Table of Content

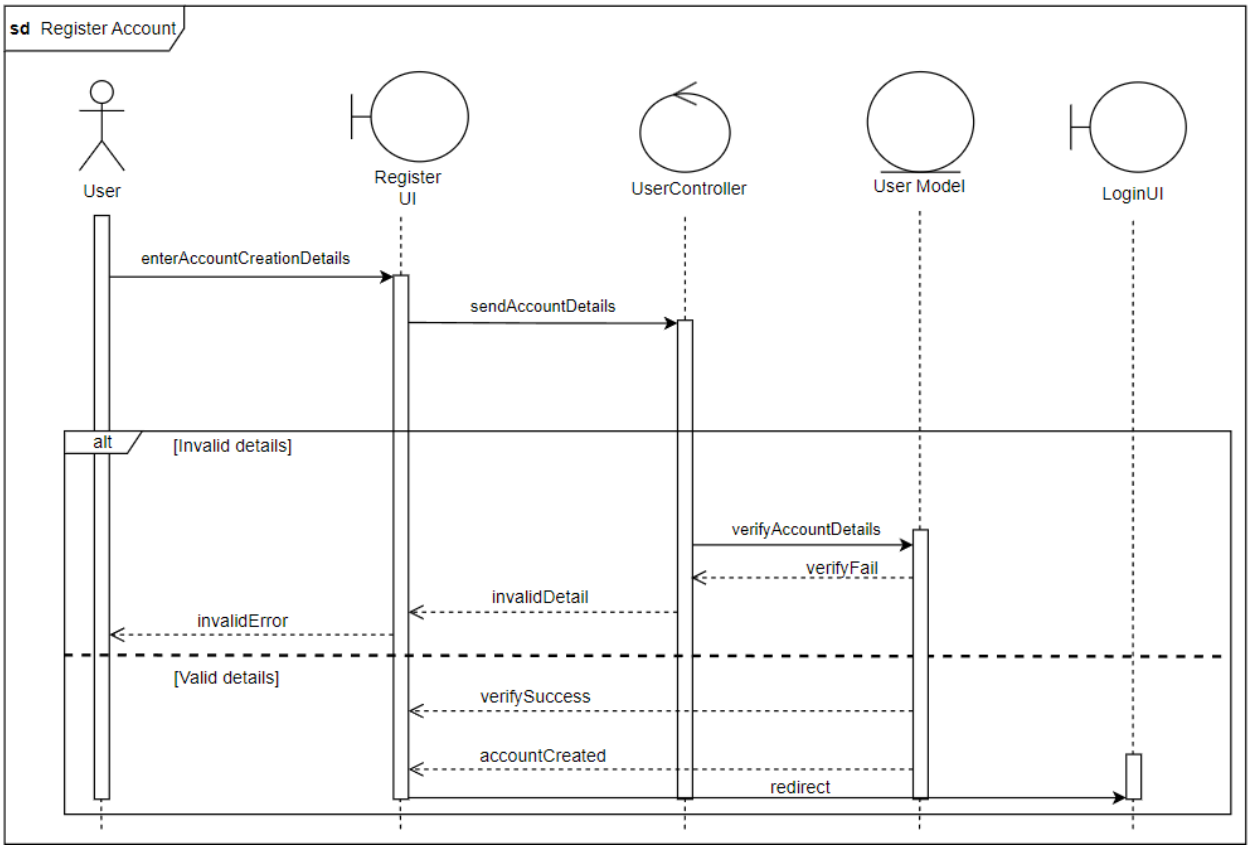
Use Case Descriptions

Register

Use Case ID:	UC_REC_1		
Use Case Name:	Register Account		
Created By:	Keng Jia Chi	Last Updated By:	Keng Jia Chi
Date Created:	25th August 2024	Date Last Updated:	25th August 2024

Actor:	User
Description:	Allows first time users to create an account in PetCare application.
Preconditions:	1. The email address given is not registered in the system.
Postconditions:	An account is successfully created for the user.
Priority:	High
Frequency of Use:	High
Flow of Events:	1. The system allows the user to register with Email and password. 2. The user chooses to register with email and password. 3. The system requests the user to input the following information fields: a. Name b. Email address c. Password d. Confirm Password 4. The user selects the “Register” button. 5. The system validates the required fields. 6. If the user’s details are valid, a PetCare account will be successfully created. 7. The system redirects the user to the PetCare application login page.
Alternative Flows:	AF-S6: If email address is already in database 1. The system will display “The email is already in use. Please use a different email.”. 2. System returns to Step 3. AF-S6: If the password does not meet the complexity requirements. 1. The system will display “Password must contain at least one uppercase letter, one lowercase letter, one numeric character, and one special character.”. 2. System returns to Step 3. AF-S6: If the password does not meet the length requirements. 1. The system will display “Password must be at least 6 characters long.”. 2. System returns to Step 3. AF-S6: If password is not the same as confirm password. 1. The system will display Password and confirm password do not match.”. 2. System returns to Step 3. AF-S6: If the user did not complete all the required fill. 1. The system will display "Please fill in all the fields.”. 2. System returns to Step 3.

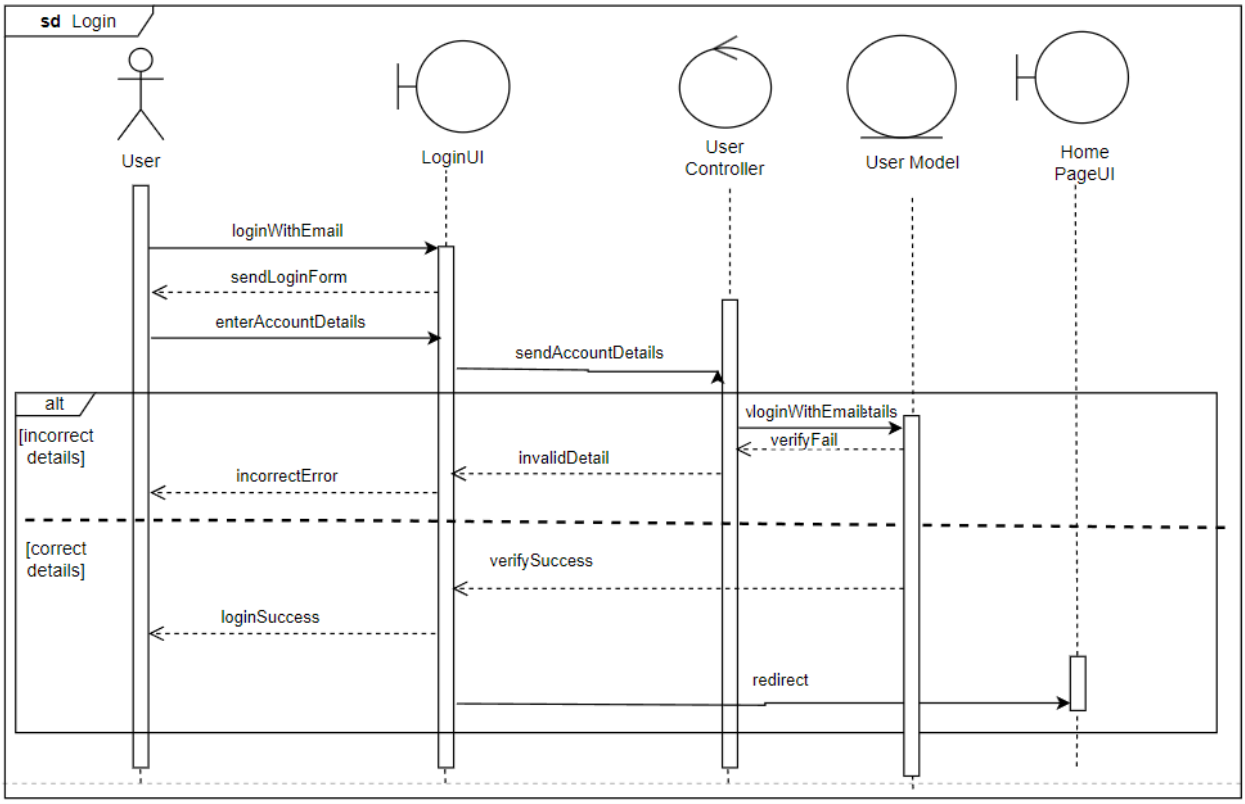
Exceptions:	None
Includes:	None
Special Requirements:	The system needs to validate user input data.
Assumptions:	None
Notes and Issues:	None



Login

Use Case ID:	UC_LOGIN_1		
Use Case Name:	Login		
Created By:	Keng Jia Chi	Last Updated By:	Keng Jia Chi
Date Created:	25th August 2024	Date Last Updated:	25th August 2024

Actor:	User
Description:	Allows users to login into their PetCare account using their email and password.
Preconditions:	User has registered for an account.
Postconditions:	User is logged into the PetCare application and is navigated to the home screen of the application.
Priority:	High
Frequency of Use:	High
Flow of Events:	1. The system allows the user to login with Email and password. 2. The user inputs his/her email and password. 3. The user selects the “Login” button. 4. If the user’s login credentials are valid, the system will direct the user to PetCare’s home page.
Alternative Flows:	AF-S3: If the user’s login credentials are invalid 1. The system shall display “Incorrect email/password. Please try again.” to the user. 2. The system returns to Step 1. AF-S3: If the user’s did not provide either email or password or both 1. The system shall display “Please fill in both email and password.” to the user. 2. The system returns to Step 1.
Exceptions:	None
Includes:	None
Special Requirements:	None
Assumptions:	The user has an existing PetCare account
Notes and Issues:	None

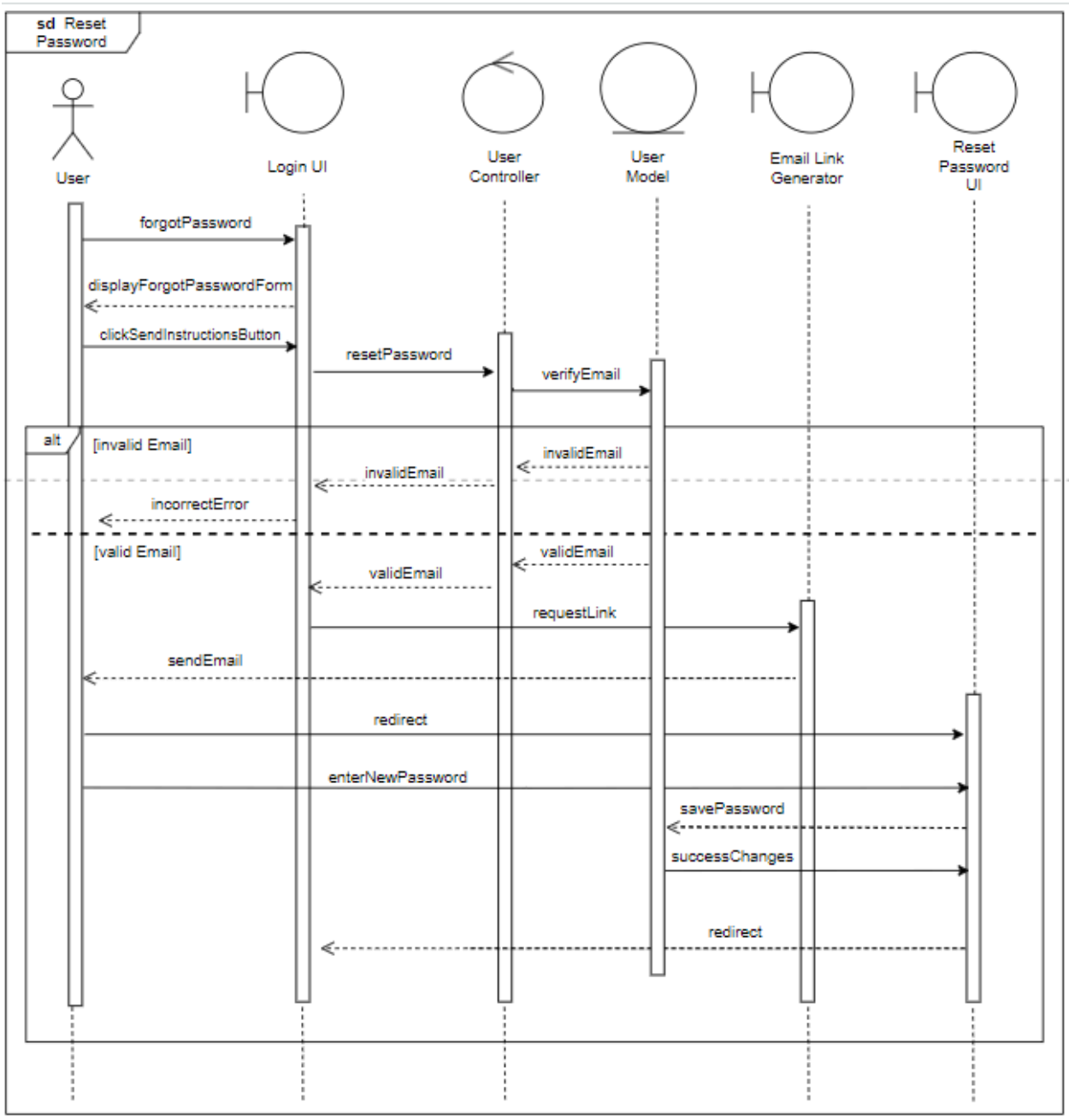

sd Login
User
LoginUI
User Controller
User Model
Home PageUI
loginWithEmail
sendLoginForm
enterAccountDetails
sendAccountDetails
alt
[incorrect details]
vloginWithEmailtails
verifyFail
invalidDetail
incorrectError
[correct details]
verifySuccess
loginSuccess
redirect

Reset Password

Use Case ID:	UC_RP_1		
Use Case Name:	Reset Password		
Created By:	Keng Jia Chi	Last Updated By:	Keng Jia Chi
Date Created:	25th August 2024	Date Last Updated:	25th August 2024

Actor:	User
Description:	Allows users to reset their password when they have forgotten it or need to change it for security reasons.
Preconditions:	1. The user has an existing PetCare account. 2. The user must have access to the email address associated with their PetCare account. 3. The user must have access to their email account.
Postconditions	1. The user password is successfully reset. 2. The user can log in with the new password
Priority:	High
Frequency of Use:	Moderate
Flow of Events:	1. The user clicks on the “Forgot Password” button. 2. The system displays the password reset page. 3. The user enters their registered email address. 4. If the email address is valid, the system sends a password reset link to the user's email. 5. The user clicks on the reset link in their email. 6. The system displays a new password creation page. 7. The user enters a new password and confirms it. 8. If the password meets the complexity requirements, the system updates the user's password in the database. 9. The system shall display “Password reset successfully” to the user. 10. The user is redirected to the login page.
Alternative Flows:	AF-S4: The user’s email address is invalid 1. The system will display “Invalid email address” to the user. 2. The system will prompt the users to re-enter their email address. 3. The system returns to Step 3. AF-S8: The user’s password does not meet the complexity requirements 1. The system will display “Invalid password” with password guidelines to the user. 2. The system will prompt the users to enter a new password. 3. The system returns to Step 7.
Exceptions:	1. The password reset link expires before the user clicks on it.
Includes:	None
Special Requirements:	1. The password reset link should expire after 24 hours. 2. The new password must meet the system’s complexity

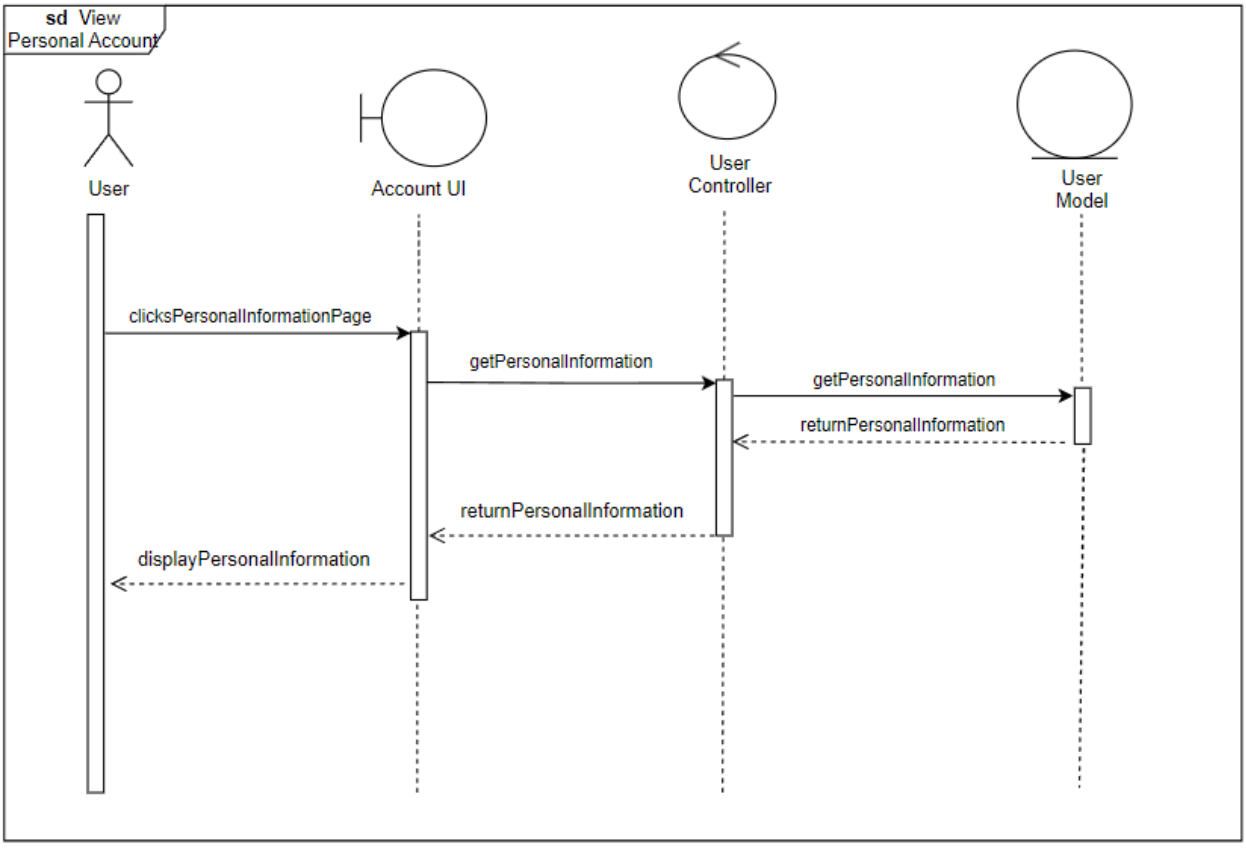
	requirements.
Assumptions:	1. The user has access to their registered email account.
Notes and Issues:	None


sd Reset Password
User
Login UI
User Controller
User Model
Email Link Generator
Reset Password UI
forgotPassword
displayForgotPasswordForm
clickSendInstructionsButton
resetPassword
verifyEmail
alt
[invalid Email]
invalidEmail
invalidEmail
incorrectError
[valid Email]
validEmail
validEmail
requestLink
sendEmail
redirect
enterNewPassword
savePassword
successChanges
redirect

