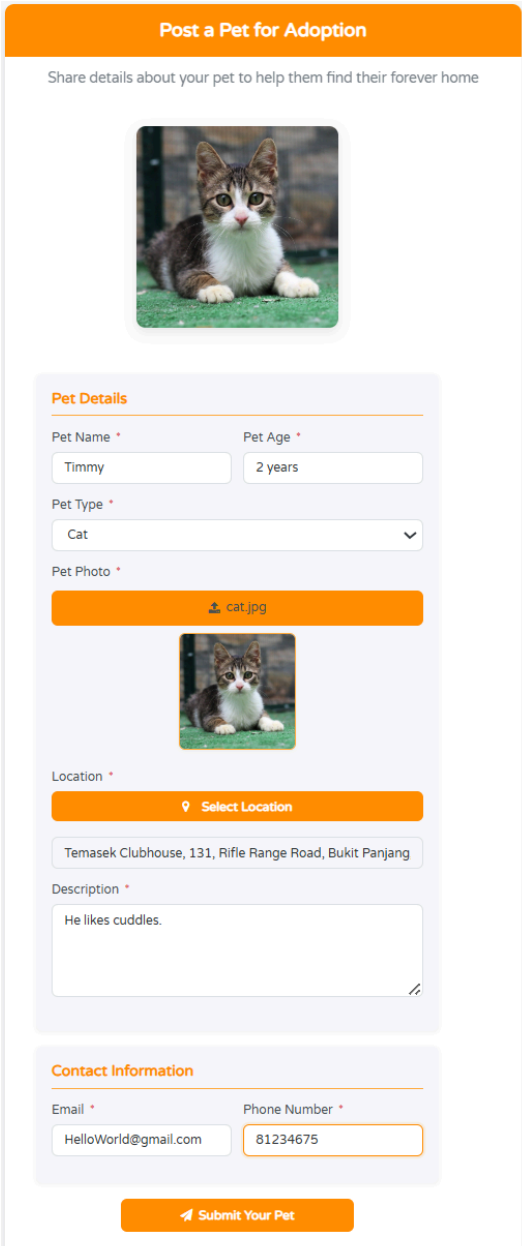
3.1.2.4 Create Listing (Use Case #2-4)

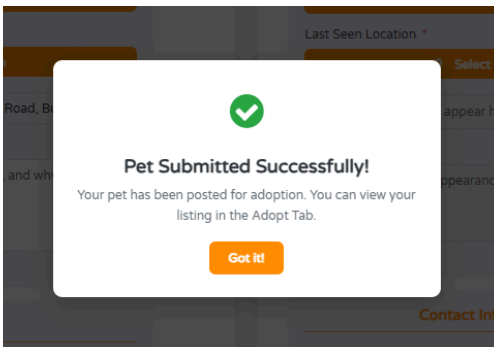
- Description: Allows owners to create listings for their missing pets and/or pets up for adoption.



3.1.2.5 Create Adopt Listing (Use Case #2-5)

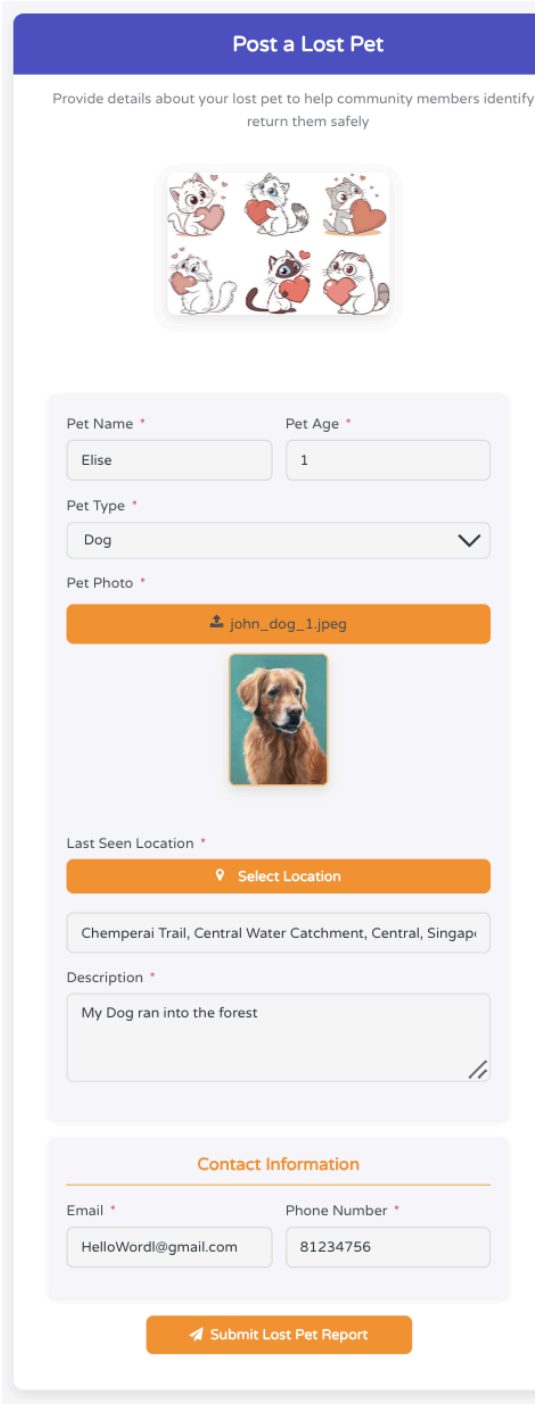
- Description: Allows owners to create a listing for pets up for adoption.

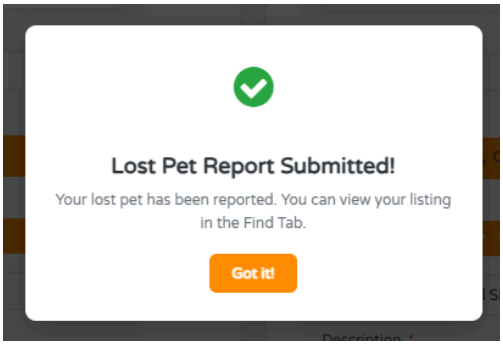




3.1.2.6 Create Lost Pet Listing (Use Case #2-6)

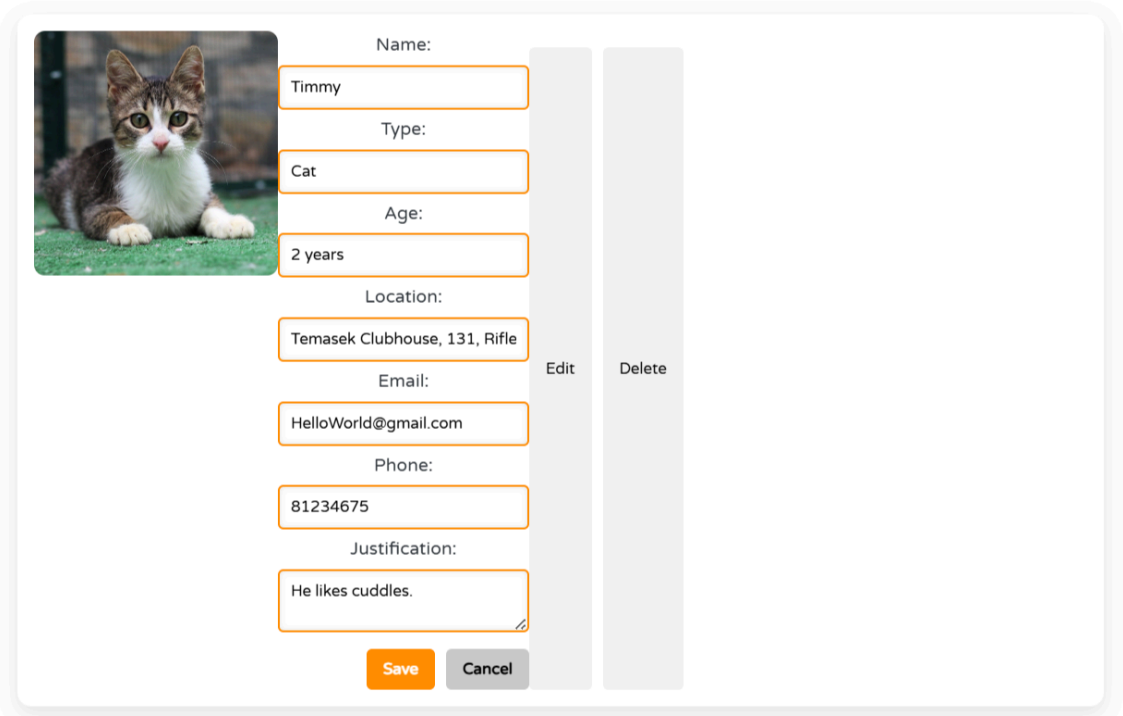
- Description: Allows owners to create a listing for their missing pets.





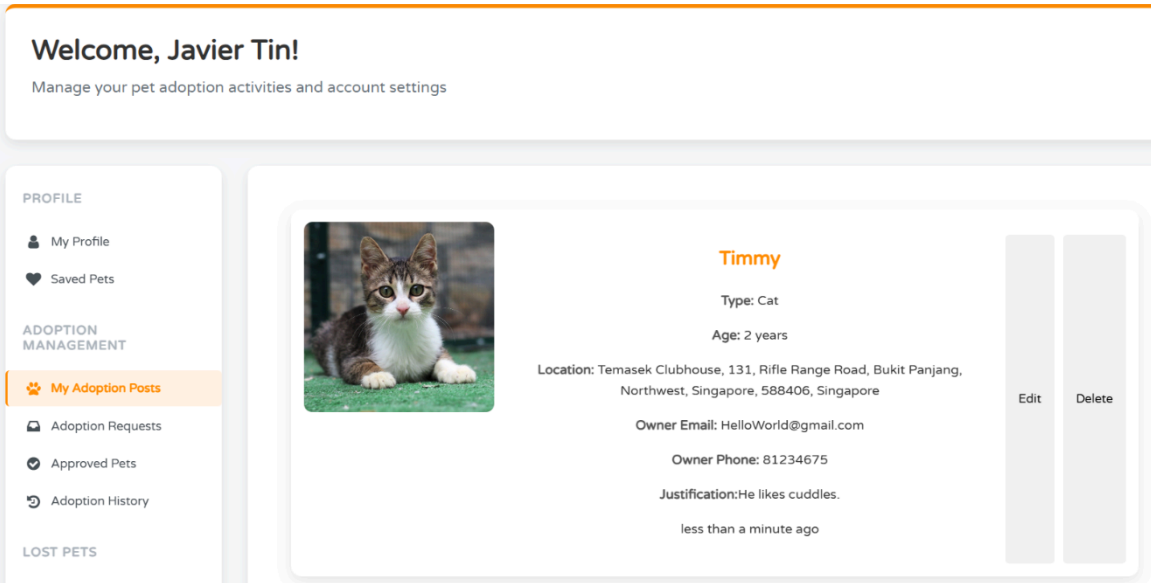
3.1.2.7 Manage My Listings (Use Case #2-7)

- Description: Allows owners to manage all their available listings (view/edit/delete).



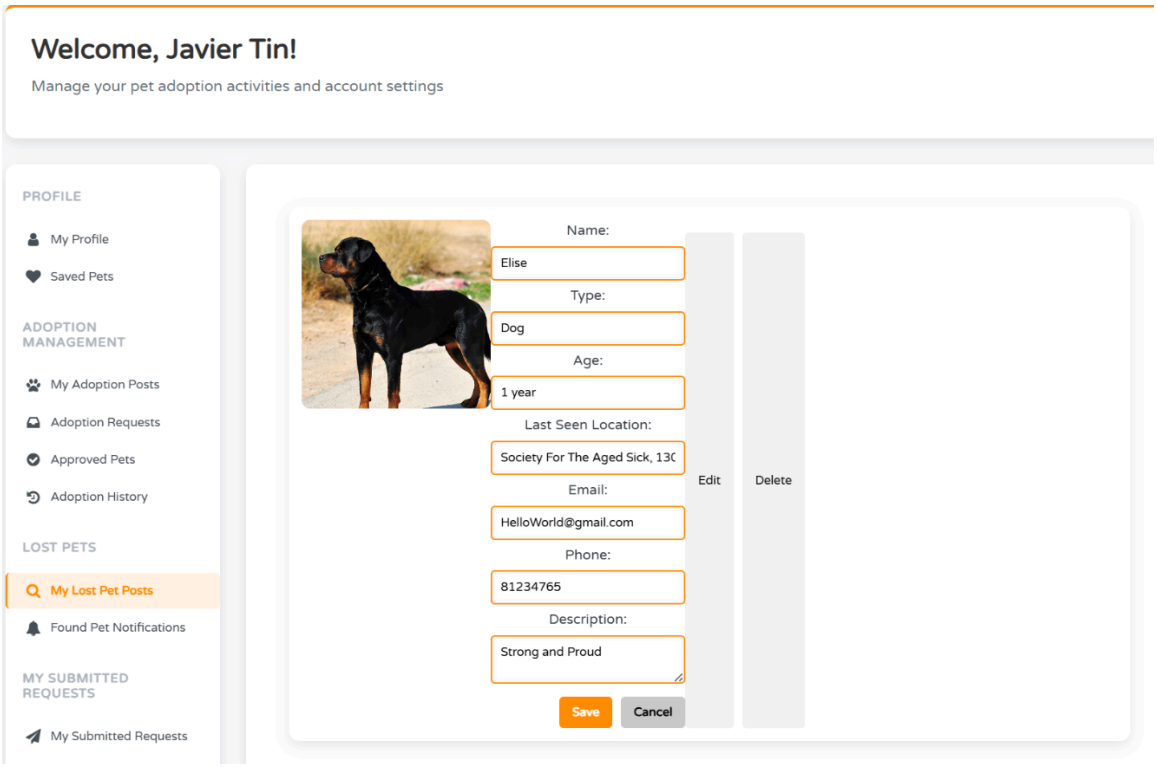
3.1.2.8 View Listing (Owner View - Use Case #2-8)

- Description: Allows owners to view all their available listings.



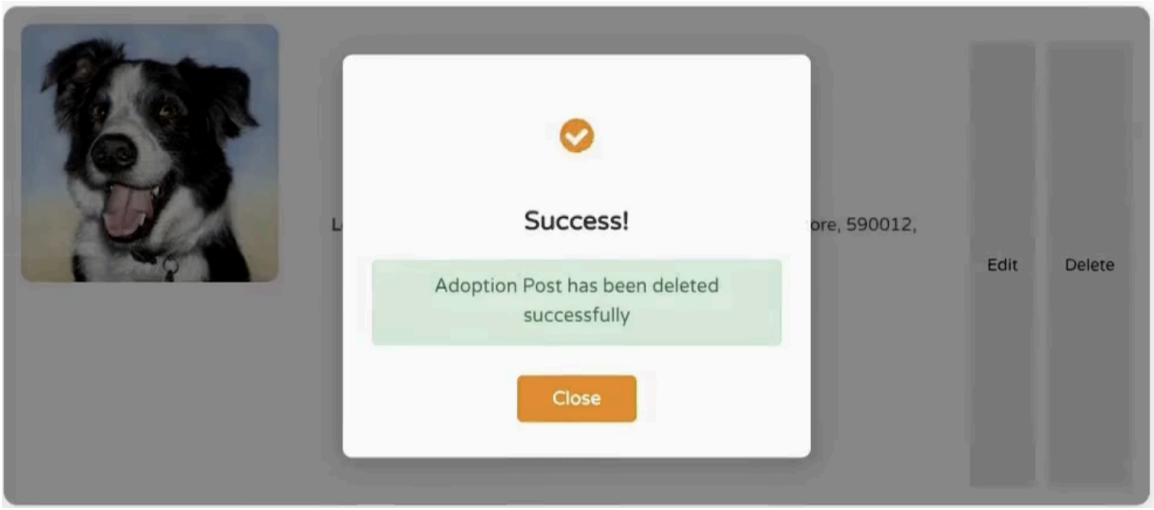
3.1.2.9 Edit Listing (Use Case #2-9)

- Description: Allows owners to edit listings for their missing pets and/or pets up for adoption.



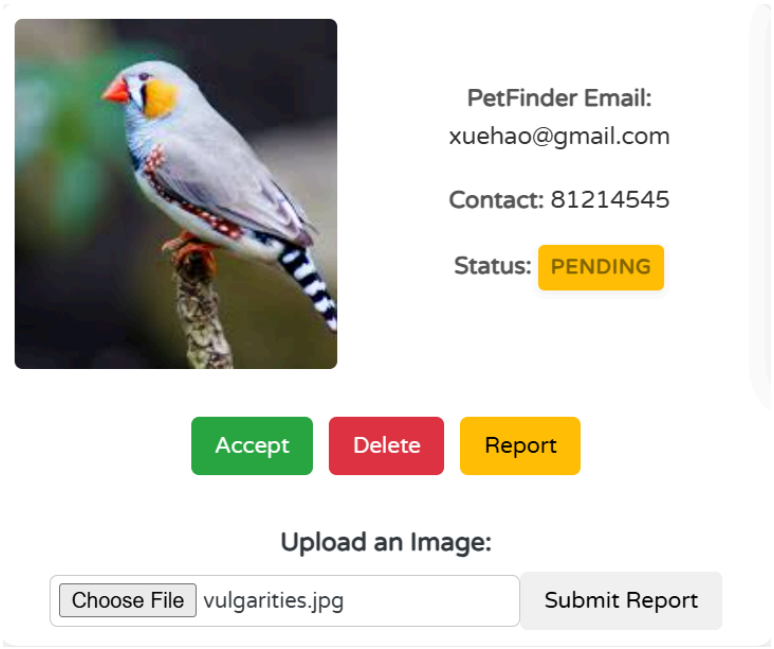
3.1.2.10 Delete Listing (Use Case #2-10)

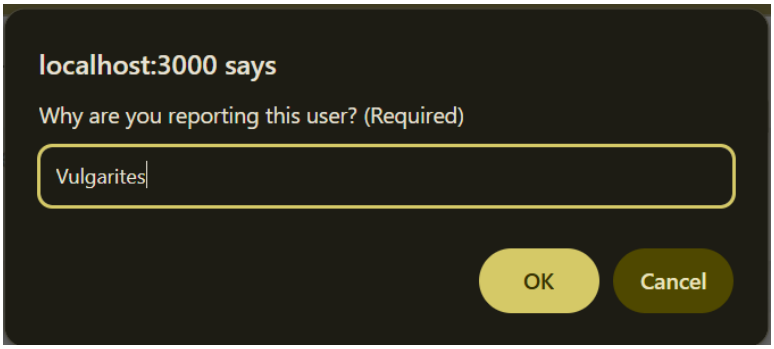
- Description: Owners are able to remove their listings.



3.1.2.11 Flag User (Use Case #2-11)

- Description: Allows owners to flag users deemed inappropriate.

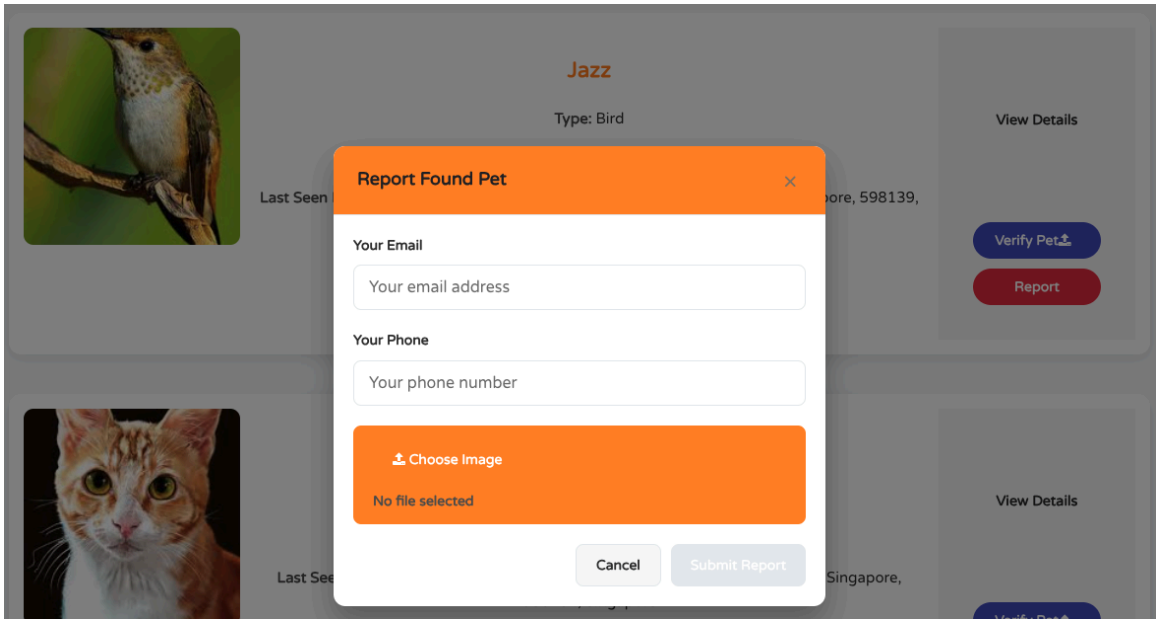




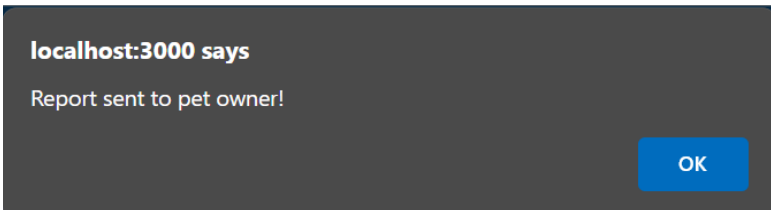
3.1.3 Functional Requirement 3: PetFinder Actions

3.1.3.1 Submit Lost Pet (Report Found Pet) (Use Case #3-1)

- Description: Pet finders can report missing pets in their vicinity by submitting information or a photo against an existing lost pet listing.



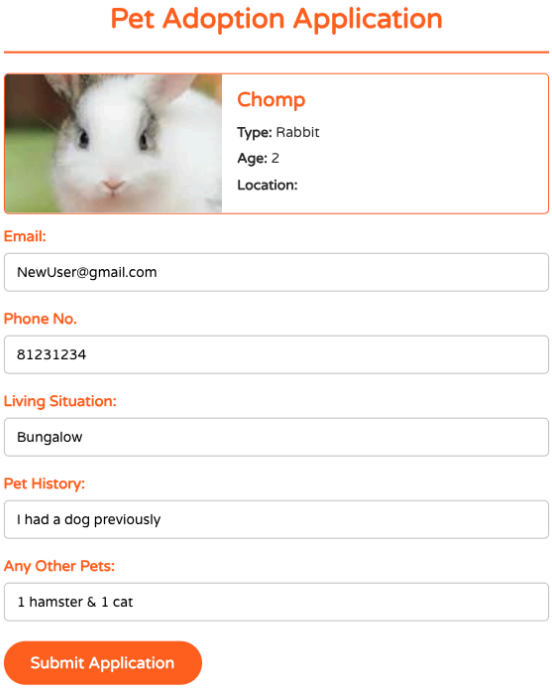
- Report would be sent to the owner upon submission.



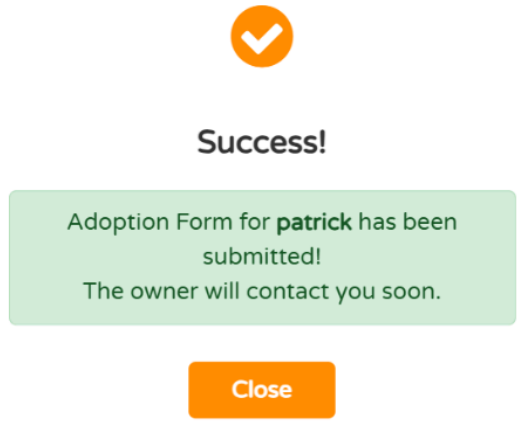
3.1.4 Functional Requirement 4: Adopter Actions

3.1.4.1 Submit Adoption Request (Use Case #4-1)

- Description: Adopters can send an adoption request to owners for pets they are interested in.

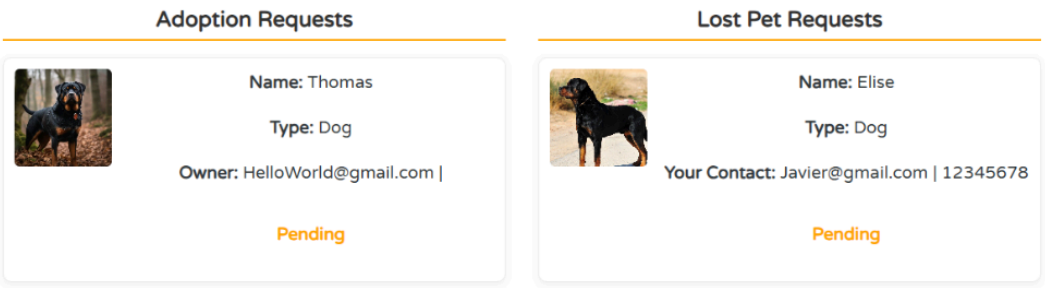


- Upon submitting the application with the relevant fields filled, the adoption request is sent to the Owner.



