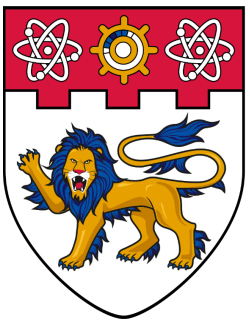


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

SC2006 - Software Engineering

Lab 3 Deliverables

Lab Group	SCMA
Team	FetchMeHome
Members	BENJAMIN YEOH JUN JIE (U2321438C)
	BASKAR KAYALVIZHI ATHITHIYA (U2322885L)
	TIN JING LUN JAVIER (U2310906E)
	TRISTAN AMADEUS SURYA (U2320674L)
	TOMAR YASHWARDHAN SINGH (U2323694E)
	ZOU XUEHAO (U2322789L)

Table of Contents

1. Use Case Model..... **3**

2. Design Model..... **4**

A. Key Class Diagram..... **4**
B. Sequence Diagram..... **8**
C. Initial Dialog Map..... **31**

3. System Architecture..... **32**

4. Application Skeleton..... **34**

A. Client..... **34**
B. Server..... **35**

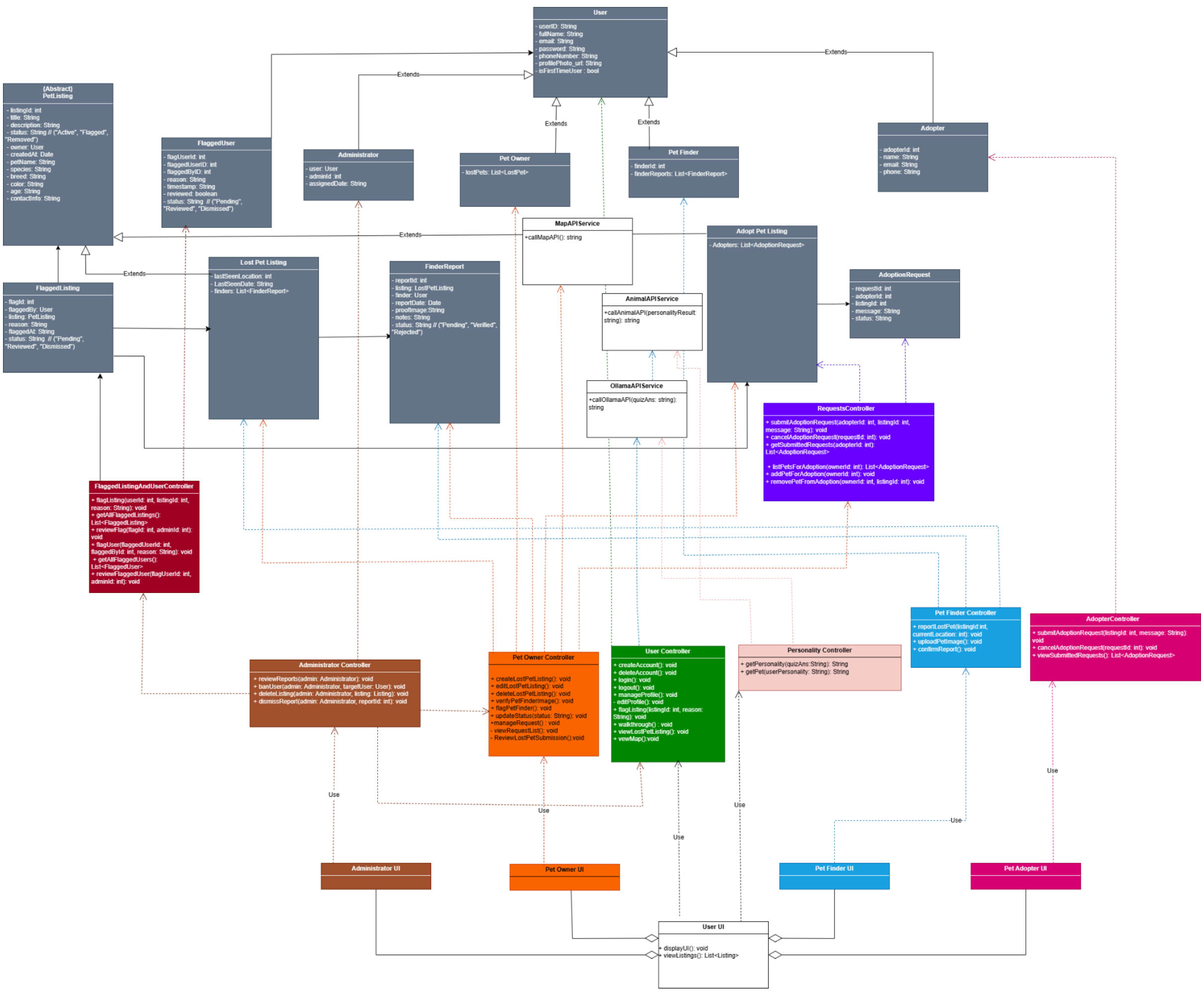
5. Appendix..... **36**

A. Identifying and Storing Persistent Data..... **36**
B. Providing Access Control..... **41**
C. Security Considerations..... **42**
D. Tech Stack..... **42**

1. Use Case Model

2. Design Model

A. Key Classes Diagram



App.js / Server.js

Start-up class for initialisation of the front and backend.

Operations:

- npm start: renders the frontend User Interface (UI)
- npm start: starts the backend application, connects to database (MongoDB) for data access and modification

User Interface (UI)

User Interfaces define the screens available to different user roles.

- **UserUI**: Main screen accessible by all users. Users can authenticate and access core functionalities.

- **AdminUI**: Includes additional management pages for administrators, allowing them to review flagged users, process reports, and ban users.

- **OwnerUI**: Allows authenticated users to post, edit, delete, and view data (CRUD) about their listings. They are able to review adoption or lost pet requests made by other users.

- **FinderUI**: Allows authenticated users to submit lost pet images to the respective owners for verification. They are able to flag users deemed inappropriate.

- **AdopterUI**: Allows authenticated users to submit applications to adopt a pet from the list of pets that are put up for adoption.

Controllers (Facade Pattern)

Application of the Facade Pattern is when the controllers serve as “Interfaces” for interacting with the application’s business logic, acting as entry points into the application. There are multiple controllers, depending on the specific use case and/or role of the User.

- AuthController: Handles user authentication (create account and log in). Accesses the **User** entity.

- UserController: Manages general user actions and allows users to view the list of pets put up for adoption as well as pets reported as lost.

- AdminController: Allows admins to review flagged users/listings, process reports, ban users, and remove listings. Accesses **LostPet**, **AdoptPet,** and **Report**.

- OwnerController: Manages pet listings. Includes the ability to post, edit, and remove listings. Accesses **LostPet** and **AdoptPet**.

- AdopterController: Manages Pet Adoption actions such as submitting adoption requests and viewing submitted requests. Access **RequestAdopt**.

- FinderController: Manages Pet Found actions such as submitting found pets.

- AdoptPetController: Manages the list of pets put up for adoption (eg. create adoption listing, edit adoption listing, delete adoption listing). Accesses **AdoptPet**.

- LostPetController: Manages the list of lost pets (eg. create lost pet listing, edit lost pet listing, delete lost pet listing). Accessed **LostPet.**

- FlagController: Manages the list of flagged users and listings(eg. allows authenticated Users to report other Users and Listings). Accesses **LostPet** and **AdoptPet.**

- PersonalityController: Manages the calls of OllamaAPI and AnimalAPI to determine which type/breed of pets are suitable for Users.

Data Access Layer

Observer Pattern

Use Case: Notifies pet owners when someone reports finding their lost pet.

Implementation:

- **LostPet** acts as the Subject for PetOwner(Observer).
- **LostPetController** triggers updates when lost pets are reported.
- **PetOwner** (Observer) receives a notification upon updates.

Benefit: Ensures timely and relevant notifications without unnecessary polling.

Factory Pattern

Use Case: The Factory Pattern is used to encapsulate the creation logic of different user types (User & Admin).

Implementation:

- When a user creates an account, the system checks whether the account being created is a normal user account or an admin account. Based on this, the **UserFactory** instantiates the appropriate class (**User** or **Admin**) and returns the corresponding object.
- The created object is then saved to the database, where it can later be retrieved during login or other operations to determine the user's role and associated privileges.

Benefits: Centralizes user creation, improves scalability, enhances maintainability, and keeps code modular and consistent.

Singleton Pattern

Use Case: Ensures only one instance of the OllamaChat service exists for interacting with the external Ollama AI model. This avoids the overhead and inconsistency that could result from creating multiple instances.

Implementation:

- The **PersonalityController.js** file defines the **OllamaChat** class.
- A single instance is created within the module as "**const ollamaChat = new OllamaChat("llama3.2")**".
- This specific instance is exported directly as "**module.exports = ollamaChat**".

- Any other module that requires **PersonalityController.js** receives the same, single, cached instance due to Node.js module caching behaviour.

Benefit: Ensures a consistent interaction point with the external AI service, prevents unnecessary extra instantiation, and potentially manages resources or state related to the AI connection.

MVC (Model-View-Controller) Pattern

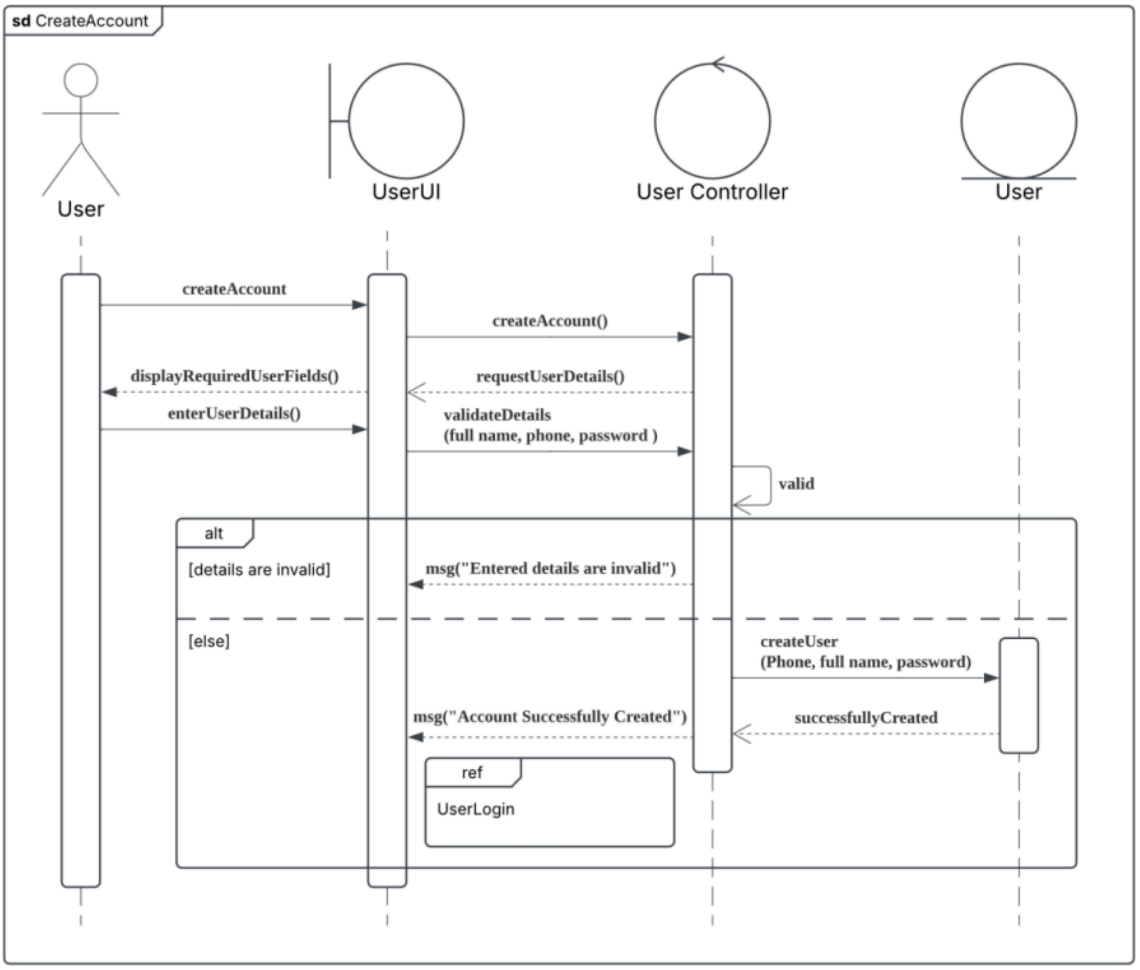
Use Case: Organizes the backend application by separating concerns into distinct layers: data management (Model), request handling/application logic (Controller), and URL path definition (Routes).

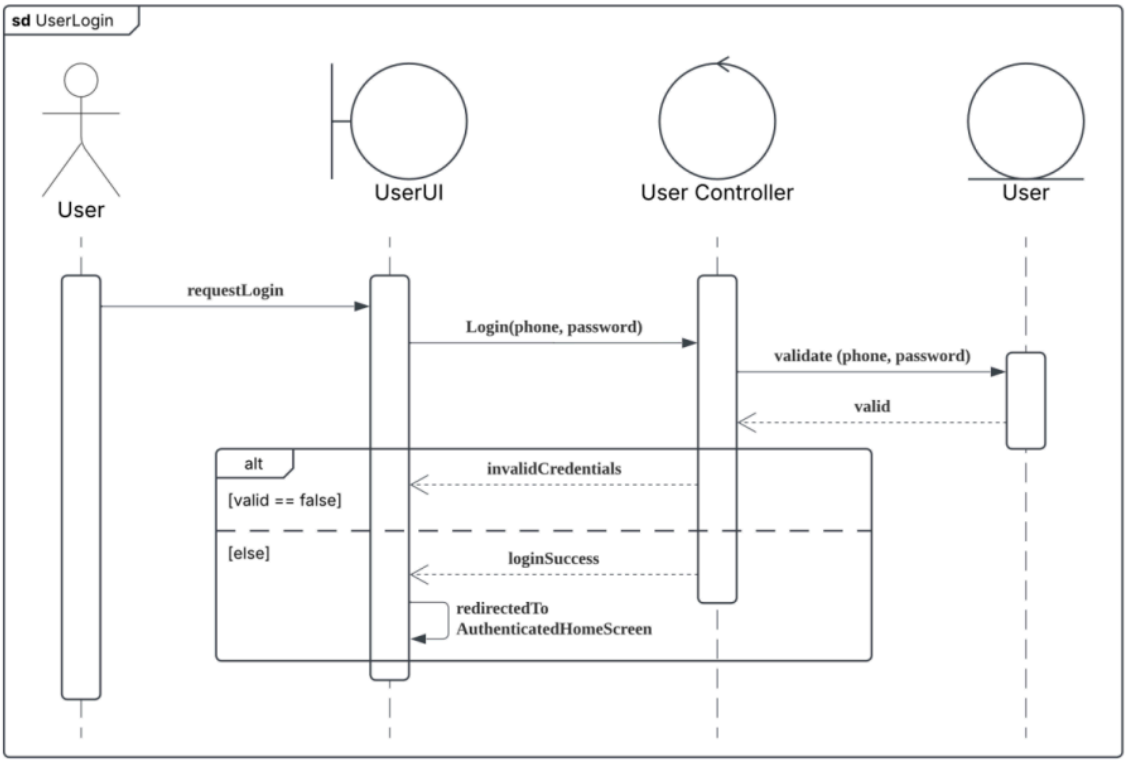
Implementation:

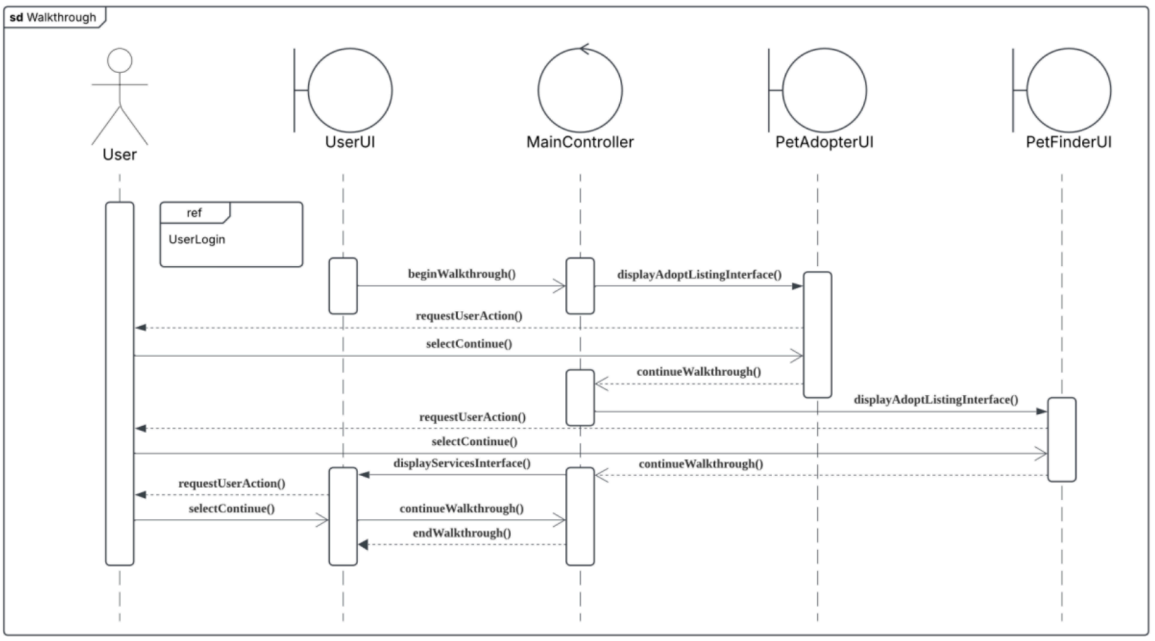
- **Model:** Implemented in the **Model** folder using Mongoose schemas (UserModel.js, PetModel.js, ReportModel.js, etc.) to define data structure and interact with the MongoDB database.
- **Controller:** Implemented in the **Controller** folder (AdoptPet.js, LostPet.js, etc.). Functions here receive requests (forwarded by Routes), use Models to perform actions (create, read, update, delete data), and formulate HTTP responses.
- **Routes:** Define API endpoints (URL paths and HTTP methods) and map them to specific functions within the Controllers. Acts as the entry point for requests after middleware.