View Personal Account

Use Case ID:	UC_ACC_1		
Use Case Name:	View Personal Account		
Created By:	Keng Jia Chi	Last Updated By:	Keng Jia Chi
Date Created:	25th August 2024	Date Last Updated:	25th August 2024

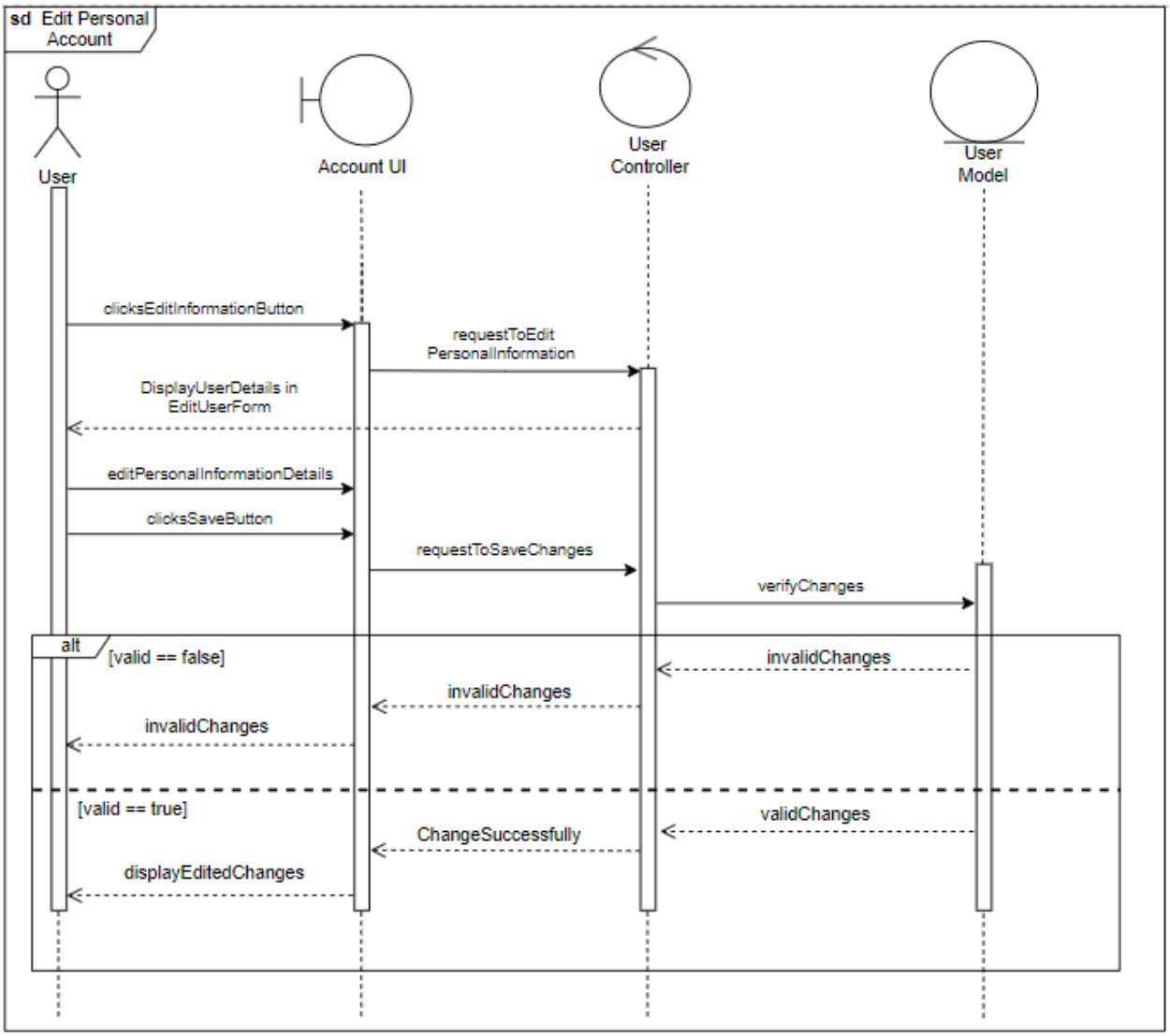
Actor:	User
Description:	Allow users to view their personal account information.
Preconditions:	1. The user is logged into their account. 2. The user has navigated to the Account Information page.
Postconditions:	The user's account information is displayed.
Priority:	Medium
Frequency of Use:	Medium
Flow of Events:	1. The user selects the "Personal Account" page. 2. The system retrieves the user's information from the database. 3. The system displays the following information: a. Name b. Email address c. Phone number d. Address 4. The user can edit their information using the included use case Edit Information. 5. The user logs out from their existing account using the included use case Log Out.
Alternative Flows:	None
Exceptions:	None
Includes:	1. Edit Information 2. Log Out
Special Requirements:	None
Assumptions:	None
Notes and Issues:	None


sd View Personal Account
User
Account UI
User Controller
User Model
clicksPersonalInformationPage
getPersonalInformation
getPersonalInformation
returnPersonalInformation
returnPersonalInformation
displayPersonalInformation

Edit Personal Information

Use Case ID:	UC_ACC_2		
Use Case Name:	Edit Personal Information		
Created By:	Keng Jia Chi	Last Updated By:	Keng Jia Chi
Date Created:	25th August 2024	Date Last Updated:	25th August 2024

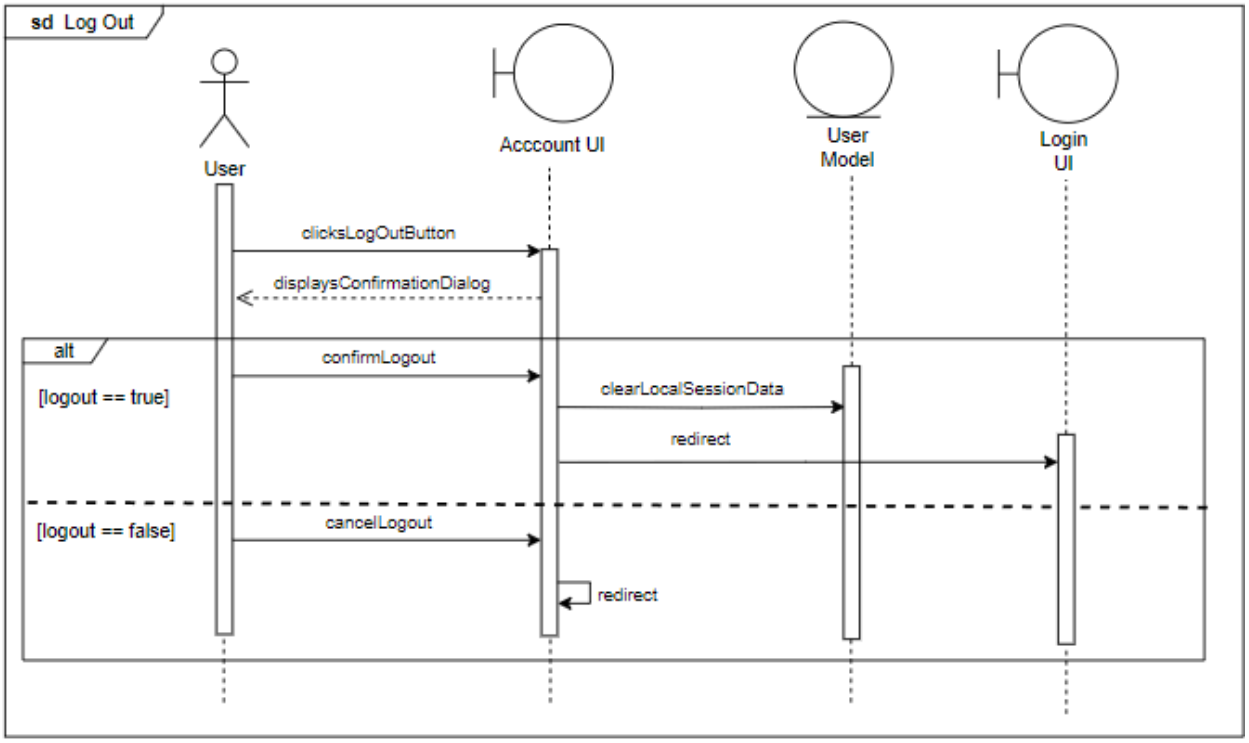
Actor:	User
Description:	Allow users to edit their personal account information.
Preconditions:	1. The user has navigated to the Account Information page.
Postconditions	The user's account information is updated with the new details.
Priority:	Medium
Frequency of Use:	Medium
Flow of events:	1. The user clicks the "Edit Information" button. 2. The system makes the field editable. 3. The user enters the new information. 4. The user clicks the "Save" button. 5. The system validates the new information. 6. If valid, the system updates the database with the new information. 7. The system displays an "Information update successfully" message to the user and shows the updated information.
Alternative Flows:	AF-S6: Information is invalid 1. The system displays an error message. 2. The user is prompted to correct the information. 3. Return to step 3.
Exceptions:	None
Includes:	None
Special Requirements:	None
Assumptions:	None
Notes and Issues:	None


sd Edit Personal Account
User
Account UI
User Controller
User Model
clicksEditInformationButton
requestToEdit PersonalInformation
DisplayUserDetails in EditUserForm
editPersonalInformationDetails
clicksSaveButton
requestToSaveChanges
verifyChanges
alt
[valid == false]
invalidChanges
invalidChanges
invalidChanges
[valid == true]
validChanges
ChangeSuccessfully
displayEditedChanges

Log Out

Use Case ID:	UC_LOGOUT_1		
Use Case Name:	Log Out		
Created By:	Keng Jia Chi	Last Updated By:	Keng Jia Chi
Date Created:	25th August 2024	Date Last Updated:	25th August 2024

Actor:	User
Description	Allows users to securely log out of their account.
Preconditions:	1. The user has navigated to the Personal Account Information page.
Postconditions:	The user is logged out of their account
Priority:	High
Frequency of Use:	High
Flow of Events:	1. The user clicks the "Log Out" button. 2. The system displays a confirmation dialog. 3. The user confirms the logout action. 4. The system terminates the user's session. 5. The system clears any local session data. 6. The system redirects the user to the login page..
Alternative Flows:	AF-S3: The user cancels log out 1. The user clicks “Cancel” on the confirmation dialog. 2. The system closes the dialog.
Exceptions:	None
Includes:	None
Special Requirements:	None
Assumptions:	None
Notes and Issues:	None


sd Log Out
User
Acccount UI
User Model
Login UI
clicksLogOutButton
displaysConfirmationDialog
alt
confirmLogout
[logout == true]
clearLocalSessionData
redirect
[logout == false]
cancelLogout
redirect

View Home Page

Use Case ID:	UC_HOME_1		
Use Case Name:	View Home Page		
Created By:	Jasmine Tye	Last Updated By:	Jasmine Tye
Date Created:	25th August 2024	Date Last Updated:	19th October 2024

Actor:	User
Description:	The Home page provides a comprehensive view of relevant information to the user. It displays details about the user's pets, shows new pets available for adoption in shelters, provides information about nearby amenities related to pet care, and includes a news article carousel featuring top pet-related articles.
Preconditions:	1. The user must have PetCare installed 2. The user must be logged in and authenticated. 3. The user has granted location permissions to the app.
Postconditions:	1. The Home page displays a personalised view including: a. Bottom navigation bar that allows users to redirect to Missing Pet, Adoption Center, Nearby, or Account page b. The user's pets with details like name and photos. c. Pet related articles in the form of a carousel view
Priority:	High
Frequency of Use:	High
Flow of Events:	1. The user opens PetCare and navigates to the Home page. 2. PetCare retrieves and displays the user's registered pets, showing each pet's name and image under the "My Pets" section. 3. PetCare displays a "+" icon to allow the user to add a new pet profile: - If there are registered pets, PetCare displays each pet's image with a "+" icon as the last item. - If there are no registered pets, PetCare displays only the "+" icon. 4. The user can tap on a pet's image or name to view, edit, or delete the pet's details. 5. PetCare retrieves and displays a news article carousel to showcase top pet-related articles. 6. Petcare automatically rotates the carousel every 5 seconds. 7. The user manually swipes through the articles in the carousel to change the article being shown in the carousel. 8. The user taps on an article in the carousel to view its full content. 9. PetCare displays a "See All" button above the news carousel, allowing the user to navigate to the "Browse Articles" page to view all articles. 10. PetCare displays a bottom navigation bar with icons that redirect the user to different sections on tap: a. Missing: Redirects to the "Missing Pets" page. b. Adoption: Redirects to the "Adoption Center" page. c. Nearby: Redirects to the "Petcare Amenities" page. d. Account: Redirects to the "Account" page.

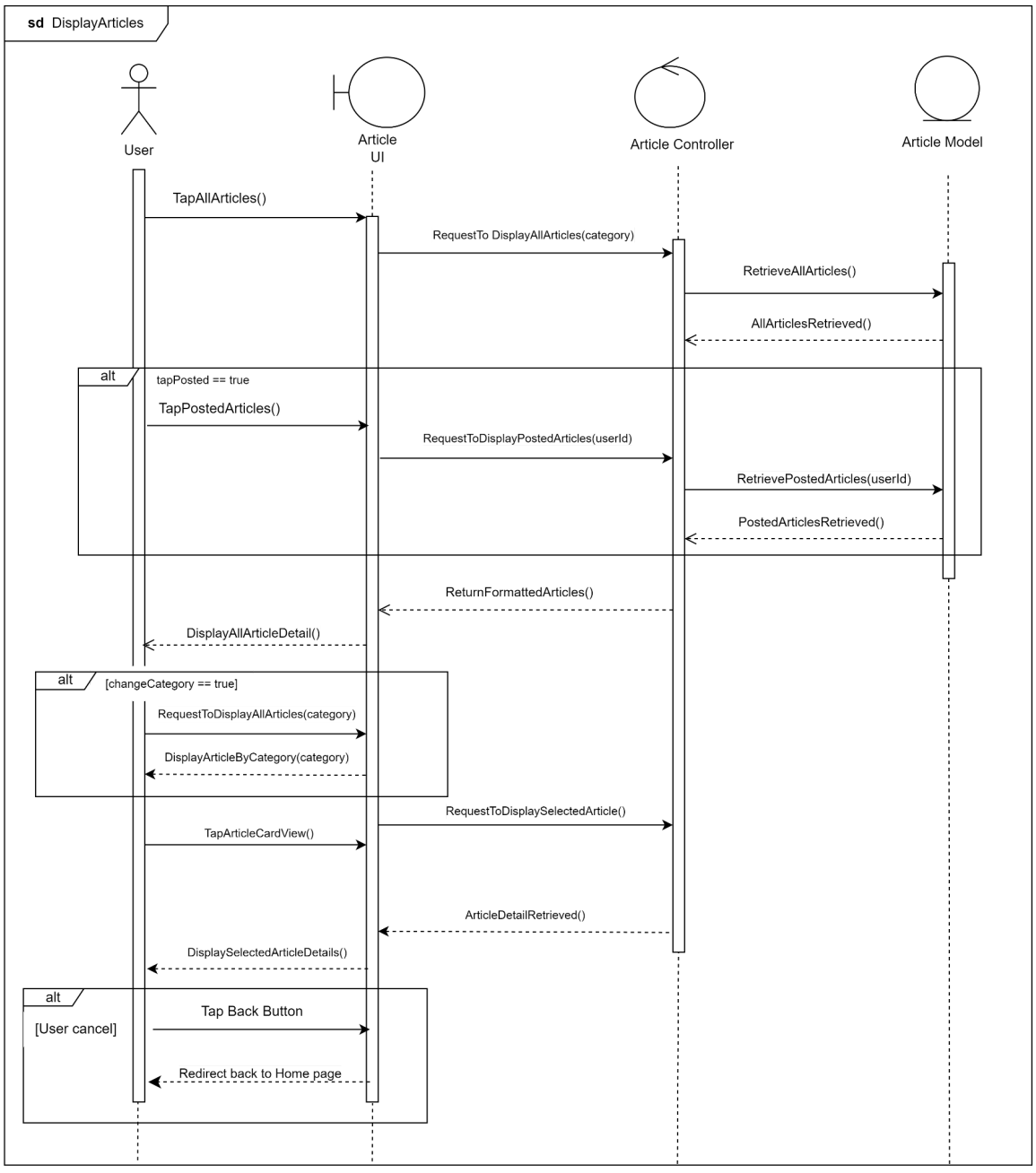
	11. PetCare displays a fixed chatbot icon at the bottom right corner, above the navigation bar, which the user can tap to access the ‘Chatbot’ page.
	AF-S2: No pets registered under user 1. System displays a message stating “You have no registered pets” AF-S3: No article added by user 1. System displays a message stating “There are currently no article”
Exceptions:	EX-1 Data Retrieval Failure 1. If any data retrieval fails, the system displays an error message prompting the user to try again later.
Includes:	UC23 - View Pet Information UC8 - View All Articles
Special Requirements:	None
Assumptions:	1. Users have Internet access
Notes and Issues:	None