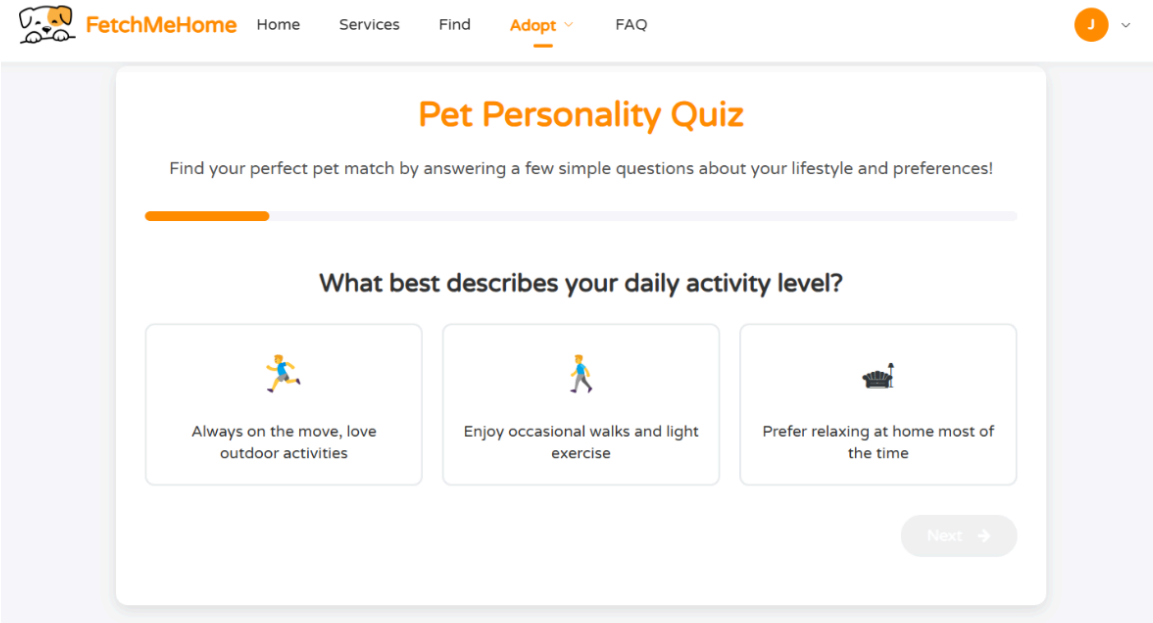
3.1.4.2 View Submitted Request List (Use Case #4-2)

- Description: Adopters can view all the adoption requests they have made for the pets they are interested in.



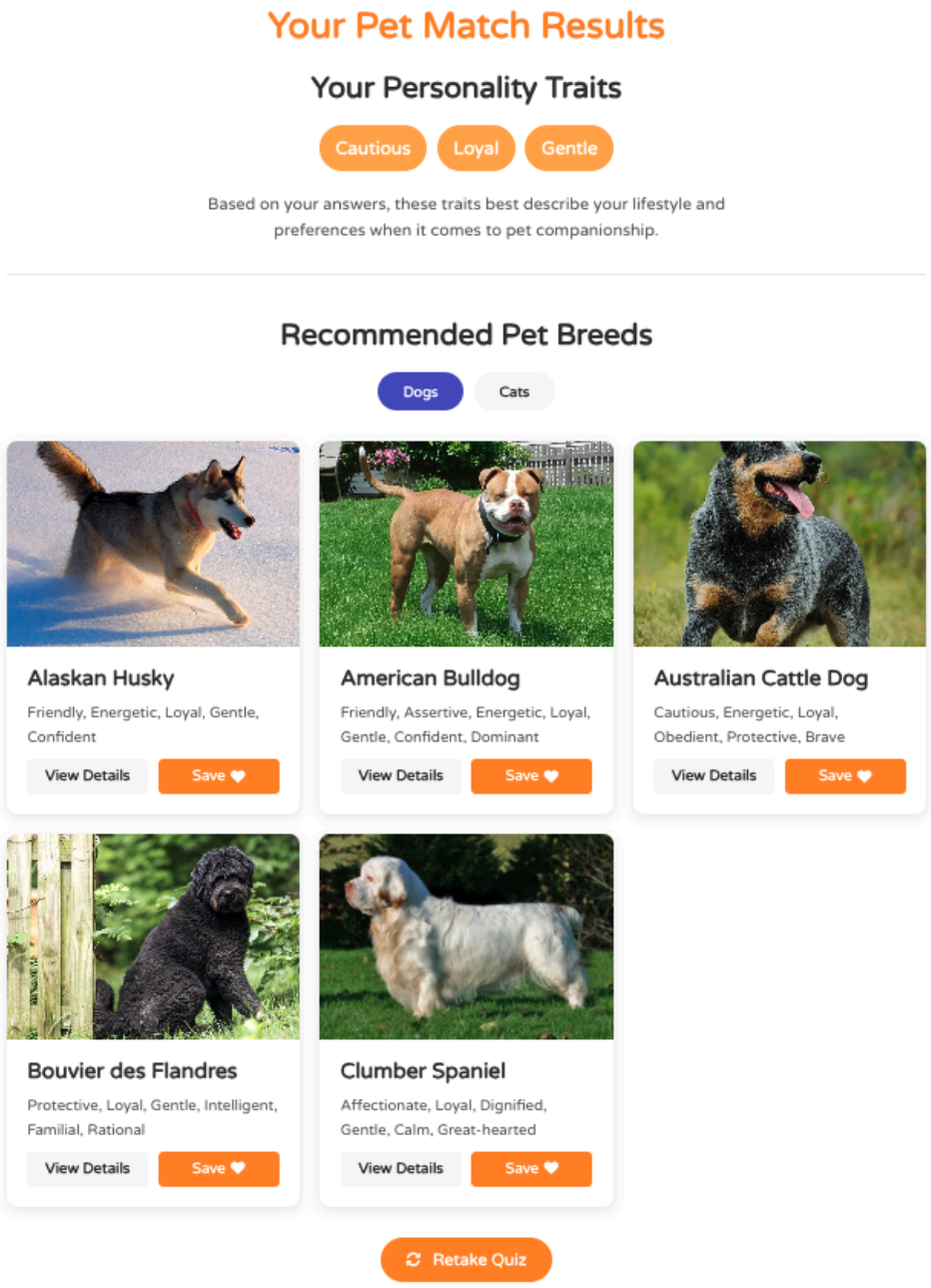
3.1.4.3 Personality Test (Use Case #4-3)

- Description: Adopters can take a personality test to determine which pet is suitable for them.



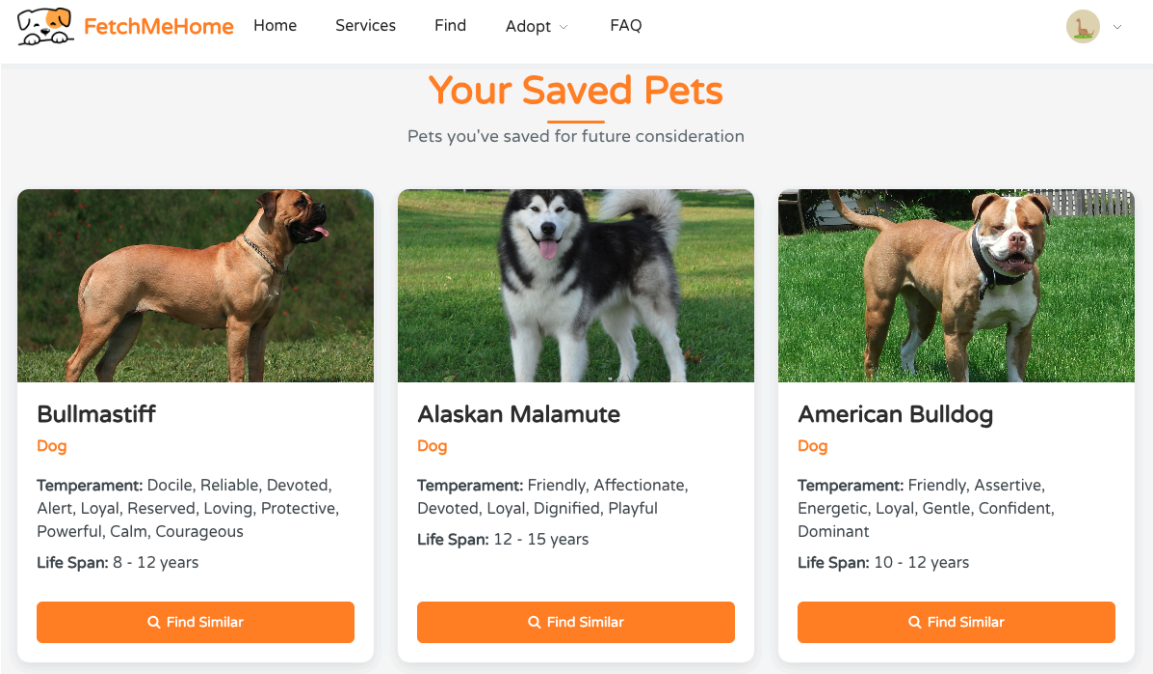
3.1.4.4 Suggest Breed (Use Case #4-4)

- Description: Adopters can get pets recommended for them based on the results of their personality test.



3.1.4.5 Save Suggested Pets (Implied Functionality)

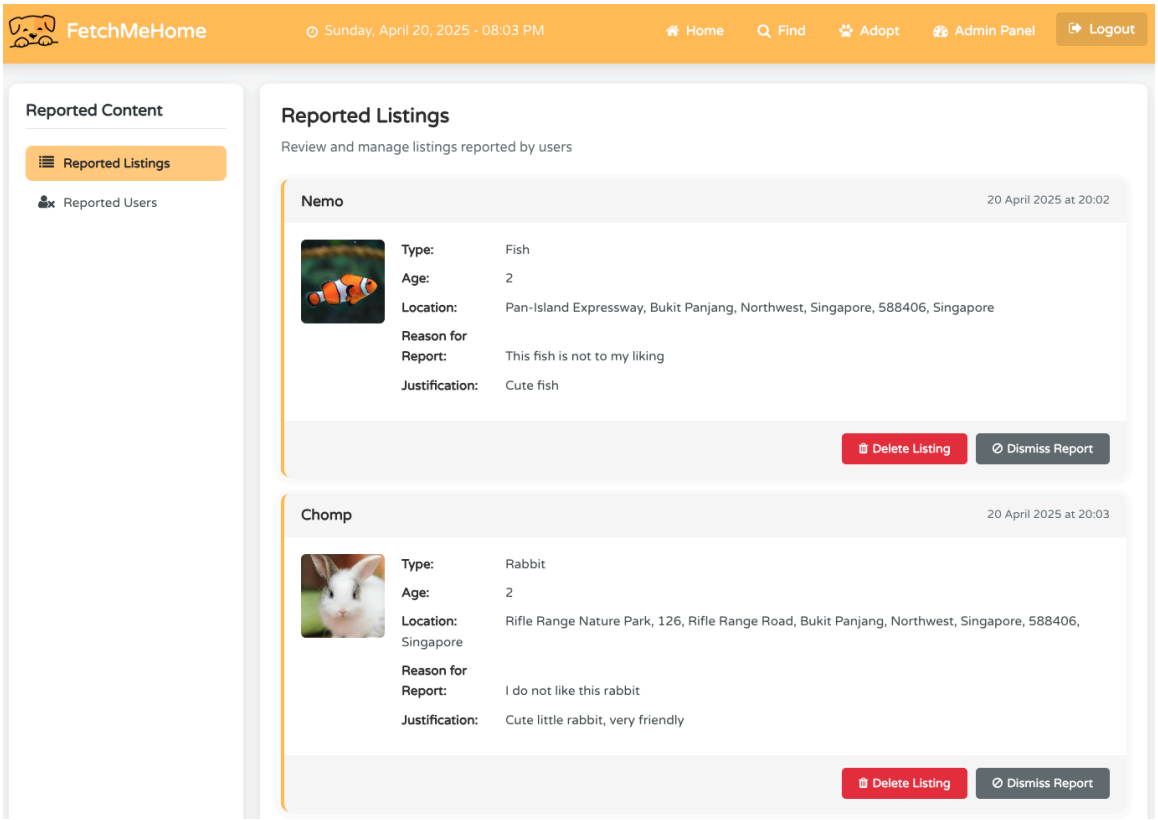
- Description: Users can save suggested pet breeds to a favorites list.

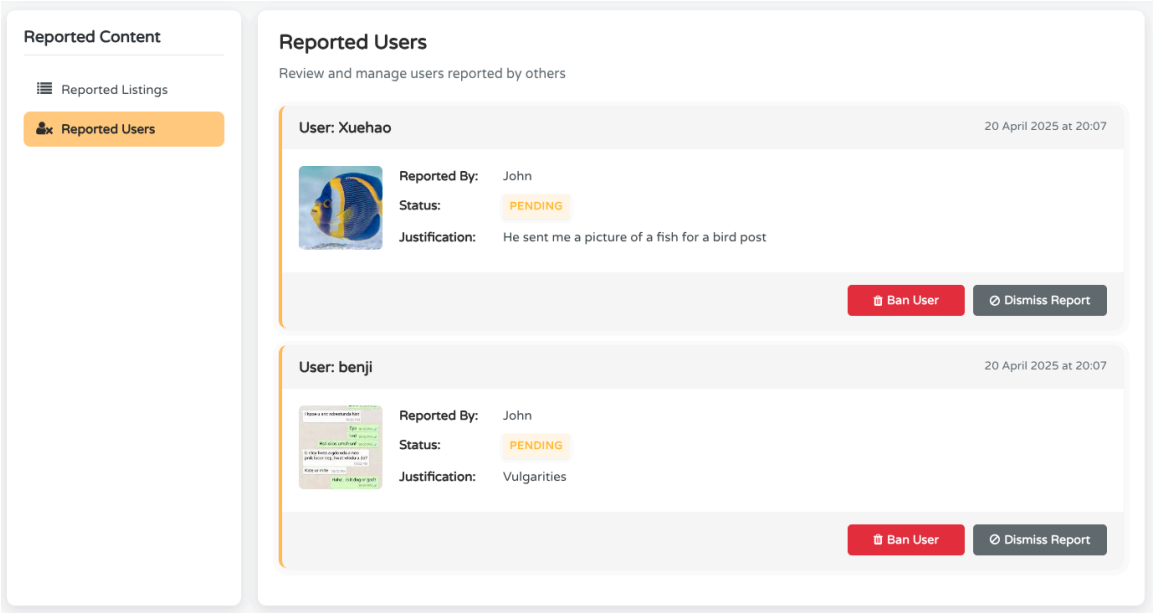


3.1.5 Functional Requirement 5: Administrator Actions

3.1.5.1 Review Activity (Use Case #5-1)

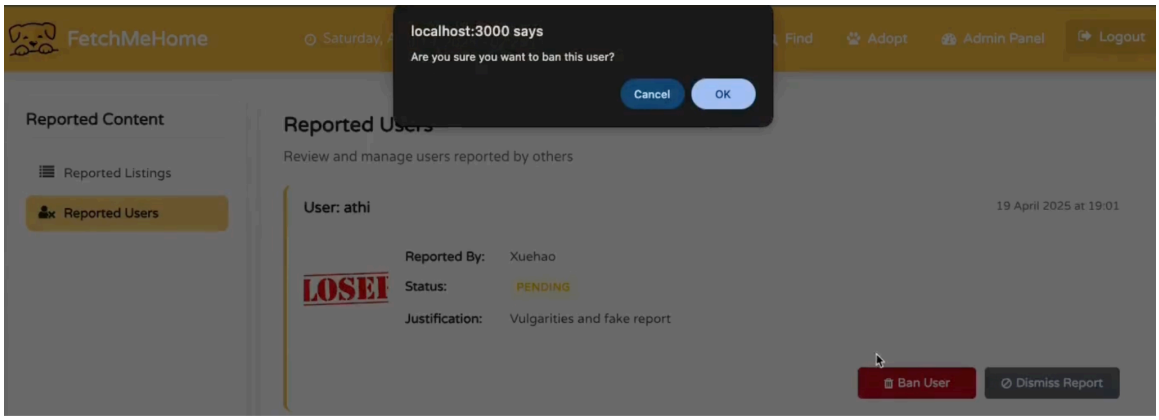
- Description: Administrators can review reported Users and Listings.





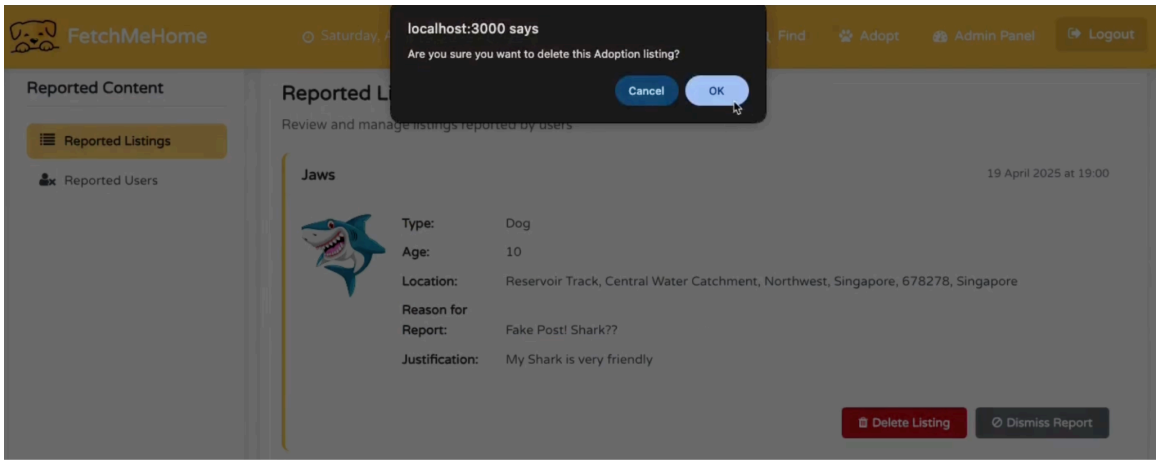
3.1.5.2 Ban User (Use Case #5-2)

- Description: Administrators can ban users deemed inappropriate.



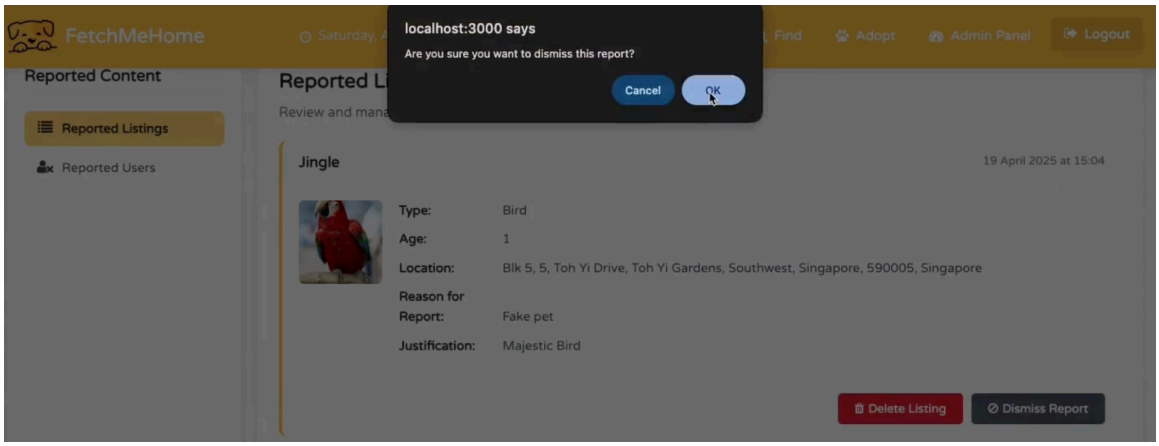
3.1.5.3 Delete Listing (Use Case #5-3)

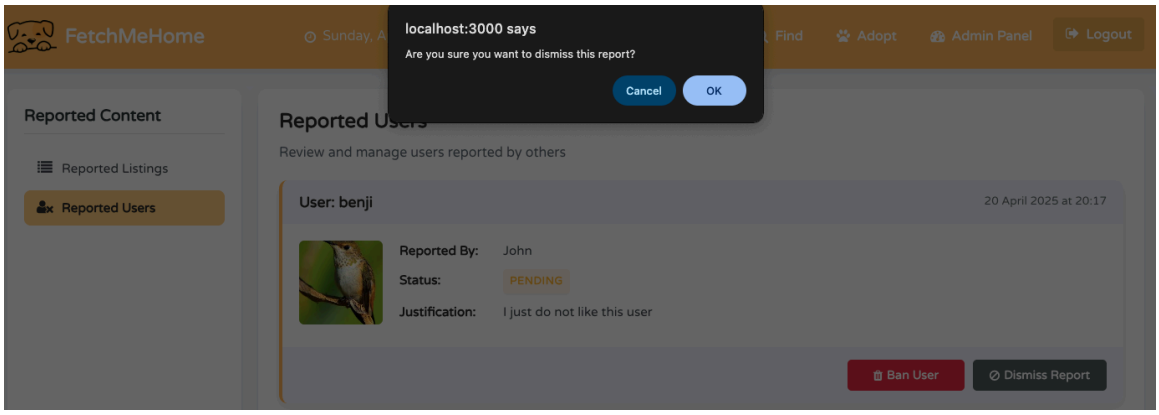
- Description: Administrators can remove listings that they deem inappropriate.



3.1.5.4 Dismiss Report (Use Case #5-4)

- Description: Administrators can remove reports that they deem unreasonable.





3.2 Hardware Interfaces

The FetchMeHome application is designed to be compatible with most laptops and desktops.

3.3 Software Interfaces

The FetchMeHome application uses a non-relational database to manage and store persistent data. Below are the primary tables and their schemas:

- **Users :**

```
name: {
type: String,
   required: true,
 },
  email: {
    type: String,
    required: true,
    unique: true,
 },
  password: {
    type: String,
    required: true,
 },
  isAdmin: {
     type: Boolean,
     default: false, // default is false
 }
```

- **AdoptPet (Listing) :**

```
name: {
      type: String,
      required: true
  },
  age: {
      type: String,
      required: true
  },
  area: {
      type: String,
      required: true
  },
  justification: {
      type: String,
      required: true
  },
  email: {
      type: String,
      required: true
  },
```

```
  phone: {
    type: String,
    required: true
  },
  type: {
    type: String,
    required: true
  },
  filename: {
    type: String,
    required: true
  },
  status: {
    type: String,
    default: "Pending"
  },
  postedBy: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true
  } // User who posted the pet
```

- **LostPet (Listing) :**

```
name: {
  type: String,
  required: true,
},
petAge: {
  type: String,
  required: true,
},
type: {
  type: String,
  required: true,
  enum: ["Dog", "Cat", "Rabbit", "Bird", "Fish", "Other"],
},
lastSeenLocation: {
  type: String,
  required: true,
},
description: {
  type: String,
  required: true,
},
filename: {
  type: String, // stores the image filename for retrieval
  required: true,
},
reportedBy: {
  type: mongoose.Schema.Types.ObjectId,
  ref: "User",
```

```
    required: true,
  },
  email: {
    type: String,
    required: true,
    match: /^[a-zA-Z0-9._-]+@gmail\.com$/, // Validate Gmail-only
emails
  },
  phone: {
    type: String,
    required: true,
  },
  status: {
    type: String,
    enum: ["Missing", "Found"], //status of pet
    default: "Missing",
  },
  createdAt: {
    type: Date,
    default: Date.now,
  },
```

- **RequestAdopt** :

```
petId: {
      type: mongoose.Schema.Types.ObjectId,
      ref: 'Pet',
      required: true
    },
    ownerId: {
      type: mongoose.Schema.Types.ObjectId,
      ref: 'User',
      required: true
    },
    adopterId: {
      type: mongoose.Schema.Types.ObjectId,
      ref: 'User',
      required: true
    },
    email: {
      type: String,
      required: true
    },
    phoneNo: {
      type: String,
      required: true
    },
    livingSituation: {
      type: String,
      required: true
    },
    previousExperience: {
```

```
    type: String,
    required: true
  },
  familyComposition: {
    type: String,
    required: true
  },
  status: {
    type: String,
    enum: ['Pending', 'Approved', 'Rejected'],
    default: 'Pending'
  }, // request status
```

- **LostPetRequest** :

```
petId: {
  type: mongoose.Schema.Types.ObjectId,
  ref: "LostPet",
  required: true
},
finderId: {
  type: mongoose.Schema.Types.ObjectId,
  ref: "User",
  required: true
}, // Reference to the lost pet
finderEmail: {
  type: String,
  required: true
},  // Finder's email
finderPhone: {
  type: String,
  required: true
},  // Finder's phone
image: {
  type: String,
  required: true
},
status: {
  type: String,
  enum: ["Pending", "Accepted"],
  default: "Pending"
}, //  New Status Field     // Image filename
createdAt: {
  type: Date,
  default: Date.now
},// Date the request was submitted
```

- **ReportListing** :

```
petId: {
  type: mongoose.Schema.Types.ObjectId,
  ref: "Pet",
  required: true
```

```
},
name: {
  type: String,
  required: true
},
type: {
  type: String,
  required: true
},
age: {
  type: String,
  required: true
},
area: {
  type: String,
  required: true
},
justification: {
  type: String,
  required: true
},
filename: {
  type: String,
  required: true
},
email: {
  type: String,
  required: true
},
phone: {
  type: String,
  required: true
},
reason: {
  type: String,
  required: true
}, // reason of report
createdAt: {
  type: Date,
  default: Date.now
}
```

- **ReportUser** :

```
finderId: {
  type: mongoose.Schema.Types.ObjectId,
  ref: "User",
  required: true
},
reportedBy: {
  type: mongoose.Schema.Types.ObjectId,
  ref: "User",
```

```
  required: true
},
image: {
  type: String,
  required: true
},
justification: {
  type: String,
  required: true
}, // reason of report
status: {
  type: String,
  default: "Pending"
},
createdAt: {
  type: Date,
  default: Date.now
},
```

3.4 Communications Interfaces

For FetchMeHome, various communication interfaces are integral to ensuring a seamless and responsive user experience:

- HTTP/HTTPS: All interactions with the web application will occur over HTTP/HTTPS protocols, ensuring compatibility with modern web browsers and secure data transmission.
- API Communication: RESTful APIs will be used for most server communications, with JSON as the primary data interchange format. For the documentation and testing of these APIs, Postman will be employed.
- Electronic Forms: Forms submitted through the platform will use secure HTTPS transactions to protect user-submitted data.
- Network Protocols: Standard network protocols such as TCP/IP will be used for network communication, ensuring adherence to established internet communication standards.