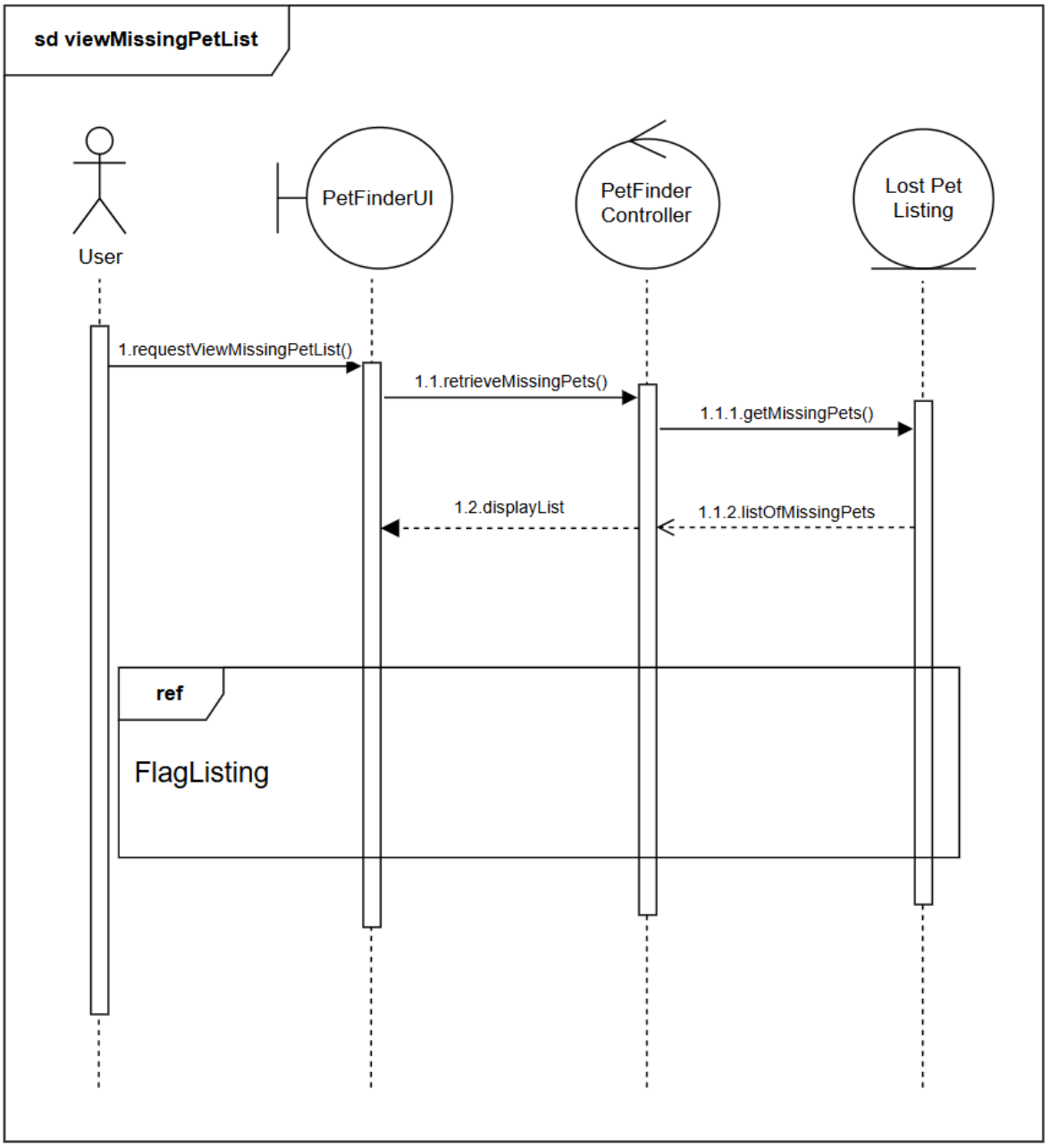
Benefit: Provides a clear separation of concerns, making the codebase more organized, easier to understand, maintain, and test. Facilitates parallel development on different application layers, where developers can work independently on Models, Controllers, or Routes without conflicts.

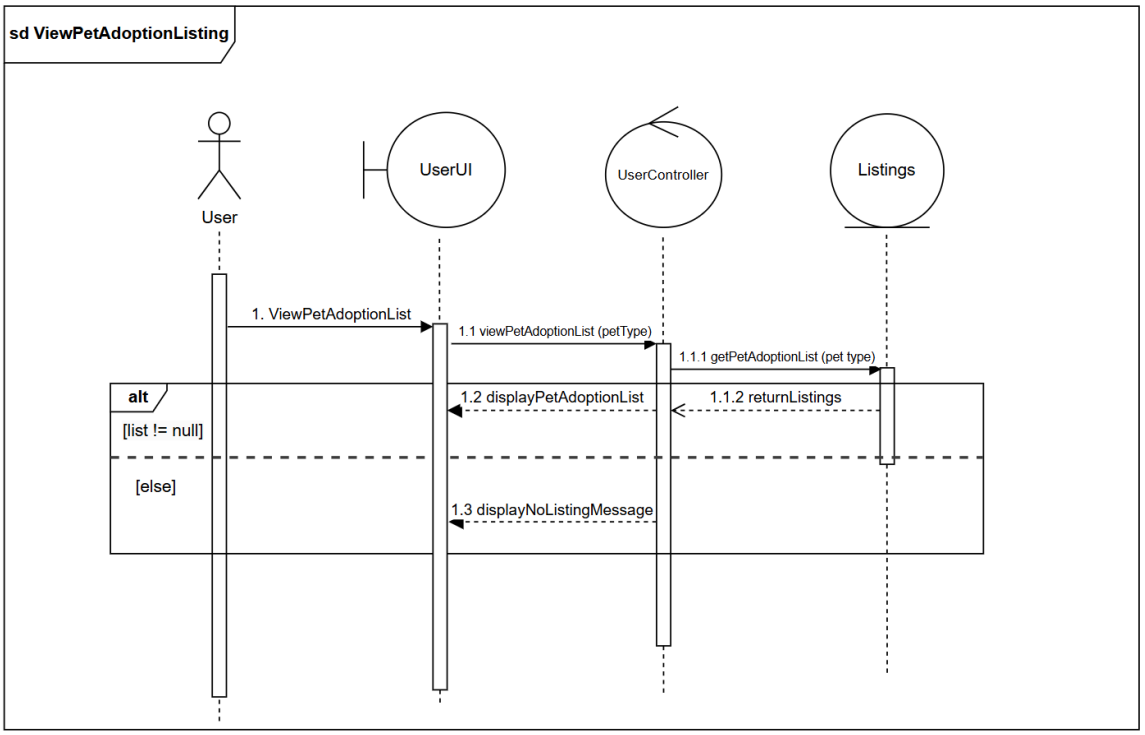
B. Sequence Diagram of Use Cases

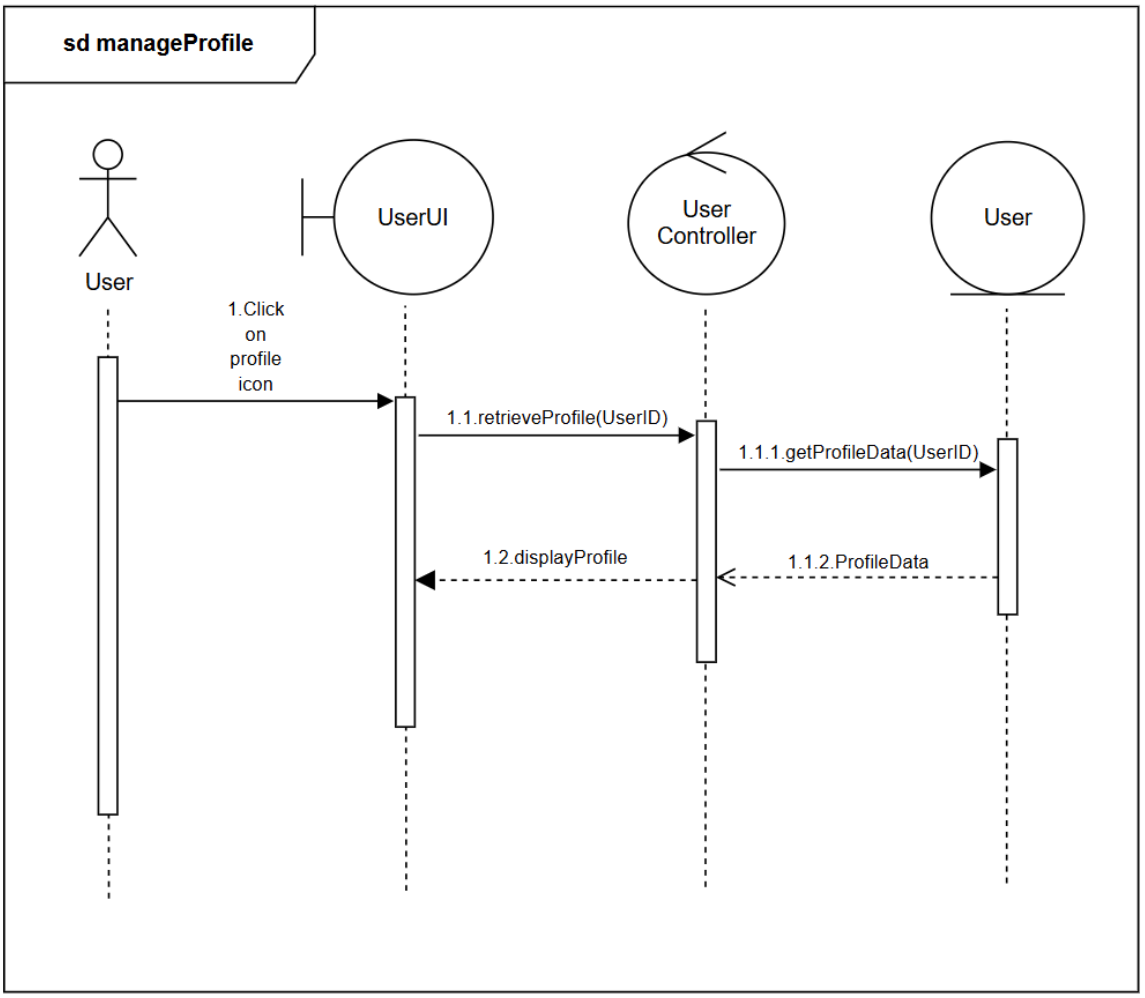


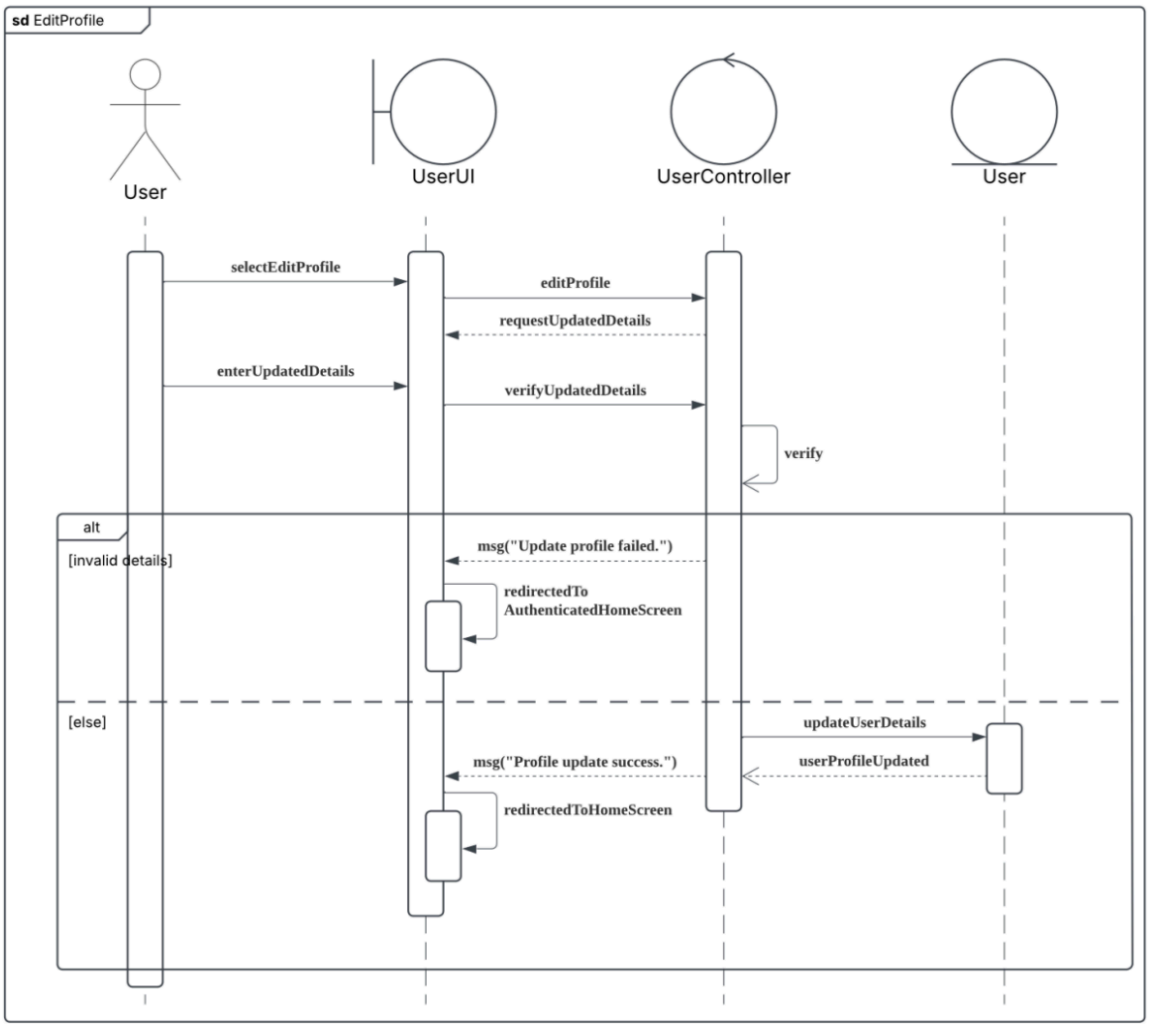

sd UserLogin
User
UserUI
User Controller
User
requestLogin
Login(phone, password)
validate (phone, password)
valid
alt
[valid == false]
invalidCredentials
[else]
loginSuccess
redirectedTo
AuthenticatedHomeScreen