View All Articles

Use Case ID:	UC_ARC_1		
Use Case Name:	Browse All Articles		
Created By:	Jasmine Tye	Last Updated By:	Jasmine Tye
Date Created:	31st August 2024	Date Last Updated:	19th October 2024

Actor:	User
Description:	Allows a user to browse, filter and search all pet-related articles.
Preconditions:	1. The user has tapped on the “See All” button in the Home page. 2. The user is logged in and authenticated. 3. The user is navigated to the Home Page.
Postconditions:	1. The articles are displayed with specific details, including a thumbnail image, title, poster's name, profile picture, and the relative time since the article was published. 2. The user can tap on any article to view its full content.
Priority:	Medium
Frequency of Use:	Medium
Flow of Events:	1. The system displays the Browse Articles section in the Home page. 2. The user taps on the “See All” button under the Browse Articles section. 3. The system redirects the user to the “Browse Articles” page. 4. The system displays a bottom navigation bar that has a “All” and “Posted” section. 5. The system highlights the “All” section in the bottom navigation bar of the “Browse Articles” page. 6. The system will highlight the “ALL” category and display all articles under all categories by default. 7. For each article, the system displays: a. A thumbnail image of the article b. Title of the article c. The name of the poster who posted the article d. The profile picture of the poster e. The relative time since the article was published. 8. The user scrolls through the list of articles. 9. The user taps on an article to view its full content. 10. Use Case continue on UC_ARC_4
Alternative Flows:	AF-S5: No Articles Available 1. If no articles are available in a selected category, PetCare displays a message saying “No articles available”
Exceptions:	None
Includes:	UC7 - Home Page UC9 - View Article Details
Special Requirements:	None

Assumptions:	None
Notes and Issues:	None



Filter Articles

Use Case ID:	UC_ARC_2		
Use Case Name:	Filter Articles		
Created By:	Jasmine Tye	Last Updated By:	Jasmine Tye
Date Created:	31st August 2024	Date Last Updated:	20th October 2024

Actor:	User
Description:	Allows a user to filter articles by category, such as Lifestyle, Grooming, Community, Health and Wellness, and Others.
Preconditions:	1. The user is on the “Browse Articles” page.
Postconditions:	1. Search results for articles related to the input keyword(s) are displayed to the user. 2. If no results are found, a message is displayed indicating no articles are available for the searched keyword.
Priority:	Medium
Frequency of Use:	Medium
Flow of Events:	1. The system categorizes and displays articles into the following sections: Lifestyle, Grooming, Community, Health and Wellness and Others. 2. The user taps a category from the available options. 3. Upon category selection, the system filters the content and displays only the articles associated with the selected category. 4. The user can switch between categories by tapping on another category. Each time a new category is selected, the system dynamically updates the displayed content to match the newly selected category. 5. The user scrolls through the list of articles within the selected category. 6. The user taps on an article to view its full content. 7. Use Case continue on UC_ARC_4
Alternative Flows:	AF-S3: No Articles Available in Category 1. If no articles are available in a selected category, PetCare displays a message saying “No articles available”
Exceptions:	None
Includes:	None
Special Requirements:	The system must filter the results without refreshing or reloading the page.

Assumptions:	The Articles page must be fully functionable and display accurate results.
Notes and Issues:	None

Search Articles

Use Case ID:	UC_ARC_3		
Use Case Name:	Search Articles		
Created By:	Jasmine Tye	Last Updated By:	Jasmine Tye
Date Created:	31st August 2024	Date Last Updated:	19th October 2024

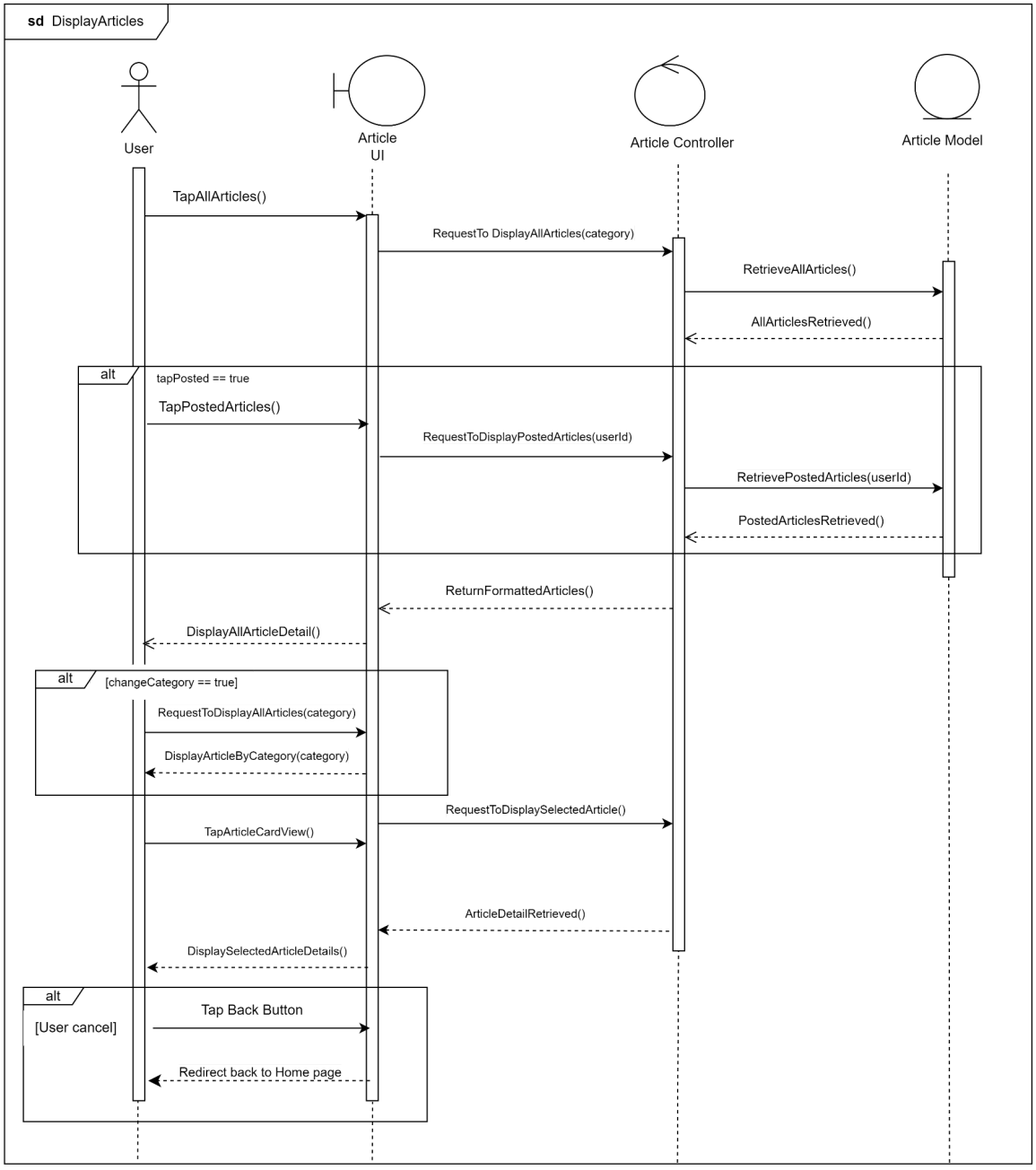
Actor:	User
Description:	Allows a user to search for pet-related articles using keywords.
Preconditions:	1. The user is on the “Browse Articles” page.
Postconditions:	1. The user can view articles filtered by the selected category. 2. If no articles are available in the selected category, a message is displayed.
Priority:	Medium
Frequency of Use:	Medium
Flow of Events:	1. The user inputs a keyword into the search bar. 2. As the user types, the system switches to the "ALL" category view and displays articles that match the input keyword. 3. The system displays a “X” icon at the right end of the search bar to clear the search. 4. If the user taps on the "X" icon, the system clears the search and returns to the default “ALL” category view. 5. User will select the article to view its full content. 6. Use Case continue on UC_ARC_4
Alternative Flows:	AF-S2: No Result Found 1. When a user is searching a keyword in the search bar, if there are no results, PetCare displays a message saying “No articles with the {user input} keyword(s) available. Please use a different keyword.”
Exceptions:	None
Includes:	None
Special Requirements:	None
Assumptions:	None
Notes and Issues:	None

View Article Details

Use Case ID:	UC_ARC_4		
Use Case Name:	View Article Details		
Created By:	Jasmine Tye	Last Updated By:	Jasmine Tye
Date Created:	31st August 2024	Date Last Updated:	19th October 2024

Actor:	User
Description:	Allows the user to view the full details of the selected article.
Preconditions:	1. The user has selected an article either from the article carousel in Home page, browse all articles page or posted articles page.
Postconditions:	1. The user can view the full details of the selected article, including the article's content and all related information.
Priority:	High
Frequency of Use:	High
Flow of Events:	1. The user opens the "Browse All" articles page and taps on an article. 2. The system opens a new page displaying the full article details: a. The thumbnail image of the article b. The category of the article c. The title of the article d. The full text of the article e. The name of the poster f. The profile picture of the poster g. The relative time since the article was published 3. The user scrolls through the article content.
Alternative Flows:	AF-S6: Accessing an Article from Home page 1. The user taps on an article from the article carousel in the Home page. 2. The user continues with Step 2. AF-S6: Accessing an Article from Posted article page 1. The user taps on an article from the "Posted Article" page. 2. The user continues with Step 2. AF-S3: Back button 1. The user taps on the 'Back' button. 2. The system redirects the user back to the previous page.
Exceptions:	EX3: Article Not Found 1. If the selected article has been deleted or is unavailable, PetCare displays a popup message saying "Article not found" 2. The system redirects the user back to the page they were previously at.
Includes:	None

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

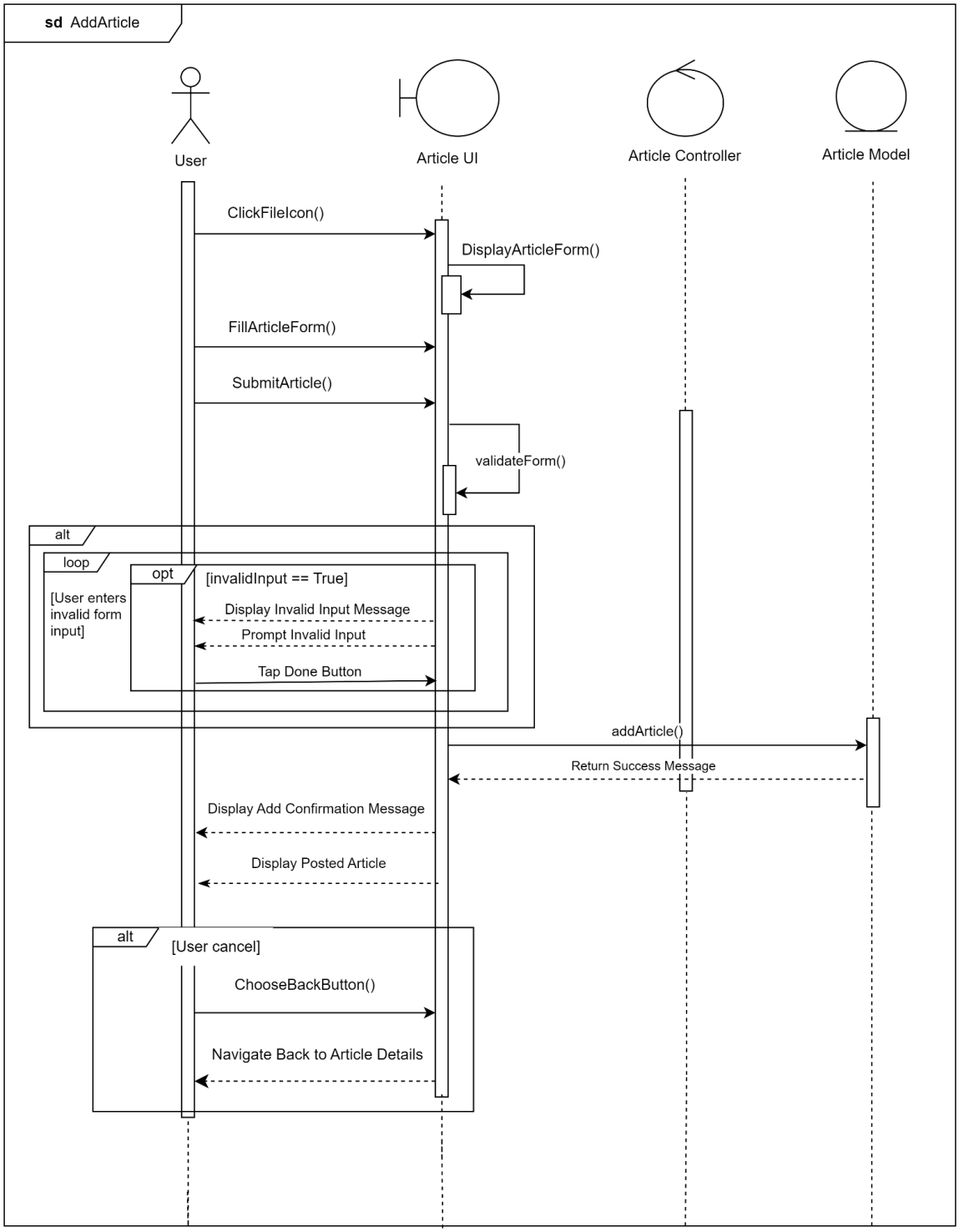


Add New Article

Use Case ID:	UC_ARC_5		
Use Case Name:	Add New Article		
Created By:	Jasmine Tye	Last Updated By:	Jasmine Tye
Date Created:	31st August 2024	Date Last Updated:	19th October 2024

Actor:	User
Description:	Allows a user to create and submit a new pet-related article to the PetCare application.
Preconditions:	1. The user is on the “Posted Articles” page of the application.
Postconditions:	1. A new article is created and stored in the system. 2. The system assigns the article to the poster in the database. 3. The new article appears in the “All Articles” section 4. The new article appears in the “Posted Articles” section only to the original poster.
Priority:	High
Frequency of Use:	Medium
Flow of Events:	1. The user taps on the file icon located at the bottom right of the bottom navigation bar in the “Posted Articles” page. 2. The system displays the article creation form. 3. The user fills in the following information: a. Title of the article b. Body text of the article c. Category of the article from the given options (Lifestyle, Grooming, Community, Health and Wellness, Others) d. Thumbnail image for the article 4. The user taps on the “Confirm” button. 5. The system saves the new article to the database. 6. The system displays a success popup message. 7. The system redirects the user to the “Posted” section where the new article is now visible and posted.
Alternative Flows:	AF-S3: User Cancels Article Creation 1. During step 3, the user can tap on the “Back” button to stop the article creation. 2. The system displays a popup message to confirm the action. 3. If the user taps on the “Ok” button, the system closes the form and returns the user back to the “Posted Articles” page without creating a new article. 4. If the user taps on the “No” button, the system closes the popup message and the user returns to the article creation form.
Exceptions:	EX3: Image Upload Failure 1. If there is an issue with uploading the image, PetCare displays an error message and asks the user to re-upload the image.
Includes:	UC11 - View Posted Articles UC8 - View All Articles

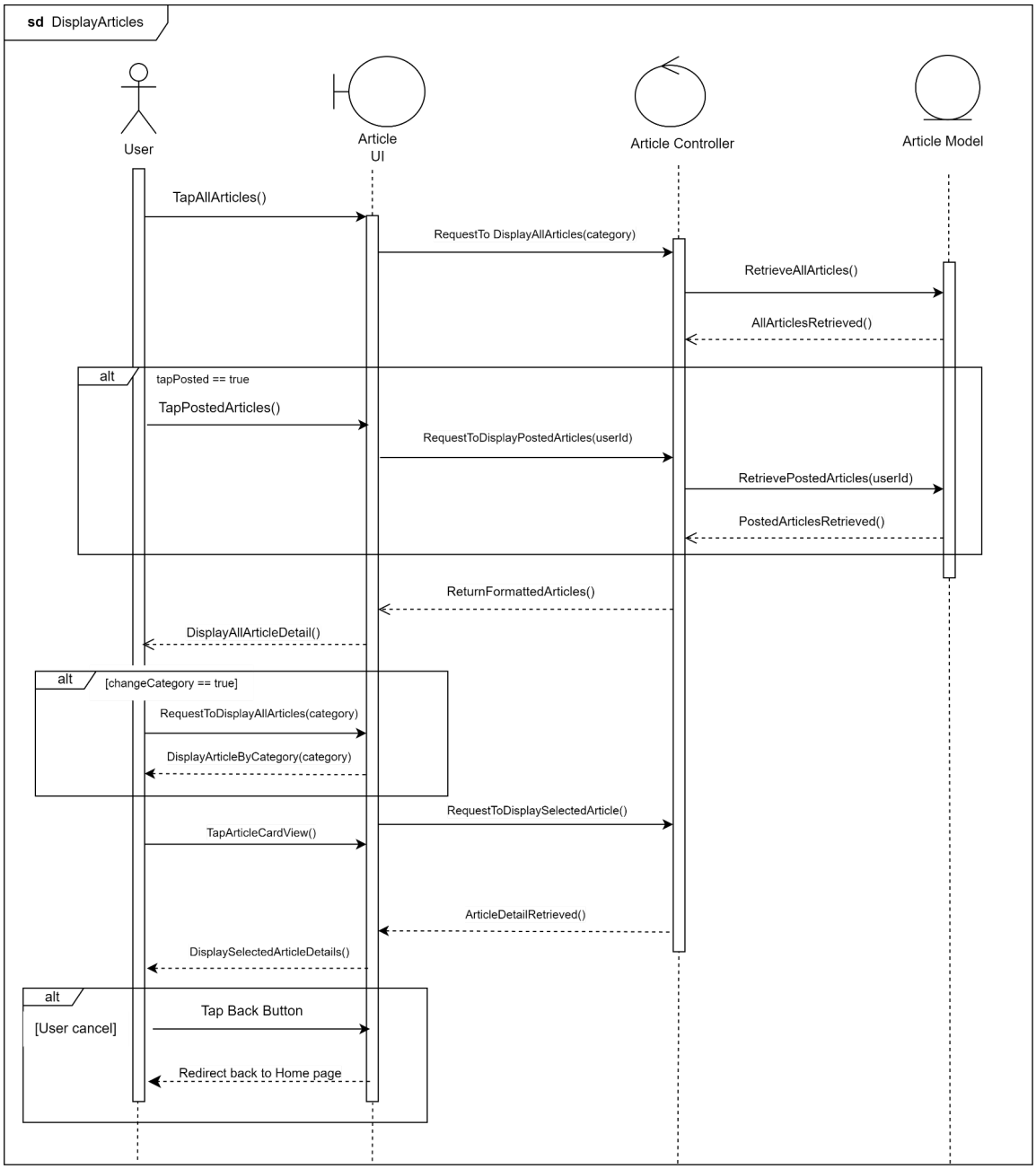
Special Requirements:	None
Assumptions:	- The user publishes articles that are related to pets. - The user selects the correct category for the article.
Notes and Issues:	None