These communication interfaces are designed to provide robust, efficient, and secure interactions within the platform, aligning with the goal of providing a reliable service for helping pets find a new home or bringing them back to their loved ones.

4. System Features

4.1 Functional Requirements

1. FetchMeHome shall allow Users to be authenticated and use the website.
    1.1. FetchMeHome shall allow Users to create an account
        1.1.1. FetchMeHome shall allow users to input their Full Name, E-Mail, Password.
        1.1.2. FetchMeHome shall create an account with the information entered by the user.
    1.2. FetchMeHome shall allow Users to log in using the account they have created previously
        1.2.1. FetchMeHome shall allow Users to enter their E-Mail and Password to log into the application.
        1.2.2. FetchMeHome shall mask the Password entered by the Users to protect the User's privacy.
        1.2.3. FetchMeHome shall verify if the E-Mail and the password keyed in by the User are valid before letting them log in.
    1.3. FetchMeHome shall allow Users to edit their profile details.
        1.3.1. FetchMeHome shall allow Users to edit their email.
        1.3.2. FetchMeHome shall allow Users to edit their profile photo.
        1.3.3. FetchMeHome shall allow Users to edit their password.
    1.4. FetchMeHome shall allow Users to view and query all features
        1.4.1. FetchMeHome shall display all features via a list interface to provide a comprehensible view of all available options.
        1.4.2. FetchMeHome shall provide a Find option that allows Users to find any missing pets near their vicinity.

1.4.3. FetchMeHome shall provide an Adopt option that allows Users to adopt a pet based on the listing.

2. FetchMeHome shall allow Users to perform general tasks without a user account.
   2.1. Users must be able to view lost pet listings.
      2.1.1. FetchMeHome shall display pet details for each lost pet listing.
      2.1.2. FetchMeHome shall allow Users to filter lost pet listings based on the type of pet and the last seen location of the lost pet.
      2.1.3. FetchMeHome shall allow Users to report inappropriate lost pet listings.
   2.2. Users must be able to view pet adoption listings.
      2.2.1. FetchMeHome shall display pet details for each adoption listing
      2.2.2. FetchMeHome shall allow Users to filter pet adoption listings based on the type of pet.
      2.2.3. FetchMeHome shall allow Users to report inappropriate pet adoption listings.
   2.3. Users must be able to view the Frequently Asked Questions (FAQ) page.
3. FetchMeHome shall allow Owners to perform Owner specific tasks
   3.1. FetchMeHome shall allow Owners to create new lost pet listings.
      3.1.1. Owners must upload images of the lost pet.
      3.1.2. Owners must enter the name of the lost pet.
      3.1.3. Owners must enter the age of the lost pet.
      3.1.4. Owners must enter the type of the lost pet.
      3.1.5. Owners must use the live map feature to indicate the location of the lost pet.
      3.1.6. Owners must enter a short description of the lost pet
   3.2. FetchMeHome shall allow Owners to interact with PetFinders
      3.2.1. FetchMeHome shall allow Owners to verify images submitted by PetFinders.
      3.2.2. FetchMeHome shall allow Owners to flag PetFinders for misconduct.

3.2.2.1. FetchMeHome shall allow the Adopter to enter a short description of the situation.

3.3. FetchMeHome shall allow Owners to create new adoption pet listings.

3.3.1. Owners must upload images of the pet for adoption.

3.3.2. Owners must enter the name of the pet for adoption.

3.3.3. Owners must enter the age of the pet for adoption.

3.3.4. Owners must enter the type of the pet for adoption.

3.3.5. Owners must use the live map feature to specify the location of the adoption meet-up.

3.3.6. Owners must enter a short description of the pet for adoption.

3.4. FetchMeHome shall allow Owners to edit details of their existing listings.

3.4.1. FetchMeHome shall allow Owners to edit any of their desired attributes.

3.5. FetchMeHome shall allow Owners to interact with Adopters.

3.5.1. FetchMeHome shall allow Owners to review Adopters' adoption requests.

3.5.2. FetchMeHome shall allow Owners to contact Adopters to liaise meeting details.

3.5.3. FetchMeHome shall allow Owners to flag Adopters for misconduct.

4. FetchMeHome shall allow PetFinders to perform PetFinder-specific tasks.

4.1. FetchMeHome shall allow PetFinders to report found pets.

4.2. FetchMeHome shall allow PetFinders to flag inappropriate listings.

5. FetchMeHome shall allow Adopters to perform Adopter specific functions.

5.1. FetchMeHome shall allow Adopters to submit adoption requests for their desired pet.

5.1.1. Adopters must enter their email.

5.1.2. Adopters must enter their phone number.

5.1.3. Adopters must state their current housing situation.

5.1.4. Adopters must fill in their past history with pets.

5.1.5. Adopters must fill in their current list of pets.

5.2. FetchMeHome shall allow Adopters to flag Owners for misconduct.

5.2.1. FetchMeHome shall allow the Adopter to enter a short description of the situation.

6. FetchMeHome shall allow Administrators to perform Administrator specific functions.

6.1. FetchMeHome shall allow Administrators to review reports made.

6.1.1. FetchMeHome shall allow Administrators to ban the Users from the platform.

6.1.2. FetchMeHome shall allow Administrators to remove listings from the platform.

6.1.3. FetchMeHome shall allow Administrators to dismiss the report.

4.2 Use Case Descriptions

4.2.1 Functional Requirement 1

4.2.1.1 Create Account

Use Case ID:	#1-1		
Use Case Name:	CreateAccount		
Created By:	Baskar Kayalvizhi Athithiya	Last Updated By:	Baskar Kayalvizhi Athithiya
Date Created:	08/02/2025	Date Last Updated:	08/02/25

Actor:	User
Description:	Allows first-time users to create an Account.
Preconditions:	None
Postconditions:	An Account is created for the User.
Priority:	High
Frequency of Use:	High
Flow of Events:	1. System will prompt the user to enter their Full Name, E-mail Address, Phone Number, Password, and Retype Password. 2. System checks all the required fields. 3. User selects “Validate Phone Number”. 4. System generates a One-Time-Password (OTP) and sends the OTP to the User’s entered Phone Number. 5. User enters the OTP.

	6. An Account is created for the User and the database is updated.
Alternative Flows:	AF-S2: User did not fill in all required fields in the form 1. User did not fill in all required fields in the form. 2. Form is not submitted, the system prompts User to fill in all required fields. 3. System returns to step 1. AF-S3: User phone number is already registered 1. User phone number is already registered with the app. 2. System will prompt the User to login using the login use case. AF-S4: User OTP entered is incorrect 1. User OTP entered is incorrect. 2. System will prompt the User to enter the correct OTP. 3. System returns to step 4.
Exceptions:	None
Includes:	Walkthrough
Special Requirements:	System needs to validate User input data
Assumptions:	None
Notes and Issues:	None

4.2.1.2 Login

Use Case ID:	#1-2
Use Case Name:	Login

Created By:	Baskar Kayalvizhi Athithiya	Last Updated By:	Baskar Kayalvizhi Athithiya
Date Created:	08/02/2025	Date Last Updated:	08/02/25

Actor:	User
Description:	Allows the User to login using their phone number and password.
Preconditions:	Users must have an existing FetchMeHome account.
Postconditions:	The User is logged in to the FetchMeHome application and is navigated to the home screen of the application.
Priority:	High
Frequency of Use:	High
Flow of Events:	1. System prompts the User to enter their phone number and password. 2. The User enters their Phone number and Password into the respective fields. 3. The User selects “Login”. 4. System checks all the required fields. 5. The User is navigated to the home screen of the FetchMeHome application.
Alternative Flows:	AF-S4: User did not fill in all required fields in the page 1. User did not fill in all required fields in the page. 2. Login is not processed, the system prompts User to fill in all required fields. 3. System returns to step 1.

Exceptions:	None
Includes:	None
Special Requirements:	System needs to validate the email and password provided by the User.
Assumptions:	User already has an existing FetchMeHome account.
Notes and Issues:	The Password will be masked as dots but the User can view them by pressing the eye icon.

4.2.1.3 ViewMissingPetList

Use Case ID:	#1-3		
Use Case Name:	ViewMissingPetList		
Created By:	Baskar Kayalvizhi Athithiya	Last Updated By:	Baskar Kayalvizhi Athithiya
Date Created:	08/02/2025	Date Last Updated:	08/02/25

Actor:	User
Description:	Users can view all available Lost Pet listings
Preconditions:	The User must be logged in to the FetchMeHome application

Postconditions:	The User will be able to view all the listings of Lost Pets
Priority:	High
Frequency of Use:	High
Flow of Events:	1. The User shall click on the “LostPet” icon on the navigation bar of the home screen of the FetchMeHome application. 2. The System shall then display all the posts of Lost Pets. 3. The User can then click on any individual post to view more details of the Lost Pets.
Alternative Flows:	None
Exceptions:	None
Includes:	None
Special Requirements:	None
Assumptions:	None
Notes and Issues:	None

4.2.1.4 ViewPetAdoptionList

Use Case ID:	#1-4		
Use Case Name:	ViewPetAdoptionList		
Created By:	Baskar Kayalvizhi Athithiya	Last Updated By:	Baskar Kayalvizhi Athithiya

Date Created:	08/02/2025	Date Last Updated:	08/02/25

Actor:	User
Description:	Users can view all listings of pets that are up for adoption
Preconditions:	The User must be logged in to the FetchMeHome application
Postconditions:	The User will be able to view all listings of the pets that are up for adoption
Priority:	High
Frequency of Use:	High
Flow of Events:	1. User shall click the “AdoptPet” icon on the navigation bar of the home screen of the FetchMeHome application. 2. The System shall then display all listings of Pets that are up for adoption. 3. Users can click on any individual post to expand it and view more details of the pet that is up for adoption.
Alternative Flows:	None
Exceptions:	None
Includes:	None
Special Requirements:	None
Assumptions:	None

Notes and Issues:	None

4.2.1.5 ManageProfile

Use Case ID:	#1-5		
Use Case Name:	ManageProfile		
Created By:	Baskar Kayalvizhi Athithiya	Last Updated By:	Baskar Kayalvizhi Athithiya
Date Created:	08/02/2025	Date Last Updated:	08/02/25

Actor:	User
Description:	Users can manage the status of their profile
Preconditions:	The User must be logged in to their FetchMeHome account
Postconditions:	The User's account details and status will be updated by the details provided by the User and the actions taken
Priority:	High
Frequency of Use:	Low
Flow of Events:	1. The User will click on the "Profile" icon on the navigation bar of the FetchMeHome home page 2. The System shall then display the profile page of the User. 3. The System shall then display the initial details provided by the User during Account creation.

	4. The System shall display the “Edit Profile” button and the “Delete Account” buttons at the bottom of the profile. 5. If the User clicks on the “Edit Profile” button then they will use the included use case EditProfile to edit their profile. 6. If the User clicks on the “Delete Account” button then they will use the included use case DeleteAccount to delete their profile. 7. If the User clicks on the “Log Out” button then they will use the included use case LogOut to log out of their account.
Alternative Flows:	None
Exceptions:	None
Includes:	EditProfile, DeleteAccount, LogOut
Special Requirements:	None
Assumptions:	None
Notes and Issues:	None

4.2.1.6 EditProfile

Use Case ID:	#1-6		
Use Case Name:	EditProfile		
Created By:	Baskar Kayalvizhi Athithiya	Last Updated By:	Baskar Kayalvizhi Athithiya
Date Created:	08/02/2025	Date Last Updated:	08/02/25

Actor:	User
Description:	Users can edit the details of their profile such as phone number, email address, and name, and add a profile photo
Preconditions:	1. The User must have an existing FetchMeHome application 2. The User must be logged in to the FetchMeHome application
Postconditions:	The information given by the User will be updated on their profile
Priority:	Low
Frequency of Use:	Low
Flow of Events:	1. The User will click the “Edit Profile” button on their profile. 2. The system shall prompt the User to key in the updated personal particulars. 3. The User shall then click the “Confirm” button to save the changes made to their profile.
Alternative Flows:	None
Exceptions:	None
Includes:	None
Special Requirements:	None
Assumptions:	The User has an existing FetchMeHome account
Notes and Issues:	None

4.2.1.7 LogOut

Use Case ID:	#1-7		
Use Case Name:	LogOut		
Created By:	Baskar Kayalvizhi Athithiya	Last Updated By:	Baskar Kayalvizhi Athithiya
Date Created:	12/02/2025	Date Last Updated:	12/02/25

Actor:	User
Description:	Allows Users to log out of their FetchMeHome account
Preconditions:	User is logged in and authenticated.
Postconditions:	User is logged out of the FetchMeHome application
Priority:	High
Frequency of Use:	Low
Flow of Events:	1. The System shall display a “Log Out” button in the profile of the User. 2. The User shall then click the “Log Out” button. 3. The System shall then display the “Confirm” and “Cancel” buttons. 4. The User shall then click the “Confirm” button. 5. The User shall then be logged out of the FetchMeHome application.

	6. The System shall then display the Login page of the FetchMeHome application.
Alternative Flows:	AF-S3: The User chooses to click the “Cancel” button instead 1. The User shall click on the “Cancel” button. 2. The System shall then return to the profile of the User.
Exceptions:	None
Includes:	None
Special Requirements:	None
Assumptions:	None
Notes and Issues:	None

4.2.1.8 FlagListing

Use Case ID:	#1-8		
Use Case Name:	FlagListing		
Created By:	Benjamin Yeoh	Last Updated By:	Benjamin Yeoh
Date Created:	08/02/2025	Date Last Updated:	08/02/25

Actor:	User
Description:	Allows Users to flag listings deemed inappropriate.
Preconditions:	Useris logged in and authenticated.
Postconditions:	Administrators are notified.
Priority:	Medium
Frequency of Use:	Low
Flow of Events:	1. The System shall display a “FlagListing” button. 2. If the User deems the listing as inappropriate, the User can click the “FlagListing” button. 3. The System will prompt the User to fill in details of the scenario in a short description box. 4. Checks are made to ensure the description box is filled. 5. User clicks the “Submit” button, a notification is sent to the Administrators and the database is updated.
Alternative Flows:	<u>AF-S4: User did not fill up required field in the form</u> 1. User did not fill up the required field in the form. 2. User clicks the “Submit” button. 3. Checks that the required field is not filled up. 4. User is prompted to fill in the required field, no changes to the form. 5. System returns to step 3.
Exceptions:	None
Includes:	None

Special Requirements:	None
Assumptions:	None
Notes and Issues:	None

4.2.2 Functional Requirement 2

4.2.2.1 ManageRequests

Use Case ID:	#2-1		
Use Case Name:	ManageRequests		
Created By:	Benjamin Yeoh	Last Updated By:	Benjamin Yeoh
Date Created:	08/02/2025	Date Last Updated:	08/02/25

Actor:	Owner
Description:	Allows Owner to manage requests made by PetFinder and/or Adopter.
Preconditions:	Owner is logged in and is authenticated.
Postconditions:	None
Priority:	High
Frequency of Use:	High
Flow of Events:	1. System shall display a “ManageRequests” button in the profile page. 2. Upon clicking the button, the Owner will be provided with 2 use cases: a. ReviewAdoptionRequest b. ReviewLostPetRequest

	3. If the Owner selects the activity ReviewAdoptionRequest, then the Owner uses the included use case ReviewAdoptionRequest to review adoption request(s) made by Adopter(s). 4. If the Owner selects the activity ReviewLostPetRequest, then the Owner uses the included use case ReviewLostPetRequest to review missing pet report(s) made by PetFinder(s).
Alternative Flows:	None
Exceptions:	None
Includes:	1. ReviewAdoptionRequest 2. ReviewLostPetRequest
Special Requirements:	None
Assumptions:	None
Notes and Issues:	None

4.2.2.2 ReviewAdoptionRequest

Use Case ID:	#2-2		
Use Case Name:	ReviewAdoptionRequest		
Created By:	Benjamin Yeoh	Last Updated By:	Benjamin Yeoh

Date Created:	08/02/2025	Date Last Updated:	08/02/25

Actor:	Owner
Description:	Allows Owner to view all available adoption requests made by the Adopter.
Preconditions:	Owner is logged in and is authenticated.
Postconditions:	None
Priority:	High
Frequency of Use:	High
Flow of Events:	1. System shall display a list of all Adopter(s) adoption requests with their personality test results attached as an image. 2. For each request, the System will provide an “Accept” and a “Decline” button for the Owner to determine if the respective Adopter is eligible for adoption. 3. Upon clicking the “Accept” button for a particular request, the System will provide the Owner with the respective Adopter’s Phone No.
Alternative Flows:	<u>AF-S2: Owner selects the “Decline” button</u> 1. If the Owner selects the “Decline” button, the System will delete the specific request. 2. System returns to step 1.

Exceptions:	None
Includes:	None
Special Requirements:	None
Assumptions:	None
Notes and Issues:	None

4.2.2.3 ReviewLostPetRequest

Use Case ID:	#2-3		
Use Case Name:	ReviewLostPetRequest		
Created By:	Benjamin Yeoh	Last Updated By:	Benjamin Yeoh
Date Created:	08/02/2025	Date Last Updated:	08/02/25

Actor:	Owner
Description:	Allows Owner to view all lost pet reports made by PetFinder.
Preconditions:	Owner is logged in and is authenticated.

Postconditions:	None
Priority:	High
Frequency of Use:	High
Flow of Events:	1. System shall display a list of all lost pet reports from PetFinder(s). 2. For each report, the System shall display the uploaded image(s) provided by the PetFinder, as well as a “Accept” and a “Decline” button. 3. Owner shall determine if any of the images uploaded is his pet. 4. If there is an image match for a particular report, the Owner will click the “Accept” button, and the system will provide the Owner with the respective PetFinder Phone No.
Alternative Flows:	AF-S2: Owner selects the “Decline” button 1. If the Owner selects the “Decline” button, the System will delete the specific report. 2. System returns to step 1.
Exceptions:	None
Includes:	None
Special Requirements:	None
Assumptions:	None
Notes and Issues:	None

4.2.2.4 CreateListing

Use Case ID:	#2-4		
Use Case Name:	CreateListing		
Created By:	Benjamin Yeoh	Last Updated By:	Benjamin Yeoh
Date Created:	08/02/2025	Date Last Updated:	08/02/2025

Actor:	Owner
Description:	Allows Owners to create listings for their missing pet(s) and/or pet(s) up for adoption.
Preconditions:	Owner is logged in and is authenticated.
Postconditions:	None
Priority:	High
Frequency of Use:	High
Flow of Events:	1. System shall display an “Add” icon in the home page. 2. Upon clicking the “Add’ icon, Owner will be provided with 2 use cases: a. CreateAdoptListing b. CreateLostPetListing

	3. If the Owner selects the activity CreateAdoptListing, then the Owner uses the included use case CreateAdoptListing to create a listing for their pet(s)to put up for adoption. 4. If the Owner selects the activity CreateLostPetListing, then the Owner uses the included use case CreateLostPetListing to create a listing for their missing pet(s).
Alternative Flows:	None
Exceptions:	None
Includes:	1. CreateAdoptListing 2. CreateLostPetListing
Special Requirements:	None
Assumptions:	None
Notes and Issues:	None

4.2.2.5 CreateAdoptListing

Use Case ID:	#2-5		
Use Case Name:	CreateAdoptListing		
Created By:	Benjamin Yeoh	Last Updated By:	Benjamin Yeoh
Date Created:	08/02/2025	Date Last Updated:	08/02/2025

Actor:	Owner
Description:	Allows Owner to create a listing for pets up for adoption.
Preconditions:	Owner is logged in and is authenticated.
Postconditions:	Listing is successfully posted.
Priority:	High
Frequency of Use:	High
Flow of Events:	1. Owner fills in all the required forms presented to them. 2. Owner clicks the “Submit” button, CreateAdoptListing is called. 3. Checks are made to ensure all the required fields are filled up. 4. Listing is created and added to the list of pets up for adoption.
Alternative Flows:	AF-S3: Owner did not fill up all required fields in the form 1. Owner did not fill up all required fields in the form. 2. Owner clicks the “Submit” button. 3. The System flags out that some of the required fields are not filled up. 4. System returns to step 1.
Exceptions:	None
Includes:	None
Special Requirements:	None
Assumptions:	None

Notes and Issues:	None

4.2.2.6 CreateLostPetListing

Use Case ID:	#2-6		
Use Case Name:	CreateLostPetListing		
Created By:	Benjamin Yeoh	Last Updated By:	Benjamin Yeoh
Date Created:	08/02/2025	Date Last Updated:	08/02/2025

Actor:	Owner
Description:	Allows Owner to create a listing for their missing pet(s).
Preconditions:	Owner is logged in and authenticated.
Postconditions:	Listing is successfully posted.
Priority:	High
Frequency of Use:	High
Flow of Events:	1. Owner fills in all the required forms presented to them. 2. Owner clicks the “Submit” button, CreateLostPetListing is called. 3. Checks are made to ensure all the required fields are filled up.

	4. Listing is sent and updated to the database.
Alternative Flows:	AF-S3: Owner did not fill up all required fields in the form 1. Owner did not fill up all required fields in the form. 2. Owner clicks the “Submit” button, CreateLostPetListing is called. 3. Checks that some required fields are not filled up. 4. The System flags out that some of the required fields are not filled up. 5. System returns to step 1.
Exceptions:	None
Includes:	None
Special Requirements:	None
Assumptions:	None
Notes and Issues:	None

4.2.2.7 ManageMyListings

Use Case ID:	#2-7		
Use Case Name:	ManageMyListings		
Created By:	Benjamin Yeoh	Last Updated By:	Benjamin Yeoh
Date Created:	08/02/2025	Date Last Updated:	08/02/2025

Actor:	Owner
Description:	Allows Owners to manage all their available listing(s).
Preconditions:	Owner is logged in and is authenticated.
Postconditions:	None
Priority:	High
Frequency of Use:	High
Flow of Events:	1. System shall display a “MyPanel” button in the profile page. 2. Upon clicking the “MyPanel” button, the system shall display all available listing(s) which the Owner had uploaded. 3. The System shall provide the Owner with 2 use cases for each listing: a. Edit b. Delete 4. If the Owner selects the activity Edit, he will use the included use case Edit to edit his listing. 5. If the Owner selects the activity Delete, he will use the included use case Delete to delete his listing.
Alternative Flows:	None
Exceptions:	None
Includes:	1. ViewListing 2. EditLising 3. DeleteListing

Special Requirements:	None
Assumptions:	Owner has existing listings.
Notes and Issues:	None

4.2.2.8 ViewListing

Use Case ID:	#2-8		
Use Case Name:	ViewListing		
Created By:	Benjamin Yeoh	Last Updated By:	Benjamin Yeoh
Date Created:	08/02/2025	Date Last Updated:	08/02/2025

Actor:	Owner
Description:	Allows Owners to view all their available listing(s).
Preconditions:	Owner is logged in and is authenticated.
Postconditions:	None
Priority:	High

Frequency of Use:	High
Flow of Events:	1. System shall display a “MyPanel” button in the profile page. 2. Upon clicking the “MyPanel” button, the system shall display all available listing(s) which the Owner had uploaded.
Alternative Flows:	None
Exceptions:	None
Includes:	None
Special Requirements:	None
Assumptions:	Owner has existing listings.
Notes and Issues:	None

4.2.2.9 EditListing

Use Case ID:	#2-9		
Use Case Name:	EditListing		
Created By:	Benjamin Yeoh	Last Updated By:	Benjamin Yeoh
Date Created:	08/02/2025	Date Last Updated:	08/02/2025

Actor:	Owner
Description:	Allows Owners to edit listings for their missing pet(s) and/or pet(s) up for adoption.
Preconditions:	Owner is logged in and is authenticated.
Postconditions:	Listing is successfully edited.
Priority:	Medium
Frequency of Use:	Medium
Flow of Events:	1. The Owner selects the “Edit” button, he would be allowed to edit on his selected listing. 2. System shall display a “Submit” button. 3. Upon clicking the “Submit” button, the system checks that all required fields are filled up. 4. The listing is updated.
Alternative Flows:	AF-S3: Owner did not fill up all required fields in the form 1. Owner did not fill up all required fields in the form. 2. Owner clicks the “Submit” button, EditListing is called. 3. Checks that some required fields are not filled up. 4. System returns to step 1.
Exceptions:	None
Includes:	None
Special Requirements:	None

Assumptions:	Owner has existing listings.
Notes and Issues:	None

4.2.2.10 DeleteListing

Use Case ID:	#2-10		
Use Case Name:	DeleteListing		
Created By:	Benjamin Yeoh	Last Updated By:	Benjamin Yeoh
Date Created:	08/02/2025	Date Last Updated:	08/02/2025

Actor:	Owner
Description:	Owner is able to remove his listing(s).
Preconditions:	Owner is logged in and authenticated.
Postconditions:	Listing is successfully deleted
Priority:	Medium
Frequency of Use:	Medium

Flow of Events:	1. The Owner selects the “Delete” button. 2. System shall display a “Confirm” button and a “Cancel” button. 3. Upon clicking the “Confirm” button, the listing is removed from the database.
Alternative Flows:	AF-S2: Owner clicks the “Cancel” button 1. If Owner clicks the “Cancel” button, the System will not remove the listing.
Exceptions:	None
Includes:	None
Special Requirements:	None
Assumptions:	Owner has existing listings
Notes and Issues:	None

4.2.2.11 FlagUser

Use Case ID:	#2-11		
Use Case Name:	FlagUser		
Created By:	Benjamin Yeoh	Last Updated By:	Benjamin Yeoh
Date Created:	08/02/2025	Date Last Updated:	08/02/2025

Actor:	Owner
Description:	Allows Owner to flag Users deemed inappropriate.
Preconditions:	Owner is logged in and authenticated.
Postconditions:	Administrators are notified.
Priority:	Medium
Frequency of Use:	Low
Flow of Events:	1. The System shall display a “FlagUser” icon. 2. If the Owner/Adopter/PetFinder deems the User’s actions (online or physical) as inappropriate, the Owner/Adopter/PetFinder can click the “FlagUser” icon. 3. The System will prompt the Owner to fill in details of the scenario in a short description box and/or any images / supporting evidence. 4. Checks are made to ensure the description box is filled. 5. Owner clicks the “Submit” button, a notification is sent to the Administrators and the database is updated.
Alternative Flows:	<u>AF-S4: Owner did not fill up required field in the form</u> Owner did not fill up the required field in the form. Owner clicks the “Submit” button. Checks that the required field is not filled up.
Exceptions:	None

Includes:	None
Special Requirements:	None
Assumptions:	Owner has communicated to PetFinder and/or Adopter.
Notes and Issues:	None

4.2.3 Functional Requirement 3

4.2.3.1 SubmitLostPet

Use Case ID:	#3-1		
Use Case Name:	SubmitLostPet		
Created By:	Baskar Kayalvizhi Athithiya	Last Updated By:	Baskar Kayalvizhi Athithiya
Date Created:	08/02/2025	Date Last Updated:	09/02/25

Actor:	PetFinder
Description:	PetFinder can report missing pets in their vicinity.
Preconditions:	1. PetFinder is logged in and authenticated. 2. PetFinder can only report the missing animal through the corresponding LostPet listing
Postconditions:	The Owner will be notified.
Priority:	High
Frequency of Use:	High
Flow of Events:	1. System shall display a list of all Lost Pet(s). 2. PetFinder clicks on the respective LostPet listing which corresponds to the pet they found. 3. System shall display a “Report” button. 4. If the PetFinder clicks on the “Report” button, the System will direct the PetFinder to upload image(s) of the pet.