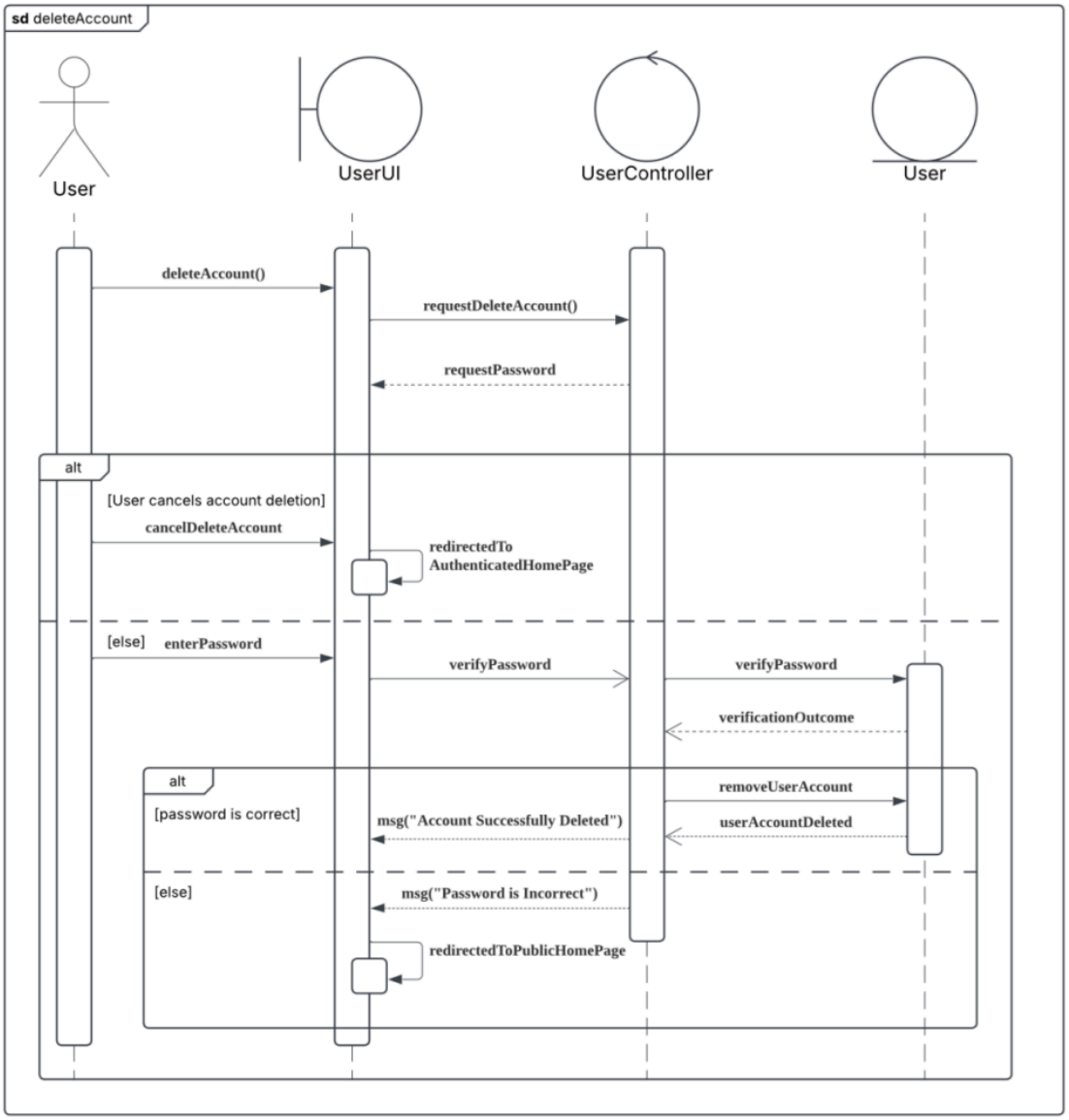

sd Walkthrough
User
UserUI
MainController
PetAdopterUI
PetFinderUI
ref
UserLogin
beginWalkthrough()
displayAdoptListingInterface()
requestUserAction()
selectContinue()
continueWalkthrough()
displayAdoptListingInterface()
requestUserAction()
selectContinue()
displayServicesInterface()
continueWalkthrough()
requestUserAction()
selectContinue()
continueWalkthrough()
endWalkthrough()

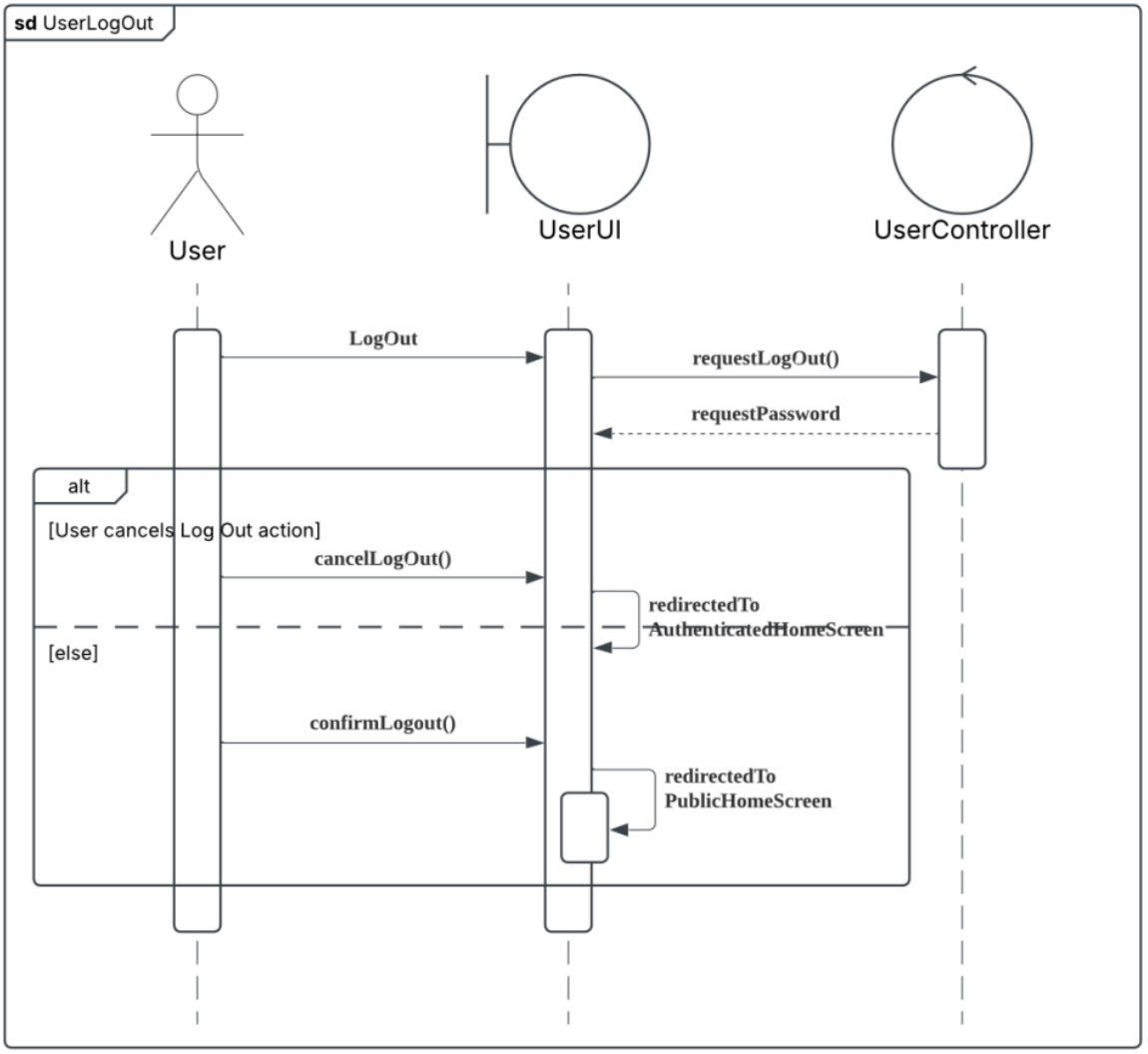

sd viewMissingPetList
User
PetFinderUI
PetFinder Controller
Lost Pet Listing
1.requestViewMissingPetList()
1.1.retrieveMissingPets()
1.1.1.getMissingPets()
1.2.displayList
1.1.2.listOfMissingPets
ref
FlagListing

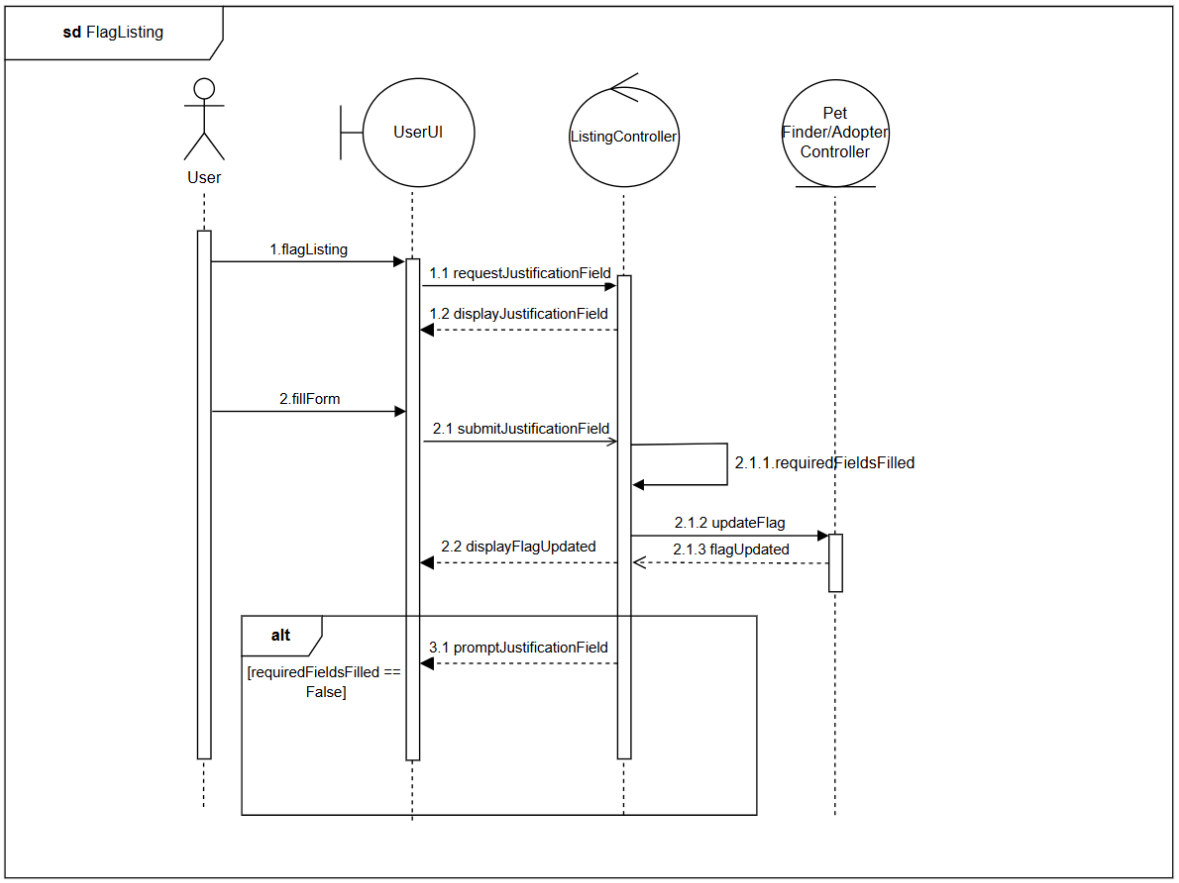

sd ViewPetAdoptionListing
User
UserUI
UserController
Listings
1. ViewPetAdoptionList
1.1 viewPetAdoptionList (petType)
1.1.1 getPetAdoptionList (pet type)
alt
[list != null]
1.2 displayPetAdoptionList
1.1.2 returnListings
[else]
1.3 displayNoListingMessage

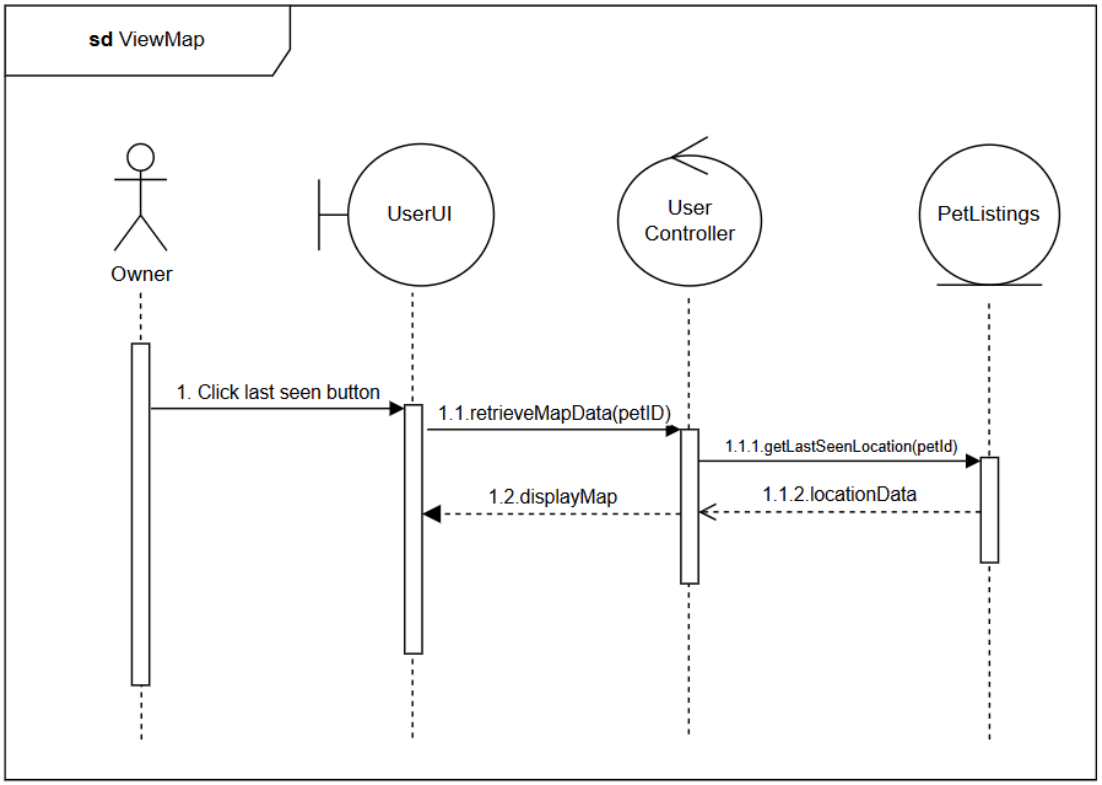

sd manageProfile
User
UserUI
User Controller
User
1.Click on profile icon
1.1.retrieveProfile(UserID)
1.1.1.getProfileData(UserID)
1.2.displayProfile
1.1.2.ProfileData

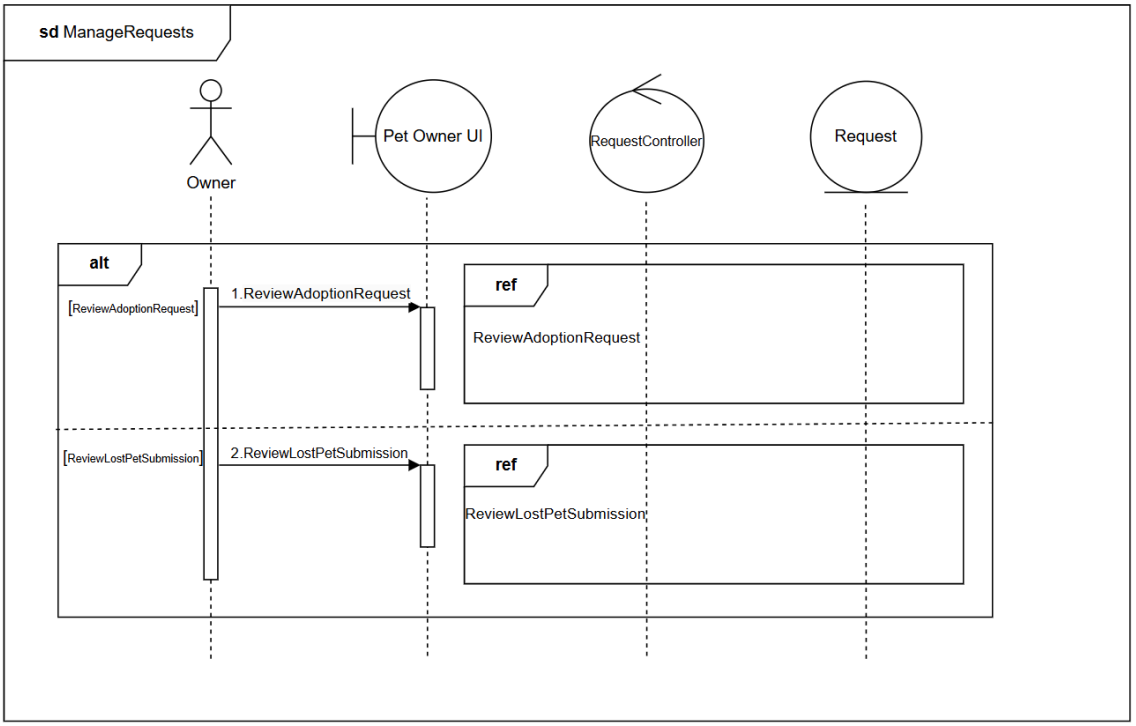

sd EditProfile
User
UserUI
UserController
User
selectEditProfile
editProfile
requestUpdatedDetails
enterUpdatedDetails
verifyUpdatedDetails
verify
alt
[invalid details]
msg("Update profile failed.")
redirectedTo
AuthenticatedHomeScreen
[else]
updateUserDetails
msg("Profile update success.")
userProfileUpdated
redirectedToHomeScreen