View Posted Articles

Use Case ID:	UC_ARC_6		
Use Case Name:	View Posted Articles		
Created By:	Jasmine Tye	Last Updated By:	Jasmine Tye
Date Created:	31st August 2024	Date Last Updated:	31st August 2024

Actor:	User
Description:	Allows users to view the articles that each user has posted onto the PetCare app.
Preconditions:	1. The user has navigated to the “Browse Articles” page and has tapped on the “Posted” section in the bottom navigation bar.
Postconditions:	1. The user can see a list of all the articles that they have posted under the “Posted” section. 2. The user is able to interact with their posted articles.
Priority:	Medium
Frequency of Use:	Medium
Flow of Events:	1. The user navigates to the “Browse Articles” page. 2. The user taps on the “Posted” section on the bottom navigation bar. 3. The system displays the “Posted Article” page. 4. The system retrieves and displays all articles posted by the user. 5. The user can scroll through the list of their posted articles. 6. The user can tap on article(s) to view the full details of the article.
Alternative Flows:	AF-S3: No articles posted 1. If the user has not posted any articles, the system displays a message stating “No articles posted. Share your articles.”
Exceptions:	EX1: Network Error 1. If there is a network or database issue while fetching the articles, PetCare displays an error message indicating that articles could not be loaded and suggests the user to try again later.
Includes:	UC_ARC_5 - Add New Article UC_ARC_4 - View Article Details
Special Requirements:	None
Assumptions:	None
Notes and Issues:	None

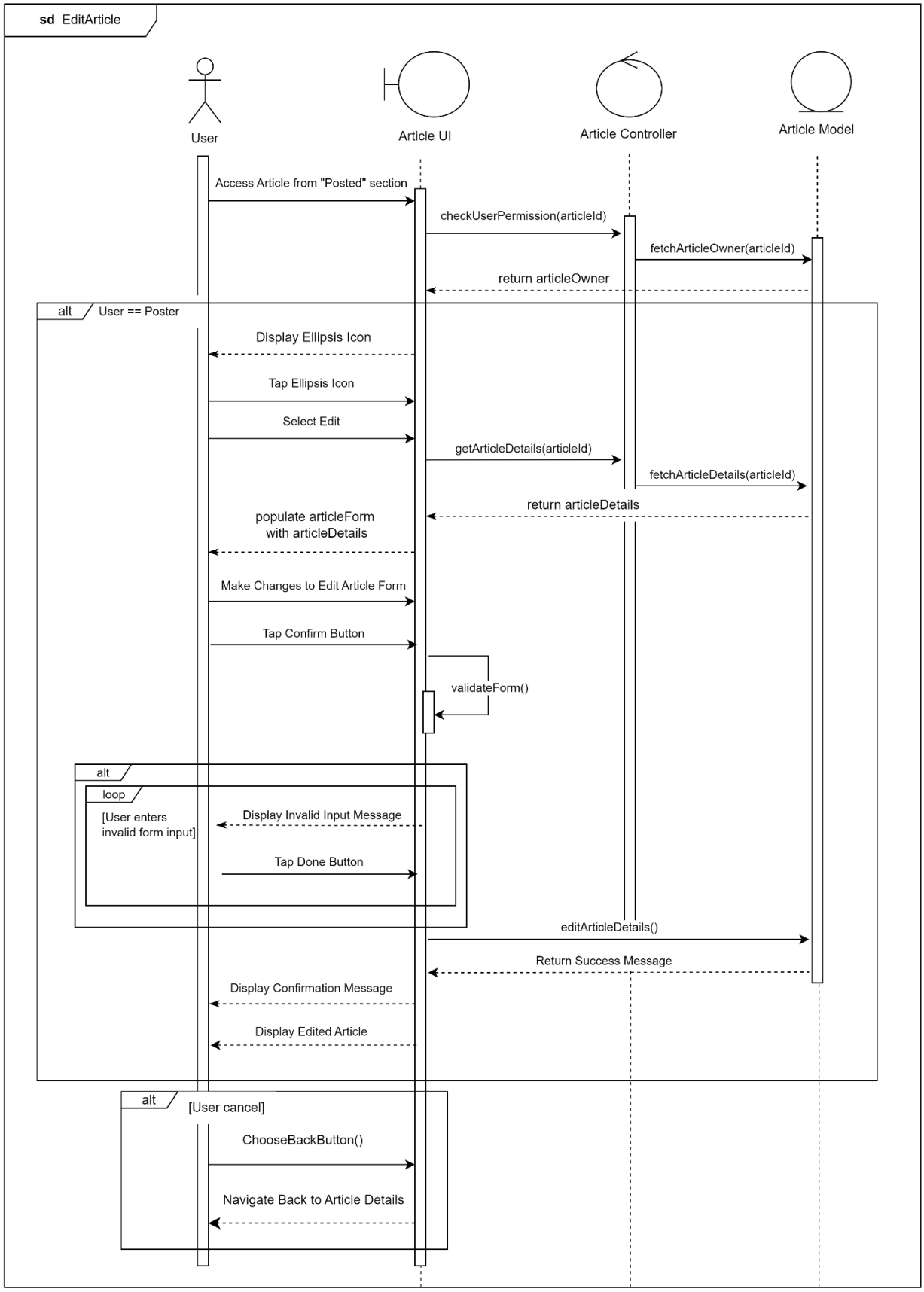

sd DisplayArticles
User
Article UI
Article Controller
Article Model
TapAllArticles()
RequestTo DisplayAllArticles(category)
RetrieveAllArticles()
AllArticlesRetrieved()
alt
tapPosted == true
TapPostedArticles()
RequestToDisplayPostedArticles(userId)
RetrievePostedArticles(userId)
PostedArticlesRetrieved()
ReturnFormattedArticles()
DisplayAllArticleDetail()
alt
[changeCategory == true]
RequestToDisplayAllArticles(category)
DisplayArticleByCategory(category)
RequestToDisplaySelectedArticle()
TapArticleCardView()
ArticleDetailRetrieved()
DisplaySelectedArticleDetails()
alt
[User cancel]
Tap Back Button
Redirect back to Home page

Edit Article

Use Case ID:	UC_ARC_7		
Use Case Name:	Edit Article		
Created By:	Jasmine Tye	Last Updated By:	Jasmine Tye
Date Created:	31st August 2024	Date Last Updated:	31st August 2024

Actor:	User
Description:	Allows a user to edit an article that they have previously posted on the PetCare app.
Preconditions:	1. The user has posted at least one article. 2. The user has selected an article from the “Posted Article” page.
Postconditions:	1. The user’s changes to the article are saved and updated in the PetCare database. 2. The edited article is updated in both the “Posted Article” and “All Article” sections.
Priority:	Medium
Frequency of Use:	Low
Flow of Events:	1. The user navigates to the “Posted” section from the bottom navigation bar in the Browse Articles page. 2. The user selects an article that they previously posted. 3. The user taps on the vertical ellipsis icon on the top right corner of the article detail page which is only visible to the original poster of the article. 4. PetCare displays the “Edit” and “Delete” options. 5. The user taps on the “Edit” options. 6. PetCare redirects the user to the Edit Article form, pre-filled with the existing details of the article: a. Title of the article. b. Body text of the article. c. Selected category of the article. d. Existing thumbnail image of the article. 7. The user can modify any of the article details: a. Changing the title b. Updating the body text c. Selecting a different category d. Uploads a new thumbnail image 8. The user taps on the “Save” button to submit the changes. 9. The system displays a success message. 10. The system updates the edited article details in both the “Posted” and “All” sections. 11. The system redirects the user back to the article view to show the updated content.
Alternative Flows:	AF-S3: User Cancels Editing 1. During steps 6-7, the user can tap on the “Back” button to stop the editing process. 2. The system displays a popup message to confirm the action. 3. If the user taps on the “Ok” button, the system closes the form and returns the user back to the article details without making any changes.

	4. If the user taps on the “No” button, the system closes the popup message and the user returns to the edit process.
Exceptions:	EX1: Network Error 1. If there is a network or database issue while updating the article details, PetCare displays an error message indicating that articles could not be updated and suggests the user to try again later.
Includes:	UC_ARC_4 - View Article Details
Special Requirements:	None
Assumptions:	None
Notes and Issues:	None

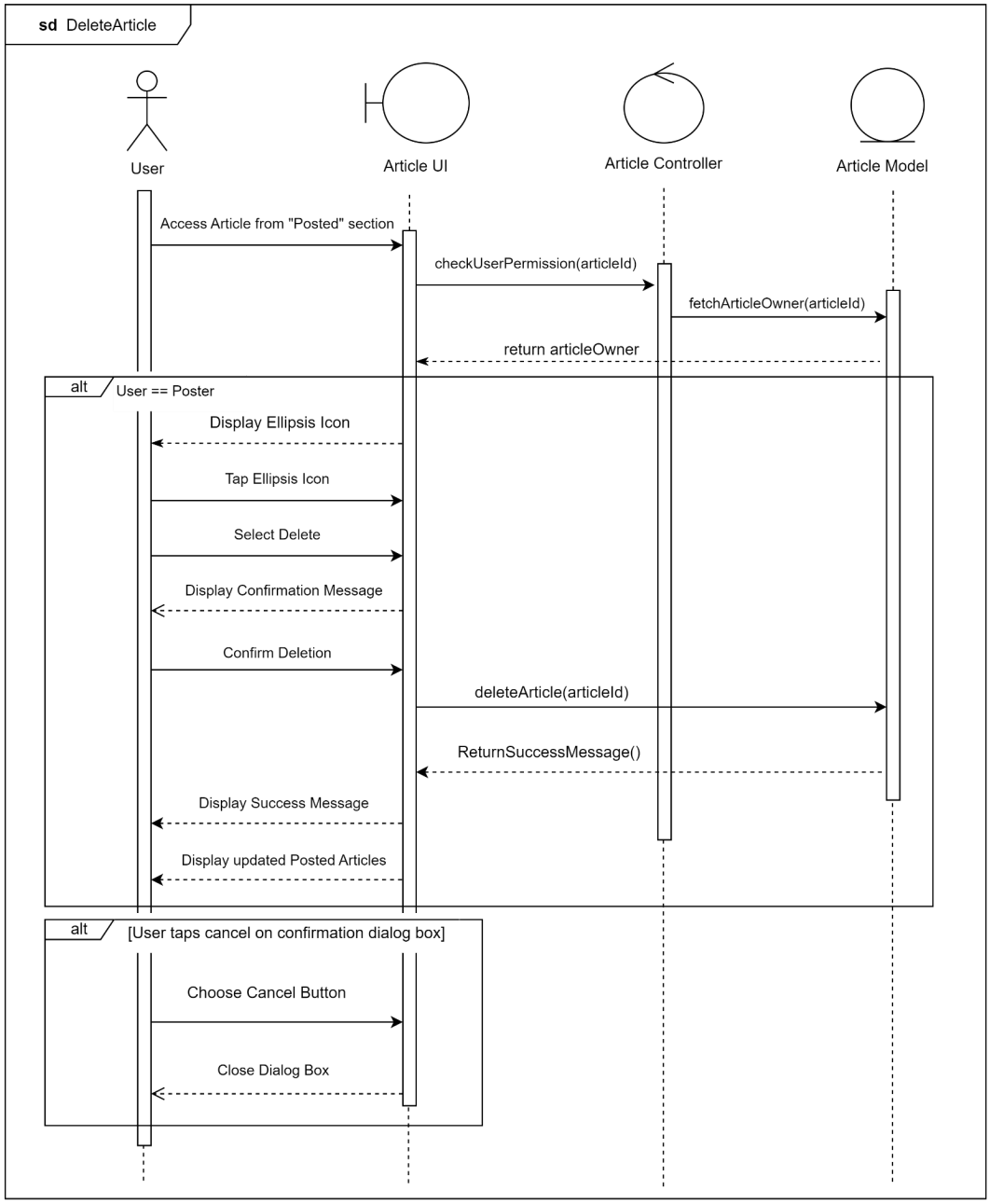

sd EditArticle
User
Article UI
Article Controller
Article Model
Access Article from "Posted" section
checkUserPermission(articleId)
fetchArticleOwner(articleId)
return articleOwner
alt
User == Poster
Display Ellipsis Icon
Tap Ellipsis Icon
Select Edit
getArticleDetails(articleId)
fetchArticleDetails(articleId)
return articleDetails
populate articleForm
with articleDetails
Make Changes to Edit Article Form
Tap Confirm Button
validateForm()
alt
loop
[User enters
invalid form input]
Display Invalid Input Message
Tap Done Button
editArticleDetails()
Return Success Message
Display Confirmation Message
Display Edited Article
alt
[User cancel]
ChooseBackButton()
Navigate Back to Article Details

Delete Article

Use Case ID:	UC_ARC_8		
Use Case Name:	Delete Article		
Created By:	Jasmine Tye	Last Updated By:	Jasmine Tye
Date Created:	31st August 2024	Date Last Updated:	31st August 2024

Actor:	User
Description:	Allows the user to delete an article that they have previously posted on the PetCare app.
Preconditions:	1. The user has posted at least one article. 2. The user has selected an article from the “Posted Article” page.
Postconditions:	1. The selected article is permanently removed from the PetCare database. 2. The deleted article is no longer visible in both the “Posted” and “All” sections in the “Browse Articles” page.
Priority:	Medium
Frequency of Use:	Low
Flow of Events:	1. The user navigates to the “Posted” section from the bottom navigation bar in the Browse Articles page. 2. The user selects an article that they previously posted. 3. The user taps on the vertical ellipsis icon on the top right corner of the article detail page which is only visible to the original poster of the article. 4. PetCare displays the “Edit” and “Delete” options. 5. The user taps on the “Delete” option. 6. PetCare displays a popup box asking the user to confirm the deletion, “Are you sure you want to delete this article? This action cannot be undone.” 7. The user taps on the “Yes” button to confirm the deletion. 8. PetCare permanently deletes the article from the database. 9. PetCare updates the “Posted” and “All” sections to remove the deleted article from view. 10. PetCare displays a confirmation message indicating that the article has been deleted successfully. 11. The system redirects the user back to the article view to show the updated content.
Alternative Flows:	AF-S3: User Cancels Deletion 1. In Step 7, if the user taps on the “No” button in the confirmation popup, PetCare closes the pop-up box without deleting the article. 2. PetCare redirects the user back to the article details without making any changes.
Exceptions:	EX1: Network Error 1. If there is a network or database issue while updating the article details, PetCare displays an error message indicating that articles could not be deleted and suggests the user to try again later.