	5. System displays a “Upload” and “Cancel” option. 6. PetFinder clicks on the “Upload” button and uploads image(s) of the pet as an image attachment with a short optional description. 7. System displays a “Confirm” and a “Cancel” button. 8. PetFinder clicks on the “Confirm” button. Uploaded images will be sent to the Owner and he/she will be notified.
Alternative Flows:	AF-S5: PetFinder clicks “Cancel” button 1. If the PetFinder clicks the “Cancel” button, the report will be dismissed and images will not be sent to the Owner. 2. The PetFinder will be brought back to the selected LostPet Listing.
Exceptions:	None
Includes:	None
Special Requirements:	None
Assumptions:	The Pet found matches the image in the listing.
Notes and Issues:	None

4.2.4 Functional Requirement 4

4.2.4.1 SubmitAdoptionRequest

Use Case ID:	#4-1		
Use Case Name:	SubmitAdoptionRequest		
Created By:	Baskar Kayalvizhi Athithiya	Last Updated By:	Baskar Kayalvizhi Athithiya
Date Created:	11/02/2025	Date Last Updated:	11/02/25

Actor:	Adopter
Description:	Adopters can send an Adoption Request to Owners for pet(s) they are interested in.
Preconditions:	Adopter is logged in and authenticated.
Postconditions:	The Adoption Request is successfully sent to the Owner of the Pet
Priority:	High
Frequency of Use:	High
Flow of Events:	1. The System displays the details of the Pet that is up for adoption. 2. The System displays the “Adopt” button. 3. The Adopter clicks on the “Adopt” button. 4. System prompts the Adopter to fill in the following details: a. Age b. Gender

	c. Message to be sent to the Owner 5. System displays the “Confirm” and “Cancel” button. 6. The Adopter clicks on the “Confirm” button. 7. System sends the Adoption Request to the Owner of the Pet Listing.
Alternative Flows:	AF-S5: PetFinder clicks “Cancel” button 1. Adopter terminates the adoption process and the request is not sent to the Owner. 2. Adopter is brought back to the Pet Details page.
Exceptions:	None
Includes:	None
Special Requirements:	None
Assumptions:	None
Notes and Issues:	None

4.2.4.2 ViewSubmittedRequestList

Use Case ID:	#4-2		
Use Case Name:	ViewSubmittedRequestList		
Created By:	Baskar Kayalvizhi Athithiya	Last Updated By:	Baskar Kayalvizhi Athithiya
Date Created:	11/02/2025	Date Last Updated:	11/02/25

Actor:	Adopter
Description:	Adopters will be able to view all the adoption requests they have made for the pets that they are interested in
Preconditions:	1. The Adopter is logged in authenticated by the System 2. The Adopter must have previously made a successful adoption request to the Owner
Postconditions:	The Adopter will be able to view the list of adoption requests they have made.
Priority:	Medium
Frequency of Use:	Medium
Flow of Events:	1. System shows the list of all adoption requests that the Adopter had submitted. 2. Adopter is able to view individual adoption request status by clicking the “View Status” button in each listing.
Alternative Flows:	None
Exceptions:	None
Includes:	None
Special Requirements:	None
Assumptions:	None

Notes and Issues:	None

4.2.4.3 PersonalityTest

Use Case ID:	#4-3		
Use Case Name:	SubmitAdoptionRequest		
Created By:	Benjamin Yeoh	Last Updated By:	Benjamin Yeoh
Date Created:	08/02/2025	Date Last Updated:	08/02/25

Actor:	Adopter
Description:	Adopters can take a personality test to determine which pet is suitable for them.
Preconditions:	Adopter is logged in and authenticated.
Postconditions:	None
Priority:	Low
Frequency of Use:	Medium
Flow of Events:	1. System shall display a “Personality” button in the profile page. 2. If the Adopter selects the “Personality” button, he will be prompted to fill in a personality form with required fields.

	3. The System checks for all required fields to be filled up. 4. The form is sent to the respective Pet Owner for him to evaluate the Adpoter's suitability for adoption.
Alternative Flows:	AF-S3: Adopter did not fill up all required fields in the form 1. Adopter did not fill up all required fields in the form. 2. Adopter clicks the "Submit" button. 3. System checks that all the required fields are not filled up. 4. The form is not sent to the Owner, the System prompts the Adopter to fill up all required fields in the form.
Exceptions:	None
Includes:	None
Special Requirements:	None
Assumptions:	Adopter wishes to adopt a pet from a particular listing.
Notes and Issues:	None

4.2.4.4 SuggestBreed

Use Case ID:	#4-4		
Use Case Name:	SuggestBreed		
Created By:	Baskar Kayalvizhi Athithiya	Last Updated By:	Baskar Kayalvizhi Athithiya
Date Created:	16/04/2025	Date Last Updated:	16/04/25

Actor:	Adopter
Description:	The Adopter can get dog and cat breeds recommended for them based on the results of their personality test
Preconditions:	Adopter is logged in and authenticated.
Postconditions:	The breed of the recommended pet is saved to the Adopters saved pets tab.
Priority:	Medium
Frequency of Use:	Low
Flow of Events:	1. System shows “Cat” and “Dog” buttons. 2. System will then display the recommended breeds based on what the Adopter chooses. 3. The System will then display “Save Pet” under each recommended breed. 4. Upon clicking the “Save Pet” button the breed will then be saved to Adopters Saved Pets tab.
Alternative Flows:	None
Exceptions:	None
Includes:	None
Special Requirements:	None

Assumptions:	Adopter has already taken the personality test and 3 key words describing their personality has been generated.
Notes and Issues:	None

4.2.5 Functional Requirement 5

4.2.5.1 ReviewActivity

Use Case ID:	#5-1		
Use Case Name:	ReviewActivity		
Created By:	Benjamin Yeoh	Last Updated By:	Benjamin Yeoh
Date Created:	08/02/2025	Date Last Updated:	08/02/25

Actor:	Administrator
Description:	Administrators will be brought to the administrator page where he/she can perform actions that can only be completed by administrators.
Preconditions:	Administrator is logged in and authenticated.
Postconditions:	Affected Users will be notified.
Priority:	High
Frequency of Use:	High
Flow of Events:	1. System displays the list of all of the reports submitted by users. 2. Administrator selects an activity and views the details of the submitted report. 3. System prompts the Administrator to select the desired activity: DeleteListing, BanUser, DismissReport.

	4. If the Administrator selects the activity DeleteListing, then the Administrator uses the included use case DeleteListing to delete the selected listing. 5. If the Administrator selects the activity BanUser, then the Administrator uses the included use case BanUser to ban the selected user. 6. If the Administrator selects the activity DismissReport, then the Administrator will be directed to the included use case DismissReport to delete the report.
Alternative Flows:	None
Exceptions:	None
Includes:	1. DeleteListing 2. BanUser 3. DismissReport
Special Requirements:	None
Assumptions:	None
Notes and Issues:	None

4.2.5.2 BanUser

Use Case ID:	#5-2		
Use Case Name:	BanUser		
Created By:	Benjamin Yeoh	Last Updated By:	Benjamin Yeoh

Date Created:	08/02/2025	Date Last Updated:	08/02/25

Actor:	Administrator
Description:	Administrators can ban Users deemed inappropriate.
Preconditions:	Administrator is logged in and authenticated.
Postconditions:	Banned Users will be notified.
Priority:	High
Frequency of Use:	High
Flow of Events:	1. System displays a “Ban User” button. 2. Administrator clicks on the “Ban User” button. 3. System shall display a “Confirm” and a “Cancel” button. 4. Upon clicking the “Confirm” button, the Administrator bans the selected User. 5. System disables the user’s account. 6. The banned user is notified of the ban.
Alternative Flows:	AF-S3: Administrator clicks the “Cancel” button 1. If the Administrator clicks the “Cancel” button, the System will return to step 1.
Exceptions:	None

Includes:	None
Special Requirements:	None
Assumptions:	None
Notes and Issues:	None

4.2.5.3 DeleteListing

Use Case ID:	#5-3		
Use Case Name:	DeleteListing		
Created By:	Benjamin Yeoh	Last Updated By:	Benjamin Yeoh
Date Created:	08/02/2025	Date Last Updated:	08/02/25

Actor:	Administrator
Description:	Administrator can remove listing that he/she deems inappropriate
Preconditions:	If the actor is an Administrator, the administrator must be logged in and verified.
Postconditions:	Listing is deleted

Priority:	High
Frequency of Use:	High
Flow of Events:	1. User selects a the listing that he/she wants to delete 2. User clicks the kebab menu and selects the “Delete Listing” icon. 3. System prompts Administrator’s confirmation and deletes the listing. 4. Owner of the listing is notified of the deletion. 5. Listing is removed from the database.
Alternative Flows:	AF-S3: Administrator cancels the confirmation: 1. Administrator selects the “Cancel” icon when prompted for deletion confirmation. 2. Administrator is brought back to the interface list of submitted reports.
Exceptions:	None
Includes:	None
Special Requirements:	None
Assumptions:	None
Notes and Issues:	None

4.2.5.4 DismissReport

Use Case ID:	#5-4		
Use Case Name:	DismissReport		
Created By:	Benjamin Yeoh	Last Updated By:	Benjamin Yeoh
Date Created:	08/02/2025	Date Last Updated:	08/02/25

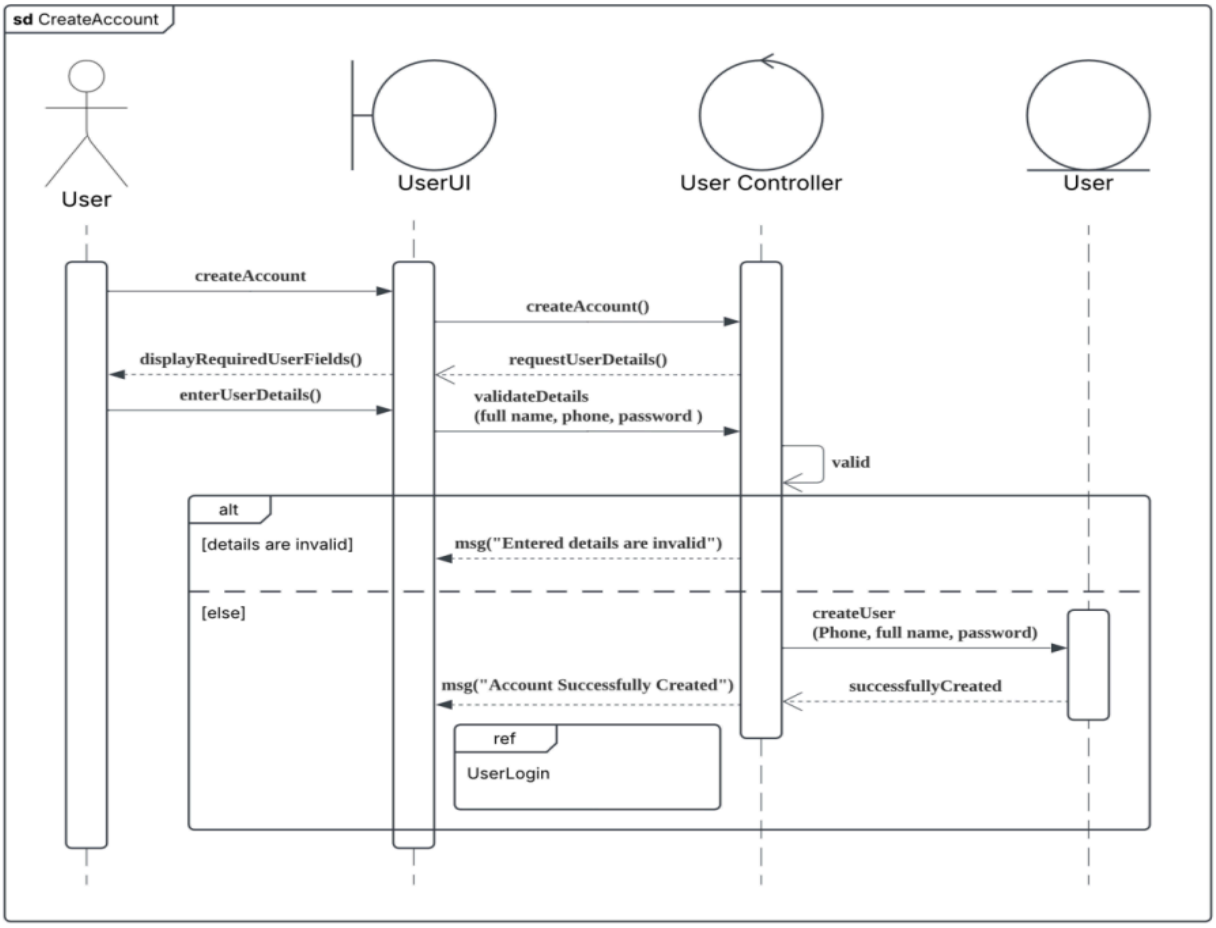
Actor:	Administrator
Description:	Administrator can remove reports that he/she deems illogical
Preconditions:	Administrator is logged in and authenticated.
Postconditions:	Report is deleted
Priority:	High
Frequency of Use:	High
Flow of Events:	1. System displays a “Dismiss Report” button. 2. Administrator clicks on the “Dismiss Report” button. 3. System shall display a “Confirm” and a “Cancel” button. 4. Upon clicking the “Confirm” button, the Administrator removes the selected report. 5. Report is removed from the database.
Alternative Flows:	AF-S3: Administrator cancels the confirmation:

	1. Administrator selects the “Cancel” icon when prompted for deletion confirmation. 2. Administrator is brought back to the interface list of submitted reports.
Exceptions:	None
Includes:	None
Special Requirements:	None
Assumptions:	None

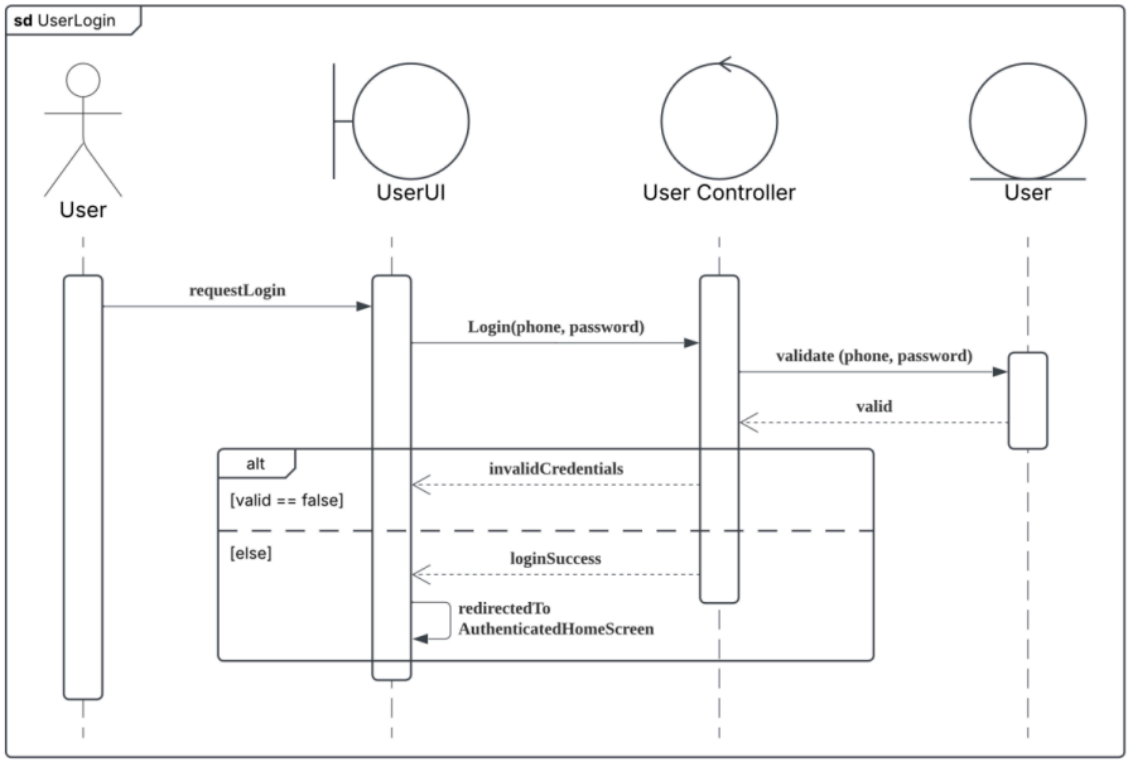
4.3 Sequence Diagrams for Use Cases

4.3.1 For Use Cases under #1

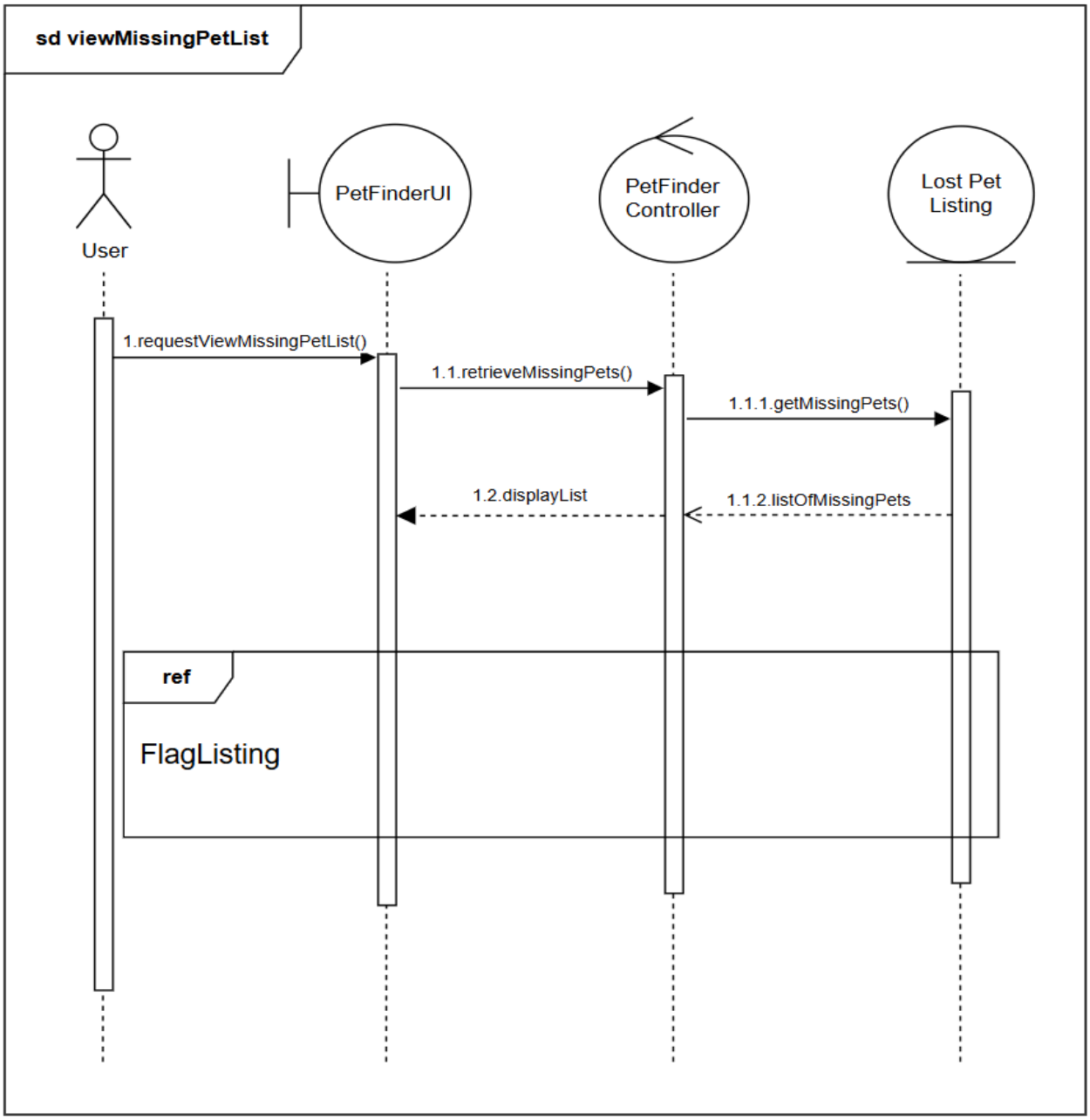
4.3.1.1 CreateAccount



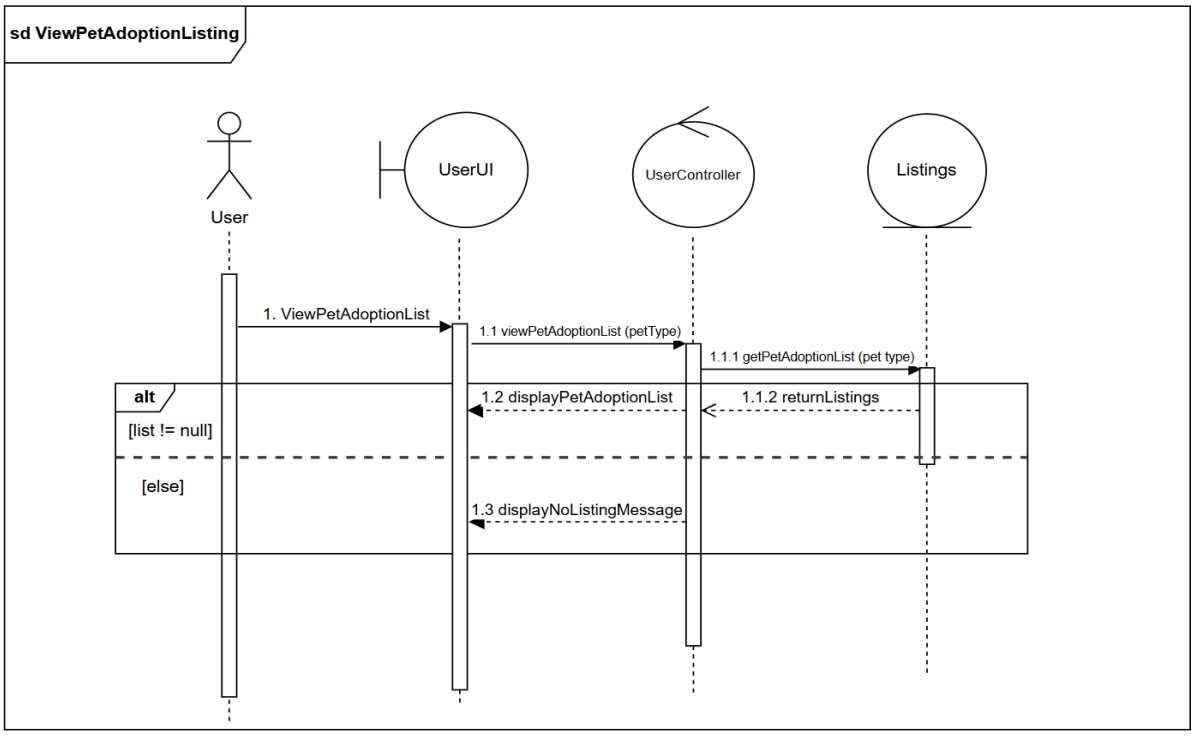
4.3.1.2 Login


sd UserLogin
User
UserUI
User Controller
User
requestLogin
Login(phone, password)
validate (phone, password)
valid
alt
[valid == false]
invalidCredentials
[else]
loginSuccess
redirectedTo
AuthenticatedHomeScreen