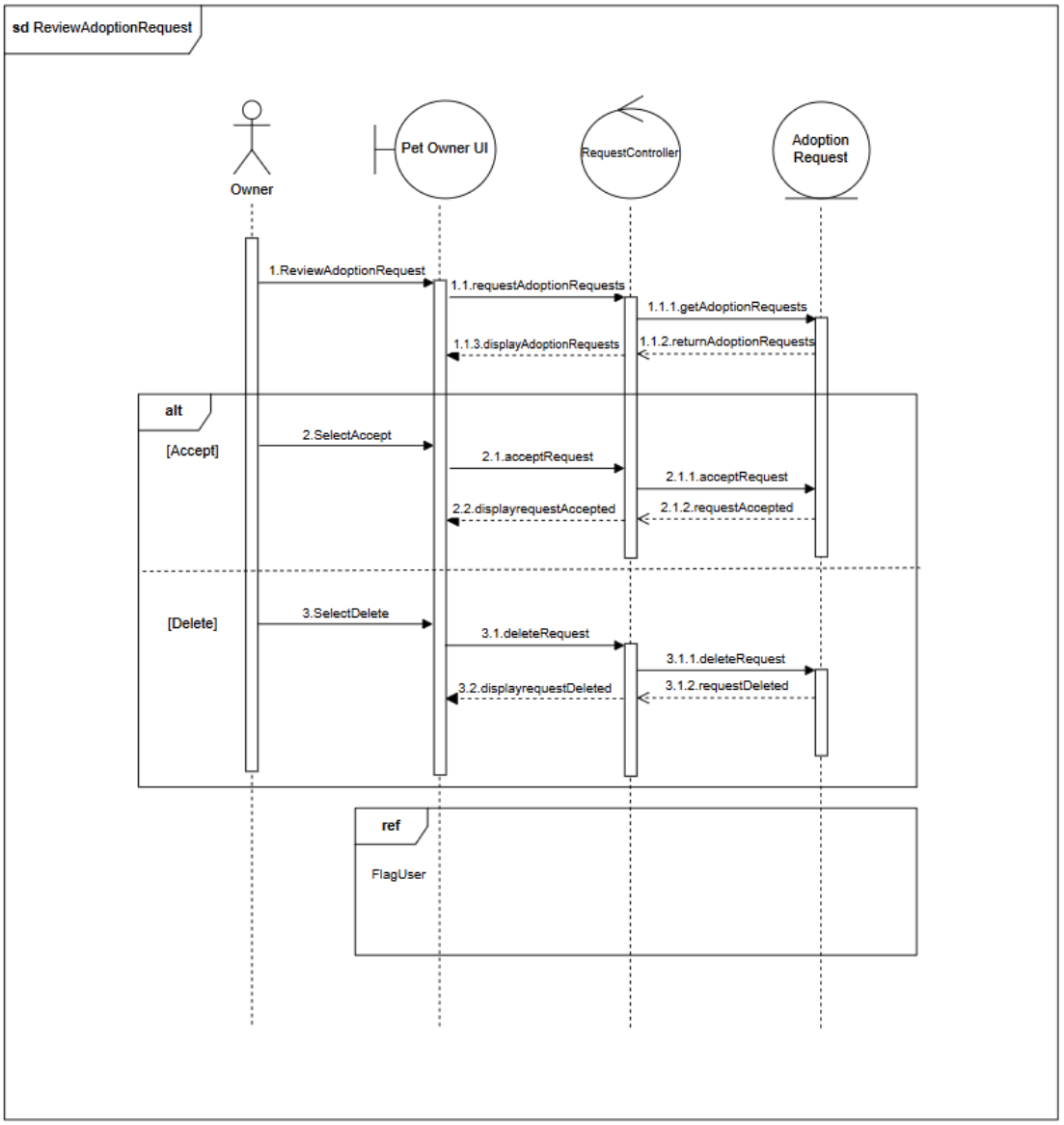

sd deleteAccount
User
UserUI
UserController
User
deleteAccount()
requestDeleteAccount()
requestPassword
alt
[User cancels account deletion]
cancelDeleteAccount
redirectedTo
AuthenticatedHomePage
[else]
enterPassword
verifyPassword
verifyPassword
verificationOutcome
alt
[password is correct]
removeUserAccount
msg("Account Successfully Deleted")
userAccountDeleted
[else]
msg("Password is Incorrect")
redirectedToPublicHomePage

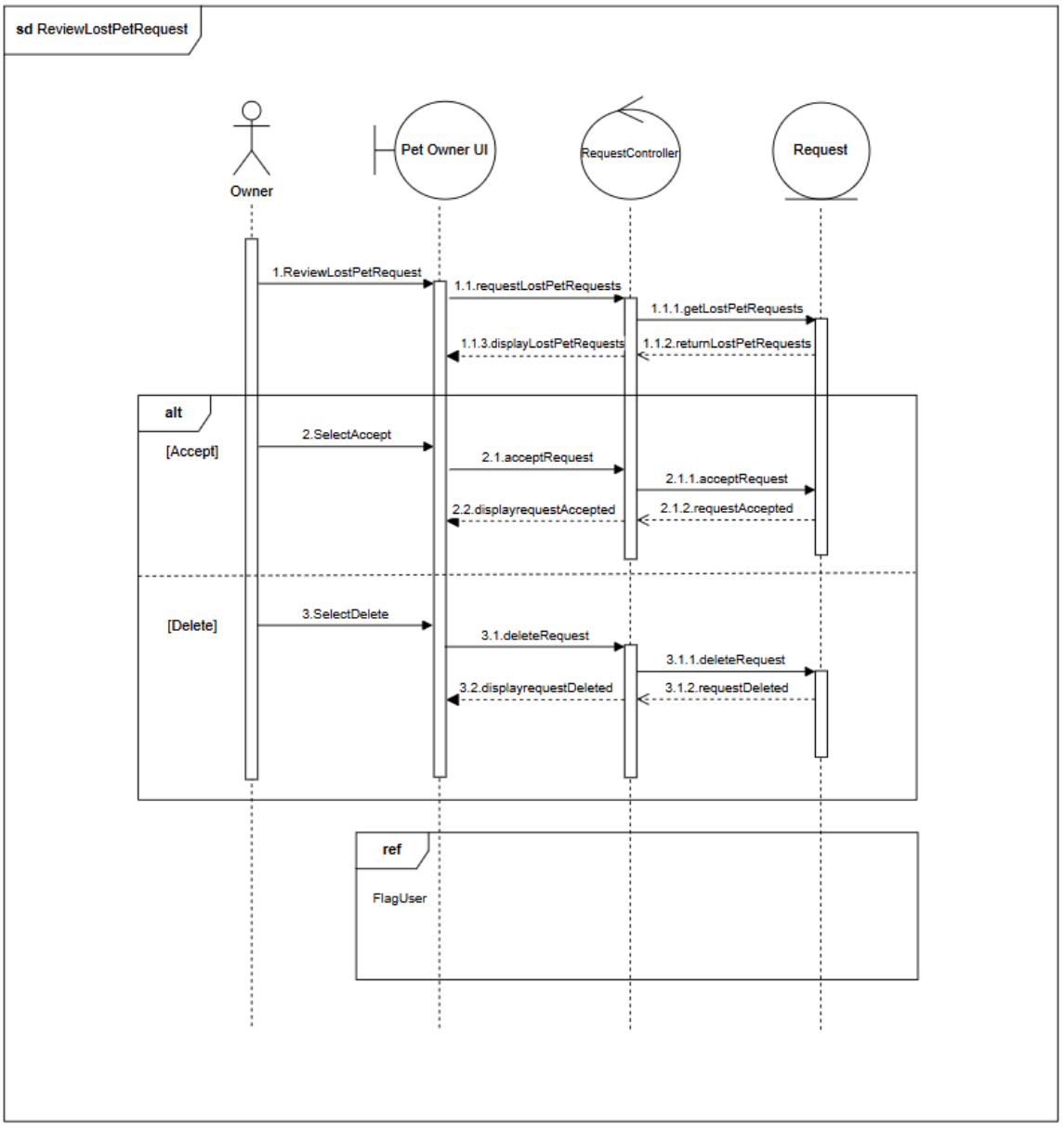

sd UserLogOut
User
UserUI
UserController
LogOut
requestLogOut()
requestPassword
alt
[User cancels Log Out action]
cancelLogOut()
redirectedTo
AuthenticatedHomeScreen
[else]
confirmLogout()
redirectedTo
PublicHomeScreen

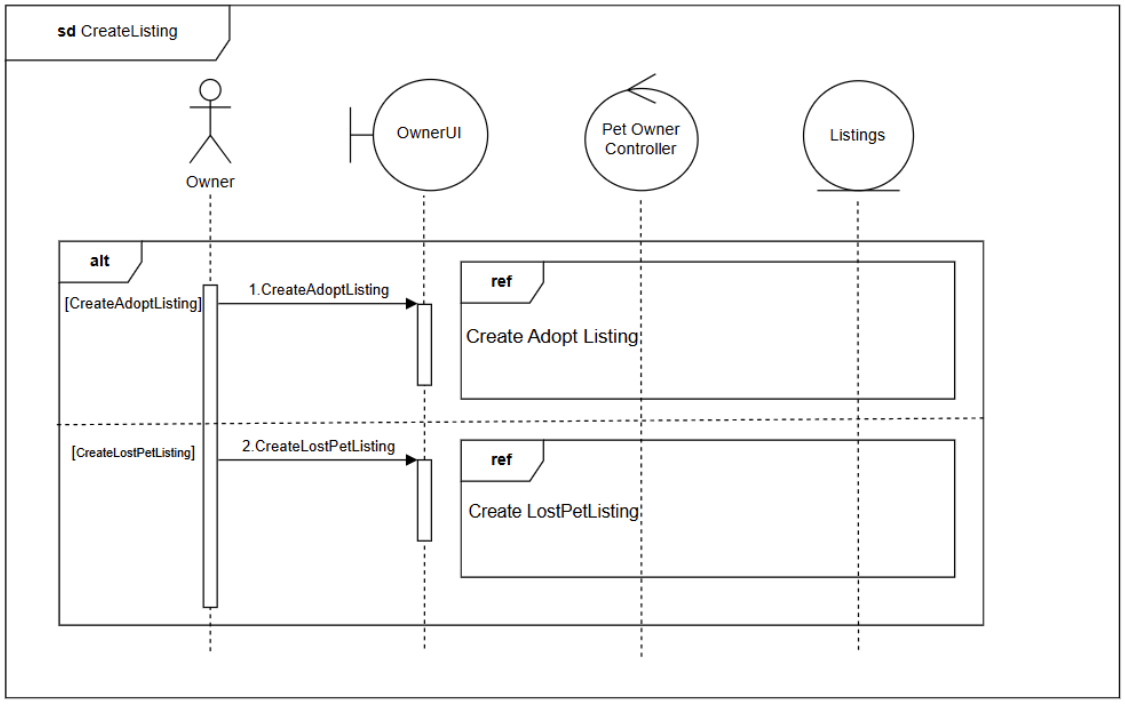

sd FlagListing
User
UserUI
ListingController
Pet Finder/Adopter Controller
1.flagListing
1.1 requestJustificationField
1.2 displayJustificationField
2.fillForm
2.1 submitJustificationField
2.1.1.requiredFieldsFilled
2.1.2 updateFlag
2.1.3 flagUpdated
2.2 displayFlagUpdated
alt
[requiredFieldsFilled == False]
3.1 promptJustificationField

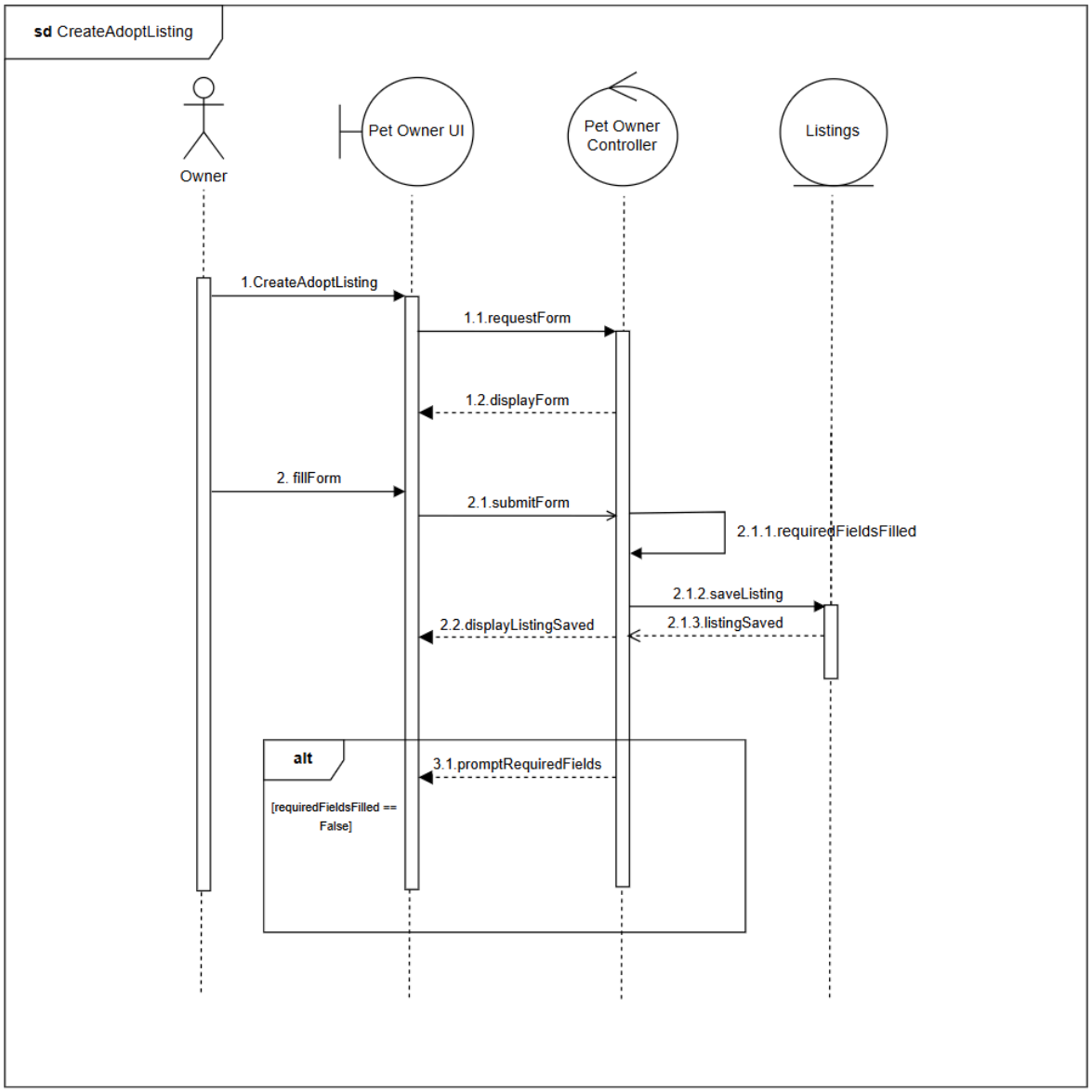

sd ViewMap
Owner
UserUI
User Controller
PetListings
1. Click last seen button
1.1.retrieveMapData(petID)
1.1.1.getLastSeenLocation(petId)
1.2.displayMap
1.1.2.locationData

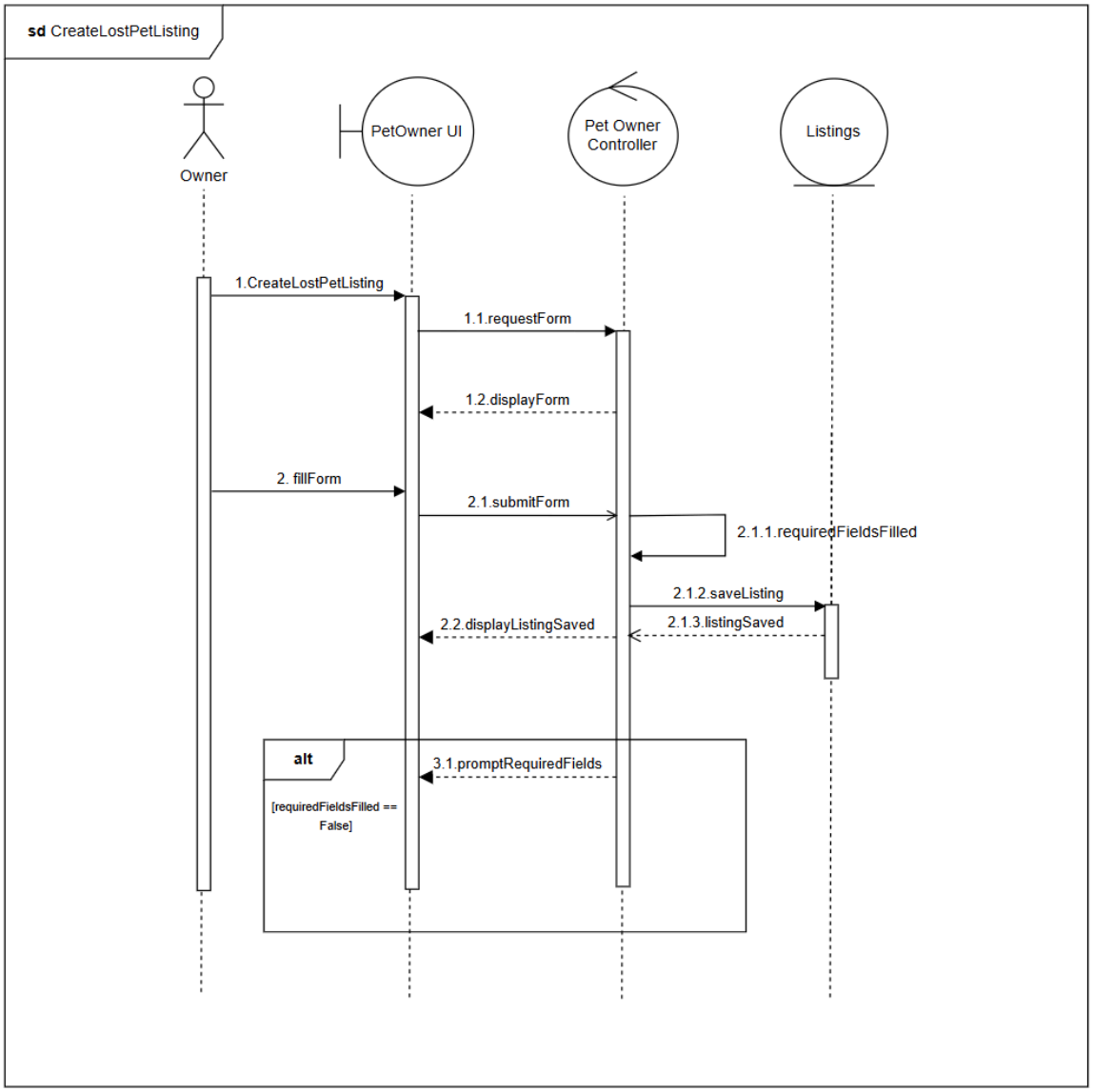

sd ManageRequests
Owner
Pet Owner UI
RequestController
Request
alt
[ReviewAdoptionRequest]
1.ReviewAdoptionRequest
ref
ReviewAdoptionRequest
[ReviewLostPetSubmission]
2.ReviewLostPetSubmission
ref
ReviewLostPetSubmission