Includes:	UC_ARC_4 - View Article Details
Special Requirements:	None
Assumptions:	None
Notes and Issues:	None

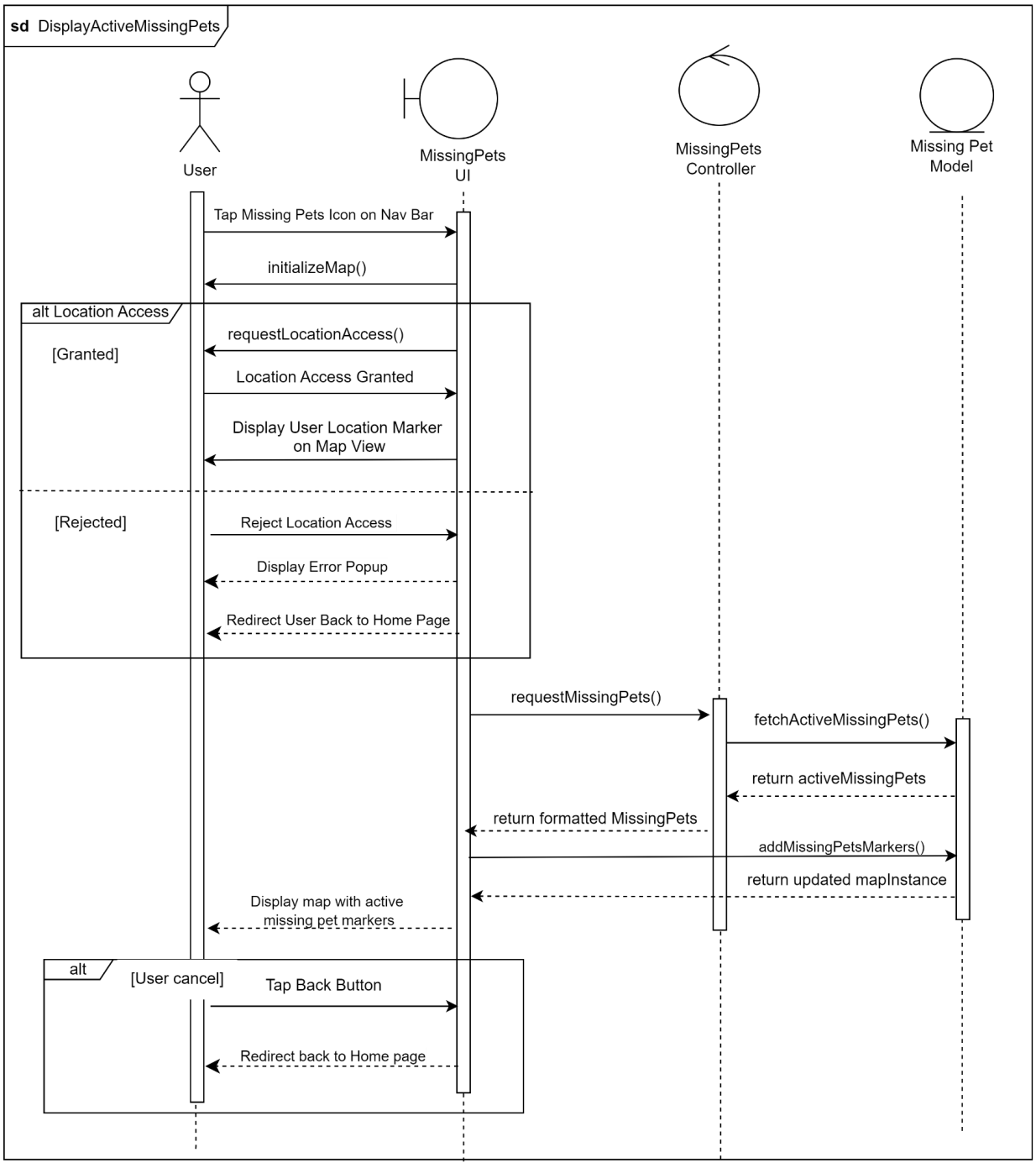


Display Active Missing Pets

Use Case ID:	UC_MP_1		
Use Case Name:	Display Active Missing Pet		
Created By:	Jasmine Tye	Last Updated By:	Jasmine Tye
Date Created:	25th August 2024	Date Last Updated:	19th October 2024

Actor:	User
Description:	Allows a user to view a list of active missing pets reported by other users.
Preconditions:	1. The user is logged in and authenticated to PetCare 2. The user has a GPS-enabled device. 3. The application has access to the user's current location.
Postconditions:	1. The system displays a map with markers showing the last seen location of missing pets. 2. The user can tap on a marker to view detailed information about the missing pet.
Priority:	High
Frequency of Use:	Medium
Flow of Events:	1. The user navigates to the "Missing Pets" section in the app. 2. The app retrieves the user's current location using GPS. 3. The system sends queries to the database for active missing pet reports nearby to the user. 4. The system displays the missing pets' images as markers on the map at their first seen locations. 5. The user taps on a marker to go to the selected missing pet's page to view more information about the selected missing pet: a. Name b. Breed c. Age d. Gender e. Photo f. Contact details of the owner g. Description posted by the owner 6. The system displays a new map interface displaying the record of multiple last seen details for the selected missing pet. 7. The system displays a GPS icon, allowing the user to quickly re-center the map on their current location with a single tap. 8. The new map interface includes: a. A timeline of sightings as markers on the map b. Date and time of each reported sighting c. Image posted by reporter d. Sighting description by reporter 9. When the user taps on the next button on the map which displays markers of sighting locations, the next sighting information is displayed.
Alternative Flows:	AF-S4: Incomplete Information 1. If some missing pets have incomplete information: a. Missing sighting image(s), system displays a default image

	b. Missing sighting description, system omits this information from the report.
Exceptions:	EX1: Location services unavailable 1. If GPS is disabled or the device is unable to determine the user's location, the system notifies the user and provides an option to manually input a location by postal code or keyword.
Includes:	None
Special Requirements:	Device permission for location access is enabled.
Assumptions:	The missing pet database is up-to-date and returns accurate information.
Notes and Issues:	None

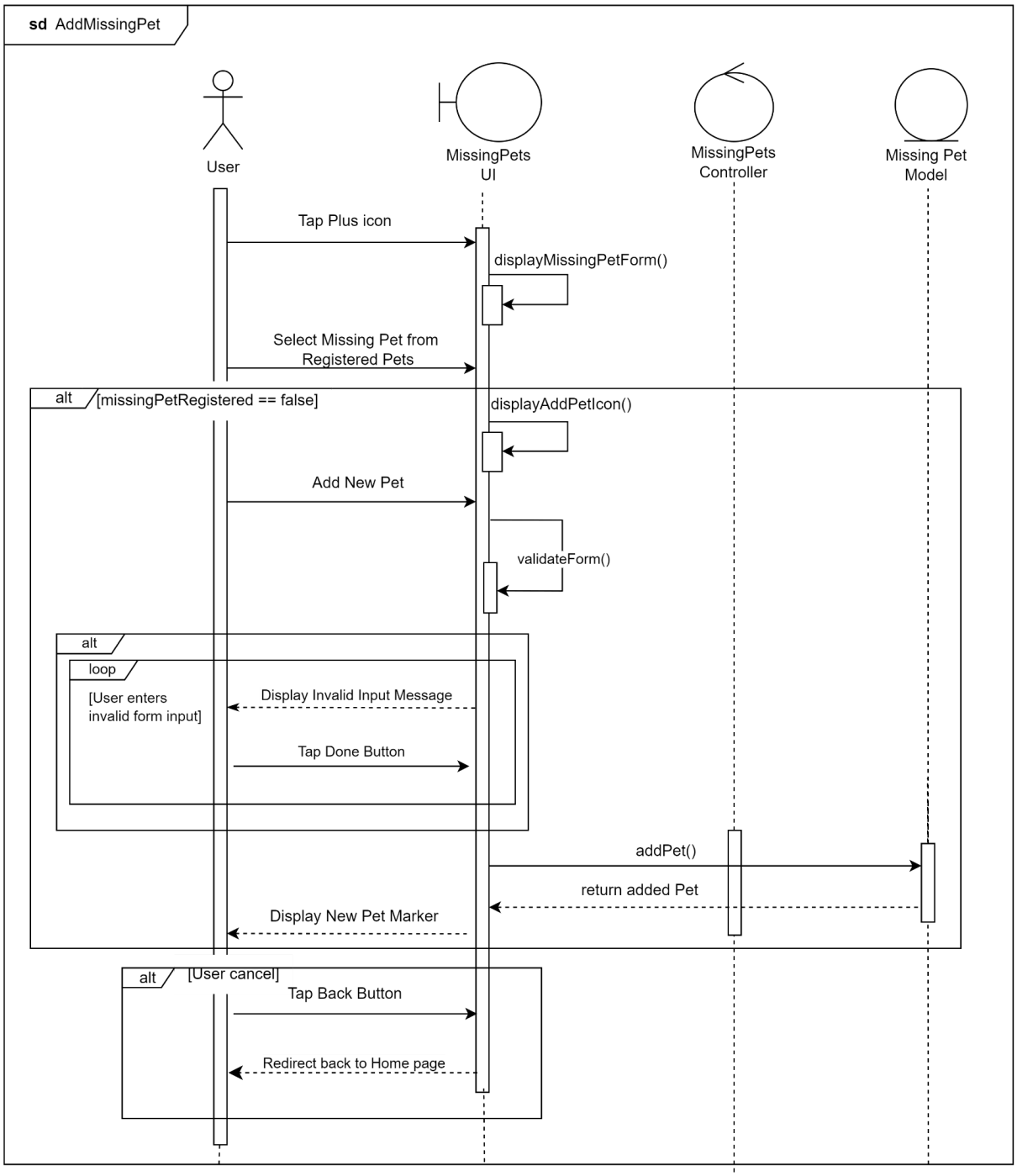

sd DisplayActiveMissingPets
User
MissingPets UI
MissingPets Controller
Missing Pet Model
Tap Missing Pets Icon on Nav Bar
initializeMap()
alt Location Access
[Granted]
requestLocationAccess()
Location Access Granted
Display User Location Marker on Map View
[Rejected]
Reject Location Access
Display Error Popup
Redirect User Back to Home Page
requestMissingPets()
fetchActiveMissingPets()
return activeMissingPets
return formatted MissingPets
addMissingPetsMarkers()
return updated mapInstance
Display map with active missing pet markers
alt
[User cancel]
Tap Back Button
Redirect back to Home page

Report Pet as Missing

Use Case ID:	UC_MP_2		
Use Case Name:	Report Pet as Missing		
Created By:	Jasmine Tye	Last Updated By:	Jasmine Tye
Date Created:	25th August 2024	Date Last Updated:	25th August 2024

Actor:	Pet Owner
Description:	Allows the pet owner to report their missing pet by providing details about the pet and the circumstances under which it went missing.
Preconditions:	1. The pet owner is logged in and authenticated. 2. The pet owner has saved the profile of the pet that went missing.
Postconditions:	1. The system successfully saves and publishes the missing pet report made by the pet owner to the app. 2. Other users are able to view the missing pet report.
Priority:	High
Frequency of Use:	Medium
Flow of Events:	1. The pet owner taps on the “Report Missing Pet” option on the “Home” page. 2. The system displays a list of the pet owner’s registered pets from the “My Pets” section. 3. The pet owner selects the profile of the pet that went missing. 4. The system automatically fills in the report form with the selected pet’s details: a. Name b. Breed c. Age d. Gender e. Photo 5. The pet owner inputs additional information in the form: a. Last seen location - postal codes only b. Last seen time c. Last seen date d. Last seen image e. Additional descriptions 6. The pet owner submits the report. 7. The system confirms the submission and displays the reported pet in the “Missing Pets” section. 8. The system displays the last seen information as the first sighting marker .
Alternative Flows:	AF-S2: No pets registered under “My Pets” 1. The system displays a prompt to add a pet profile before reporting it as missing so that the pet owner can get updates. 2. Once the user adds the missing pet’s profile, the pet owner can continue with step 2.
Exceptions:	EX3: Form submission error Form submission fails due to network issues or server errors, the system displays an error popup message asking the pet

	owner to try again at a later time. EX3: Invalid Last Seen Location 1. When the pet owner inputs an invalid postal code in last seen location, the system prompts the user to re-input a different valid postal code
Includes:	UC22-View Pet Information UC25-Add Pet
Special Requirements:	None
Assumptions:	1. The pet owner's registered pet information is accurate. 2. The pet owner consents to share the missing pet's information with other users.
Notes and Issues:	None

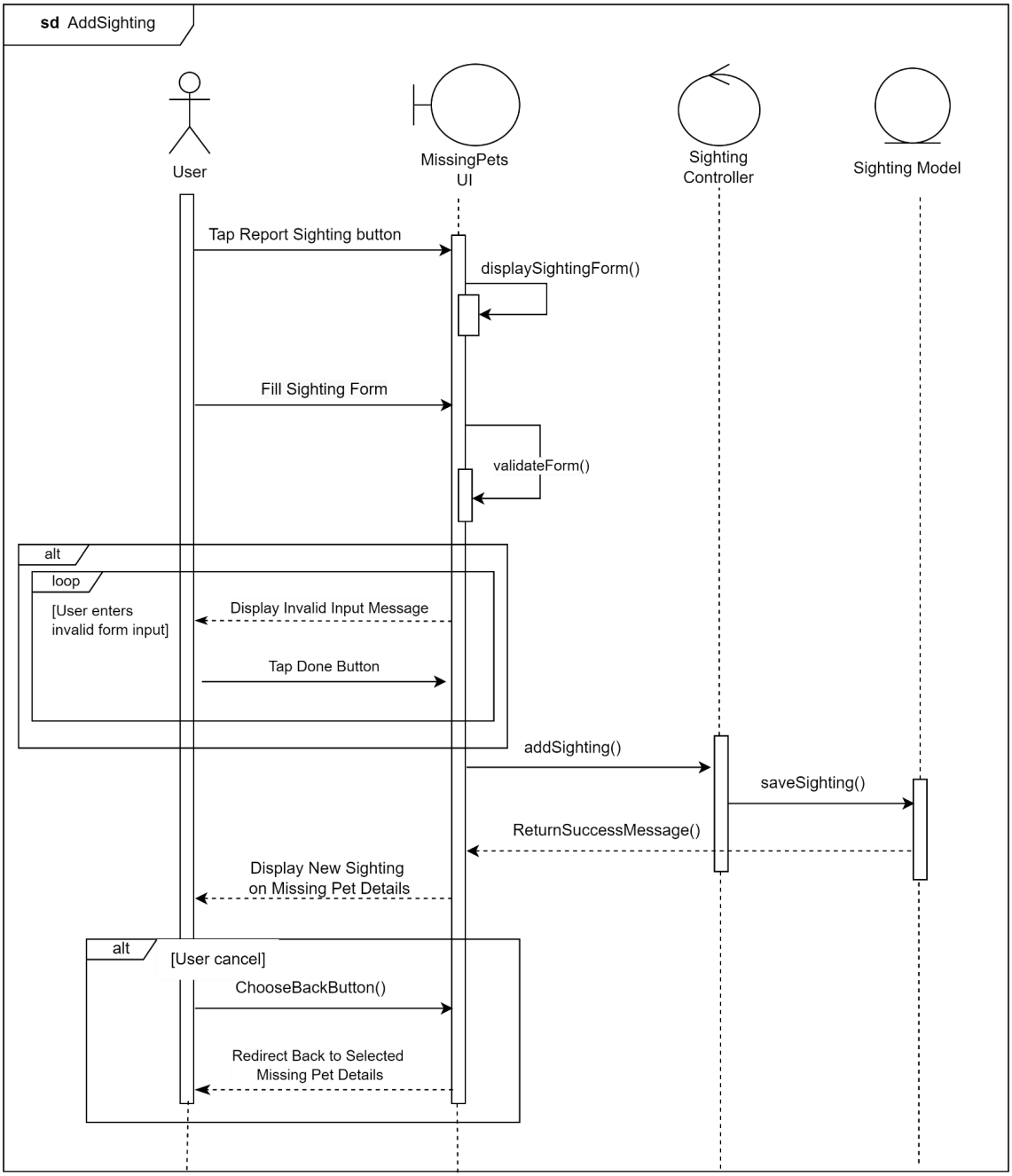

sd AddMissingPet
User
MissingPets UI
MissingPets Controller
Missing Pet Model
Tap Plus icon
displayMissingPetForm()
Select Missing Pet from Registered Pets
alt
[missingPetRegistered == false]
displayAddPetIcon()
Add New Pet
validateForm()
alt
loop
[User enters invalid form input]
Display Invalid Input Message
Tap Done Button
addPet()
return added Pet
Display New Pet Marker
alt
[User cancel]
Tap Back Button
Redirect back to Home page

Update Last Seen Location

Use Case ID:	UC_MP_3		
Use Case Name:	Update Last Seen Location		
Created By:	Jasmine Tye	Last Updated By:	Jasmine Tye
Date Created:	25th August 2024	Date Last Updated:	19th October 2024

Actor:	User (other than the pet owner/ reporting user)
Description:	Allows other users to update the last seen location, time and date of a missing pet.
Preconditions:	1. The original pet owner has made a missing report. 2. The selected missing pet marker is an active case in the system.
Postconditions:	1. The system updates the map with the new last seen location, time, and date for the missing pet. 2. All users of the app can see the updated information on the map. 3. The original reporting user receives a notification about the updated details.
Priority:	High
Frequency of Use:	Medium
Flow of Events:	1. The user navigates to the “Display Missing Pets” page in the app. 2. The system retrieves and displays the map with active missing pet markers. 3. The user selects an active missing pet marker on the map. 4. The system displays a pop-up of the details of the missing pet. 5. The user taps on the “Add Report” icon. 6. The user enters the new time, date, location, sighting image and sighting description when the user last saw the pet. 7. The system validates the entered data. 8. The system updates the map with the new last seen location, time and date for the missing pet.
Alternative Flows:	AF-S2: User inputs incorrect data 1. If the user inputs an incorrect or incomplete address, time or date, the system displays an error message and prompts the user to change the input. AF-S7: User cancels form submission 1. The user can cancel the form submission by tapping on the “Back” button and return back to the “Display Active Missing Pets” page. AF-S4: Image upload error 1. If there is an error with the upload process of the image, the system displays an error message and prompts the user to re-upload a different image. 2. The selected image is not uploaded to the system.
Exceptions:	EX3: Form submission error 1. Form submission fails due to network issues or server errors,

	the system displays an error popup message asking the user to try again at a later time.
Includes:	UC_MP_1 - Display Active Missing Pets
Special Requirements:	None
Assumptions:	Users provide accurate and honest updates about the missing pet.
Notes and Issues:	None