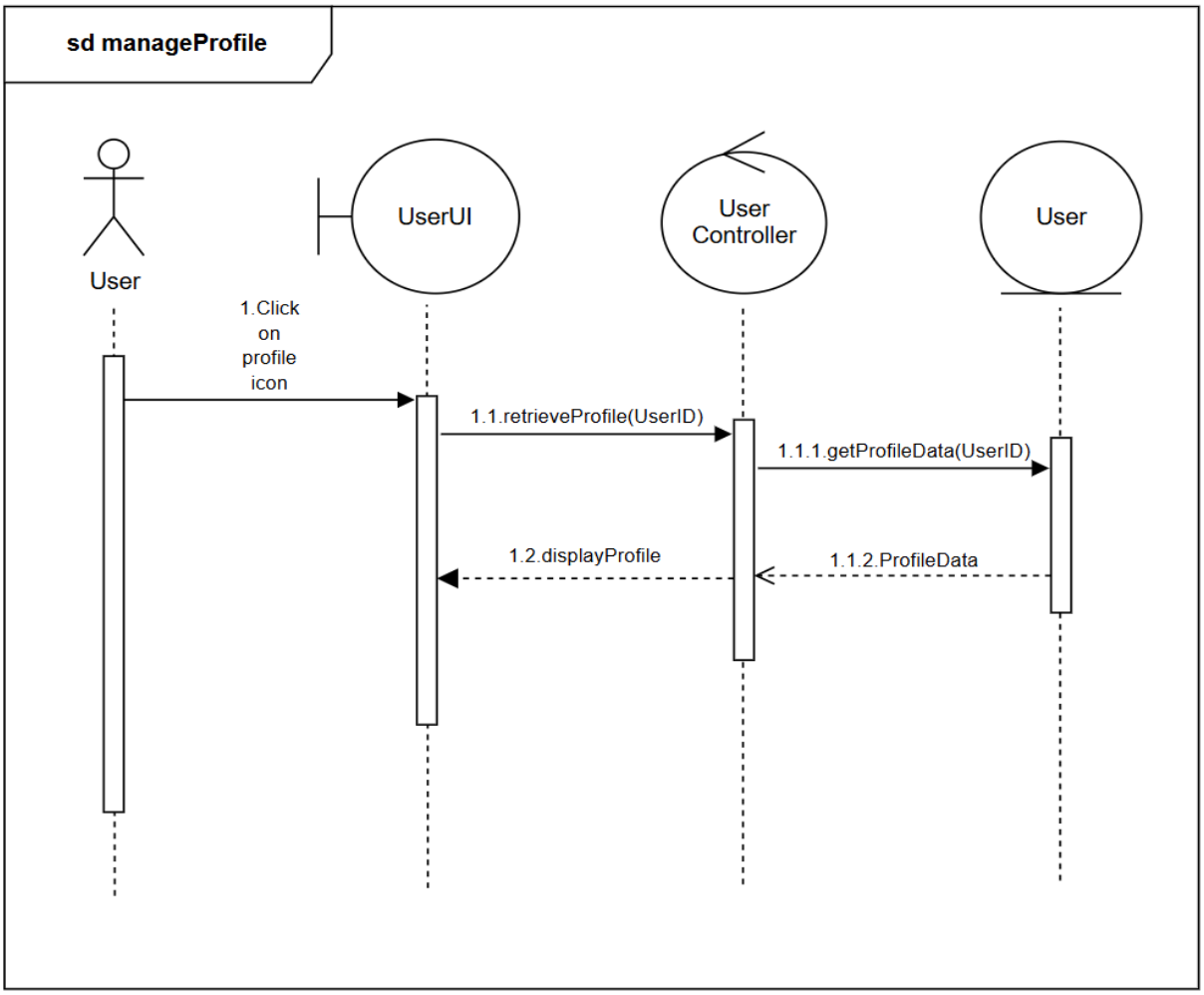
4.3.1.3 ViewMissingPetList



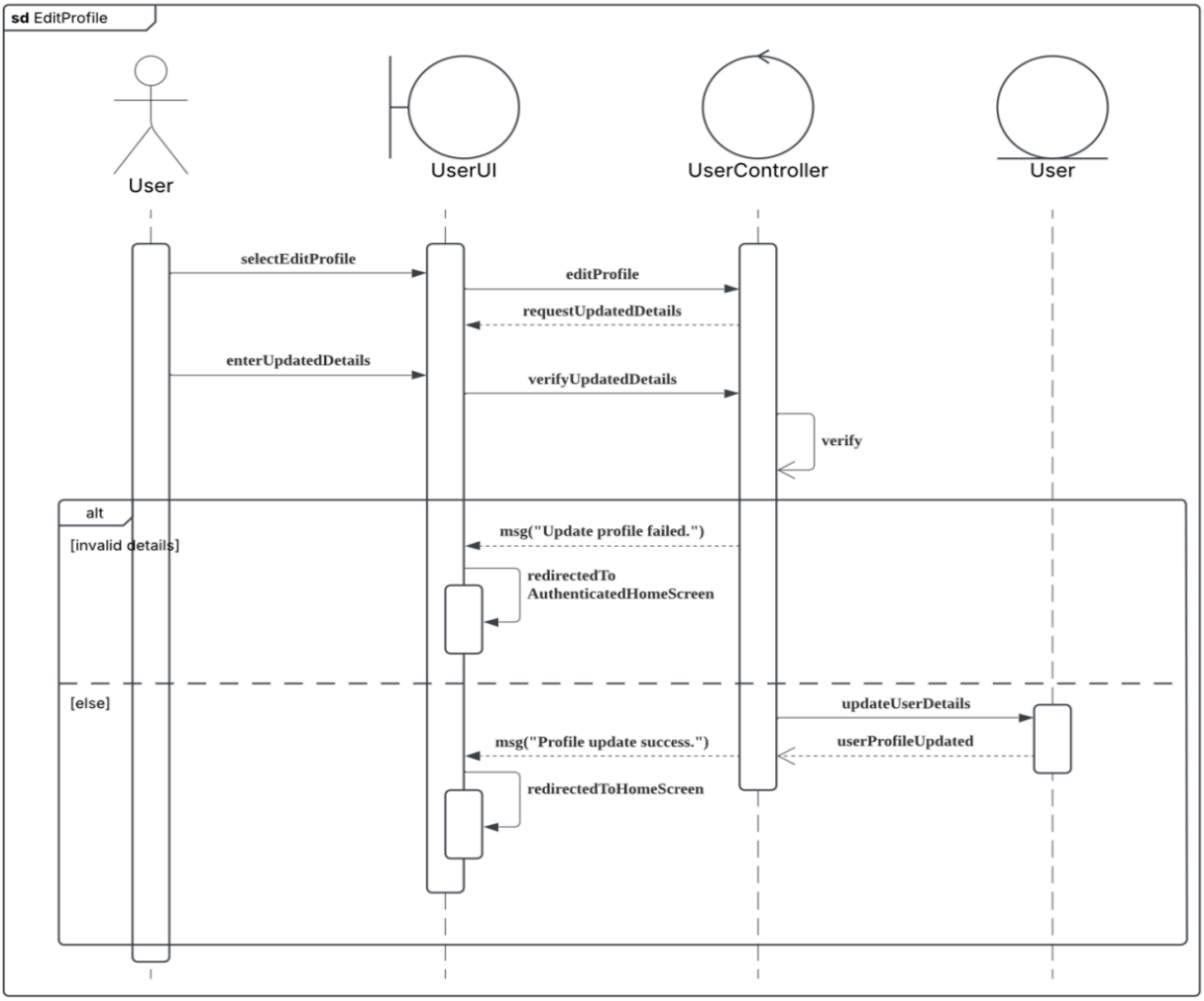
4.3.1.4 ViewPetAdoptionListing



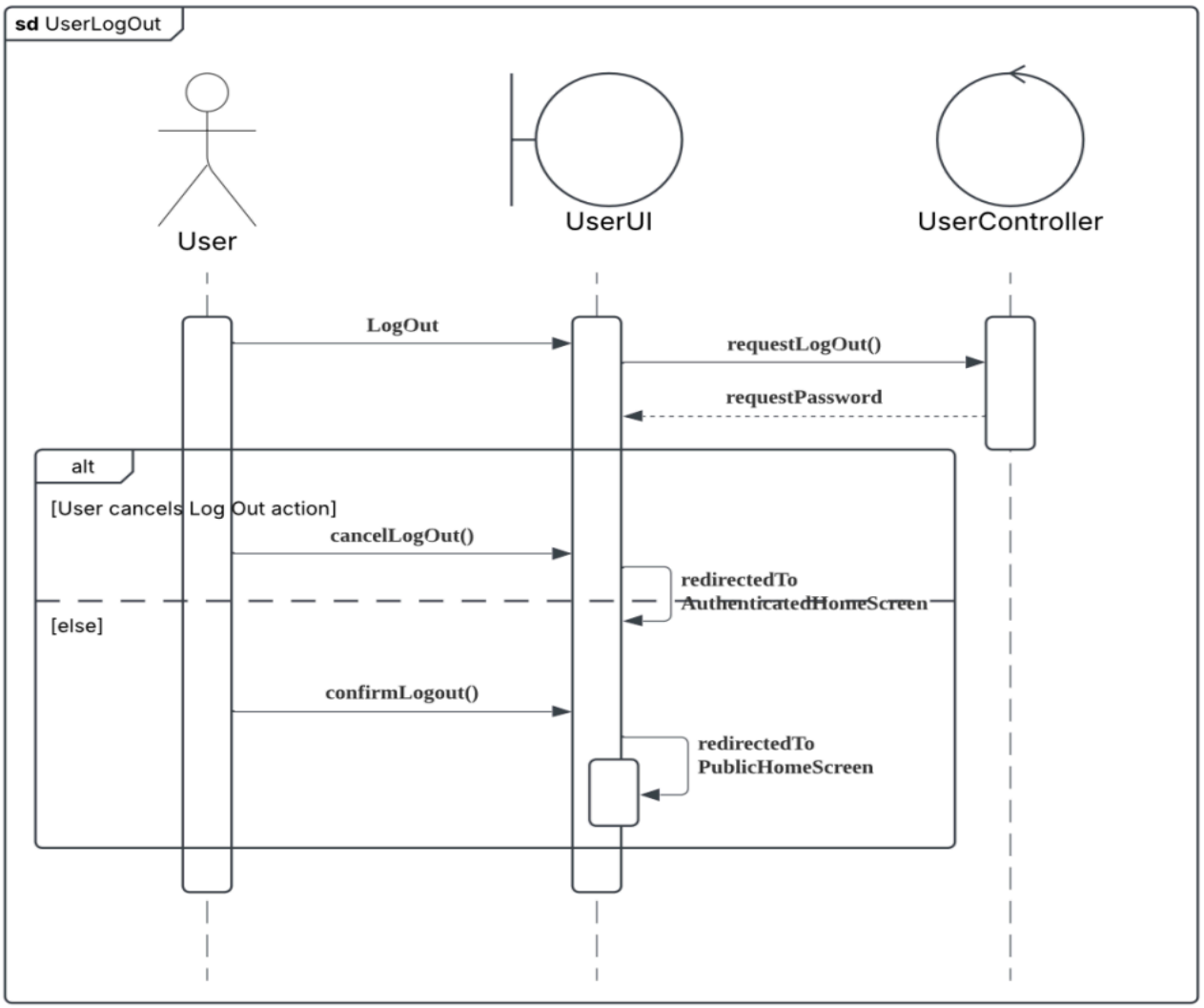
4.3.1.5 ManageProfile



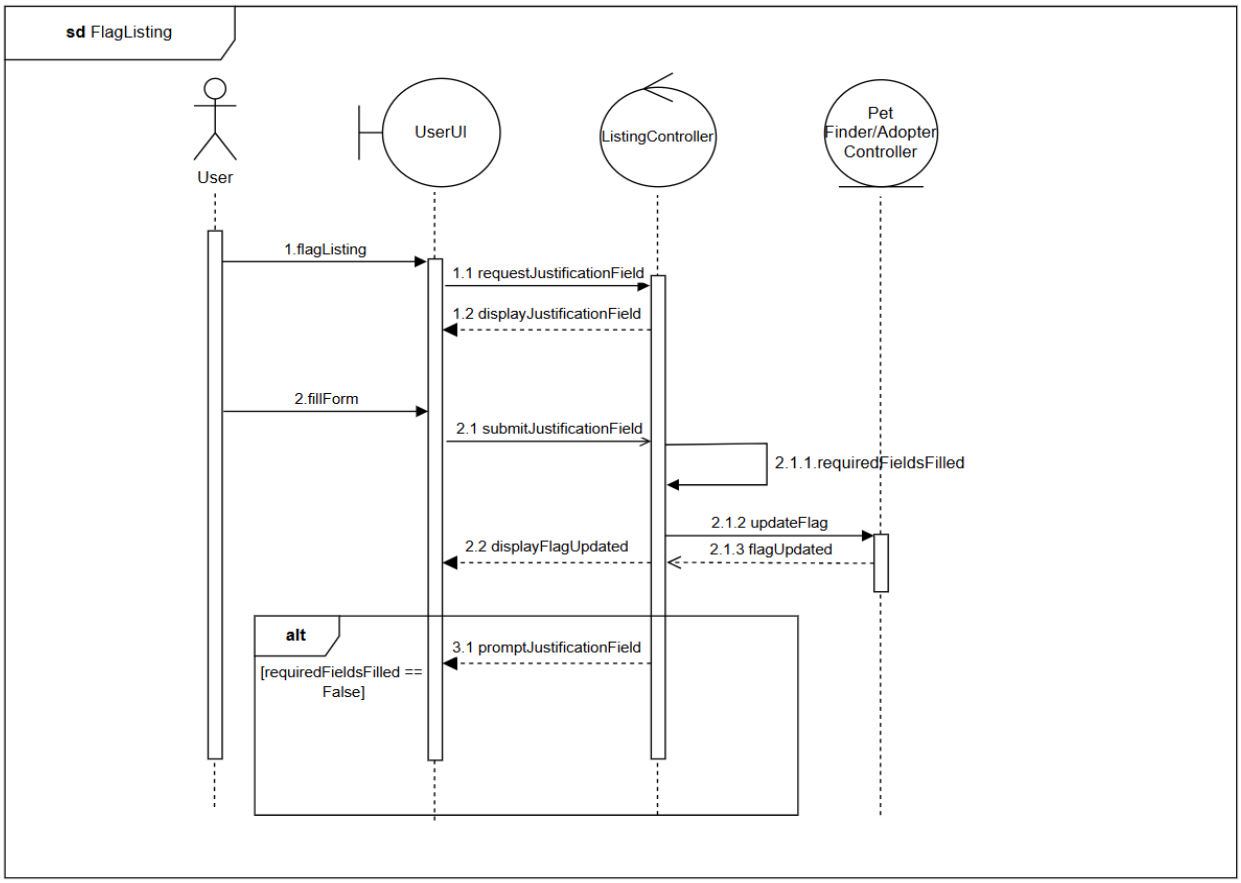
4.3.1.6 EditProfile



4.3.1.7 LogOut

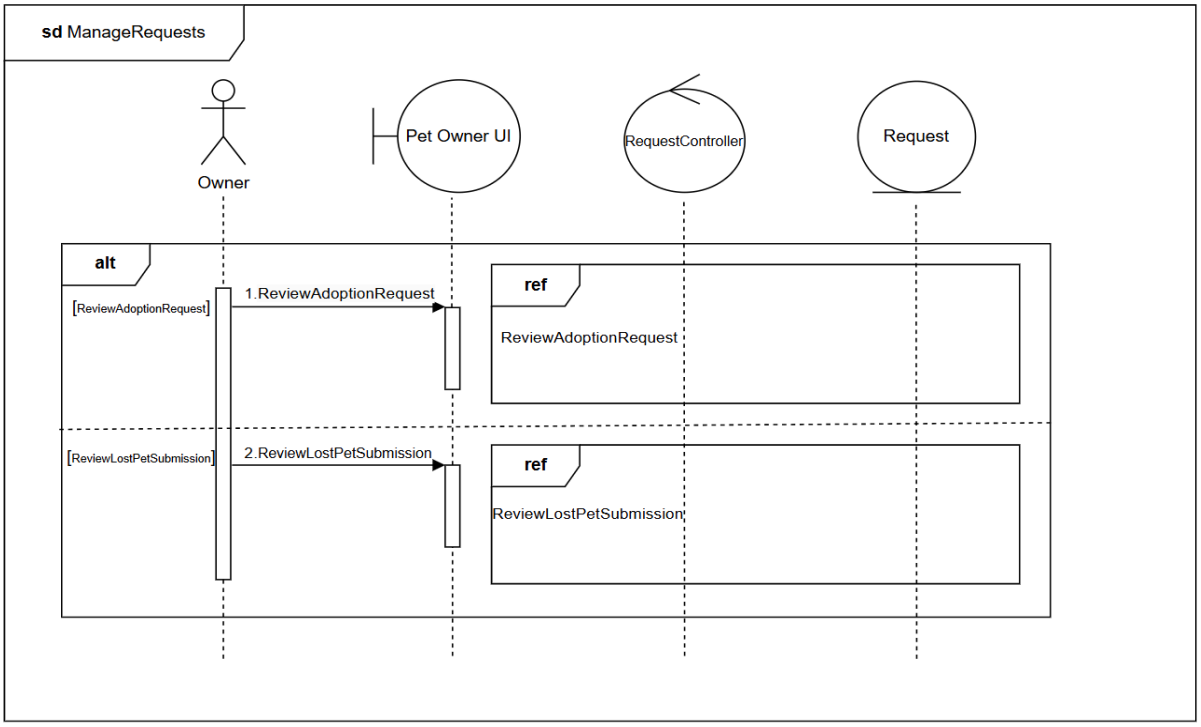

sd UserLogOut
User
UserUI
UserController
LogOut
requestLogOut()
requestPassword
alt
[User cancels Log Out action]
cancelLogOut()
redirectedTo
AuthenticatedHomeScreen
[else]
confirmLogout()
redirectedTo
PublicHomeScreen