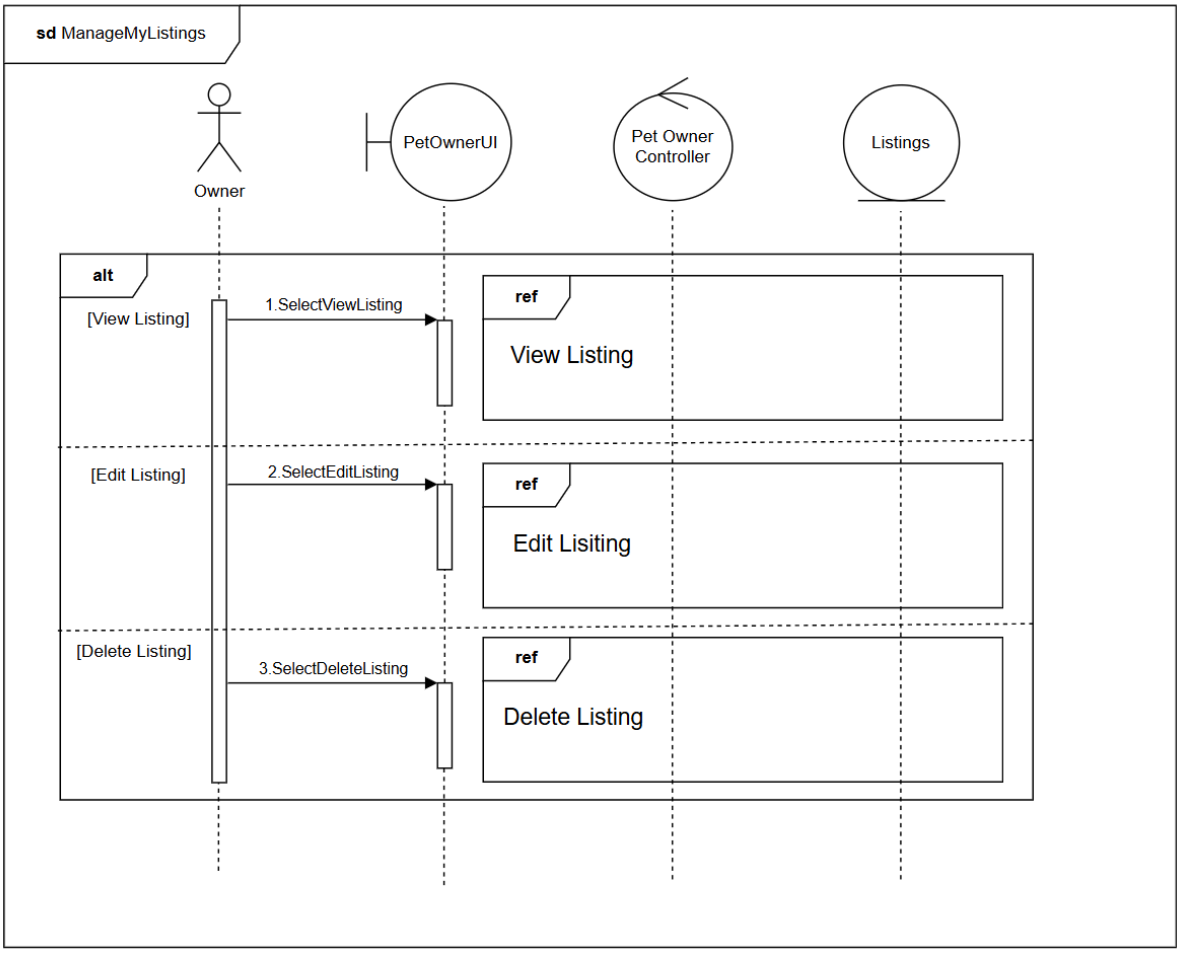

sd ReviewAdoptionRequest
Owner
Pet Owner UI
RequestController
Adoption Request
1.ReviewAdoptionRequest
1.1.requestAdoptionRequests
1.1.1.getAdoptionRequests
1.1.3.displayAdoptionRequests
1.1.2.returnAdoptionRequests
alt
[Accept]
2.SelectAccept
2.1.acceptRequest
2.1.1.acceptRequest
2.2.displayrequestAccepted
2.1.2.requestAccepted
[Delete]
3.SelectDelete
3.1.deleteRequest
3.1.1.deleteRequest
3.2.displayrequestDeleted
3.1.2.requestDeleted
ref
FlagUser

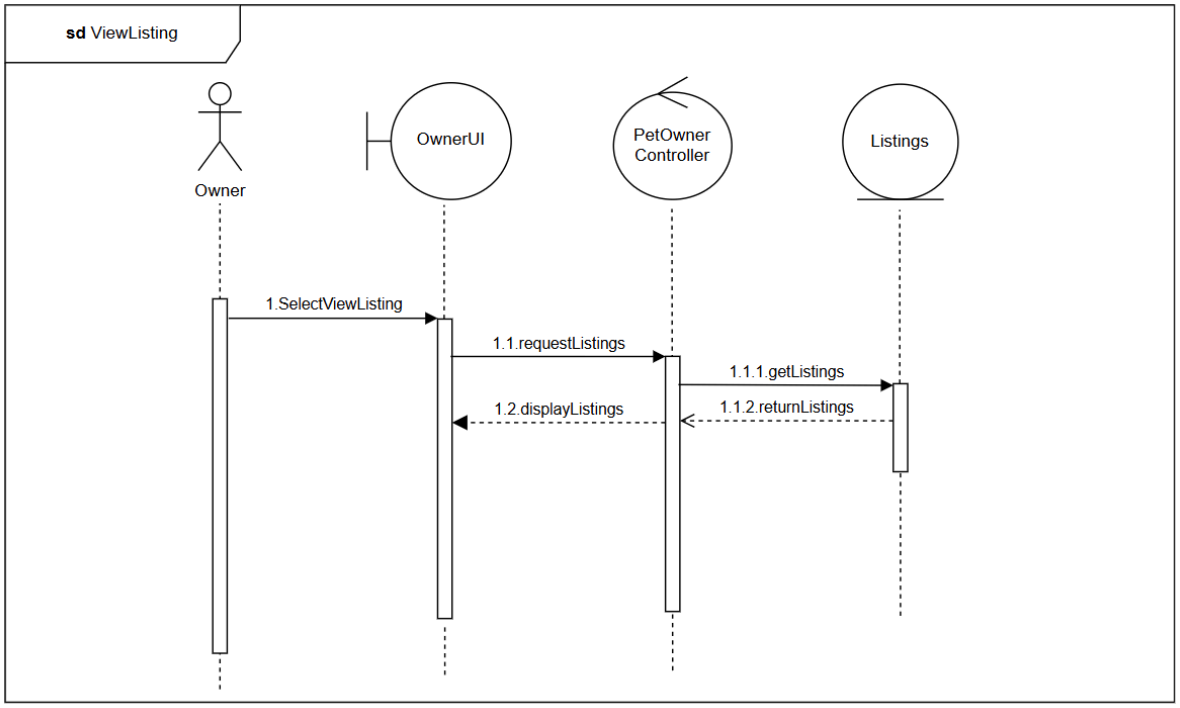

sd ReviewLostPetRequest
Owner
Pet Owner UI
RequestController
Request
1.ReviewLostPetRequest
1.1.requestLostPetRequests
1.1.1.getLostPetRequests
1.1.3.displayLostPetRequests
1.1.2.returnLostPetRequests
alt
[Accept]
2.SelectAccept
2.1.acceptRequest
2.1.1.acceptRequest
2.2.displayrequestAccepted
2.1.2.requestAccepted
[Delete]
3.SelectDelete
3.1.deleteRequest
3.1.1.deleteRequest
3.2.displayrequestDeleted
3.1.2.requestDeleted
ref
FlagUser

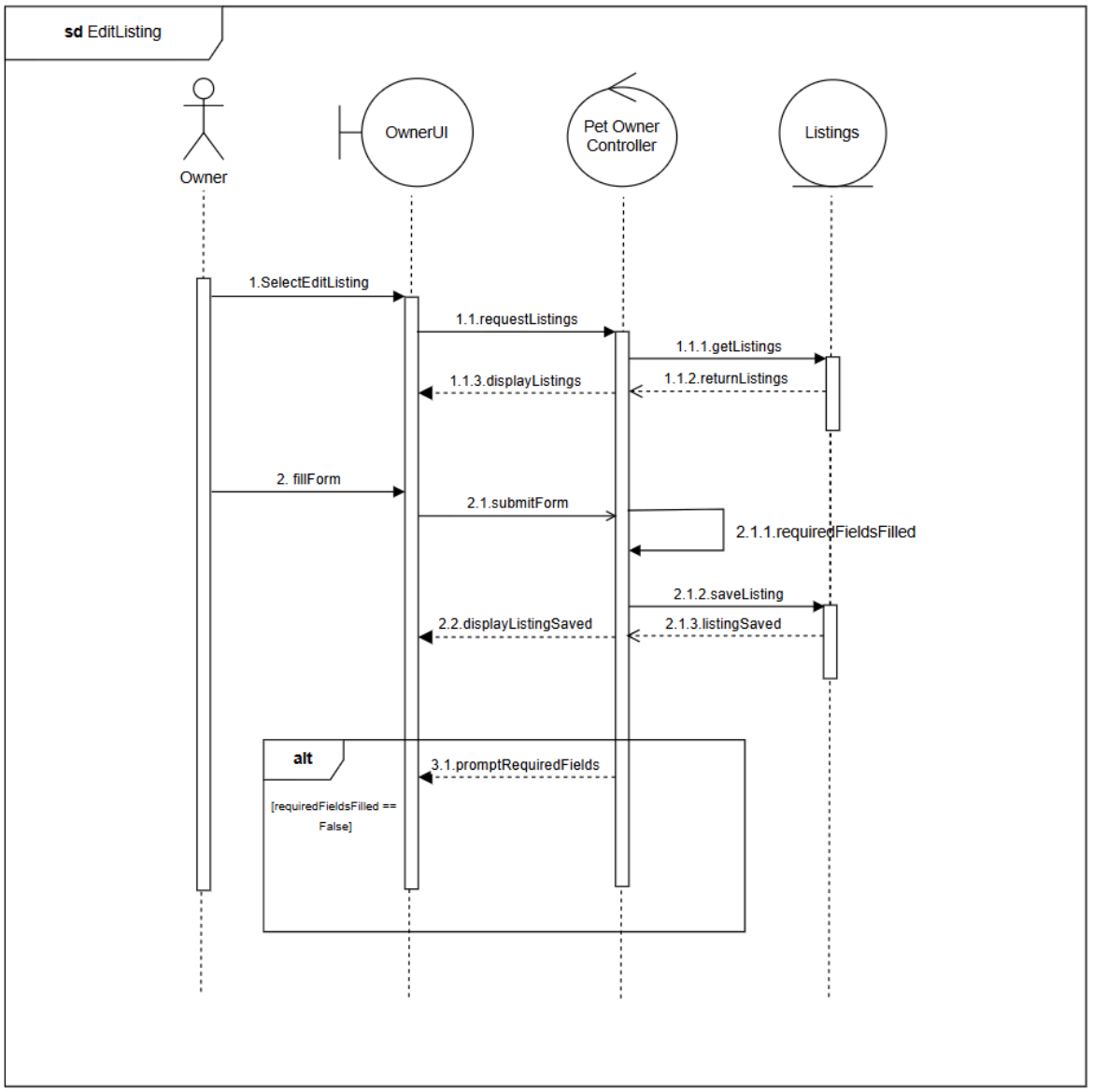

sd CreateListing
Owner
OwnerUI
Pet Owner Controller
Listings
alt
[CreateAdoptListing]
1.CreateAdoptListing
ref
Create Adopt Listing
[CreateLostPetListing]
2.CreateLostPetListing
ref
Create LostPetListing

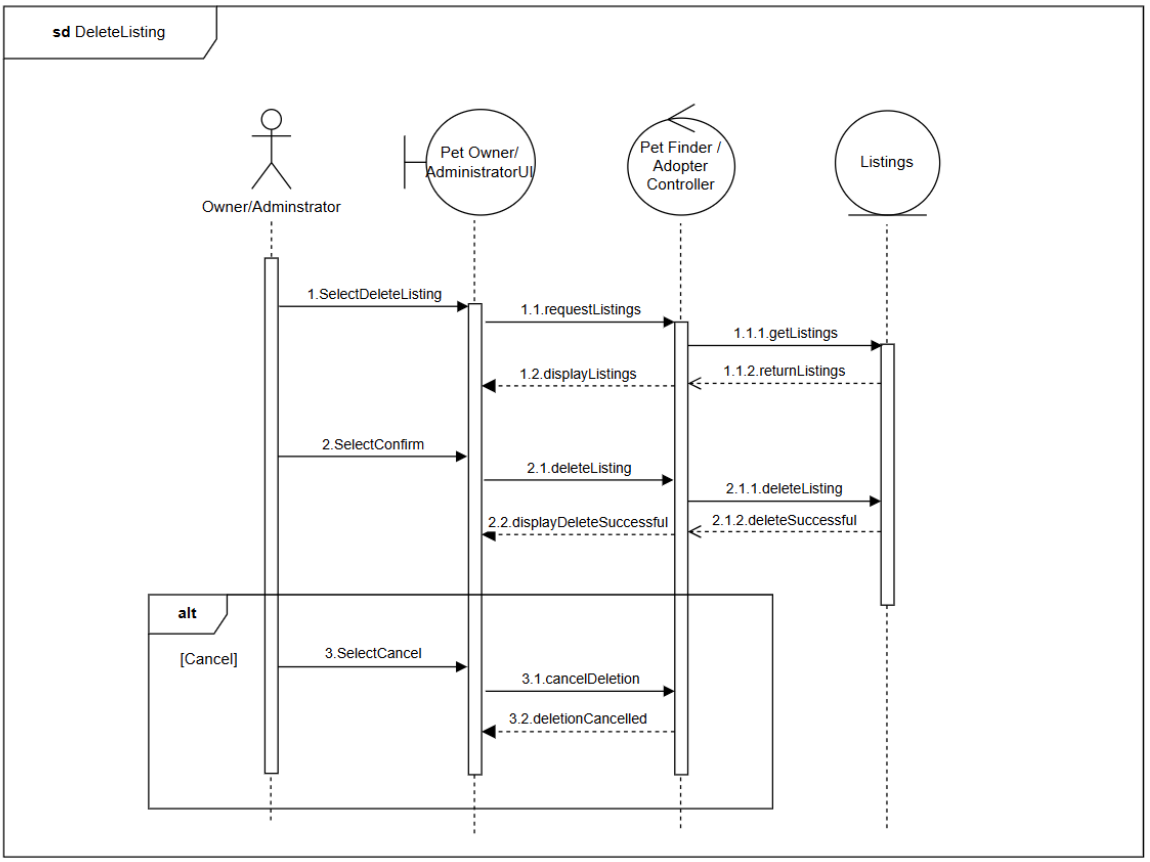

sd CreateAdoptListing
Owner
Pet Owner UI
Pet Owner Controller
Listings
1.CreateAdoptListing
1.1.requestForm
1.2.displayForm
2. fillForm
2.1.submitForm
2.1.1.requiredFieldsFilled
2.1.2.saveListing
2.1.3.listingSaved
2.2.displayListingSaved
alt
[requiredFieldsFilled == False]
3.1.promptRequiredFields