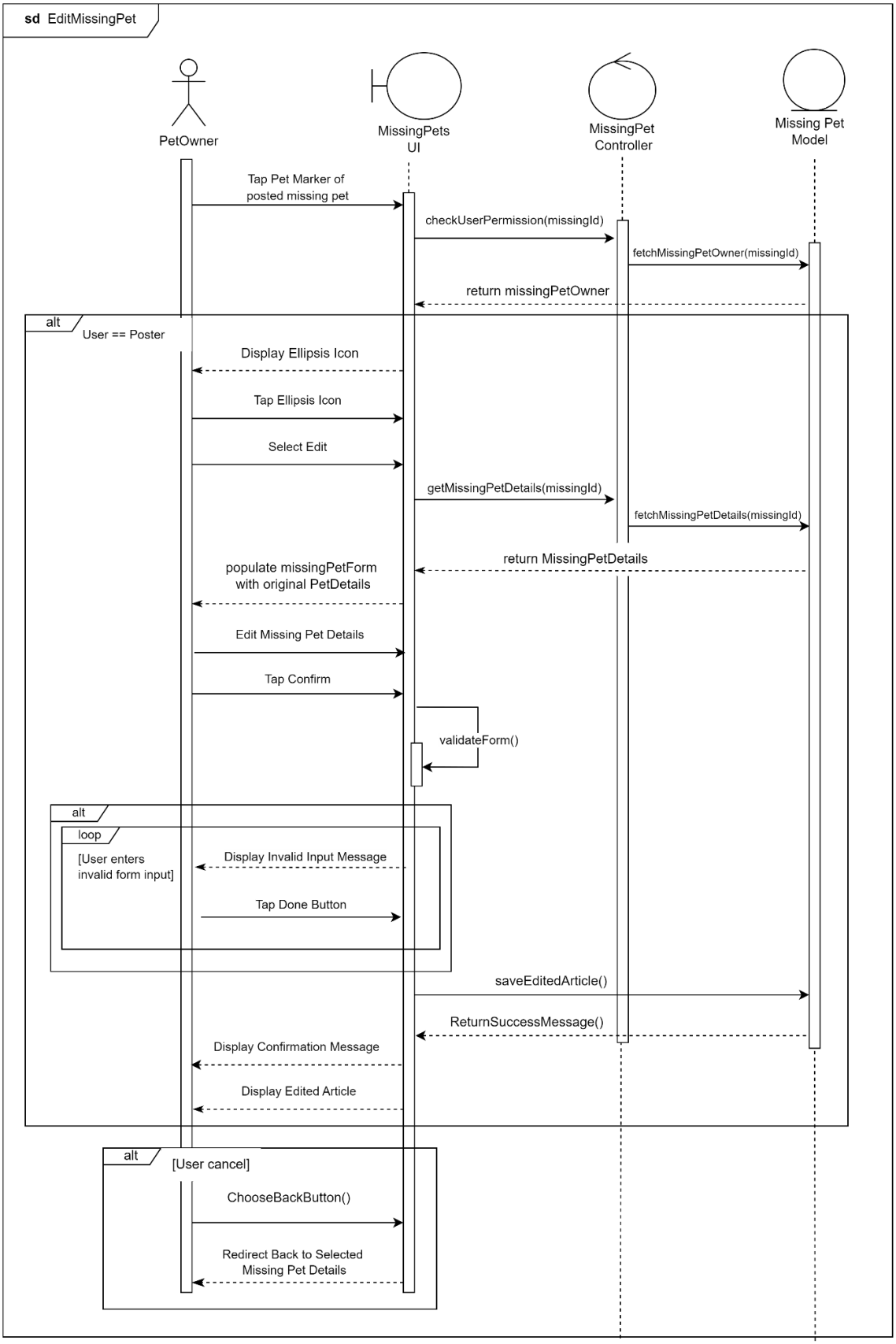


Edit Missing Pet Report

Use Case ID:	UC_MP_4		
Use Case Name:	Edit Missing Pet Report		
Created By:	Jasmine Tye	Last Updated By:	Jasmine Tye
Date Created:	28th August 2024	Date Last Updated:	19th October 2024

Actor:	Pet Owner
Description:	Allows a pet owner to make changes to details of their posted missing pet report.
Preconditions:	1. The pet owner has previously created a missing pet report of their registered pet. 2. The missing pet report is currently active in the database. 3. The user is logged in and authenticated as the owner of the selected missing pet report.
Postconditions:	1. The system reflects the changes made by the pet owner. 2. The updated information is displayed in the “Display Active Missing Pets” page.
Priority:	Medium
Frequency of Use:	Medium
Flow of Events:	1. The pet owner navigates to the “Display Missing Pets” page. 2. The pet owner selects the marker of their missing pet on the map. 3. The system displays the details of the selected missing pet report. 4. The pet owner taps on the “Edit” button. 5. The pet owner makes updates to one or more of the input below: a. Last seen location - postal codes only b. Last seen time c. Last seen date d. Last seen image e. Owner’s contact details f. Additional descriptions 6. The pet owner taps on the ‘Save’ button. 7. The system updates the details of the missing pet report. 8. The system displays a pop-up to confirm the change. 9. The pet owner is redirected back to the selected missing pet page where the updated details of their report is updated.
Alternative Flows:	AF-S3: The pet owner cancels the Edit process 1. The user taps on the back button. 2. The system displays a pop-up to inform the pet owner that no changes will be made to the missing pet report. 3. The user taps on the ‘Ok’ button. 4. The missing pet details remain unchanged. AF-S3: Missing or Incomplete form 1. The system displays a prompt to inform users that there are missing fields/ wrong data types that require attention. 2. Once all inputs are validated, the system returns back to step 7.

Exceptions:	EX2: Data update failure 1. If the system is unable to update the pet's status due to network issues or server issues. 2. The system displays an error message and prompts the user to try again later. 3. The pet status has not changed.
Includes:	None
Special Requirements:	None
Assumptions:	The pet owner provides accurate and most up-to-date information during editing.
Notes and Issues:	None

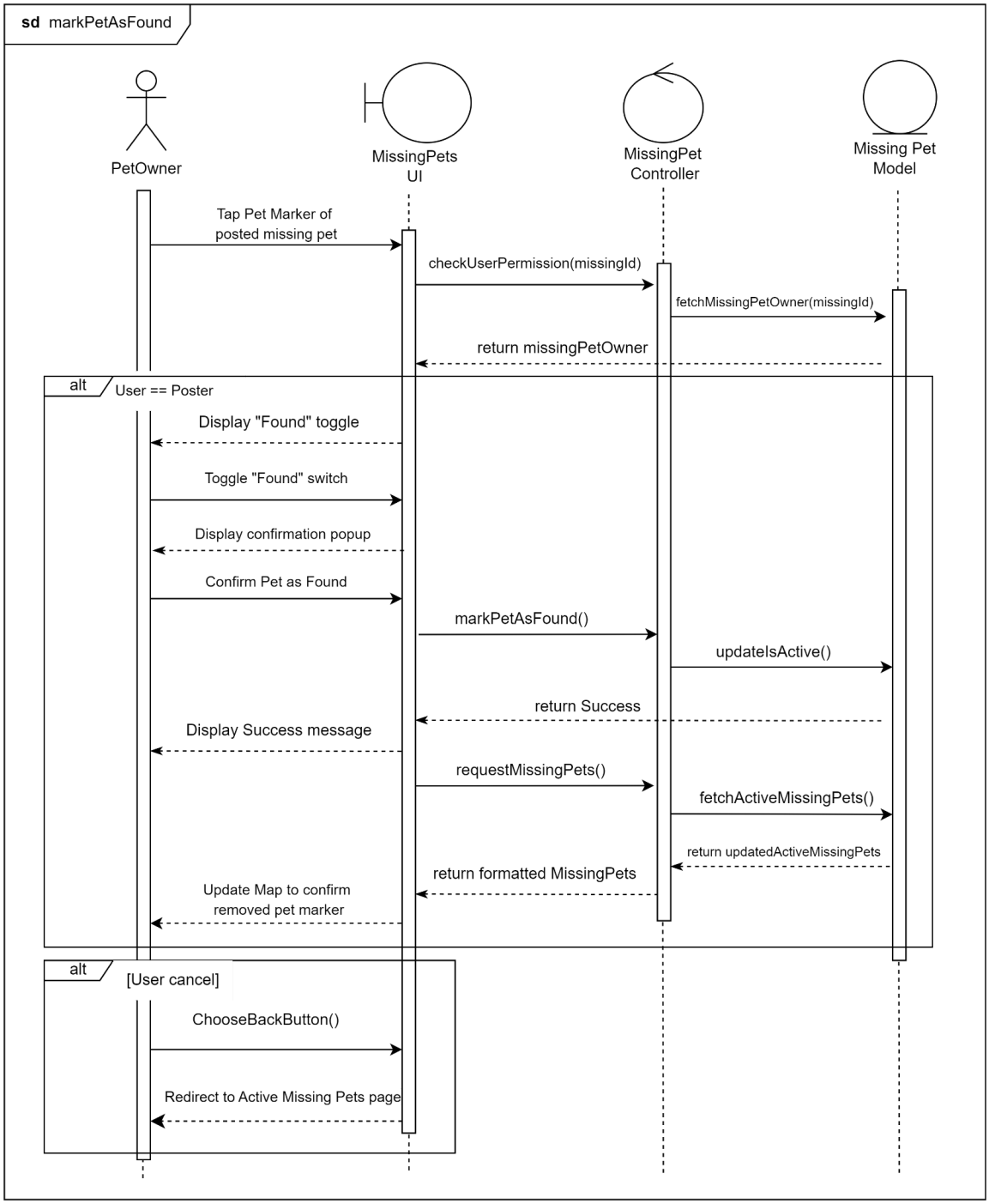

sd EditMissingPet
PetOwner
MissingPets UI
MissingPet Controller
Missing Pet Model
Tap Pet Marker of posted missing pet
checkUserPermission(missingId)
fetchMissingPetOwner(missingId)
return missingPetOwner
alt
User == Poster
Display Ellipsis Icon
Tap Ellipsis Icon
Select Edit
getMissingPetDetails(missingId)
fetchMissingPetDetails(missingId)
return MissingPetDetails
populate missingPetForm with original PetDetails
Edit Missing Pet Details
Tap Confirm
validateForm()
alt
loop
[User enters invalid form input]
Display Invalid Input Message
Tap Done Button
saveEditedArticle()
ReturnSuccessMessage()
Display Confirmation Message
Display Edited Article
alt
[User cancel]
ChooseBackButton()
Redirect Back to Selected Missing Pet Details

Mark Missing Pet as Found

Use Case ID:	UC_MP_5		
Use Case Name:	Mark Missing Pet as Found		
Created By:	Jasmine Tye	Last Updated By:	Jasmine Tye
Date Created:	26th August 2024	Date Last Updated:	26th August 2024

Actor:	Pet Owner
Description:	Allows the pet owner to mark their missing pet report.
Preconditions:	1. The pet is currently in the missing pets database. 2. The pet owner has reported a pet as missing and it is currently active in the “Display Missing Pets” page.
Postconditions:	1. The system updates the pet’s status to inactive in the database. 2. The missing pet report is no longer being displayed on the “Display Missing Pets” page.
Priority:	High
Frequency of Use:	Low
Flow of Events:	1. The pet owner navigates to the “Display Missing Pets” page in the app. 2. The pet owner selects the marker of their missing pet on the map. 3. The details of the selected missing report is shown. 4. The pet owner taps on the “Found” toggle switch which is only accessible to the pet owner. 5. The toggle switch changes to “ON” to indicate the pet has been marked as found. 6. A pop-up message displays to confirm that the pet has been found. 7. The pet owner taps on the “OK” button. 8. The system updates the pet’s status to inactive in the database. 9. The pet listing is hidden from the “Display Missing Pets” map interface. 10. The system shows a popup message to confirm that the report has been closed.
Alternative Flows:	AF-S3: Canceling the toggling of “Found” switch 1. The user taps on the “Cancel” button. 2. The pop-up messages closes without making any changes to the pet’s status. 3. The missing pet continues to be displayed in the “Display Missing Pets” map interface.
Exceptions:	EX2: Data update failure 1. If the system is unable to update the pet’s status due to network issues or server issues. 2. The system displays an error message and prompts the user to try again later. 3. The pet status has not changed.
Includes:	UC_MP_1 - Display Active Missing Pets

Special Requirements:	None
Assumptions:	The pet owner is authorized to update the status of the missing pet.
Notes and Issues:	None.

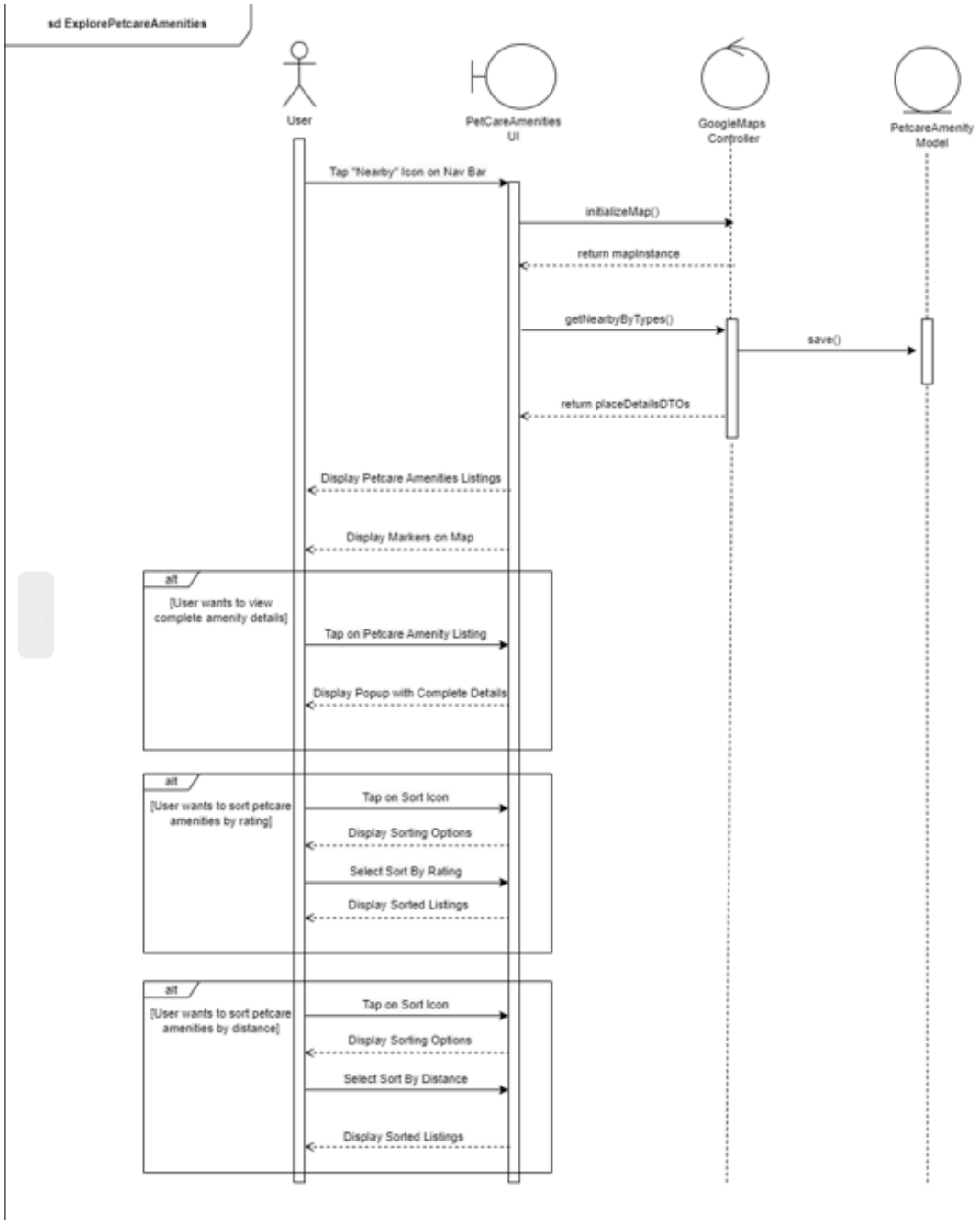


Explore Petcare Amenities

Use Case ID:	UC_PA_1		
Use Case Name:	Explore Petcare Amenities		
Created By:	Sih Jia Qi	Last Updated By:	Sih Jia Qi
Date Created:	25th August 2024	Date Last Updated:	15th September 2024

Actor:	User
Description:	Allows a user to search for nearby petcare amenities including veterinarian clinics, groomers and pet adoption centers based on their current location using a GPS-enabled mobile device. The system will display the search results as landmarks on a map interface. It will also display a list of nearby amenities on the Petcare Amenities page, providing options to sort, filter and search for a specific amenity by keywords.
Preconditions:	1. The user has a device with PetCare installed. 2. The user is logged in and authenticated. 3. The user has a device with GPS capabilities. 4. The application has access to the user's current location. 5. The user's device has Internet access. 6. The GoogleMaps API is fully functionable and returns the latest pet amenities information.
Postconditions:	1. The system displays a map with nearby pet amenities marked. 2. The user can interact with the map to view the detailed information about a selected vet. 3. The user can search, filter and sort the list of nearby pet amenities.
Priority:	High
Frequency of Use:	High
Flow of Events:	1. The user clicks on the Petcare Amenities icon on the navigation bar. 2. The system prompts the user to grant location permissions. 3. The system retrieves the user's current location using GPS. 4. The system queries the **GoogleMaps API** to search for nearby petcare amenities. 5. The system receives a list of nearby petcare amenities. 6. The system displays the returned locations as markers on the map. 7. The system also displays them as listings with their information on the Petcare Amenities page. 8. The user can sort the amenities by rating in descending order or by distance in ascending order and the app will display the results in the order that user selects. 9. The user can filter the pet amenities by ratings and/ or whether the amenity is open now. 10. The system will display the results according to the filter applied. 11. The user can search for a specific pet amenity by name or a keyword using the search bar. 12. The user can tap on a marker in the map or a listing to view more detailed information about the amenity, including its storefront image, distance, rating, name, address, contact

	information, website, open now and operating hours.
Alternative Flows:	AF-S2: User declines location access 1. The system prompts the user to manually enter their location or search a specific area. 2. The system displays the nearby pet amenities based on the user input location. AF-S5: No nearby pet amenities found 1. The system displays a message indicating that no results were found. 2. The system prompts the user to manually input a location or keyword and display it as a marker on the map and show its information in a listing below the map. AF-S7: Specific Information unavailable 1. Certain information of the selected amenity is unavailable: a. Storefront image - Display a default image provided by system b. Other info: Display “-” AF-S7: User wants to sort the amenities by rating 1. The system allows the user to sort the ratings of the petcare amenities in descending order or distance in ascending order. 2. The system reorders the list of petcare amenities based on the selected sorting options by the user.
Exceptions:	EX-S1: Location services unavailable 1. If the GPS is disabled or the system is unable to determine the user’s location, the system notifies the user and provides an option to manually input a location. EX-S2: Google Maps API Error 1. If the Google Maps API fails or returns an error, the system displays an error message that suggests the user to try again later.
Includes:	UC_PA_2 - Filter Petcare Amenities UC_PA_3 - Search for Petcare Amenities
Special Requirements:	Location service is enabled on the user’s device.
Assumptions:	Google Maps API provides accurate and up-to-date data for the pet amenities.
Notes and Issues:	None