4.3.1.8 FlagListing

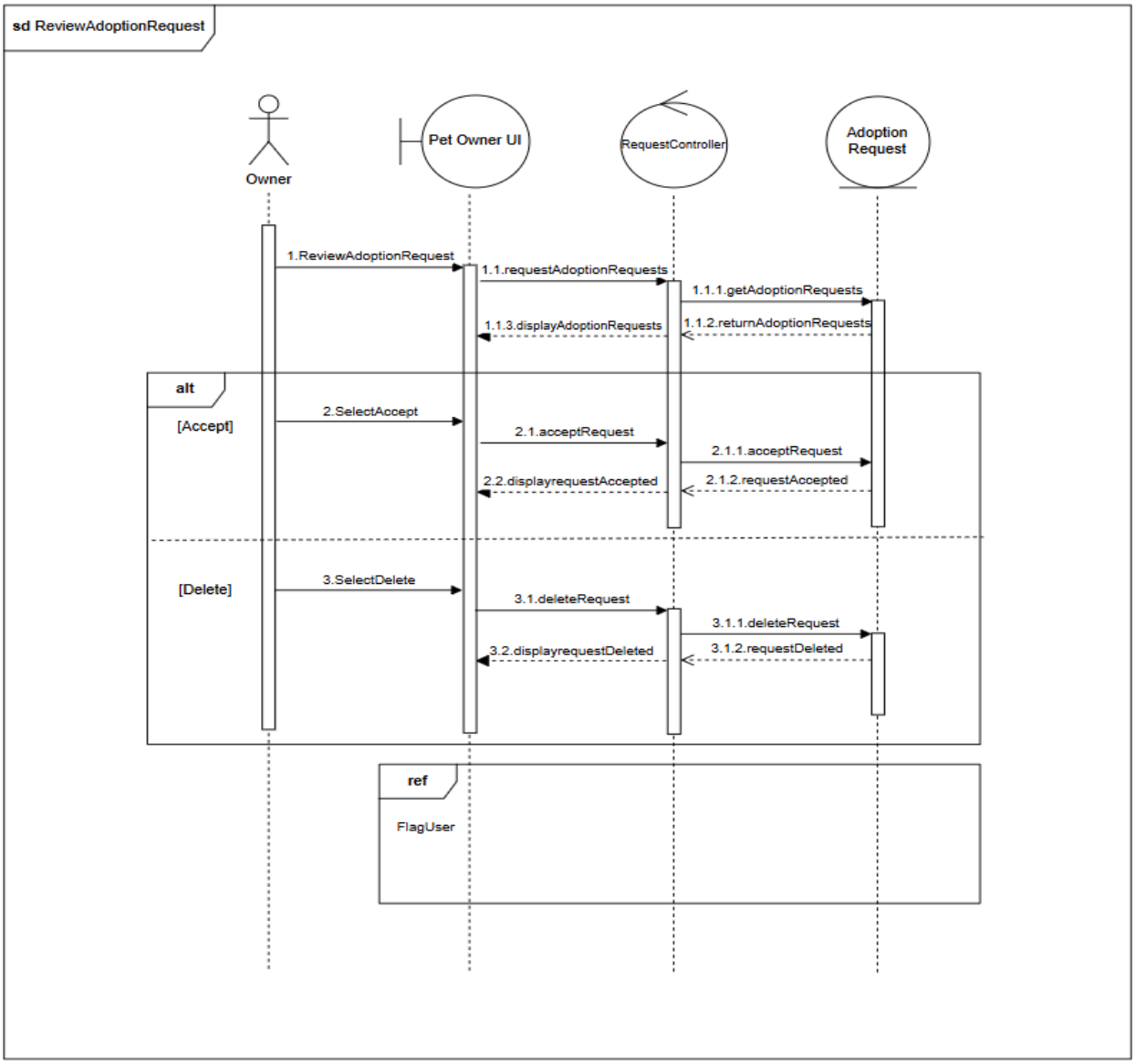


4.3.2 For Use Cases under #2

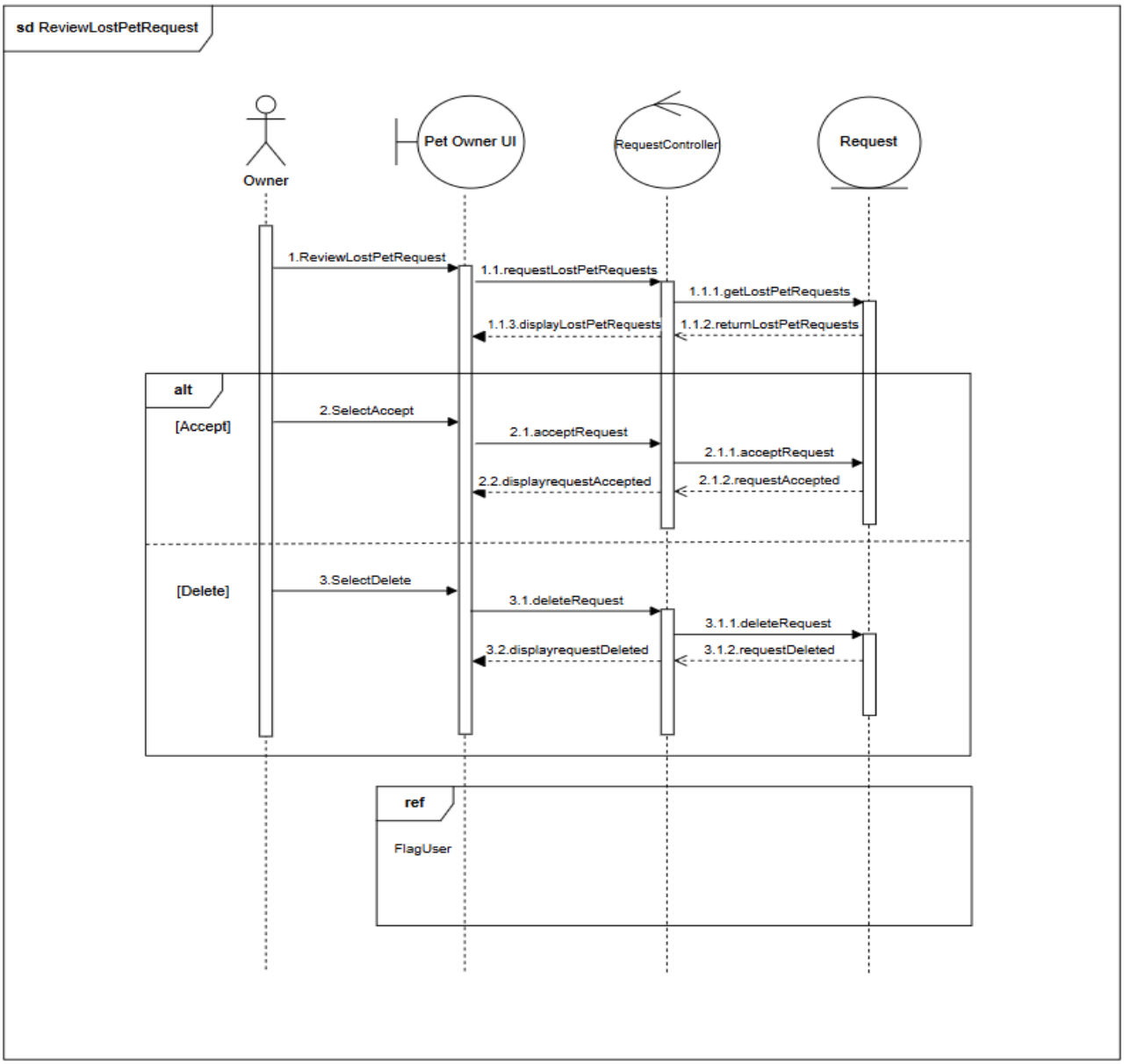
4.3.2.1 ManageRequests



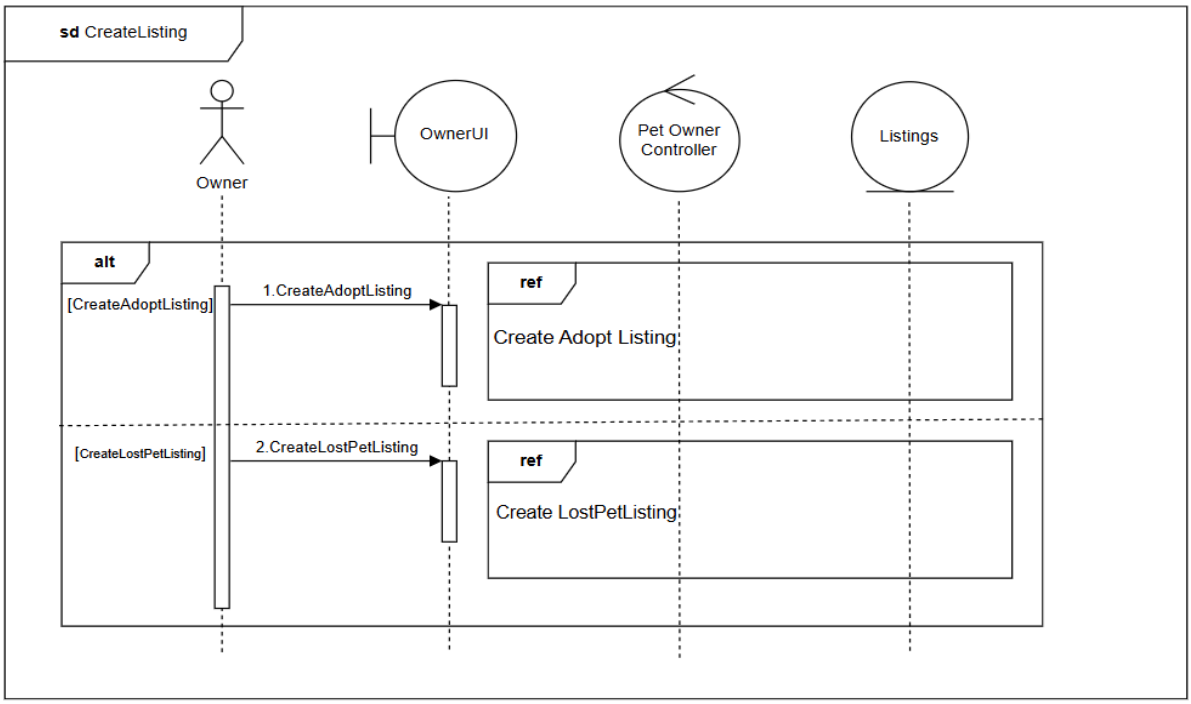
4.3.2.2 ReviewAdoptionRequest



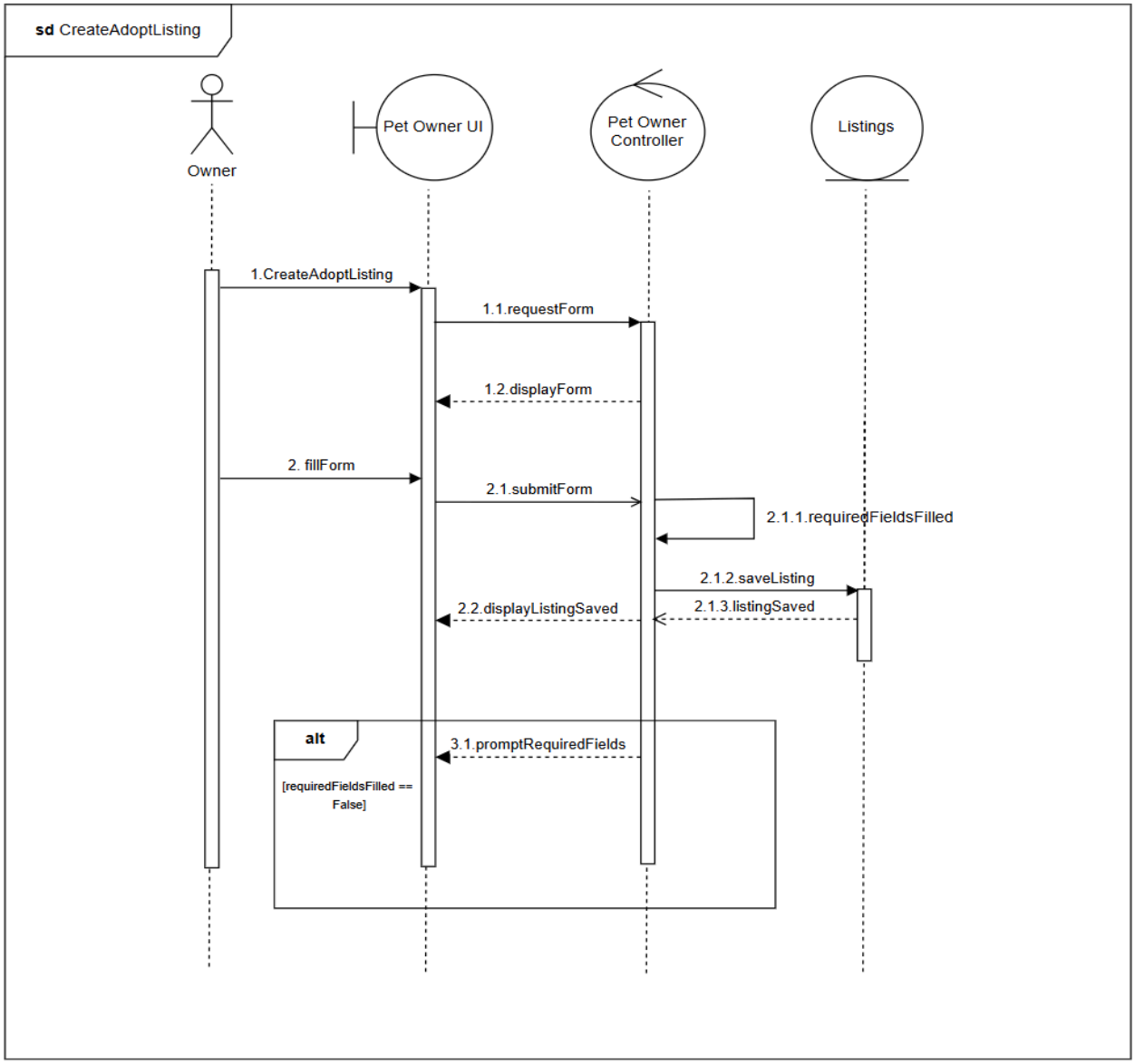
4.3.2.3 ReviewLostPetRequest



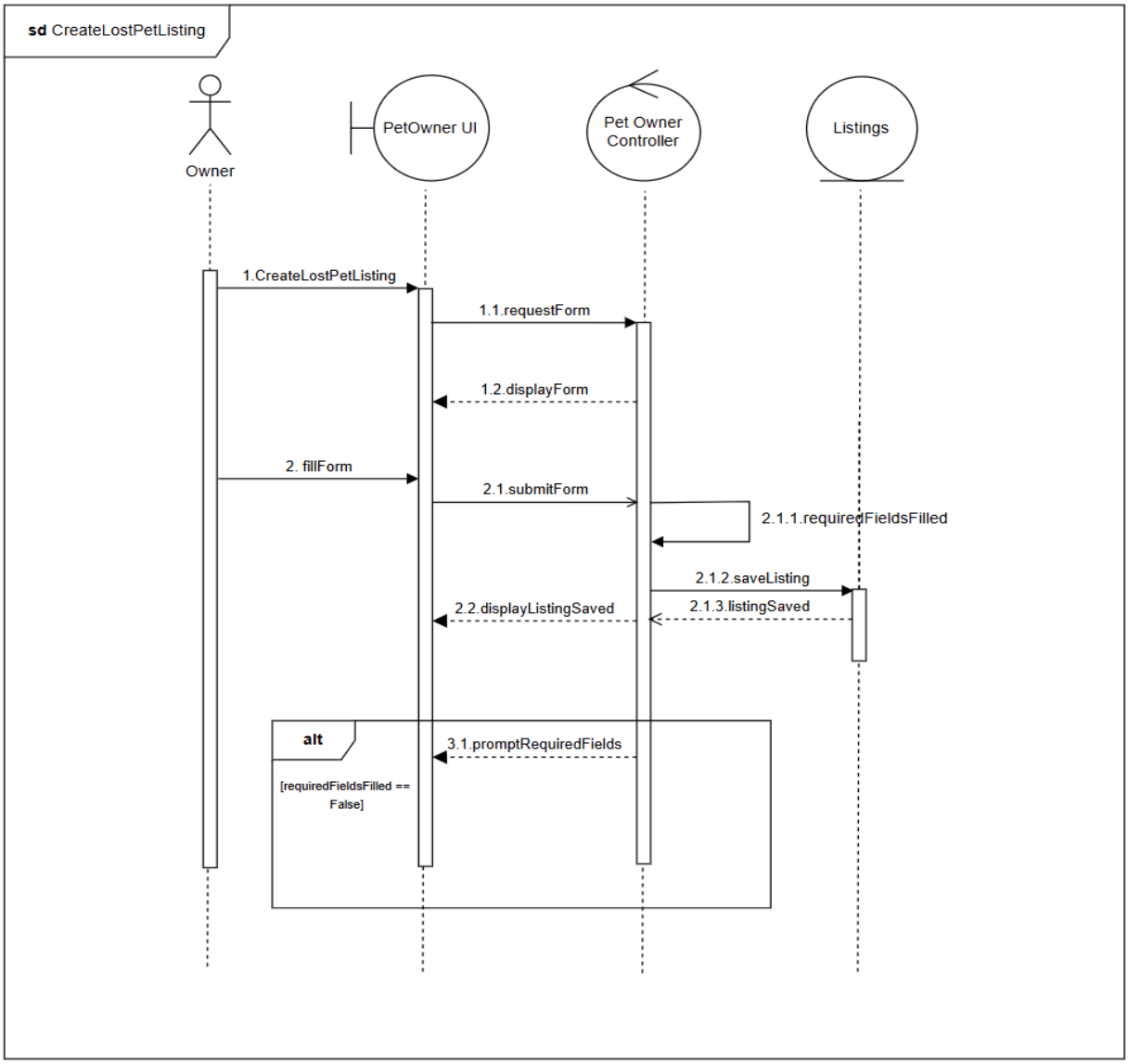
4.3.2.4 CreateListing



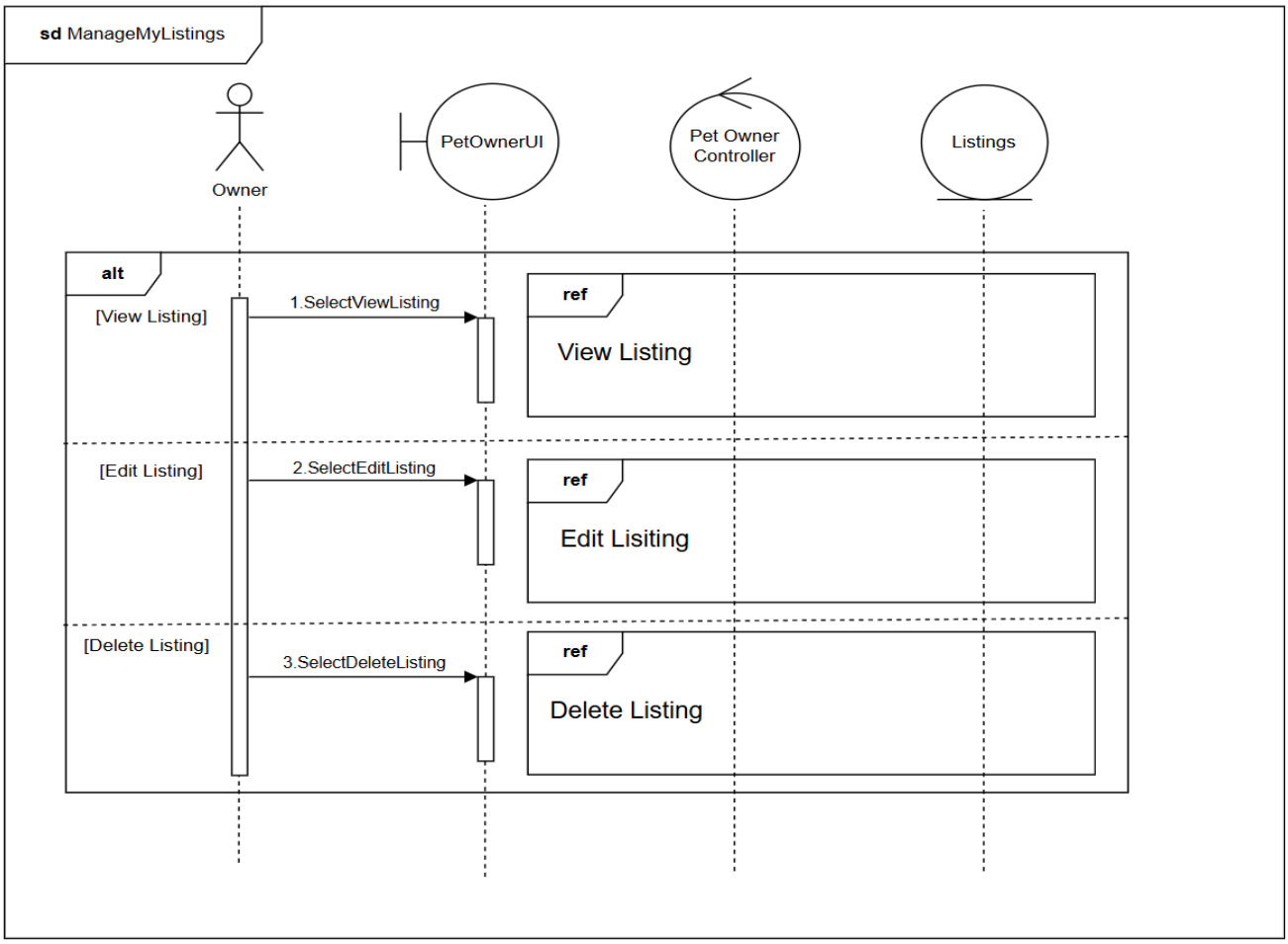
4.3.2.5 CreateAdoptListing



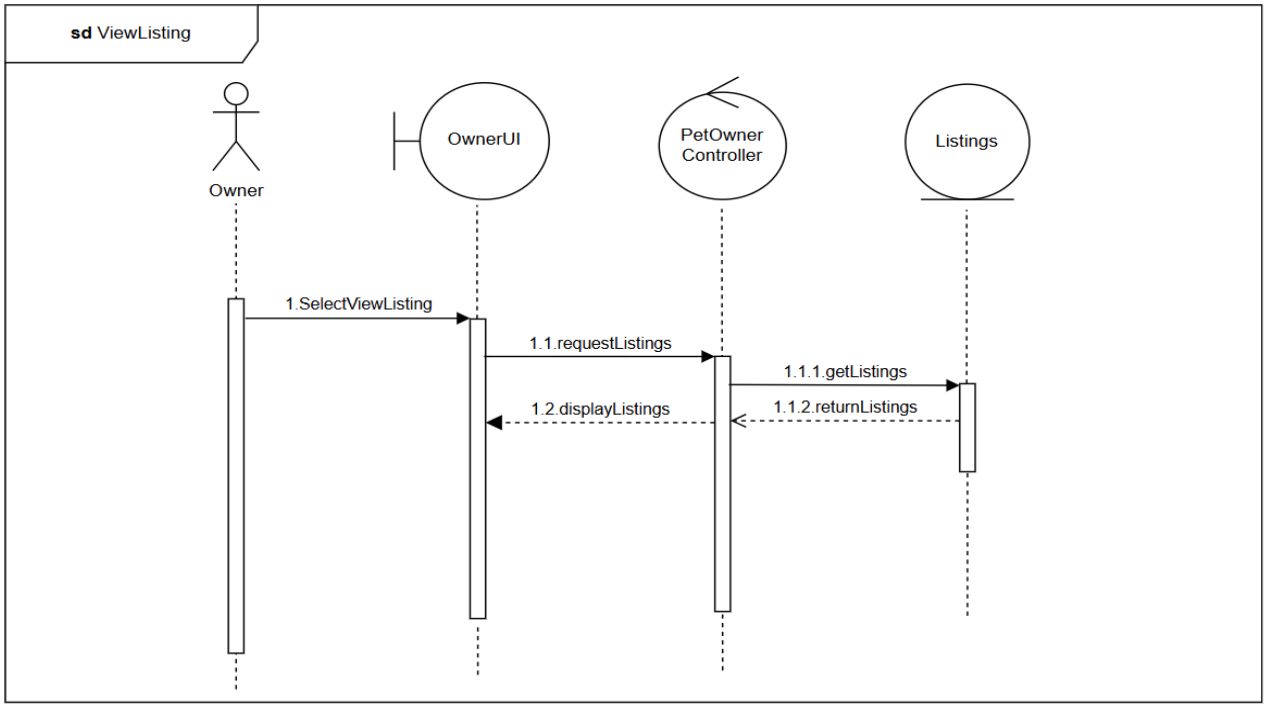
4.3.2.6 CreateLostPetListing



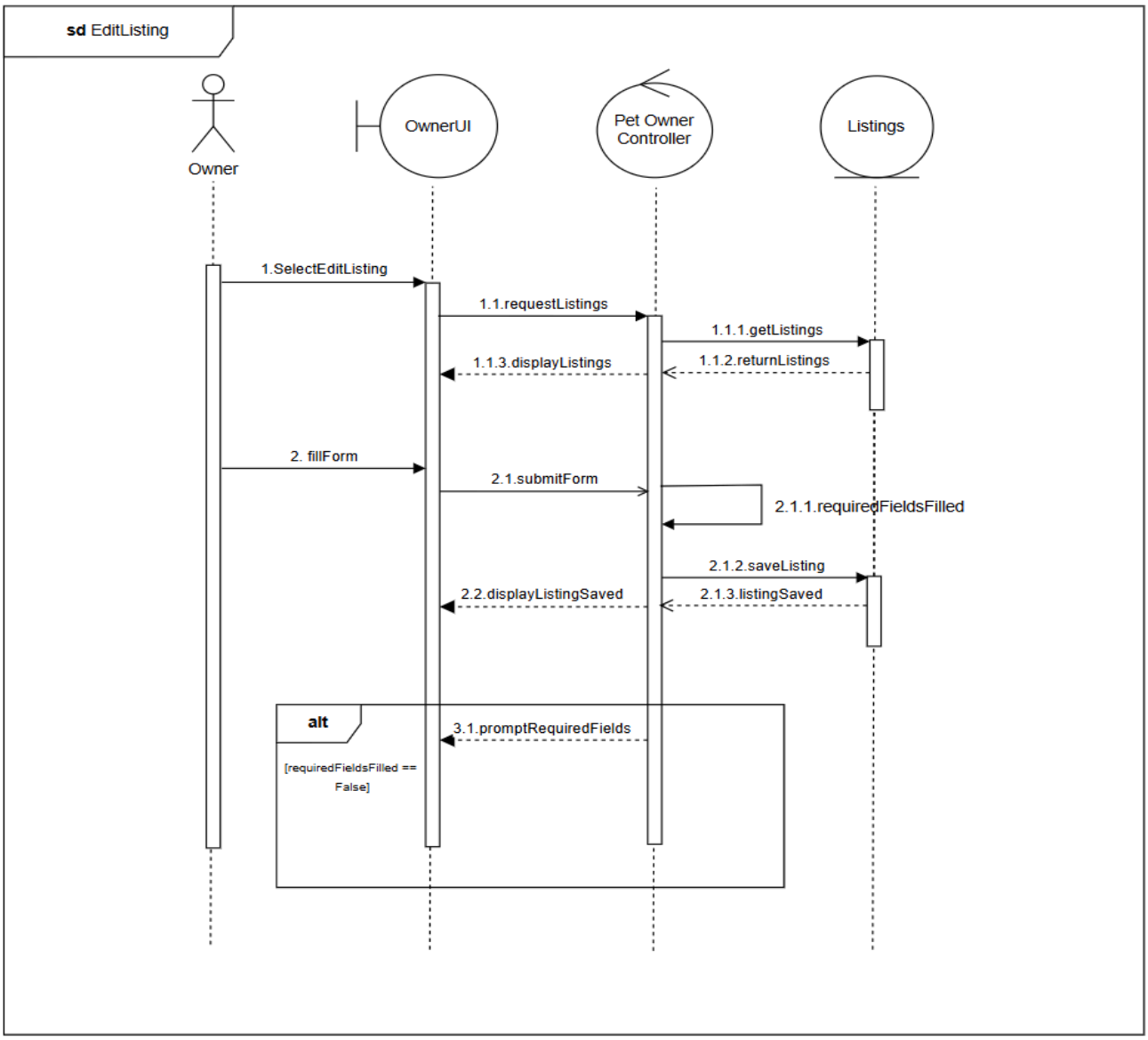
4.3.2.7 ManageMyListings



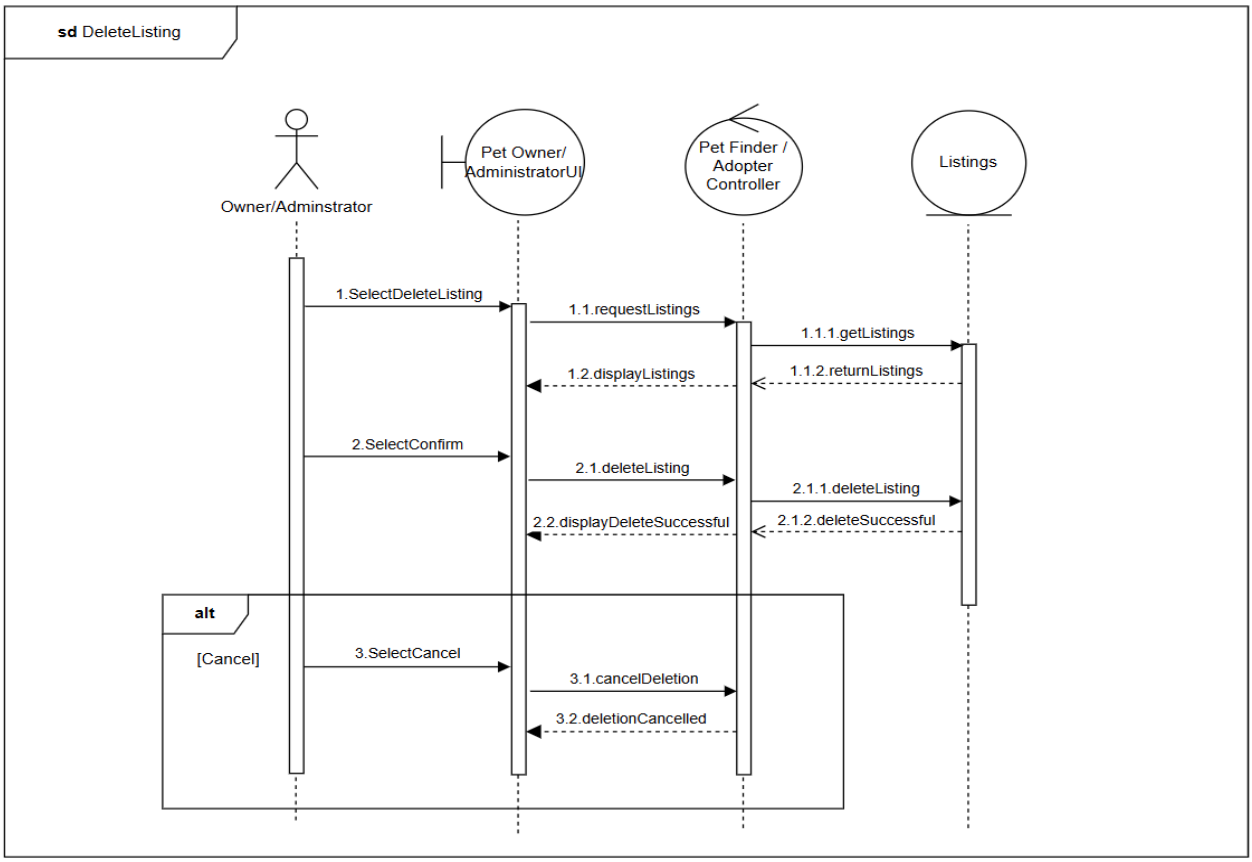
4.3.2.8 ViewListing



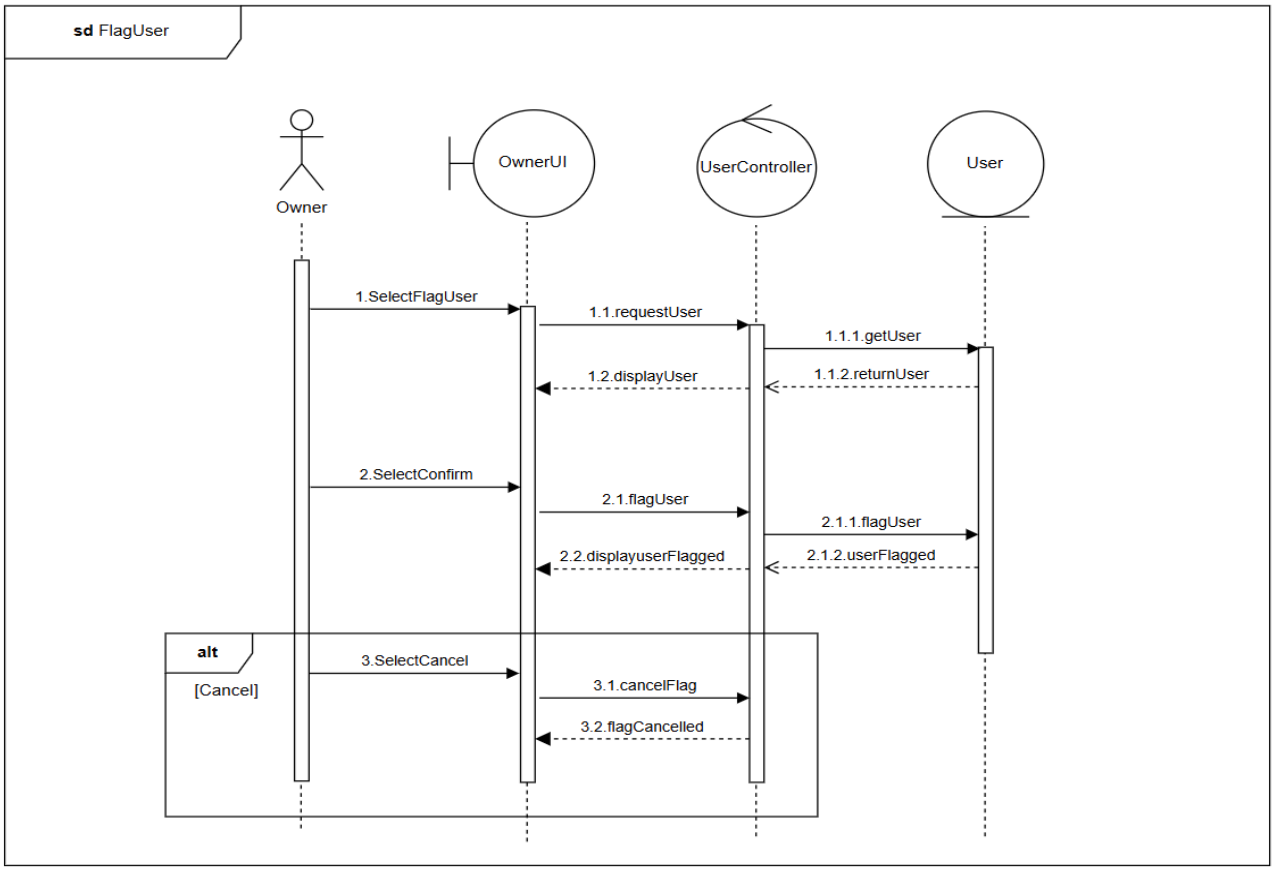
4.3.2.9 EditListing



4.3.2.10 DeleteListing

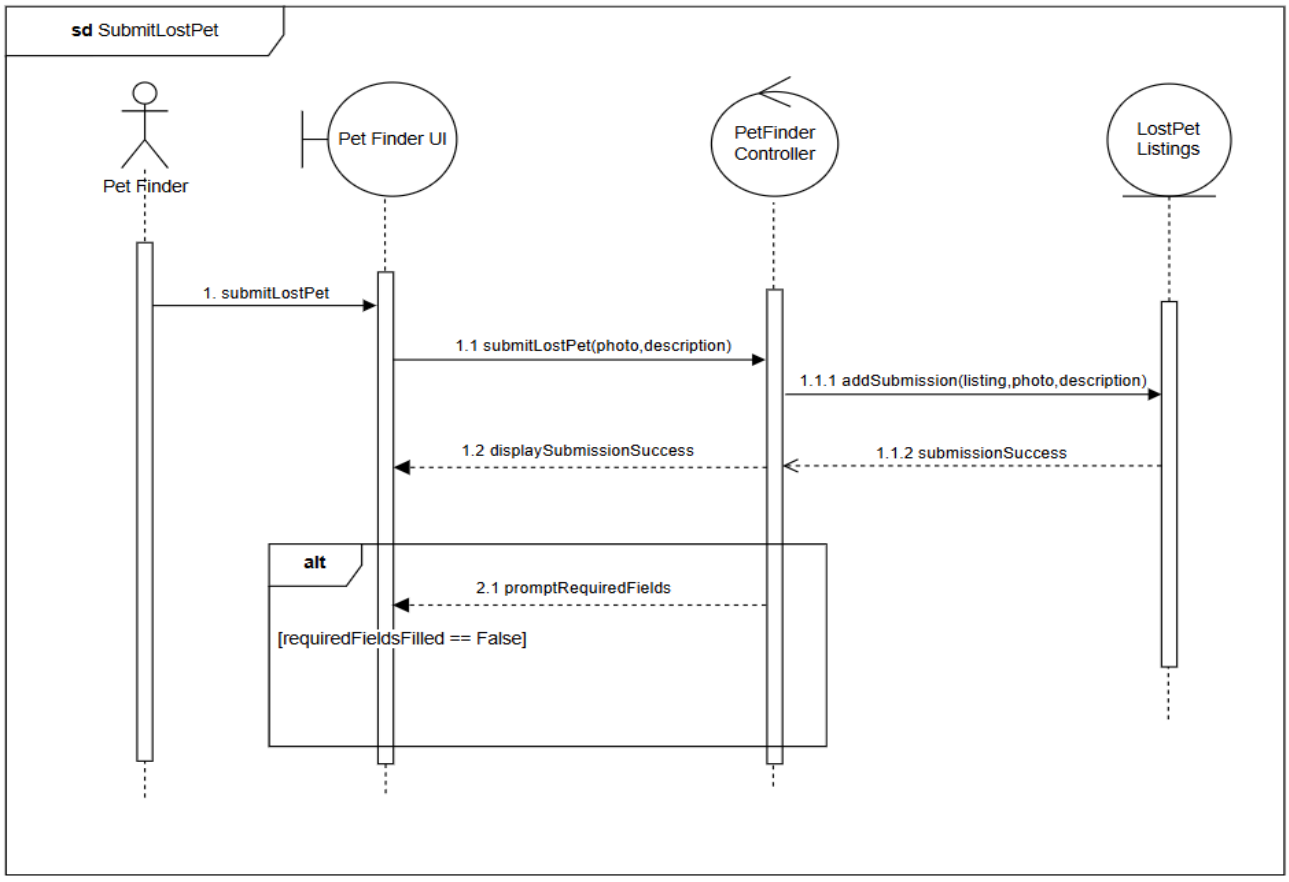


4.3.2.11 FlagUser

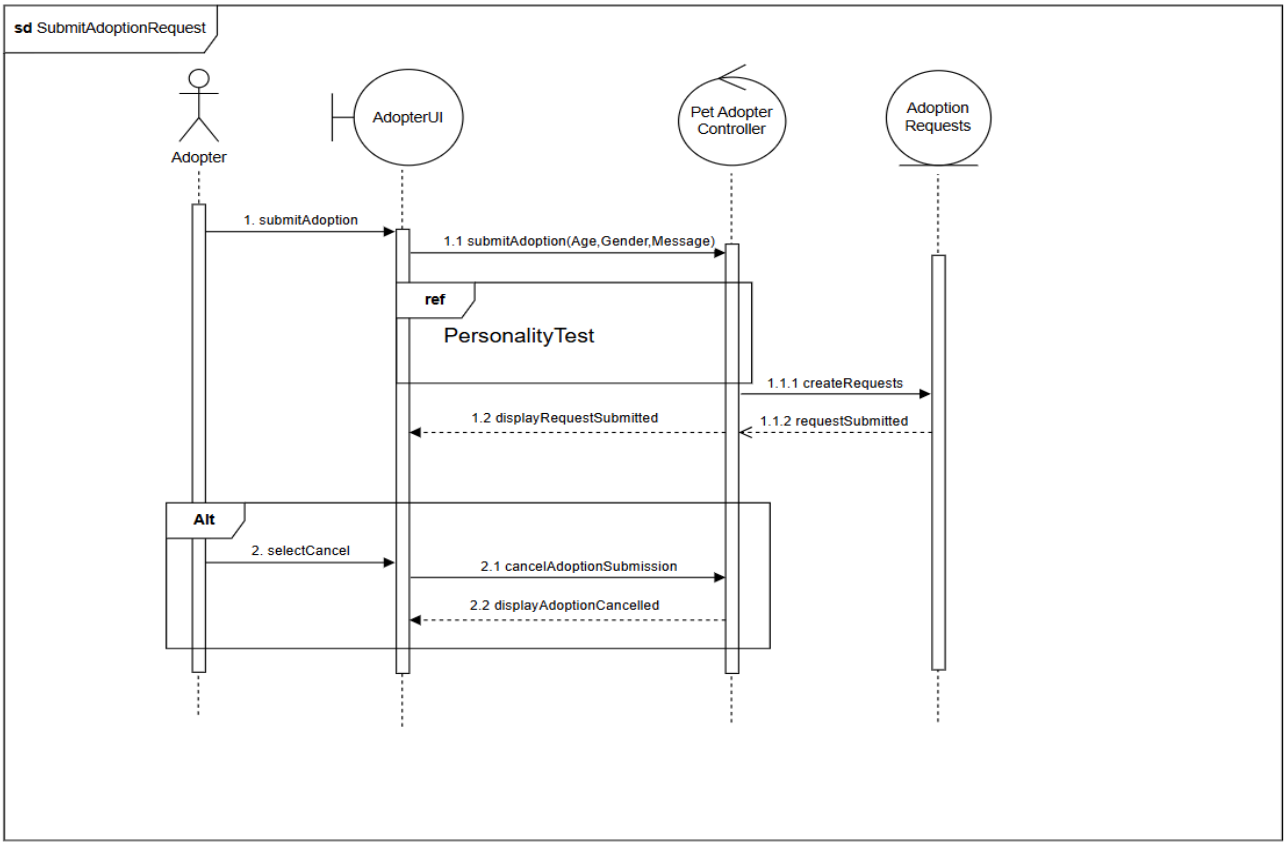


4.3.3 For Use Cases under #3

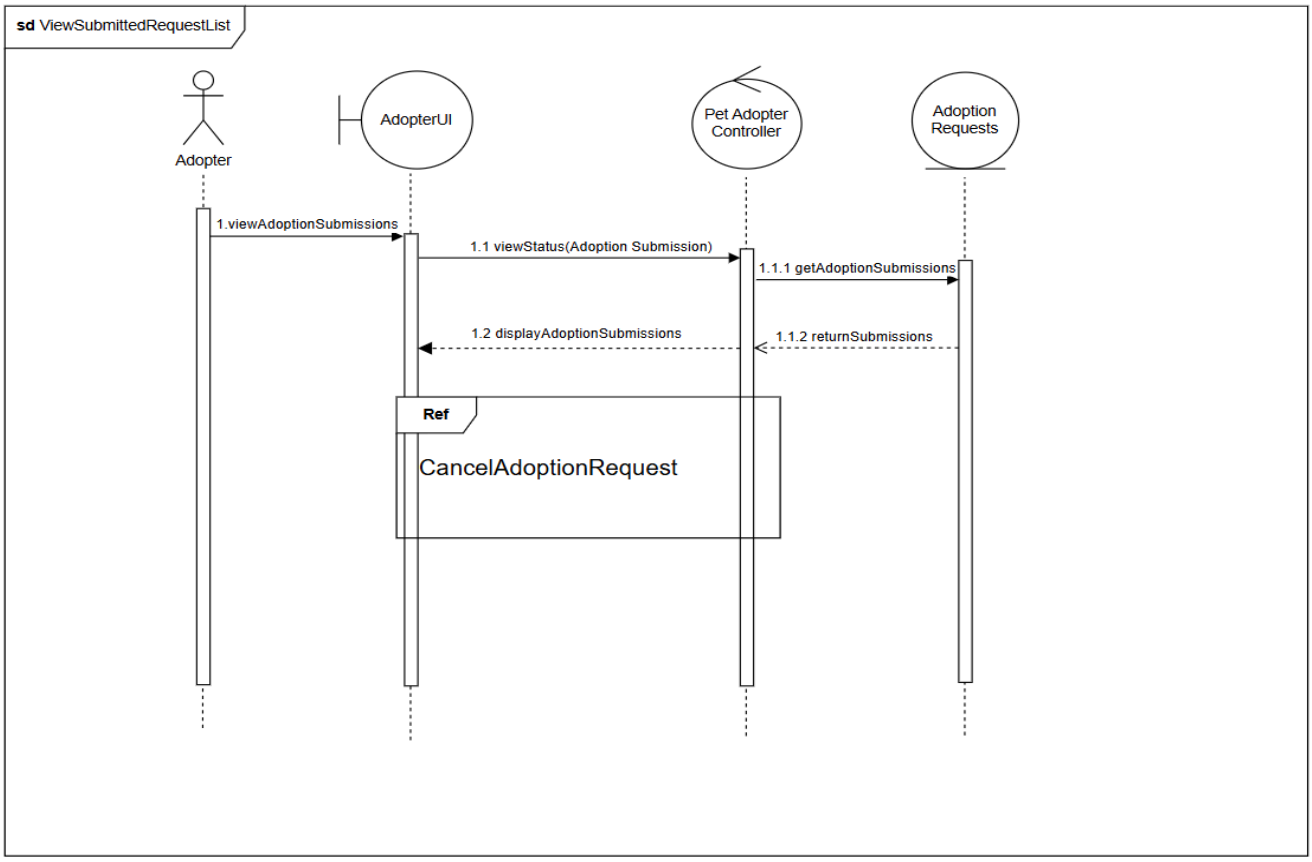
4.3.3.1 SubmitLostPet



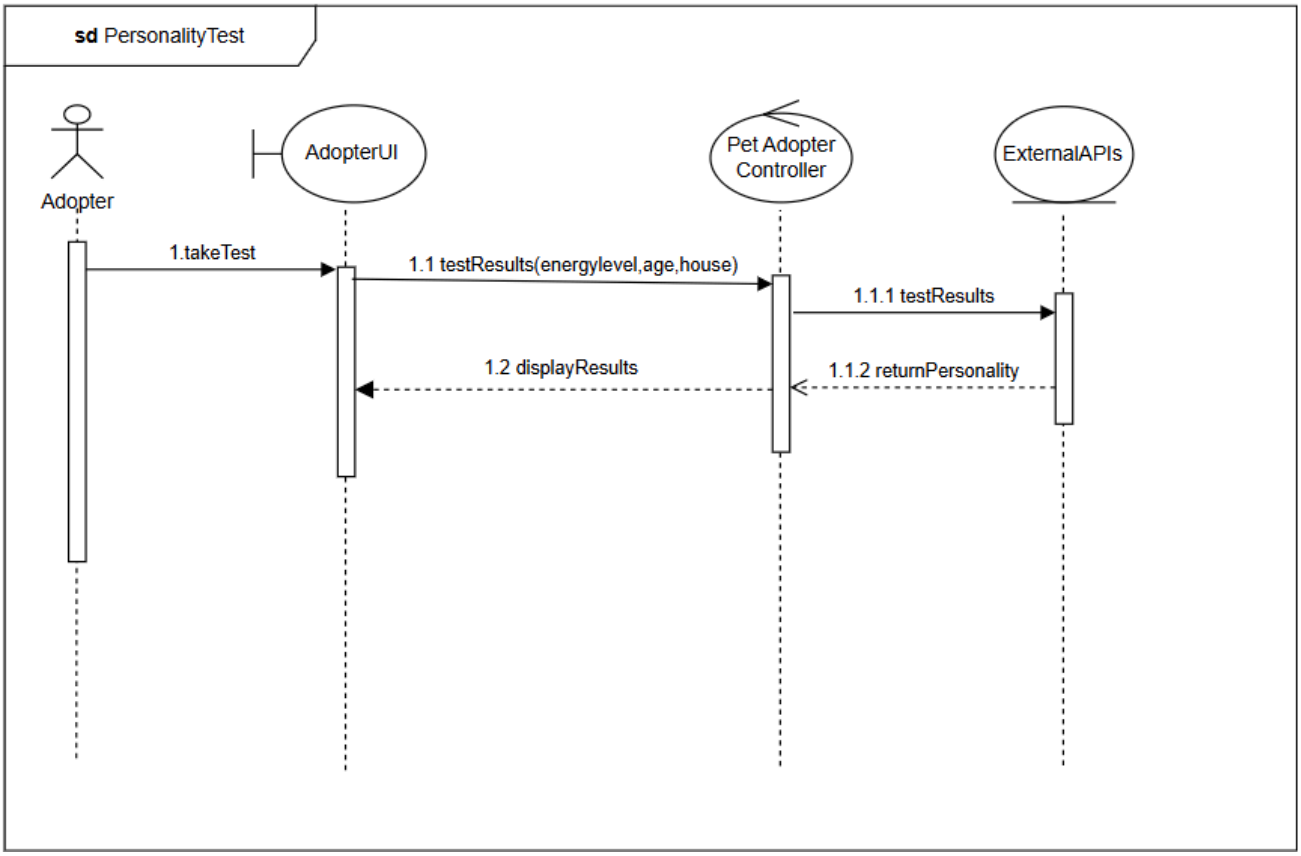
4.3.3.2 SubmitAdoptionRequest



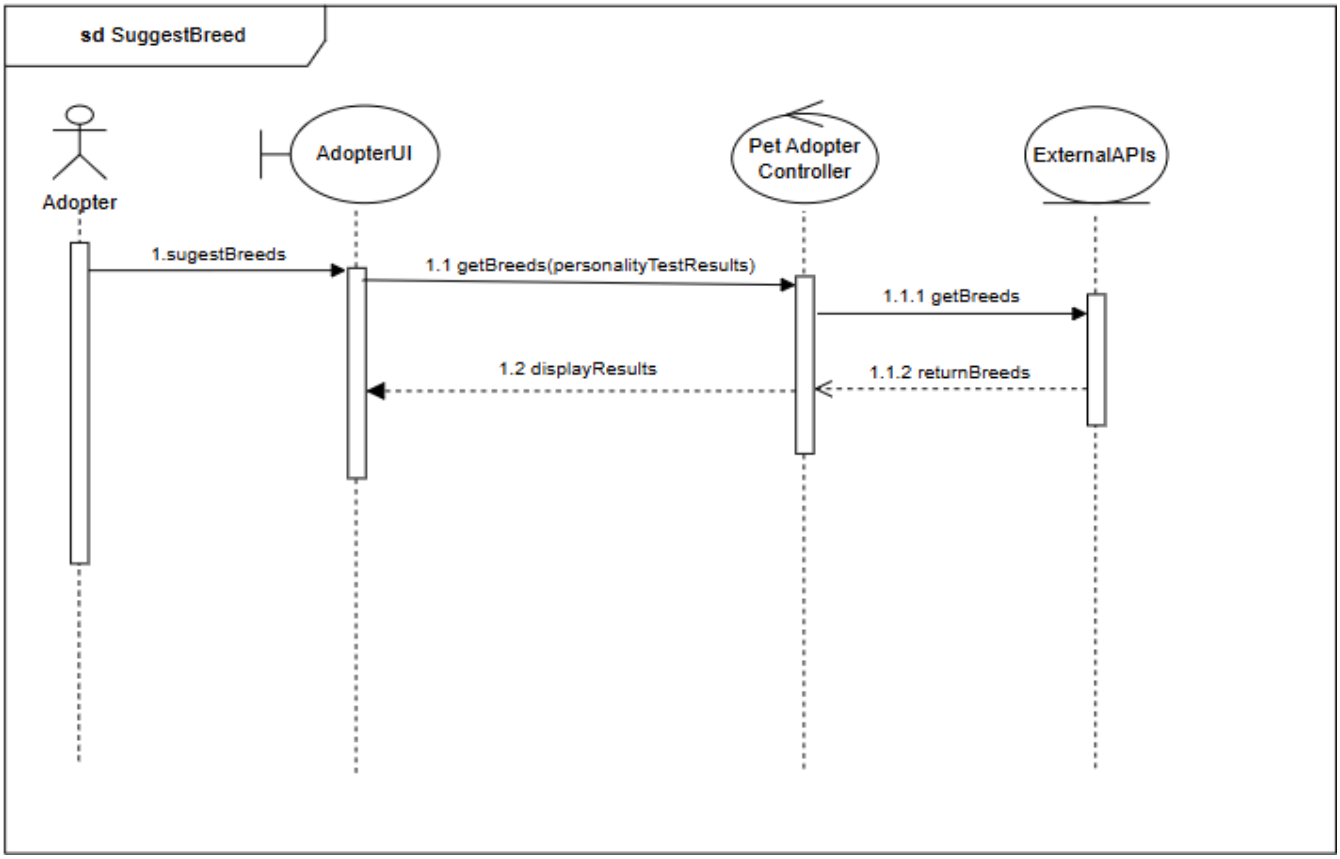
4.3.3.3 ViewSubmittedRequestList



4.3.3.4 PersonalityTest

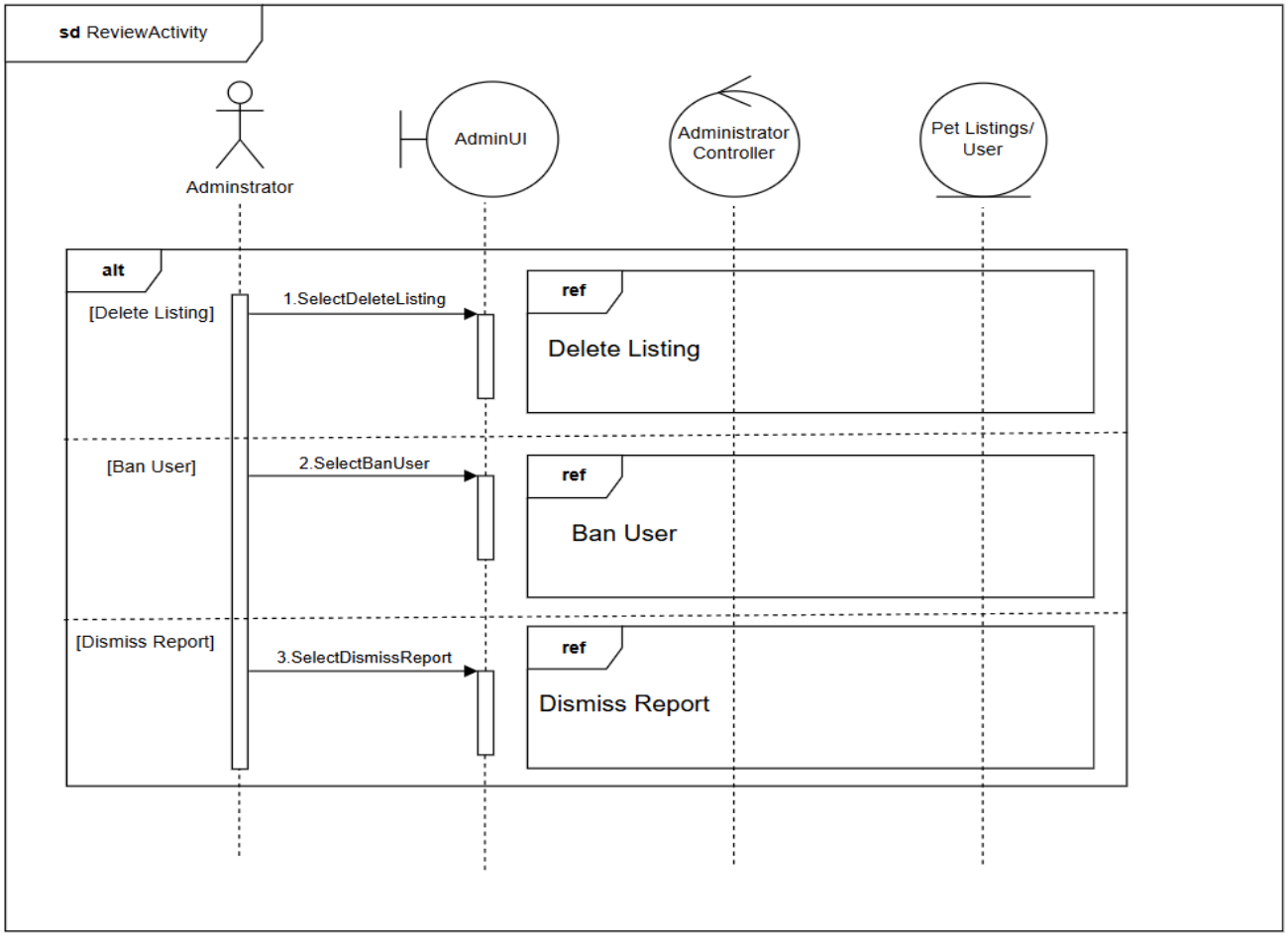


4.3.3.5 SuggestBreed

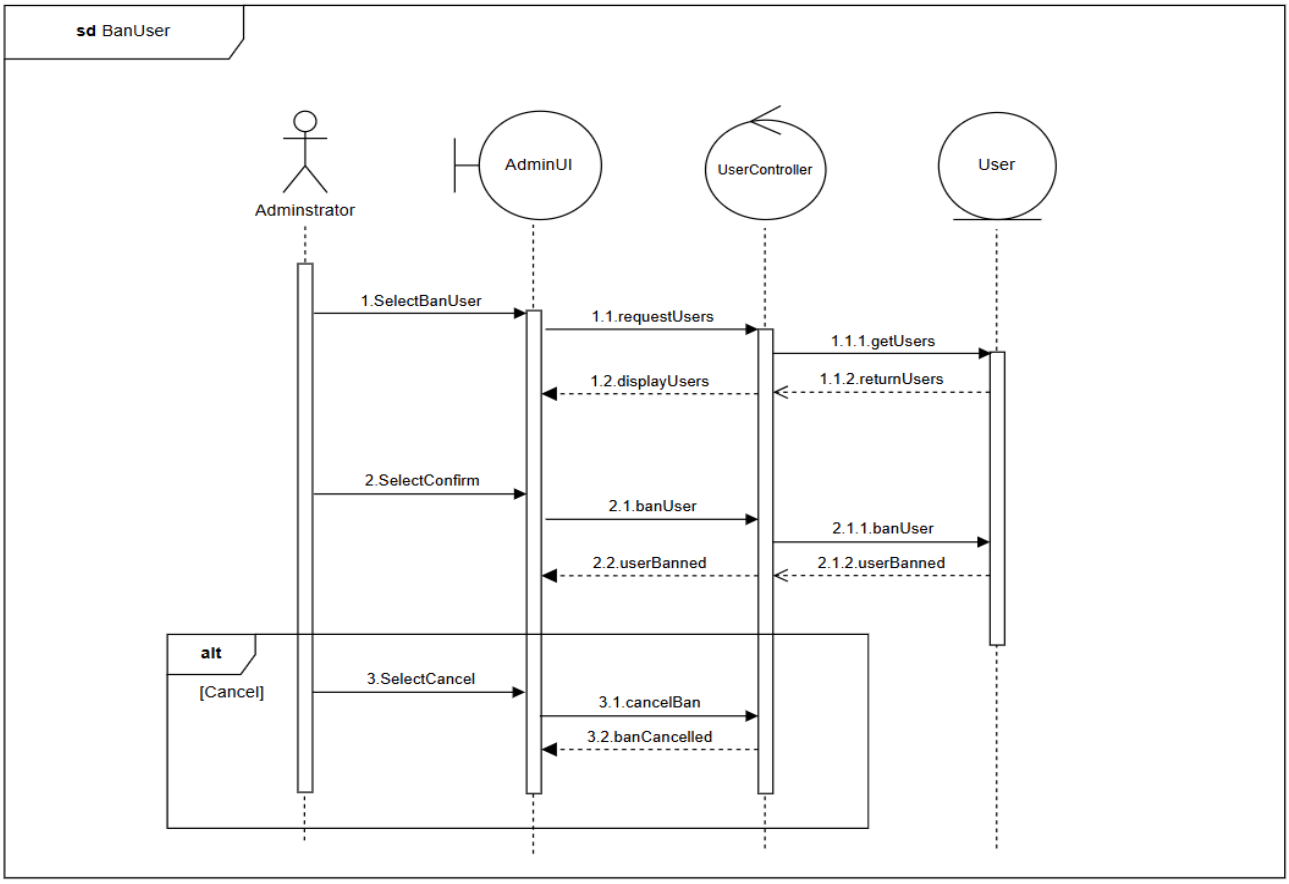


4.3.4 For Use Cases under #4

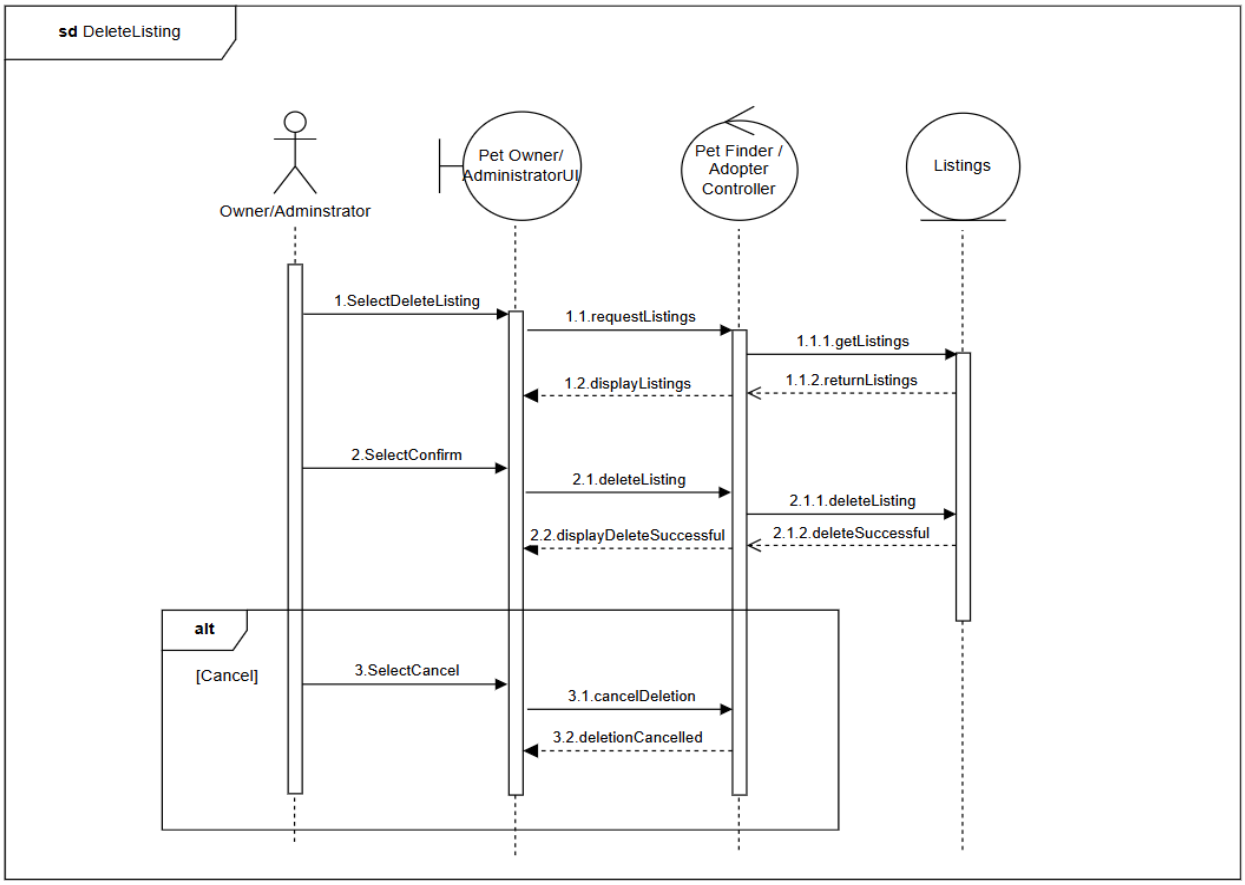
4.3.4.1 ReviewActivity



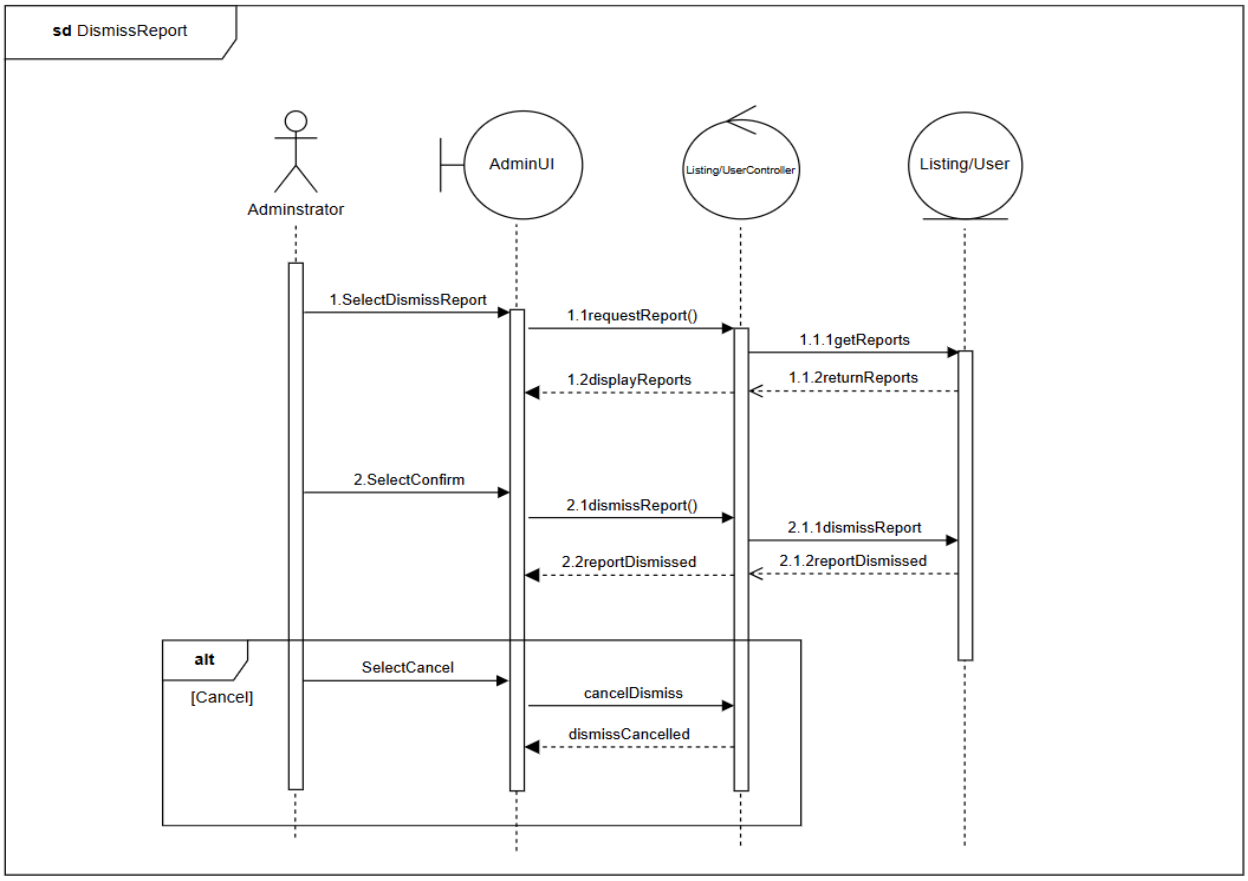
4.3.4.2 BanUser



4.3.4.3 DeleteListing



4.3.4.4 DismissReport



5. Other Nonfunctional Requirements

5.1 Usability Requirements

5.1.1 Responsive User Interface

1. 90% of users should be able to navigate the 2 different categories and the home page.

5.1.2 Quick Application Response

1. 90% of users should be able to login in 5 seconds.
2. The system should process the Users preferences and personality test results to return the most suitable pet to them in under 20 seconds.
3. The system should display the list of lost pets and pets for adoption in 5 seconds.

5.2 Safety Requirements

1. The system shall allow Users to report inappropriate listings to the Administrators.
2. The system shall allow Pet Owners receiving Adoption or Lost Pet requests to report those who are misbehaving inappropriately / trolls on the Internet to the Administrators.
3. The application will not require the sharing of any personal information between non-admin users to ensure the safety of all users.

5.3 Security Requirements

1. The user's profile and other personal information shall be kept confidential and shall not be disclosed to other users of the application or the public, in line with the PDPA rules.
2. User authentication will always be necessary before accessing other features of the app.
3. The system should implement password hashing for storing the users' passwords to protect users' accounts.
4. There should be zero tolerance for data leaks or compromises and the database should be stored securely by limiting access to the database to key developers and admins of the application.
5. All user-provided input on both the frontend and backend shall be validated to prevent common web vulnerabilities (e.g., cross-site scripting [XSS], injection attacks). Mongoose schema validation will be utilized on the backend.

6. Access to backend API endpoints will be protected using role-based authorization where necessary, ensuring users can only access data and perform actions appropriate for their role (User, Admin).

5.4 Software Quality Attributes

1. The code will always be accompanied with the relevant code comments indicating the purpose of the following block of code, to allow for interpretability, code reusability and future maintenance.
2. The system will display the appropriate success or error messages when actions are done by users, to improve testability of the product.

5.5 Business Rules

5.5.1 Encryption

1. Password to be encrypted before storing inside the database.

5.5.2 Providing Access Control

1. Role-based authentication to restrict admin and user functionalities.

Actors	**Users**	**Listings**	**Flag (Users/Listings)**
Owner	createUser() updateUser()	createListings() updateListings() deleteListings() getAllListings()	flagUser() flagListings()
PetFinder	createUser() updateUser()	getAllListings()	flagUser() flagListings()
Adopter	createUser() updateUser()	getAllListings()	flagUser() flagListings()
Admin	createUser() updateUser() getAllFlaggedUsers() banUser()	createListings() updateListings() deleteListings() getAllFlaggedListings()	removeFlag()

Actors	**AdoptionRequest**	**LostPetRequest**
Owner	reviewAdoptRequest()	reviewLostPetRequest()
PetFinder	-	createLostPetRequest()
Adopter	createAdoptRequest()	-
Admin	-	-

5.6 Verification Approach

1. Testing will utilize black-box methods (Equivalence Class Partitioning, Boundary Value Analysis) for validating functional requirements from a user perspective, focusing on key inputs and outputs, particularly for controller logic.
2. White-box testing (Basis Path Analysis) will be applied to selected backend methods with significant logical complexity to ensure path coverage.
3. Testing documentation will include detailed test cases and execution results (Input, Expected, Actual) as per project deliverables.