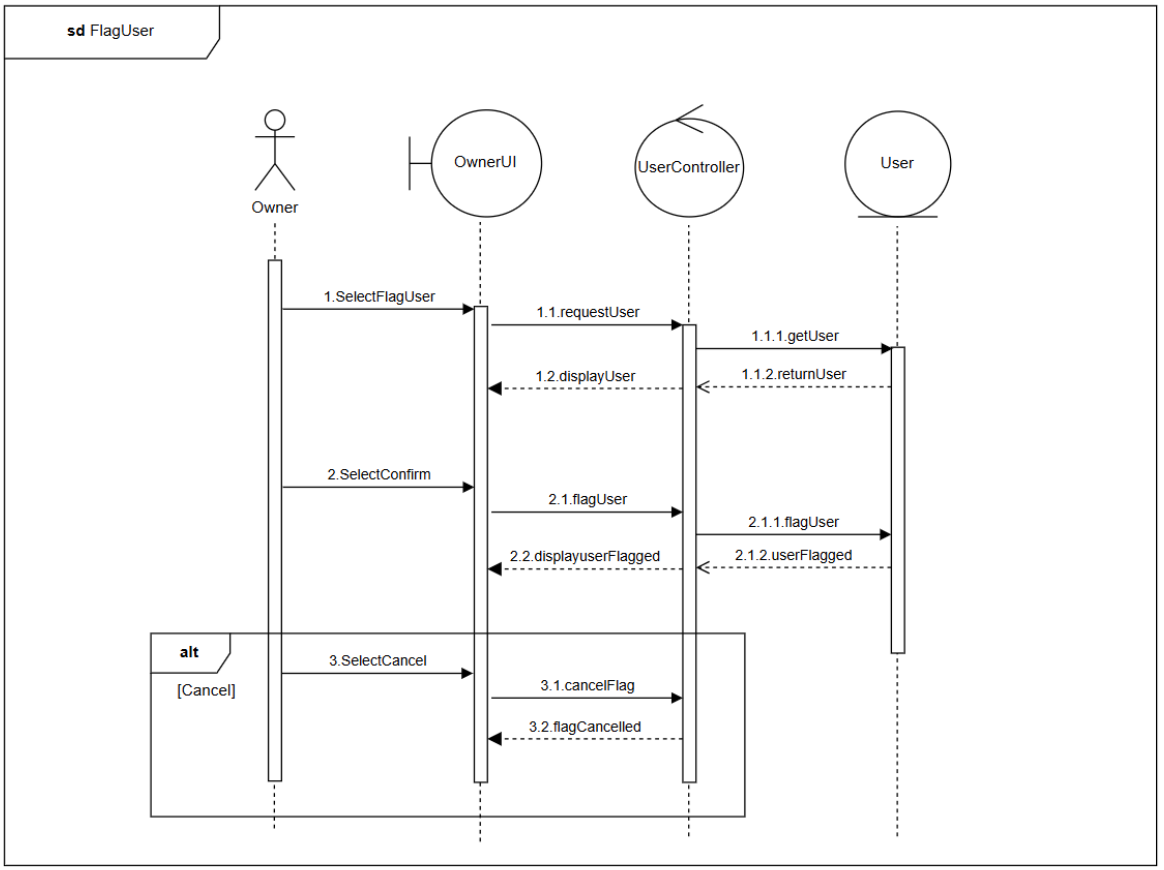

sd CreateLostPetListing
Owner
PetOwner UI
Pet Owner Controller
Listings
1.CreateLostPetListing
1.1.requestForm
1.2.displayForm
2. fillForm
2.1.submitForm
2.1.1.requiredFieldsFilled
2.1.2.saveListing
2.1.3.listingSaved
2.2.displayListingSaved
alt
[requiredFieldsFilled == False]
3.1.promptRequiredFields

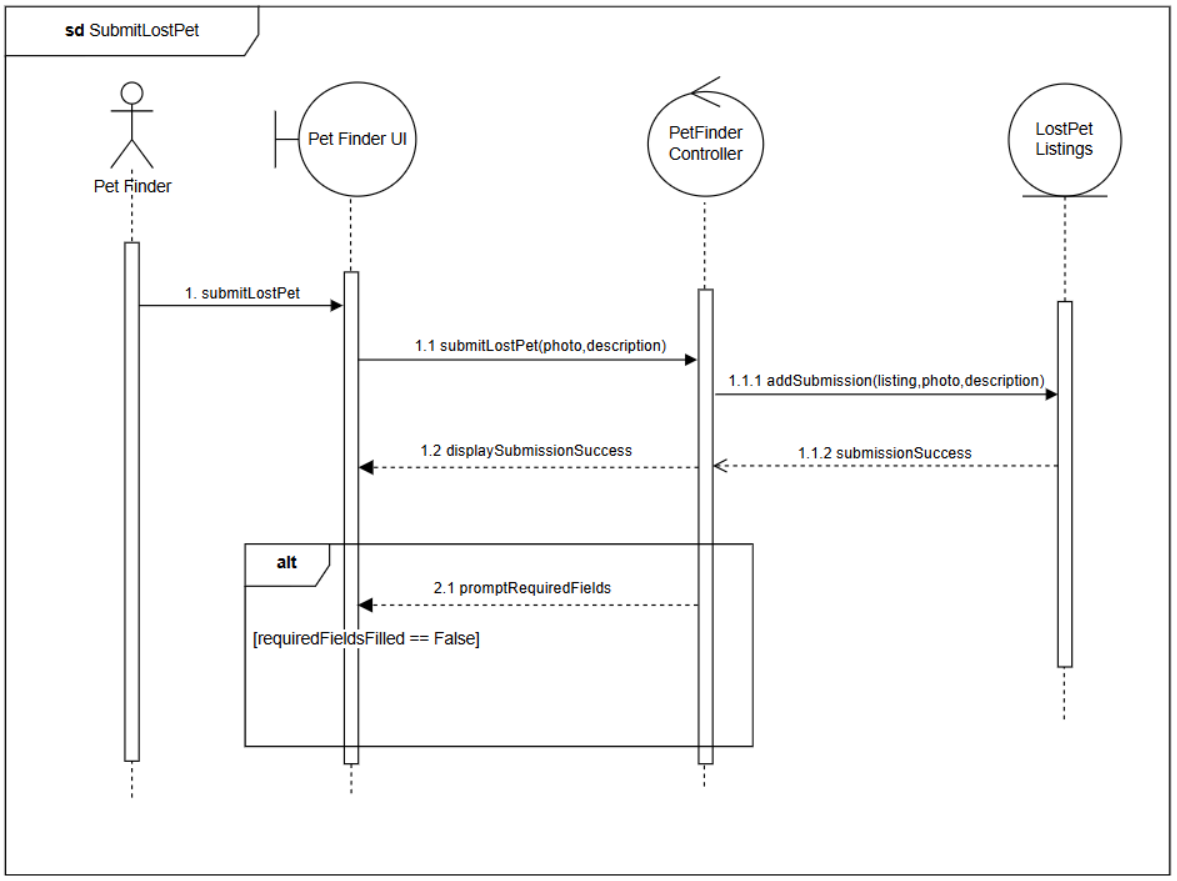

sd ManageMyListings
Owner
PetOwnerUI
Pet Owner Controller
Listings
alt
[View Listing]
1.SelectViewListing
ref
View Listing
[Edit Listing]
2.SelectEditListing
ref
Edit Lisiting
[Delete Listing]
3.SelectDeleteListing
ref
Delete Listing

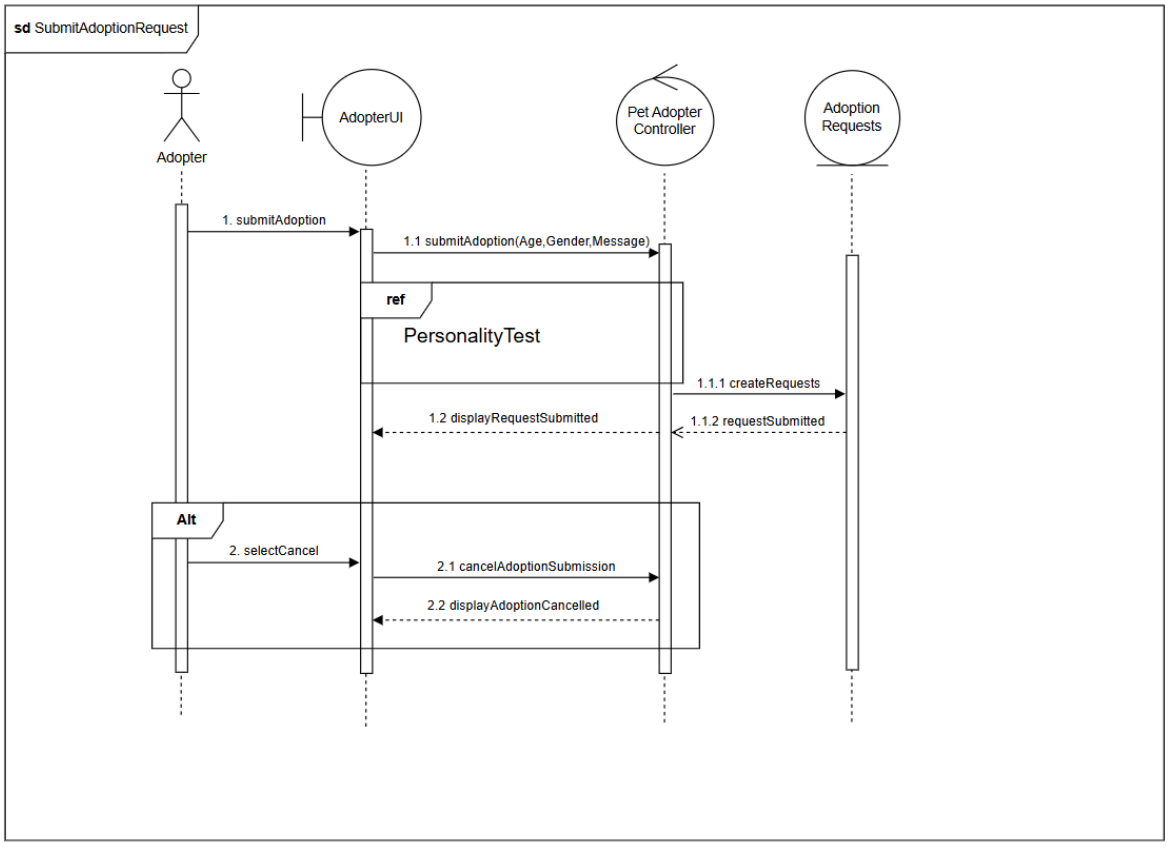

sd ViewListing
Owner
OwnerUI
PetOwner Controller
Listings
1.SelectViewListing
1.1.requestListings
1.1.1.getListings
1.2.displayListings
1.1.2.returnListings

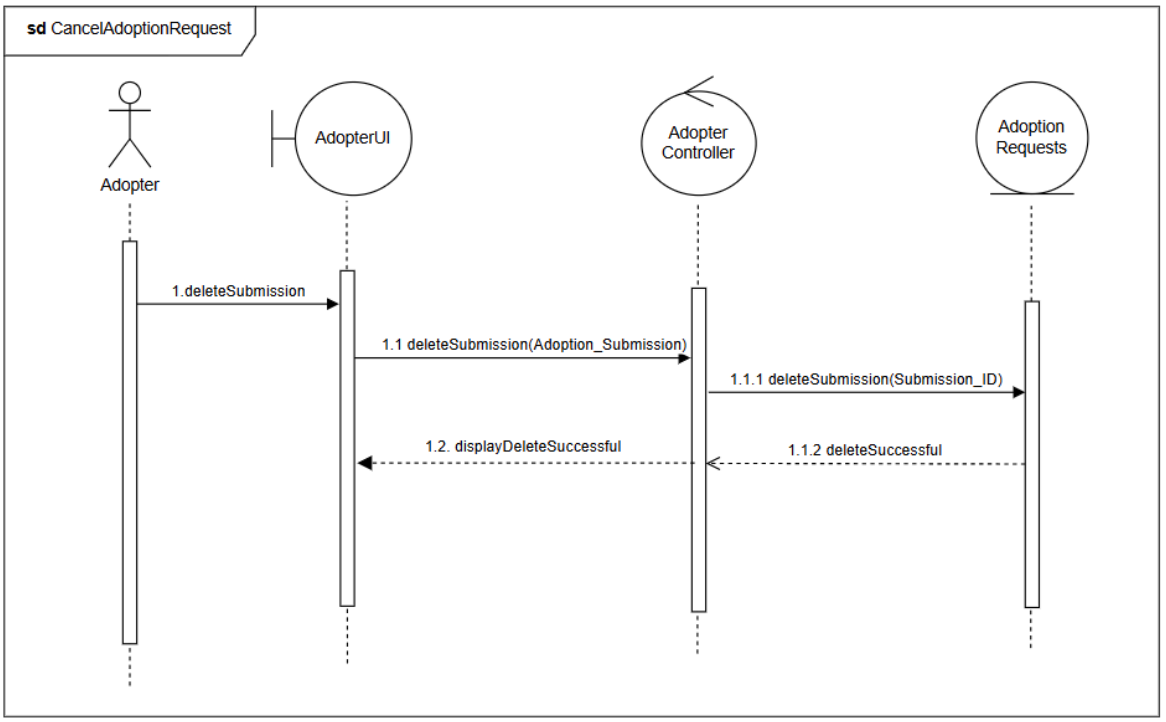

sd EditListing
Owner
OwnerUI
Pet Owner Controller
Listings
1.SelectEditListing
1.1.requestListings
1.1.1.getListings
1.1.3.displayListings
1.1.2.returnListings
2. fillForm
2.1.submitForm
2.1.1.requiredFieldsFilled
2.1.2.saveListing
2.2.displayListingSaved
2.1.3.listingSaved
alt
[requiredFieldsFilled == False]
3.1.promptRequiredFields

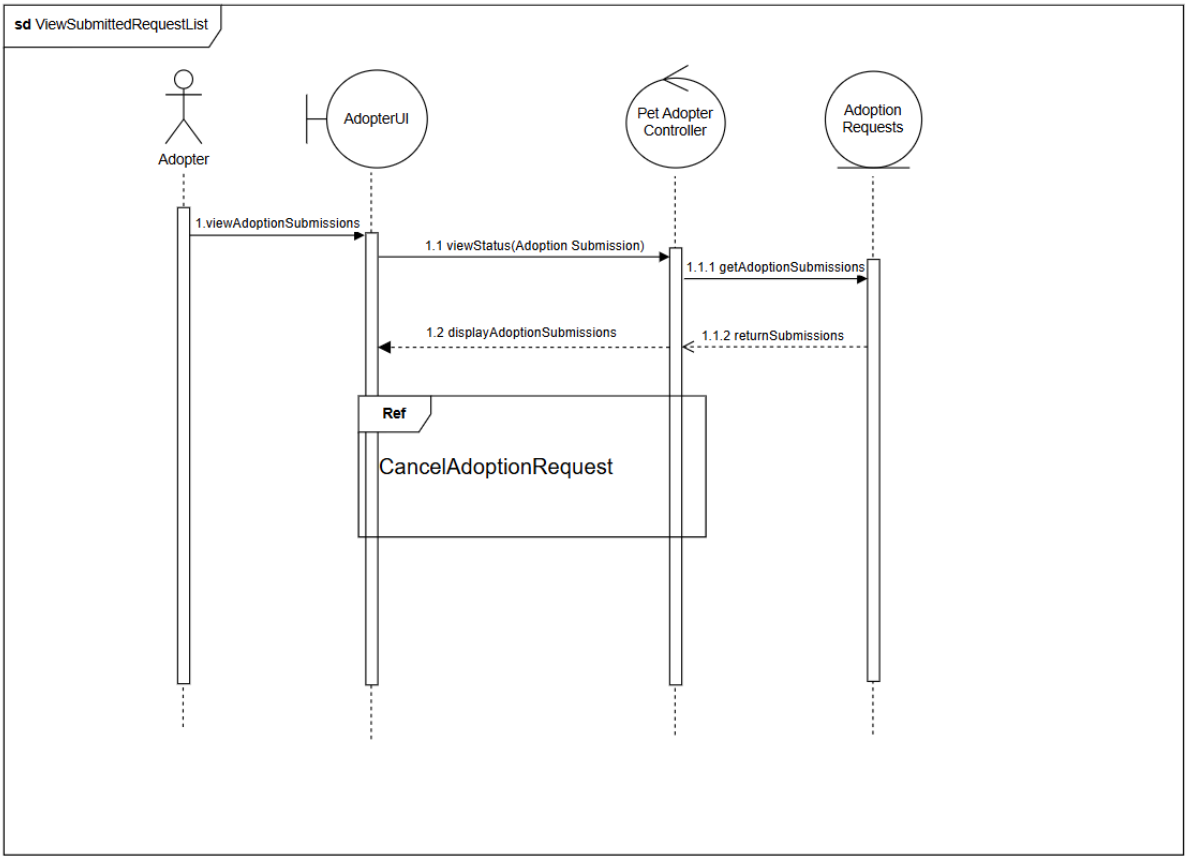

sd DeleteListing
Owner/Adminstrator
Pet Owner/ AdministratorUI
Pet Finder / Adopter Controller
Listings
1.SelectDeleteListing
1.1.requestListings
1.1.1.getListings
1.2.displayListings
1.1.2.returnListings
2.SelectConfirm
2.1.deleteListing
2.1.1.deleteListing
2.2.displayDeleteSuccessful
2.1.2.deleteSuccessful
alt
[Cancel]
3.SelectCancel
3.1.cancelDeletion
3.2.deletionCancelled