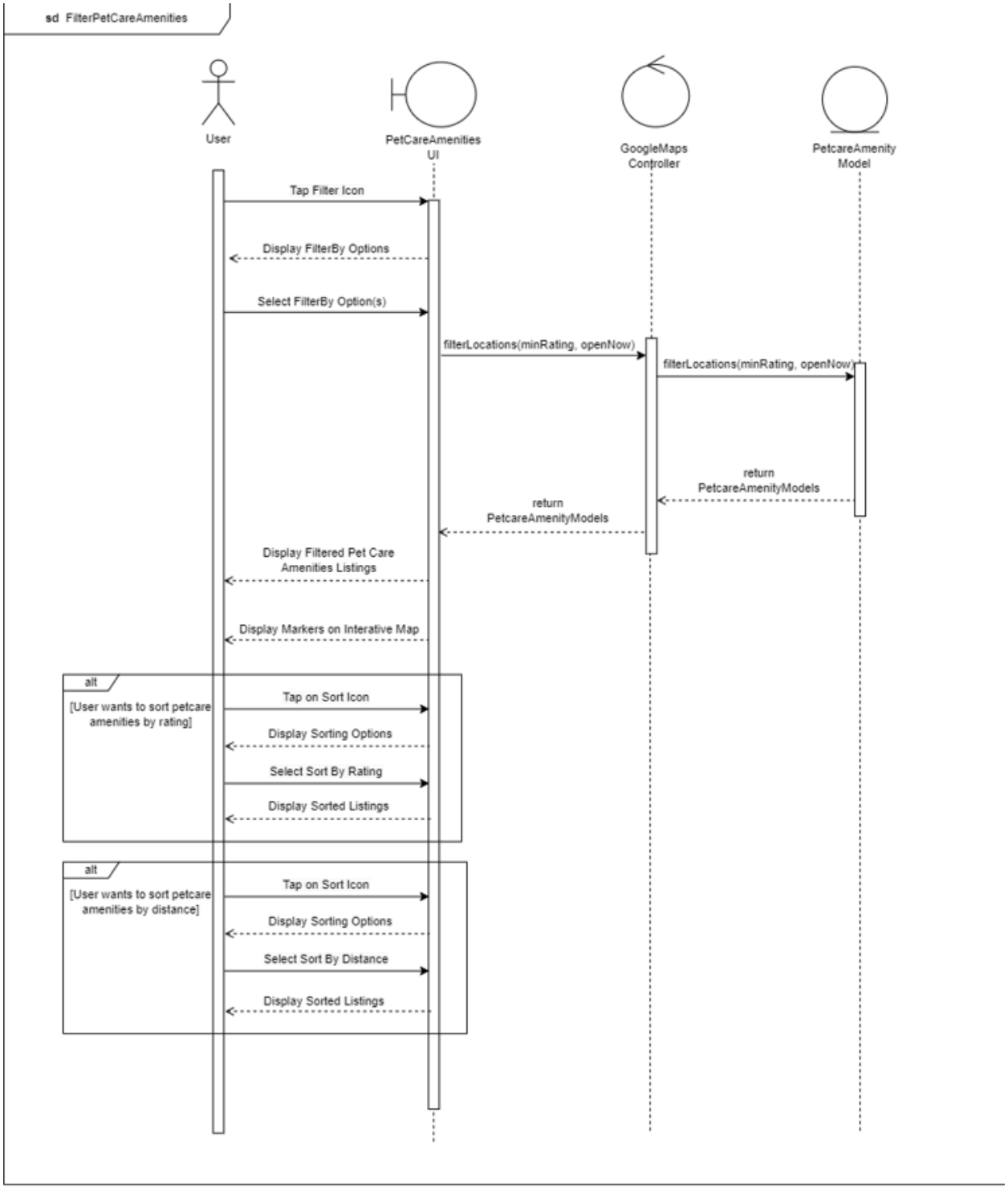

sd ExplorePetcareAmenities
User
PetCareAmenities UI
GoogleMaps Controller
PetcareAmenity Model
Tap "Nearby" Icon on Nav Bar
initializeMap()
return mapInstance
getNearbyByTypes()
save()
return placeDetailsDTOs
Display Petcare Amenities Listings
Display Markers on Map
alt
[User wants to view complete amenity details]
Tap on Petcare Amenity Listing
Display Popup with Complete Details
alt
[User wants to sort petcare amenities by rating]
Tap on Sort Icon
Display Sorting Options
Select Sort By Rating
Display Sorted Listings
alt
[User wants to sort petcare amenities by distance]
Tap on Sort Icon
Display Sorting Options
Select Sort By Distance
Display Sorted Listings

Filter Petcare Amenities

Use Case ID:	UC_PA_2		
Use Case Name:	Filter Petcare Amenities		
Created By:	Sih Jia Qi	Last Updated By:	Sih Jia Qi
Date Created:	26th August 2024	Date Last Updated:	15th September 2024

Actor:	User
Description:	Allows a user to sort the list of petcare amenities by rating in descending order or by distance in ascending order.
Preconditions:	1. The user has a device with PetCare installed. 2. The user is logged in and authenticated. 3. The user has a device with GPS capabilities. 4. The application has access to the user's current location. 5. The user's device has Internet access. 6. The GoogleMap API is fully functionable and returns the latest pet amenities information.
Postconditions:	1. The system displays the list of nearby petcare amenities sorted and/or filtered according to the options selected by the user. 2. The user is able to view the updated locations on the map and from the listings.
Priority:	High
Frequency of Use:	High
Flow of Events:	1. The user browses the Petcare Amenities page. 2. The user taps on the "Filter" icon on the screen. 3. The system allows the following filtering conditions:) a. Rating (eg. Any, 2 stars and above, 3 stars and above, 4 stars and above, 5 stars) b. Open now (eg. Any, Open now, Open 24 hours) 4. The user selects one or more filtering options. 5. The system applies the filters and only displays amenities that fit the filtering conditions. 6. The user can interact and tap on any of the displayed amenities on the map to view its details.
Alternative Flows:	AF-S5: No results returned after filtering 1. If no amenities match the filtering options, "No Location Found" message will be displayed on the screen. AF-S7: User wants to sort the amenities by rating 1. The system allows the user to sort the petcare amenity ratings in descending order or distance in ascending order. 2. The system reorders the list of petcare amenities based on the selected sorting options by the user.
Exceptions:	EX-S1: Location services unavailable 1. If the GPS is disabled or the system is unable to determine the user's location, the system notifies the user and provides an option to manually input a location
Includes:	None

Special Requirements:	The system must sort and filter the results without refreshing or reloading the page.
Assumptions:	The Petcare Amenities page must be fully functionable and display accurate results.
Notes and Issues:	None

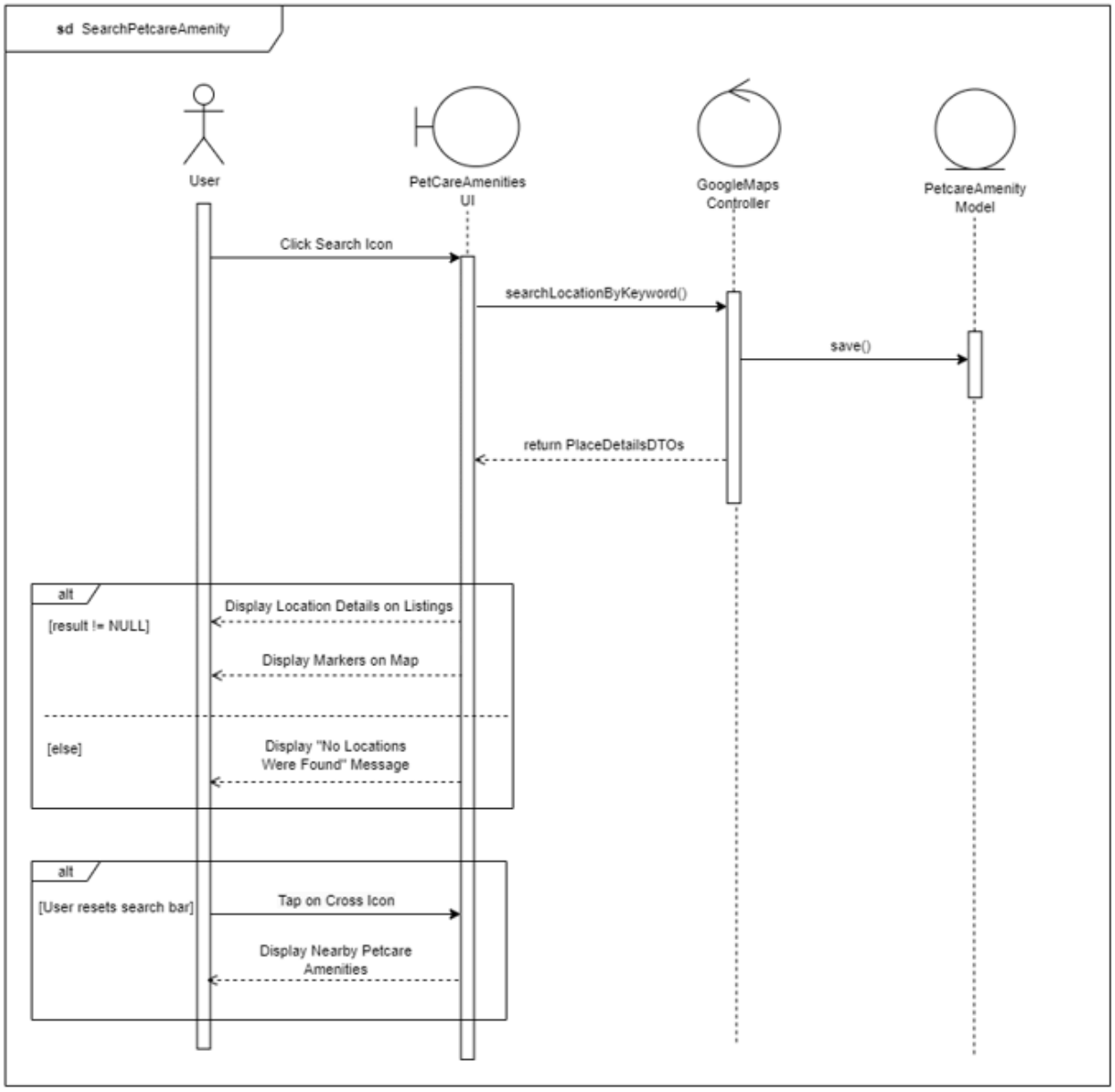

sd FilterPetCareAmenities
User
PetCareAmenities UI
GoogleMaps Controller
PetcareAmenity Model
Tap Filter Icon
Display FilterBy Options
Select FilterBy Option(s)
filterLocations(minRating, openNow)
filterLocations(minRating, openNow)
return PetcareAmenityModels
return PetcareAmenityModels
Display Filtered Pet Care Amenities Listings
Display Markers on Interative Map
alt
[User wants to sort petcare amenities by rating]
Tap on Sort Icon
Display Sorting Options
Select Sort By Rating
Display Sorted Listings
alt
[User wants to sort petcare amenities by distance]
Tap on Sort Icon
Display Sorting Options
Select Sort By Distance
Display Sorted Listings

Search Petcare Amenities

Use Case ID:	UC_PA_3		
Use Case Name:	Search Petcare Amenities		
Created By:	Sih Jia Qi	Last Updated By:	Sih Jia Qi
Date Created:	26th August 2024	Date Last Updated:	15th September 2024

Actor:	User
Description:	Allows a user to search for a specific pet amenity by its name or a keyword.
Preconditions:	1. The user has a device with PetCare installed. 2. The user is logged in and authenticated. 3. The user has a device with GPS capabilities. 4. The application has access to the user's current location. 5. The user's device has Internet access. 6. The GoogleMaps API is fully functionable and returns the latest pet amenities information.
Postconditions:	The system displays the petcare amenities that match the user's search query.
Priority:	Medium
Frequency of Use:	Medium
Flow of Events:	1. The user browses the Petcare Amenities page. 2. The user inputs a location or keyword in the search bar and taps the "Search" icon. 3. The system uses **Google Maps API** to query for petcare amenities that match the user input. 4. The map updates to only display the petcare amenities that match the user input. 5. The user can interact and tap on any of the displayed petcare amenities to view its details.
Alternative Flows:	AF-S4: Clear search bar 1. The user clears the input in the search bar. 2. The system displays all the nearby petcare amenities on the map and shows the listings below. AF-S4: No results were found 1. If no petcare amenity matches the search query (Google Maps API does not return any result), the system displays a message indicating that no results were found.
Exceptions:	EX1: Location services unavailable 1. If the GPS is disabled or the system is unable to determine the user's location, the system notifies the user and provides an option to manually input a location
Includes:	None
Special Requirements:	None
Assumptions:	Google Maps API can query based on user input and return results based on the search query.

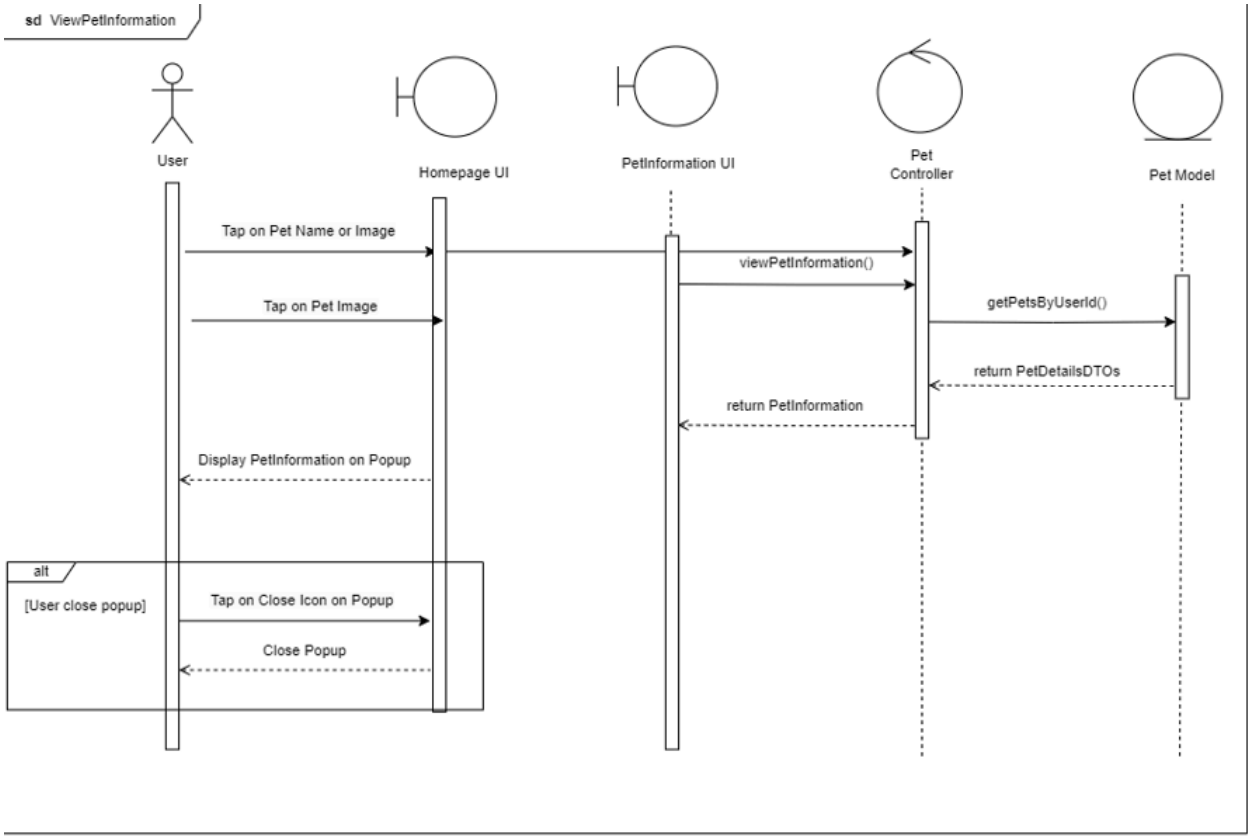
Notes and Issues:	None



View Pet Information

Use Case ID:	UC_PI_01		
Use Case Name:	View Pet Information		
Created By:	Sih Jia Qi	Last Updated By:	Sih Jia Qi
Date Created:	26th August 2024	Date Last Updated:	26th August 2024