4. API testing will be performed using tools like Postman to verify endpoint functionality and responses directly.

6. Application Testing

6.1 Black Box Testing

6.1.1 Selected Control Class

The control class selected for testing will be the OllamaChat class (PersonalityController). The OllamaChat class manages interaction with the Ollama API for generating AI responses, including sending prompts to the AI model and determining personality traits based on quiz answers.

6.1.2 Equivalence Class Testing

The **ask()** and **determinePersonality()** methods require discrete values as inputs. As such, Boundary Value Testing will not be applicable.

6.1.2.1 ask() Function

- Valid Equivalence Class: Prompt input is a non-empty string in proper format.
- Invalid Equivalence Class: Prompt input is empty, non-string value, or extremely long.

6.1.2.2 determinePersonality() Function

- Valid Equivalence Class: Quiz answers are provided as a well-formed object with all required question-answer pairs.
- Invalid Equivalence Class: Quiz answers object is incomplete (missing some required questions), empty object, or non-object value.

6.1.3 Test Cases and Testing Results

6.1.3.1 Test Cases for ask() Method

Input Parameters:

1. Prompt

Test Case Name	Test Input	Expected Output	Actual Output	Test Result
Ask-01 (Valid)	prompt: "What is the capital of France?"	A non-empty string response about Paris	A non-empty string response about Paris	Pass
Ask-02 (Invalid)	prompt: "" (Empty string)	"Error fetching response from AI."	"Error fetching response from AI."	Pass
Ask-04 (Invalid)	prompt: "a".repeat(10000) (Extremely long string)	error message	error message	Pass
Ask-05 (Invalid)	prompt: null	"Error fetching response from AI."	"Error fetching response from AI."	Pass
Ask-06 (Invalid)	prompt: undefined	"Error fetching response from AI."	"Error fetching response from AI."	Pass

6.1.3.2 Test Cases for determinePersonality() Method

Input Parameters:

1. Prompt (Quiz answers by users)

Test Case Name	Test Input	Expected Output	Actual Output	Test Result
Personality-01 (Valid)	quizAnswers: {"Question 1": "Option A", "Question 2": "Option B", "Question 3": "Option C"}	Three comma-separated personality traits (e.g., "Adventurous, Active, Relaxed")	Three comma-separated personality traits	Pass
Personality-02 (Invalid)	quizAnswers: {} (Empty object - user didn't fill any options)	"Unknown, Please, Retry"	"Unknown, Please, Retry"	Pass
Personality-03 (Invalid)	quizAnswers: {"Question 1": "Option A"} (Incomplete - user only filled 1 of 3 questions)	"Unknown, Please, Retry"	"Unknown, Please, Retry"	Pass
Personality-04 (Invalid)	quizAnswers: {"Question 1": "Option A", "Question 2": "Option B"} (Incomplete - user only filled 2 of 3 questions)	Three comma-separated personality traits (possibly less accurate)	Three comma-separated personality traits	Pass
Personality-05 (Invalid)	quizAnswers: null (System error)	"Unknown, Please, Retry"	"Unknown, Please, Retry"	Pass

6.2 White Box Testing

The 2 methods selected for white box testing will be:

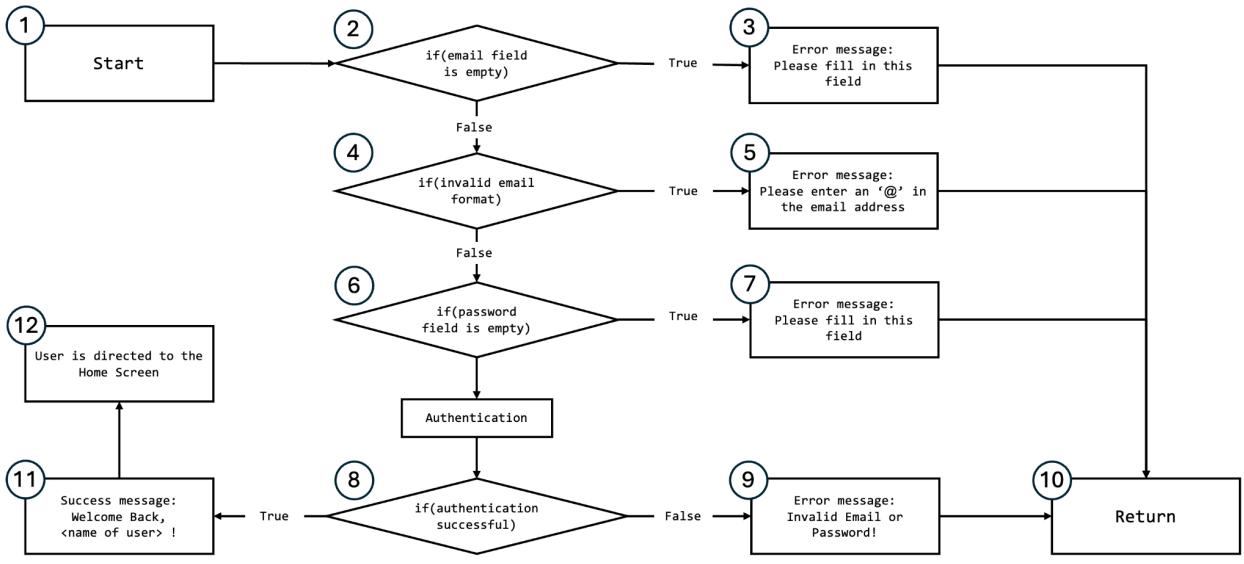
1. Login
2. PostLostPetListing

The **Login** function is responsible for authenticating users by validating their email and password inputs. It checks for missing fields, ensures the email format is correct, verifies the user's existence in the database, and compares the input password with the stored password hash. Upon successful validation, it generates and returns an authentication token for the user to log into the registered account.

The **PostLostLetListing** function enables users to submit a report for a lost pet. It performs a series of validations to ensure that all required fields are filled, the uploaded image is valid, the description does not exceed a set word limit, and both phone number and email formats are correct. Upon verification that all fields are filled up correctly, it publishes the listing for all users to view.

6.2.1 Login

6.2.1.1 Control Flow Graph



6.2.1.2 Cyclomatic Complexity

Cyclomatic Complexity (CC) = | binary decision points | + 1 = | 4 | + 1 = 5

6.2.1.3 Basis Paths

Login Method Basis Paths

- Basis Path #1 (Baseline): 1 → 2 → 4 → 6 → 8 → 11 → 12 (Successful Login)
- Basis Path #2: 1 → 2 → 3 → 10 (Email field empty)
- Basis Path #3: 1 → 2 → 4 → 5 → 10 (Invalid email format)
- Basis Path #4: 1 → 2 → 4 → 6 → 7 → 10 (Password field empty)
- Basis Path #5: 1 → 2 → 4 → 6 → 8 → 9 → 10 (Failed authentication)

6.2.1.4 Test Cases and Testing Results

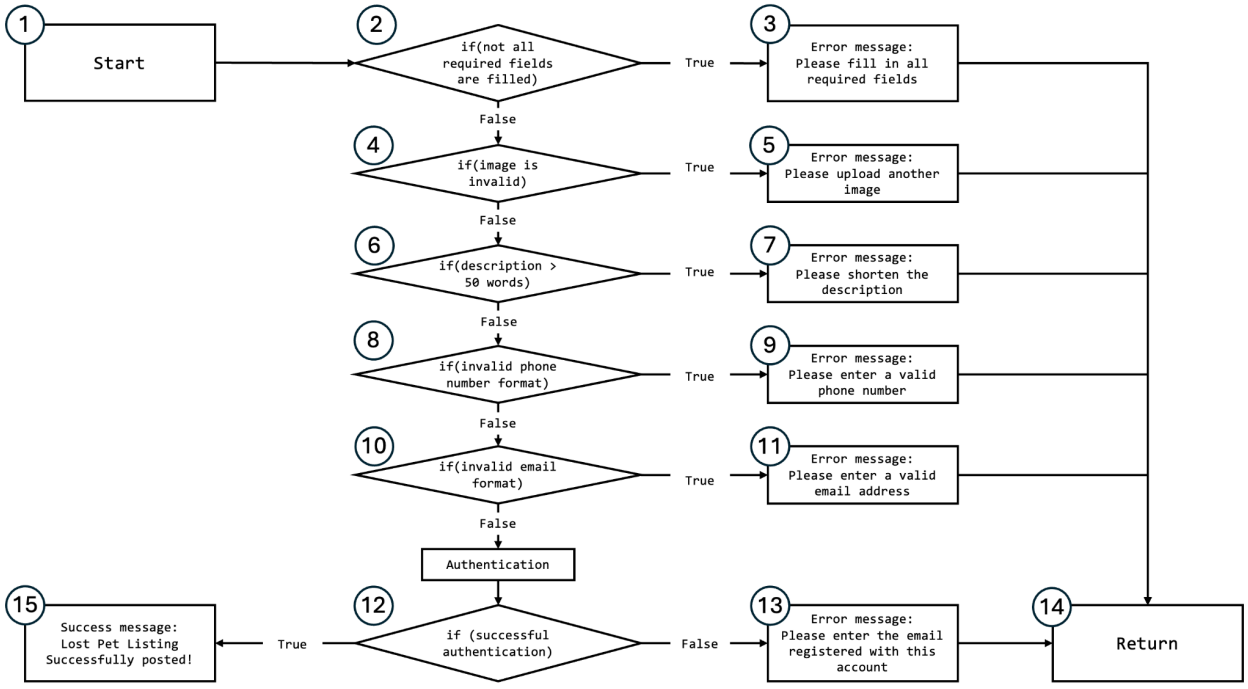
Input Parameters:

1. Email
2. Password

No.	Test Input	Expected Output	Actual Output	Test Result
1	Email: admin@gmail.com Password: admin	“Welcome back, Admin!”	“Welcome back, Admin!”	Pass
2	Email: Password: admin	“Please fill in this field!”	“Please fill in this field!”	Pass
3	Email: admin Password: admin	“Please include an ‘@’ in the email address”	“Please include an ‘@’ in the email address”	Pass
4	Email: admin@gmail.com Password:	“Please fill in this field!”	“Please fill in this field!”	Pass
5	Email: xyz@gmail.com Password: admin	“Invalid email or password!”	“Invalid email or password!”	Pass