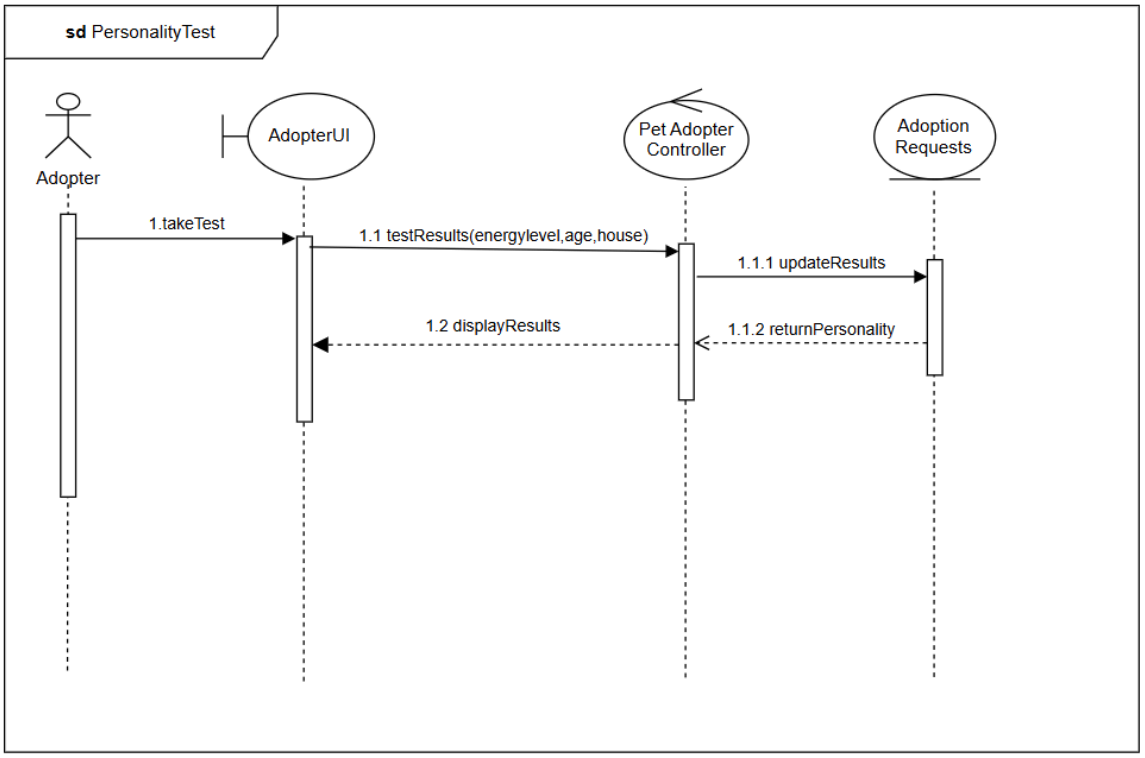

sd FlagUser
Owner
OwnerUI
UserController
User
1.SelectFlagUser
1.1.requestUser
1.1.1.getUser
1.2.displayUser
1.1.2.returnUser
2.SelectConfirm
2.1.flagUser
2.1.1.flagUser
2.2.displayuserFlagged
2.1.2.userFlagged
alt
[Cancel]
3.SelectCancel
3.1.cancelFlag
3.2.flagCancelled

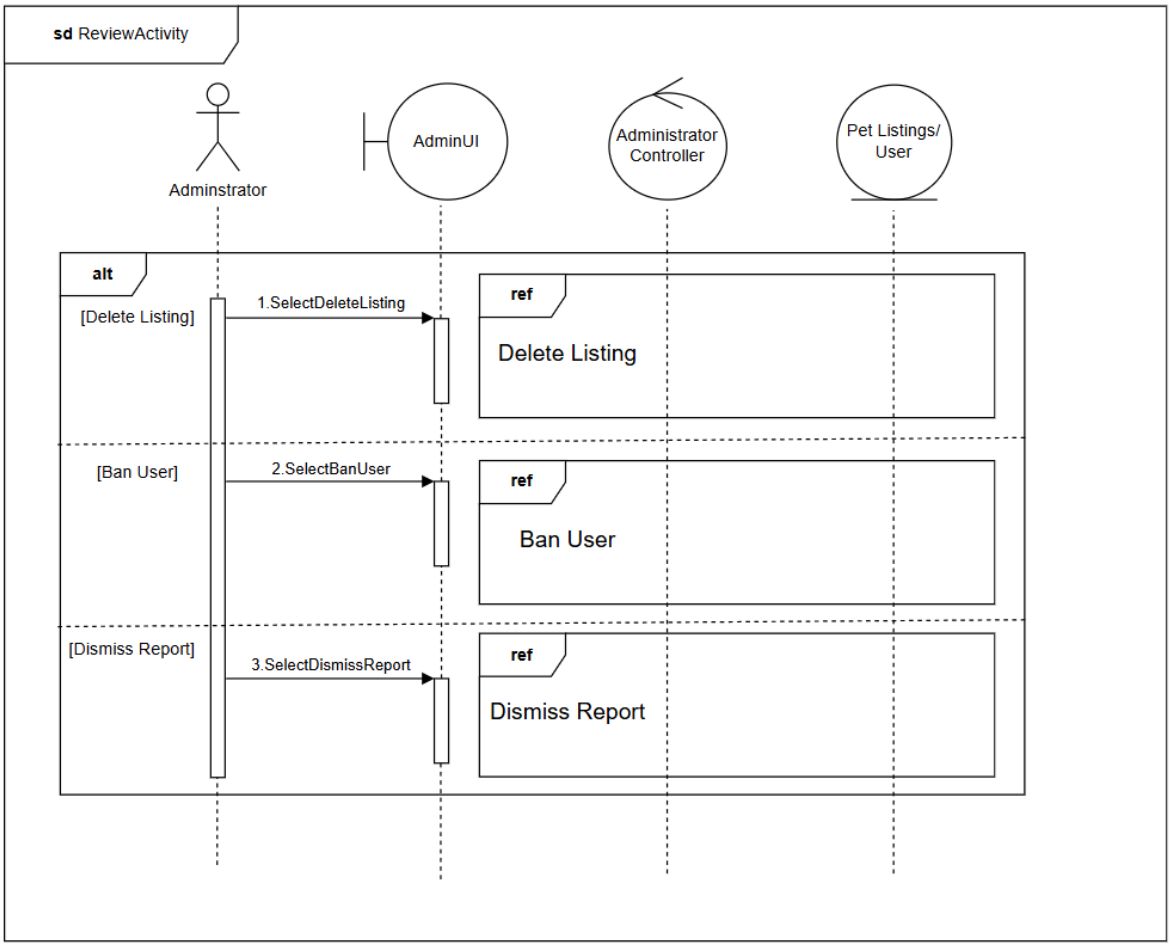

sd SubmitLostPet
Pet Finder
Pet Finder UI
PetFinder Controller
LostPet Listings
1. submitLostPet
1.1 submitLostPet(photo,description)
1.1.1 addSubmission(listing,photo,description)
1.2 displaySubmissionSuccess
1.1.2 submissionSuccess
alt
2.1 promptRequiredFields
[requiredFieldsFilled == False]

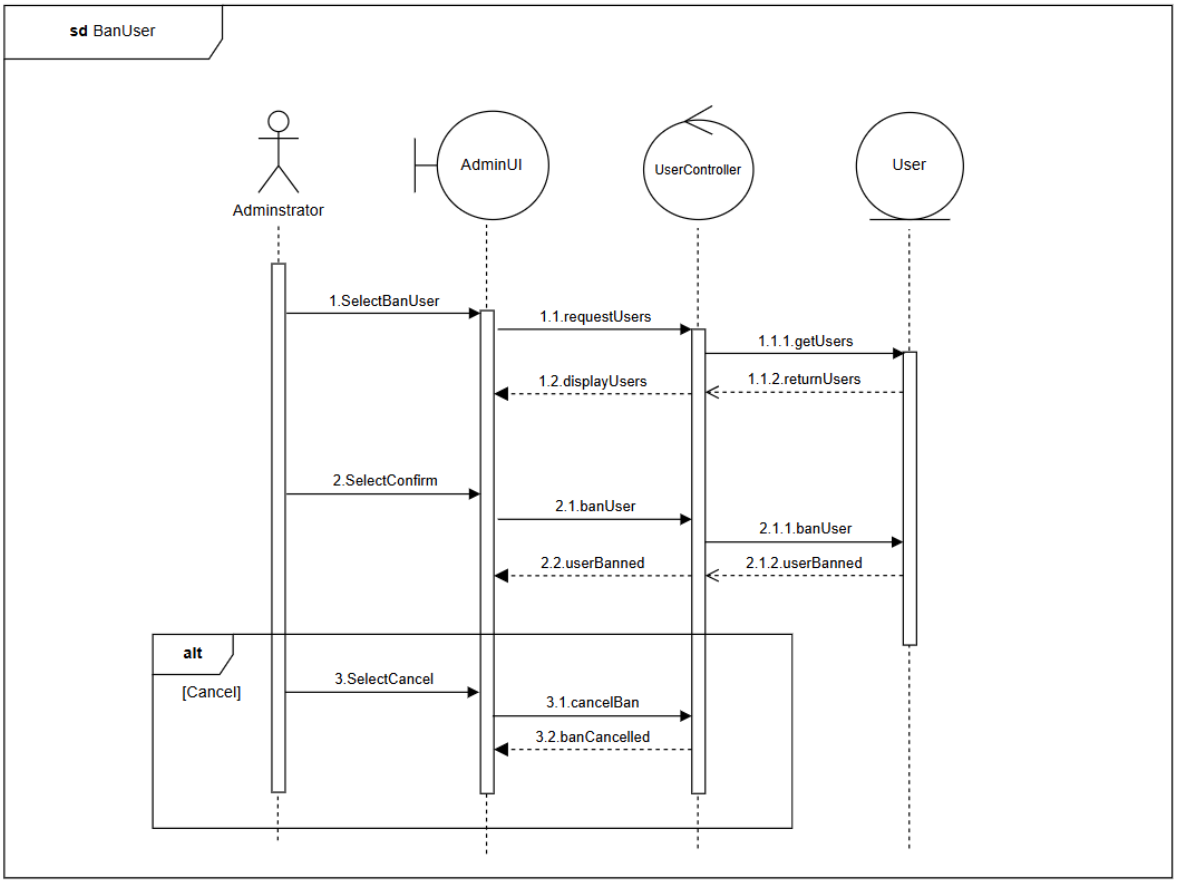

sd SubmitAdoptionRequest
Adopter
AdopterUI
Pet Adopter Controller
Adoption Requests
1. submitAdoption
1.1 submitAdoption(Age,Gender,Message)
ref
PersonalityTest
1.1.1 createRequests
1.2 displayRequestSubmitted
1.1.2 requestSubmitted
Alt
2. selectCancel
2.1 cancelAdoptionSubmission
2.2 displayAdoptionCancelled

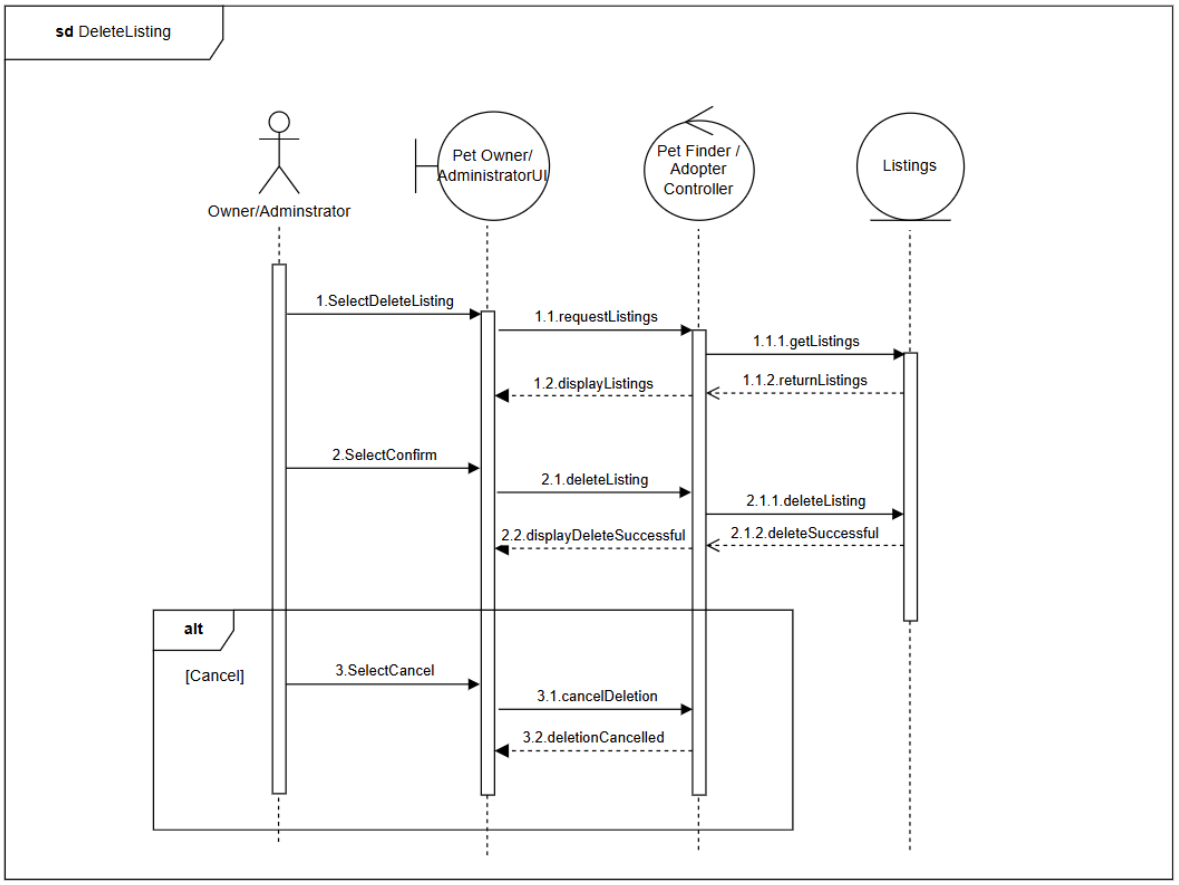

sd CancelAdoptionRequest
Adopter
AdopterUI
Adopter Controller
Adoption Requests
1.deleteSubmission
1.1 deleteSubmission(Adoption_Submission)
1.1.1 deleteSubmission(Submission_ID)
1.2. displayDeleteSuccessful
1.1.2 deleteSuccessful