6	Email: admin@gmail.com Password: xyz	“Invalid email or password!”	“Invalid email or password!”	Pass

6.2.2 PostLostPetListing

6.2.2.1 Control Flow Graph



6.2.2.2 Cyclomatic Complexity

Cyclomatic Complexity (CC) = | binary decision point | + 1 = | 6 | + 1 = 7

6.2.2.3 Basis Paths

handleAdoptionRequest Method Basis Paths

Basis Path #1 (Baseline): 1 → 2 → 4 → 6 → 8 → 12 → 15 (Successful posting)

Basis Path #2: 1 → 2 → 3 → 14 (Required fields not filled)

Basis Path #3: 1 → 2 → 4 → 5 → 14 (Invalid image)

Basis Path #4: 1 → 2 → 4 → 6 → 7 → 14 (Description too long)

Basis Path #5: 1 → 2 → 4 → 6 → 8 → 9 → 14 (Invalid phone)

Basis Path #6: 1 → 2 → 4 → 6 → 8 → 10 → 11 → 14 (Invalid email)

Basis Path #7: 1 → 2 → 4 → 6 → 8 → 10 → 12 → 13 → 14 (Invalid authentication)

6.2.2.4 Test Cases and Testing Results

Input Parameters:

1. PetName
2. PetAge
3. PetType
4. PetPhoto
5. LastSeenLocation
6. Description
7. Email
8. PhoneNumber

No.	Test Input	Expected Output	Actual Output	Test Result
1	Pet Name: TestPet Pet Age: 1 Pet Type: Dog Pet Photo: <Valid Image> Last Seen Location: <Location> Description: Small Email: User@gmail.com Phone Number: 88888888	“Lost Pet Listing Successfully Posted”	“Lost Pet Listing Successfully Posted”	Pass
2	Pet Name: Pet Age: Pet Type: Pet Photo: Last Seen Location: Description: Email: Phone Number:	“Please fill in all required fields”	“Please fill in all required fields”	Pass
3	Pet Name: TestPet Pet Age: 1 Pet Type: Dog Pet Photo: <INVALID Image>	“Please upload another image”	“Please upload another image”	Pass

	Last Seen Location: <Location> Description: Small Email: User@gmail.com Phone Number: 88888888			
4	Pet Name: TestPet Pet Age: 1 Pet Type: Dog Pet Photo: <Valid Image> Last Seen Location: <Location> Description: <A description longer than 50 words> Email: User@gmail.com Phone Number: 88888888	“Please shorten the description”	“Please shorten the description”	Pass
5	Pet Name: TestPet Pet Age: 1 Pet Type: Dog Pet Photo: <Valid Image> Last Seen Location: <Location> Description: <A description longer than 50 words> Email: admin@gmail.com Phone Number: zzz	“Please enter a valid phone number”	“Please enter a valid phone number”	Pass
6	Pet Name: TestPet Pet Age: 1 Pet Type: Dog Pet Photo: <Valid Image> Last Seen Location: <Location> Description: <A description longer than 50 words> Email: User Phone Number: 88888888	“Please enter a valid email address”	“Please enter a valid email address”	Pass
7	Pet Name: TestPet	“Please enter the	“Please enter	Pass

	Pet Age: 1 Pet Type: Dog Pet Photo: <Valid Image> Last Seen Location: <Location> Description: <A description longer than 50 words> Email: notUser@gmail.com Phone Number: 88888888	email address registered with this account”	the email address registered with this account”	

7. Other Requirements

7.1 Deployment Considerations

1. The application is designed for cloud deployment. The React frontend is suitable for static hosting platforms (e.g., Vercel, Netlify).
2. The Node.js/Express backend API is intended for deployment on Platform-as-a-Service (PaaS) providers (e.g., Heroku, Render).
3. The MongoDB database will utilize a cloud hosting service (e.g., MongoDB Atlas).
4. Production environment configurations (database URIs, API keys, JWT secrets) will be managed securely using environment variables provided by the hosting platform.

7.2 Initial Data / Seeding

1. The system requires at least one administrative user account to be created for initial setup and management tasks. Procedures for database seeding or initial admin user creation will be documented separately (e.g., in the README).

Appendix A: Data Dictionary

Term	Definition
User	An individual who interacts with the FetchMeHome application. Users can be pet owners, finders,adopters or administrators.
Authentication	The process of verifying a user's identity through email and password.
Profile	A collection of user information, including full name, email, phone number, and profile picture.
Verification	The process by which Owners approve or reject a PetFinder's reported pet sighting.
Report User	A feature allowing users to flag misconduct and notify administrators.
Report Listing	A feature to flag listings when it is fake or inappropriate and this will notify administrators.
Lost Pet Listing	A post created by an Owner containing details about a missing pet.
Review Lost Pet Request	Owners can verify if the pet in the photo submission by the Pet FInder is really their pet or not.
View Adoption Request	For the owner to verify and give their contact details upon verification of the adopter's information.
Adoption Listing	A record created by an Owner containing details about a pet available for adoption.
Pet Owner	The Owner can post the listings for adoption requests and post the lost dog listing.
Pet Finder	A person who finds the pet and/or uploads photos and/or return the pet to the owner
Adopter	A person who wishes to adopt a pet from the listings.
Administrator	An administrator who manages the platform and handles misconduct reports and listings.
Personality Test	A series of questions to determine which breed of pets are suitable for the user.