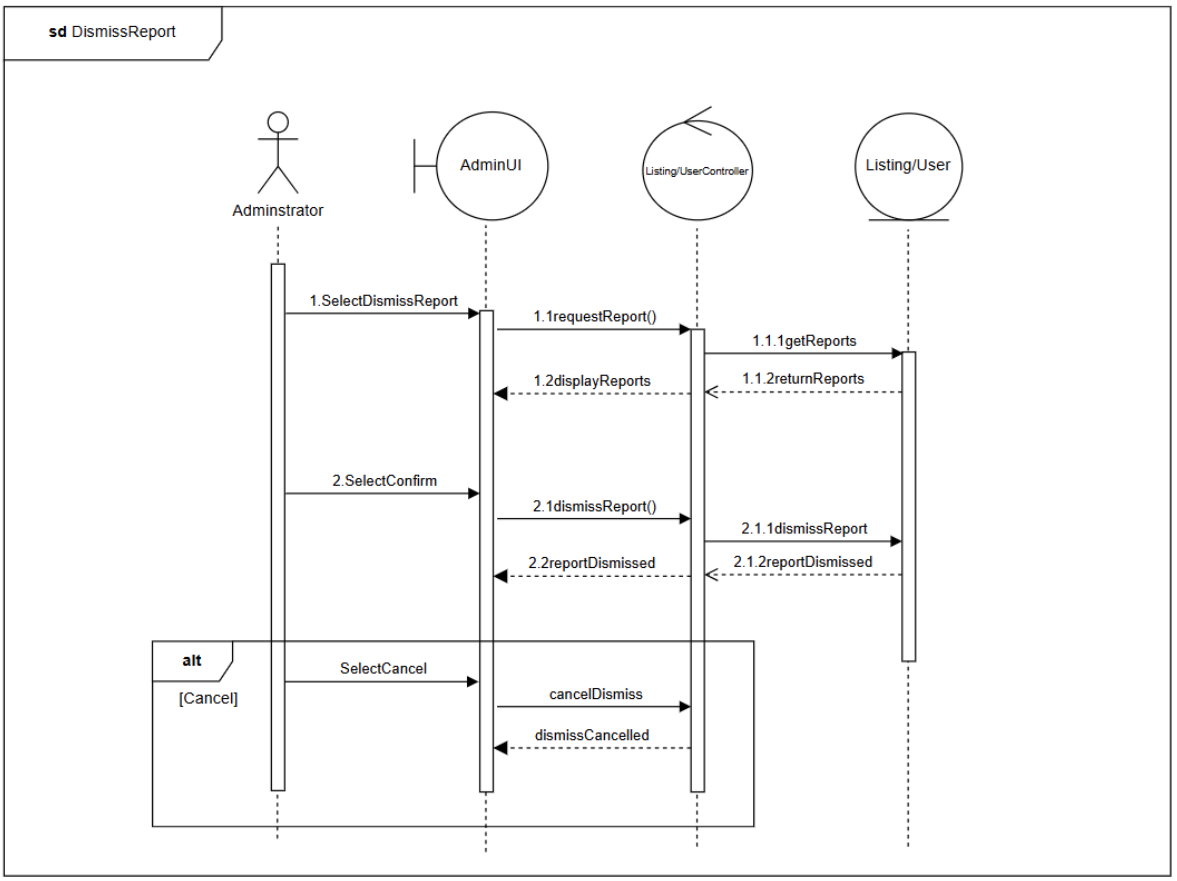

sd ViewSubmittedRequestList
Adopter
AdopterUI
Pet Adopter Controller
Adoption Requests
1.viewAdoptionSubmissions
1.1 viewStatus(Adoption Submission)
1.1.1 getAdoptionSubmissions
1.2 displayAdoptionSubmissions
1.1.2 returnSubmissions
Ref
CancelAdoptionRequest

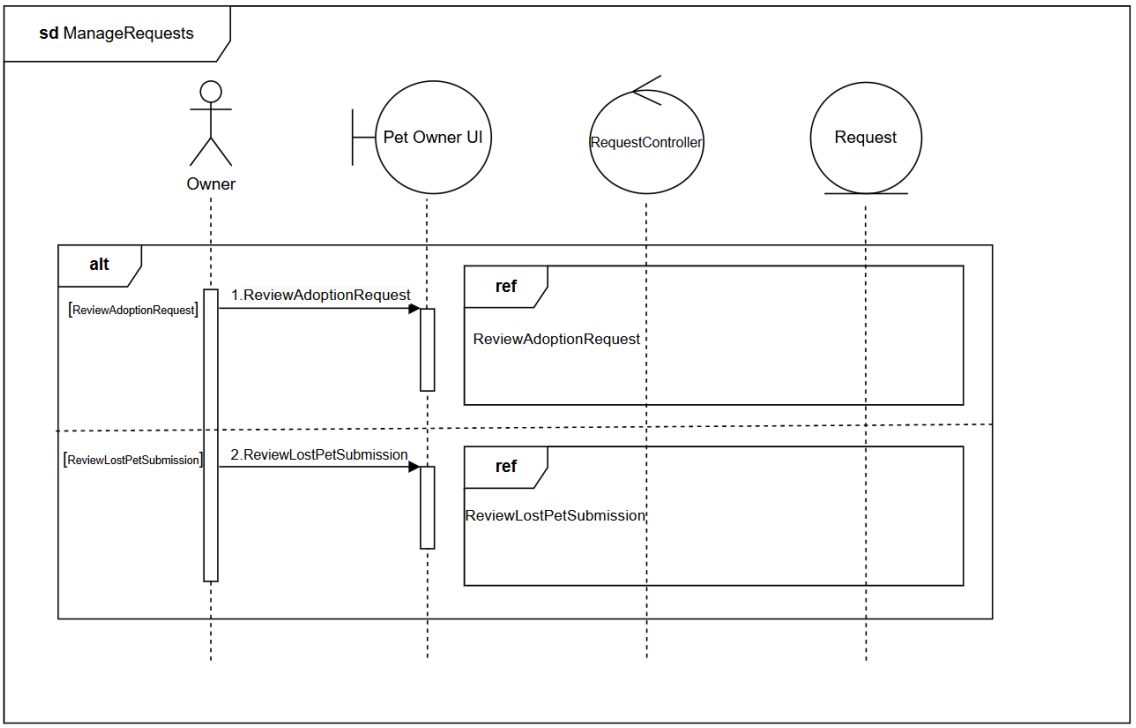

sd PersonalityTest
Adopter
AdopterUI
Pet Adopter Controller
Adoption Requests
1.takeTest
1.1 testResults(energylevel,age,house)
1.1.1 updateResults
1.2 displayResults
1.1.2 returnPersonality


sd ReviewActivity
Adminstrator
AdminUI
Administrator Controller
Pet Listings/ User
alt
[Delete Listing]
1.SelectDeleteListing
ref
Delete Listing
[Ban User]
2.SelectBanUser
ref
Ban User
[Dismiss Report]
3.SelectDismissReport
ref
Dismiss Report

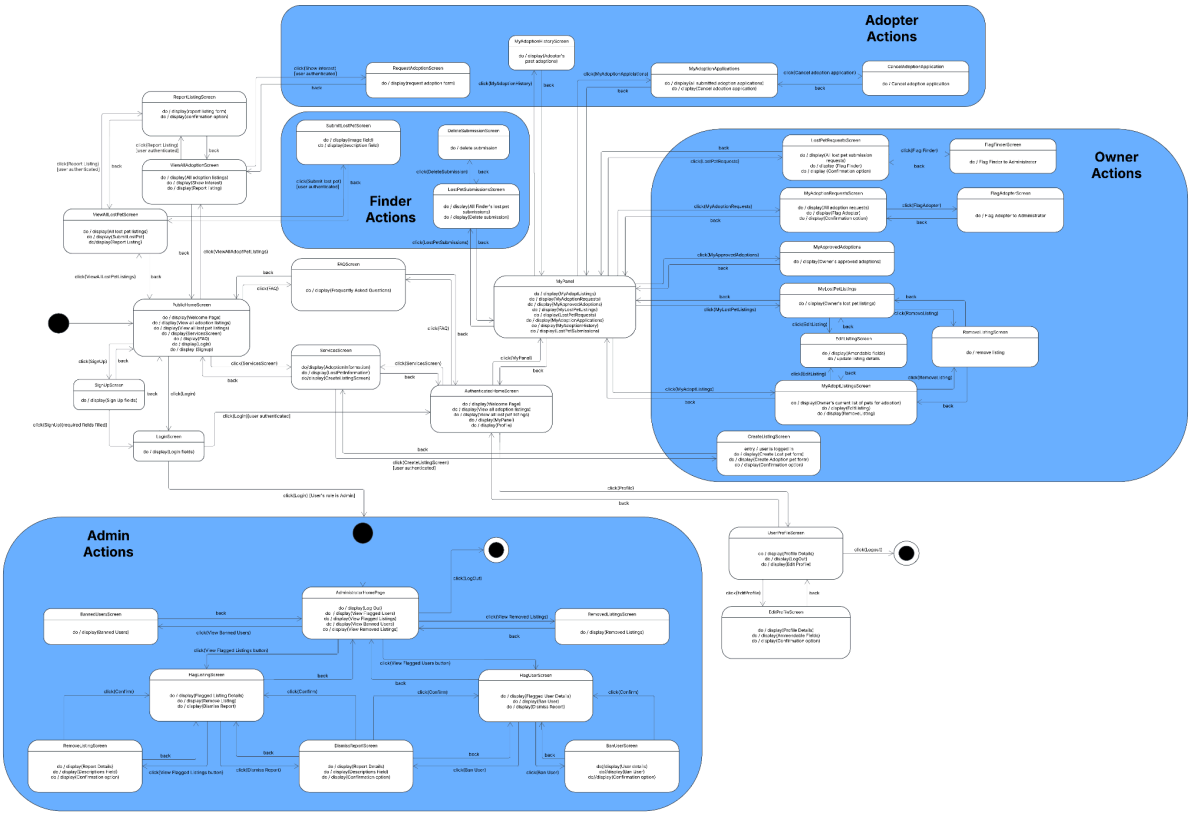

sd BanUser
Adminstrator
AdminUI
UserController
User
1.SelectBanUser
1.1.requestUsers
1.1.1.getUsers
1.2.displayUsers
1.1.2.returnUsers
2.SelectConfirm
2.1.banUser
2.1.1.banUser
2.2.userBanned
2.1.2.userBanned
alt
[Cancel]
3.SelectCancel
3.1.cancelBan
3.2.banCancelled


sd DeleteListing
Owner/Adminstrator
Pet Owner/ AdministratorUI
Pet Finder / Adopter Controller
Listings
1.SelectDeleteListing
1.1.requestListings
1.1.1.getListings
1.2.displayListings
1.1.2.returnListings
2.SelectConfirm
2.1.deleteListing
2.1.1.deleteListing
2.2.displayDeleteSuccessful
2.1.2.deleteSuccessful
alt
[Cancel]
3.SelectCancel
3.1.cancelDeletion
3.2.deletionCancelled


sd DismissReport
Adminstrator
AdminUI
Listing/UserController
Listing/User
1.SelectDismissReport
1.1requestReport()
1.1.1getReports
1.2displayReports
1.1.2returnReports
2.SelectConfirm
2.1dismissReport()
2.1.1dismissReport
2.2reportDismissed
2.1.2reportDismissed
alt
[Cancel]
SelectCancel
cancelDismiss
dismissCancelled

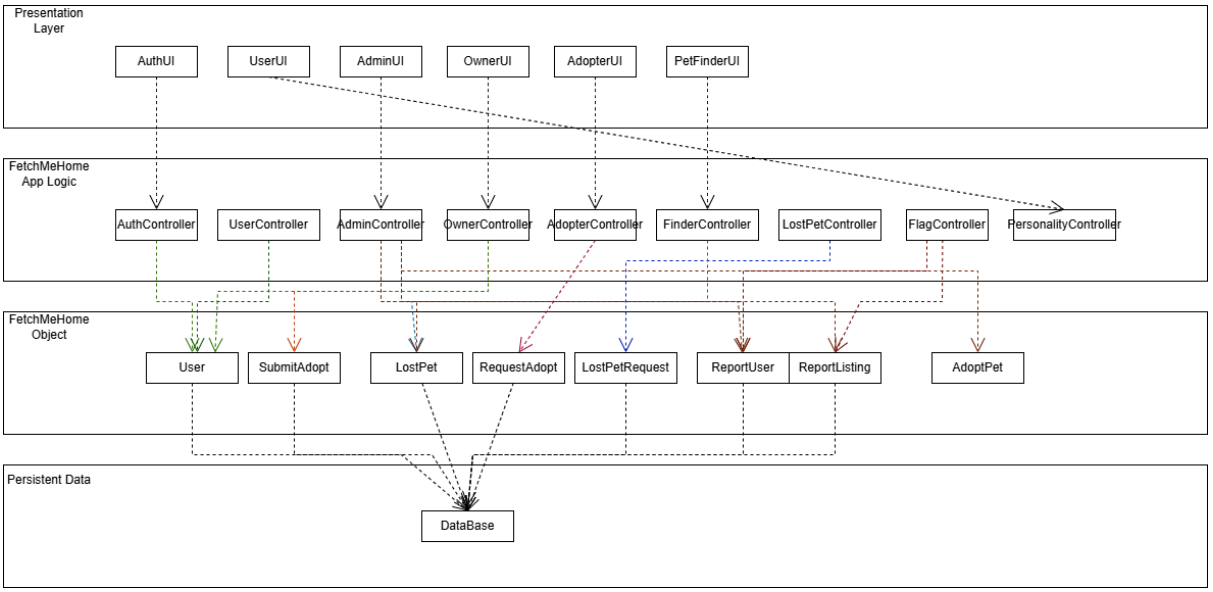

sd ManageRequests
Owner
Pet Owner UI
RequestController
Request
alt
[ReviewAdoptionRequest]
1.ReviewAdoptionRequest
ref
ReviewAdoptionRequest
[ReviewLostPetSubmission]
2.ReviewLostPetSubmission
ref
ReviewLostPetSubmission

C. Initial Dialogue Map



Refer to the link below if Initial Dialog Map unclear:
https://lucid.app/lucidchart/cc80998b-7264-4f82-8751-bafe2cc27222/edit?page=0_0#

3. System Architecture



Presentation Layer

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

1. **Login UI**
    - Allows users to log in with an email and password.
2. **Main UI**
    - Uses services from UserController and AuthenticatedUserController.
3. **Admin UI**
    - Accessible only to admins. Allows administrators to manage reports, ban users, and remove inappropriate listings.
4. **Authenticated User UI**
    - Allows authenticated users to post pet listings (adoption or lost pets), report lost pets, and report users.
5. **User UI**
    - Allows general users to view pet listings but not interact.

Application Logic Layer

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

1. **AuthController**: Handles user authentication (create account and log in). Accesses the User entity.

2. **UserController**: Manages general user actions and allows users to view the list of pets put up for adoption as well as pets reported as lost.

3. **AdminController**: Allows admins to review flagged users/listings, process reports, ban users, and remove listings. Accesses LostPet, AdoptPet, and Report.

4. **OwnerController**: Manages pet listings. Includes the ability to post, edit, and remove listings. Accesses LostPet and AdoptPet.

5. **AdopterController**: Manages Pet Adoption actions such as submitting adoption requests and viewing submitted requests. Access RequestAdopt.

6. **FinderController**: Manages Pet Found actions such as submitting found pets.

7. **AdoptPetController**: Manages the list of pets put up for adoption (eg. create adoption listing, edit adoption listing, delete adoption listing). Accesses AdoptPet.

8. **LostPetController**: Manages the list of lost pets (eg. create lost pet listing, edit lost pet listing, delete lost pet listing). Accessed LostPet.

9. **FlagController**: Manages the list of flagged users and listings(eg. allows authenticated Users to report other Users and Listings). Accesses LostPet and AdoptPet.

10. **PersonalityController**: Manages the calls of OllamaAPI and AnimalAPI to determine which type/breed of pets are suitable for Users.

App Object Layer

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

1. **AuthenticatedUser**
    - Represents all users who have created an account. It also contained administrator accounts.

2. **AdoptPetListing**
    - Represents pets being put up for adoption and are available for pet adopters to adopt.

3. **LostPetListing**
    - Represents pets that are reported as lost and are awaiting for pet finders to find.

4. **AdoptPetRequest**
    - Represents all adoption requests submitted by pet adopters.

5. **LostPetSubmission**
    - Represents all lost pet submissions submitted by pet finders.

6. **ReportedListings**
    - Stores flagged or reported listings.

7. **ReportedUsers**
    - Stores information about flagged users.

Persistent Data Layer

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

4. Application Skeleton

a. Client

i. 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.

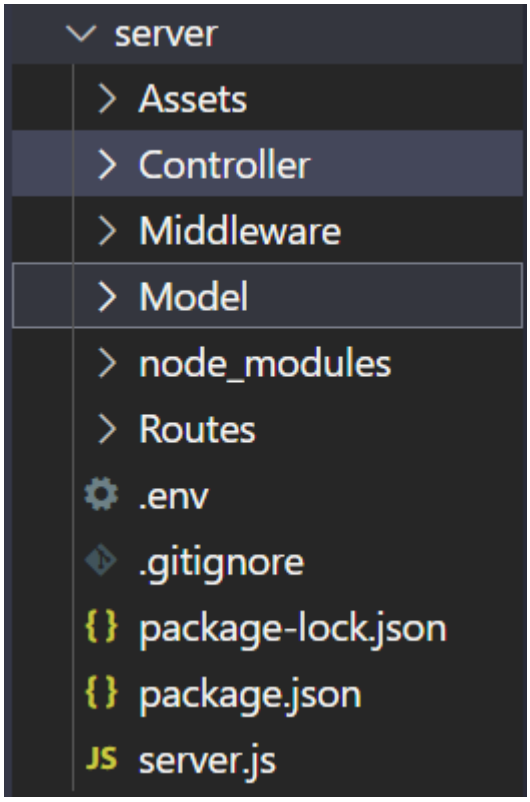
Navigation is a folder where we have our frontend routes configurations.This is where we manage how users navigate between different pages or components in the App.

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

App.js is the entry point for the Client.

b. Server

i. Built with Express.js and Node.js
ii. 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.

5. Appendix

Key Design Issues

A. Identifying and storing persistent data

- **Non-Relational Database (Models/Entity)**

    - **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
},
```

B. Providing Access Control

- Passwords are encrypted before being stored.
- 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	-	-

C. Security Considerations

- Use JWT authentication for secure logins.
- Implement input validation to prevent malicious data injection.

D. Tech Stack

- **Frontend:** React
- **Backend:** Express.js / Node.js
- **Database:** MongoDB
- **APIs Used:** OneMap API, Ollama API, Animal API