Actor:	User
Description:	Allow users to view their pet information such as name, date of birth, breed, sex, weight, coat color, special markings and medic conditions.
Preconditions:	1. The user has a device with Petcare installed. 2. The user is logged in and authenticated. 3. The user has added at least a pet.
Postconditions:	Display the pet(s) information on the Pet Information pop-up.
Priority:	Medium
Frequency of Use:	Medium
Flow of Events:	1. The user clicks on the picture of the pet in the My Pets section located on the Home page to view the details of their registered pet. 2. The system displays the following details on the Pet Information popup: - Pet name - Date of birth - Breed - Sex - Weight - Coat color - Special markings - Medic conditions 3. On the Pet Information popup, the user can click on the “Edit” icon to edit the pet’s details. 4. On the Pet Information popup page, the user can click on the “Delete” icon to delete the pet and its details.
Alternative Flows:	AF-S2: If the database returns an empty result (no pets were found), the system displays a message showing that the user has not registered any pets.
Exceptions:	EX1: Data retrieval error 1. If the system is unable to connect to the database, the system returns a message prompting the user to try again later.
Includes:	UC_PI_02 - Edit Pet Information UC_PI_04 - Delete Pet
Special Requirements:	None
Assumptions:	Database is connected successfully.
Notes and Issues:	None

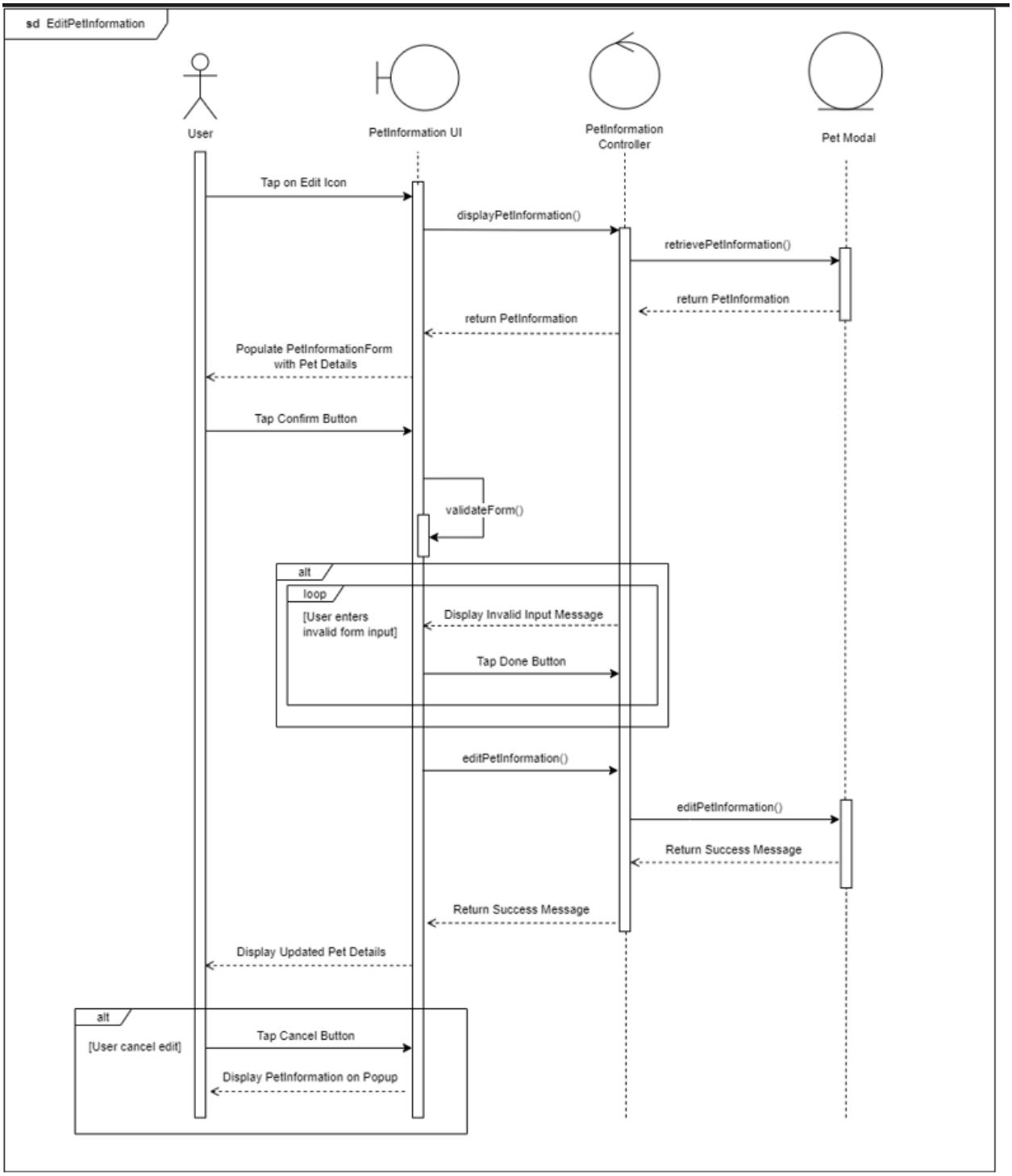

sd ViewPetInformation
User
Homepage UI
PetInformation UI
Pet Controller
Pet Model
Tap on Pet Name or Image
viewPetInformation()
Tap on Pet Image
getPetsByUserId()
return PetDetailsDTOs
return PetInformation
Display PetInformation on Popup
alt
[User close popup]
Tap on Close Icon on Popup
Close Popup

Edit Pet Information

Use Case ID:	UC_PI_02		
Use Case Name:	Edit Pet Information		
Created By:	Sih Jia Qi	Last Updated By:	Sih Jia Qi
Date Created:	26th August 2024	Date Last Updated:	26th August 2024

Actor:	User
Description:	Allow users to update their pet information such as name, date of birth, breed, sex, weight, coat color, special markings and medic conditions.
Preconditions:	1. The user has a device with PetCare installed. 2. The user is logged in and authenticated. 3. The user has registered at least a pet.
Postconditions:	Save and display the user's updated pet information.
Priority:	Medium
Frequency of Use:	Low
Flow of Events:	1. The user taps on the image or name of one of their pets on the My Pets section. 2. The user taps on the "Edit" icon on the Pet Information popup. 3. The user is redirected to the EditPetForm page. 4. The user can update the details of their pet including: - Pet name - Date of birth - Breed - Sex - Weight - Coat color - Special markings - Medic conditions 5. The user taps on the "Done" button after editing the details. 6. The system redirects the user to the Pet Information popup with the updated pet information displayed.
Alternative Flows:	AF-S4: The user does not save the changes 1. If the user clicks on the "Cancel" button on the editing page, all the edits will not be saved to the database. 2. The user will be redirected to the Pet Information popup with no pet information updated.
Exceptions:	EX4: Data update failure 1. If the system is unable to update the pet due to network issues or server issues, the system displays an error message and prompts the user to try again later. 2. The pet information will not change.
Includes:	None
Special Requirements:	None
Assumptions:	Database is connected.

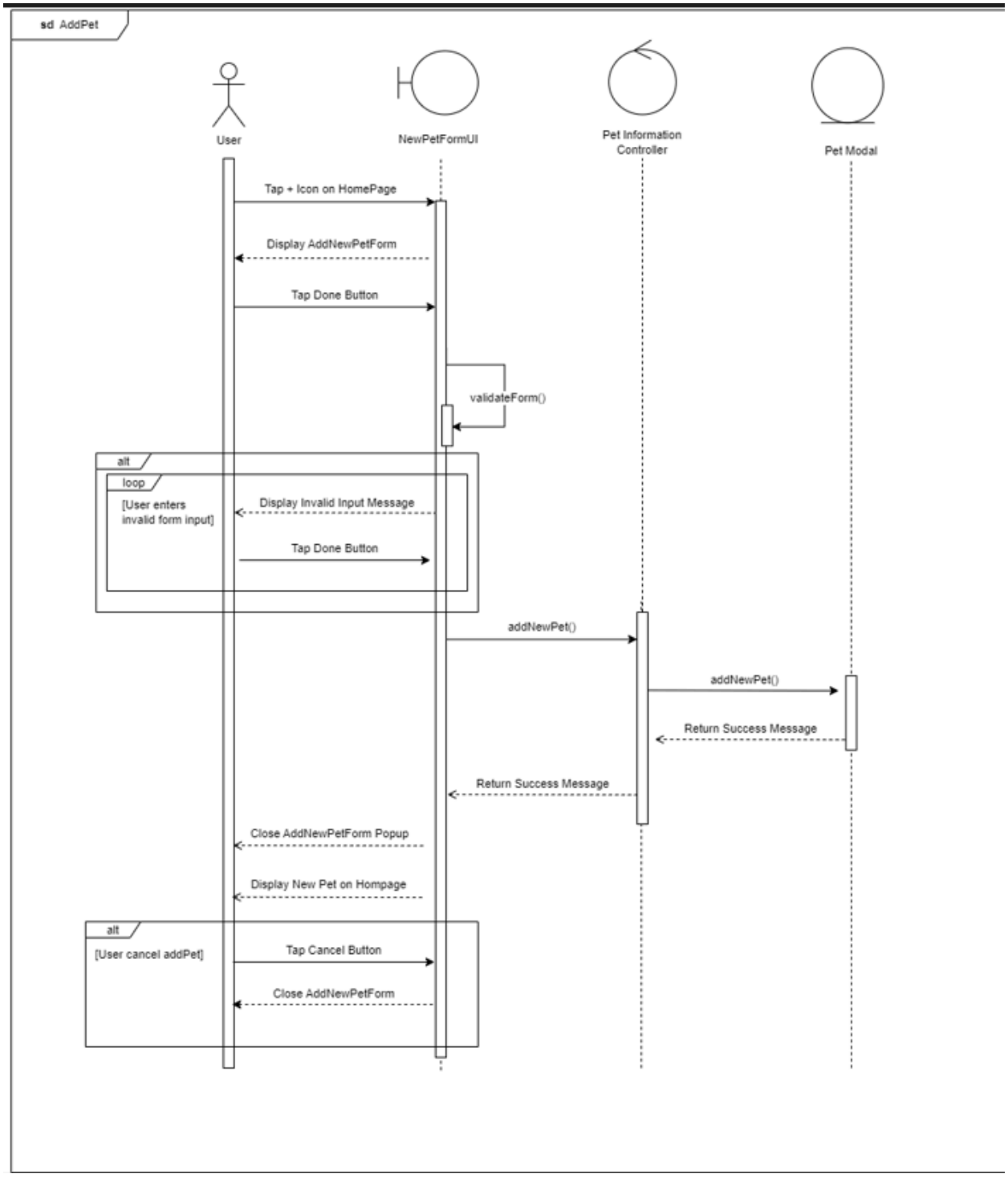
Notes and Issues:	None


sd EditPetInformation
User
PetInformation UI
PetInformation Controller
Pet Modal
Tap on Edit Icon
displayPetInformation()
retrievePetInformation()
return PetInformation
return PetInformation
Populate PetInformationForm with Pet Details
Tap Confirm Button
validateForm()
alt
loop
[User enters invalid form input]
Display Invalid Input Message
Tap Done Button
editPetInformation()
editPetInformation()
Return Success Message
Return Success Message
Display Updated Pet Details
alt
[User cancel edit]
Tap Cancel Button
Display PetInformation on Popup

Add Pet

Use Case ID:	UC_PI_03		
Use Case Name:	Add Pet		
Created By:	Sih Jia Qi	Last Updated By:	Sih Jia Qi
Date Created:	26th August 2024	Date Last Updated:	26th August 2024

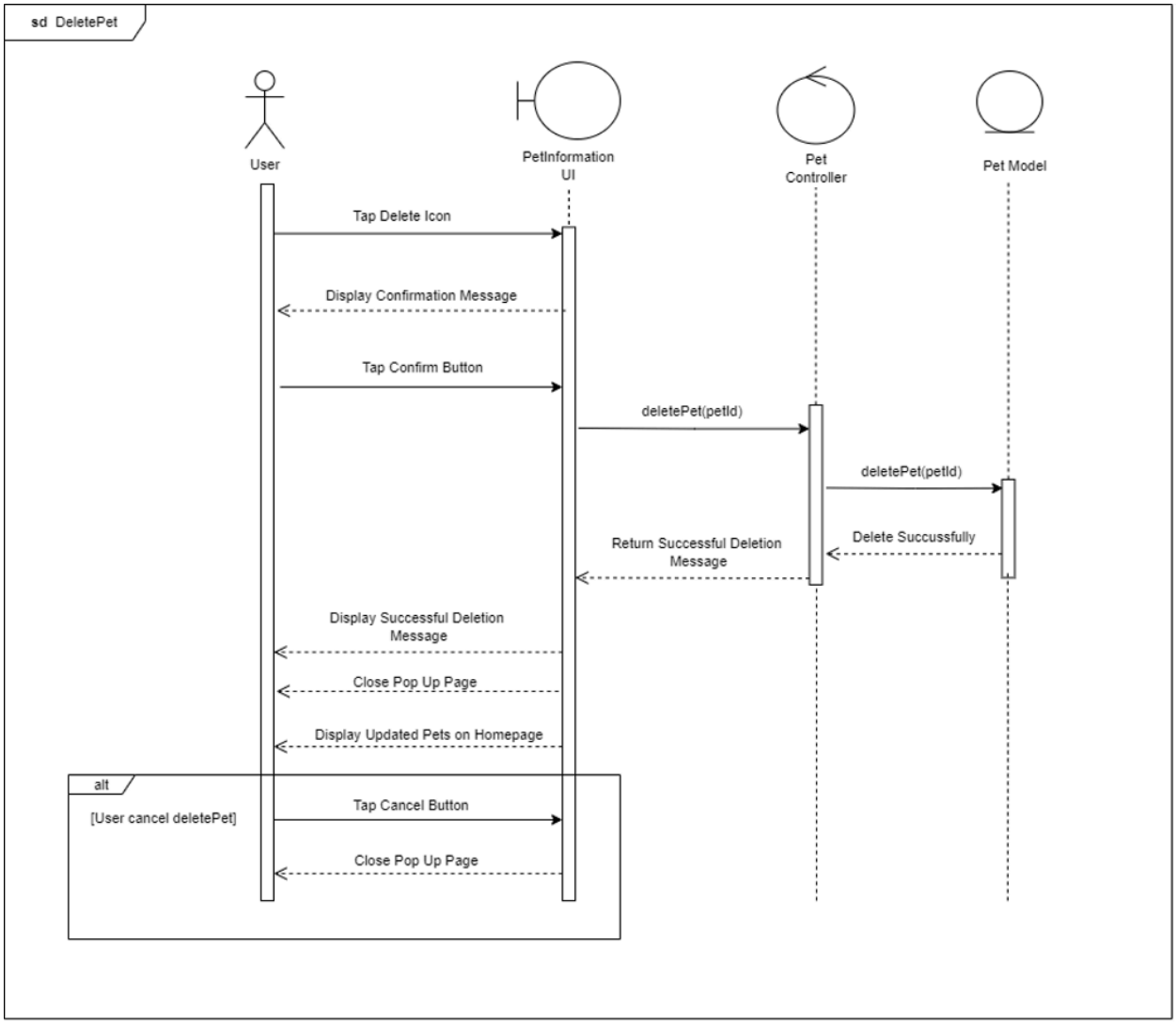
Actor:	User
Description:	Allow users to add a new pet and its information such as name, date of birth, breed, sex, weight, coat color, special markings and medic conditions.
Preconditions:	1. The user has a device with PetCare installed. 2. The user is logged in and authenticated. 3. The user has registered at least a pet.
Postconditions:	The new registered pet is added successfully into the database and the system displays it in the My Pets section on the Home page
Priority:	Medium
Frequency of Use:	Medium
Flow of Events:	1. The user clicks on the '+' icon in the My Pets section on the Home page. 2. The system redirects the user to the AddPetForm UI with the following input boxes: a. Pet name b. Date of birth c. Breed d. Sex e. Weight f. Coat color g. Special markings h. Medic conditions 3. The user taps on the "Done" button to submit the form after entering the information of its pet. 4. The system redirects the user to the Home page.
Alternative Flows:	AF-S3: The user does not want to add a new pet 1. If the user taps on the "Cancel" button, the system will redirect the user back to the Home page with no new pet added.
Exceptions:	EX3: Form submission error 1. Form submission fails due to network issues or server errors, the system displays an error pop-up message asking the user to try again later.
Includes:	None
Special Requirements:	None
Assumptions:	Database is connected.
Notes and Issues:	None


sd AddPet
User
NewPetFormUI
Pet Information Controller
Pet Modal
Tap + Icon on HomePage
Display AddNewPetForm
Tap Done Button
validateForm()
alt
loop
[User enters invalid form input]
Display Invalid Input Message
Tap Done Button
addNewPet()
addNewPet()
Return Success Message
Return Success Message
Close AddNewPetForm Popup
Display New Pet on Hompage
alt
[User cancel addPet]
Tap Cancel Button
Close AddNewPetForm

Delete Pet

Use Case ID:	UC_PI_04		
Use Case Name:	Delete Pet		
Created By:	Sih Jia Qi	Last Updated By:	Sih Jia Qi
Date Created:	26th August 2024	Date Last Updated:	26th August 2024

Actor:	User
Description:	Allow users to delete their pet and its information.
Preconditions:	1. The user has a device with PetCare installed. 2. The user is logged in and authenticated. 3. The user has registered at least a pet.
Postconditions:	The deleted pet will be removed from the My Pets section.
Priority:	Medium
Frequency of Use:	Low
Flow of Events:	1. The user taps on the image or name of one of their pets in the My Pets section on the Home page. 2. The user taps on the “Delete” icon on the Pet Information popup. 3. The system displays a popup with the “Confirm” and “Cancel” buttons. 4. The user taps on the “Confirm” button. 5. The user is redirected to the Home page.
Alternative Flows:	AF-S3: Cancel delete option 1. If the user taps on the “Delete” button, the user will be redirected back to the Home page with no changes made to the pet information.
Exceptions:	EX4: Data deletion failure 1. If the system is unable to delete the pet due to network issues or server issues, the system displays an error message and prompts the user to try again later. 2. The pet is not deleted and will be displayed in the My Pets section.
Includes:	None
Special Requirements:	None
Assumptions:	Database is connected.
Notes and Issues:	None

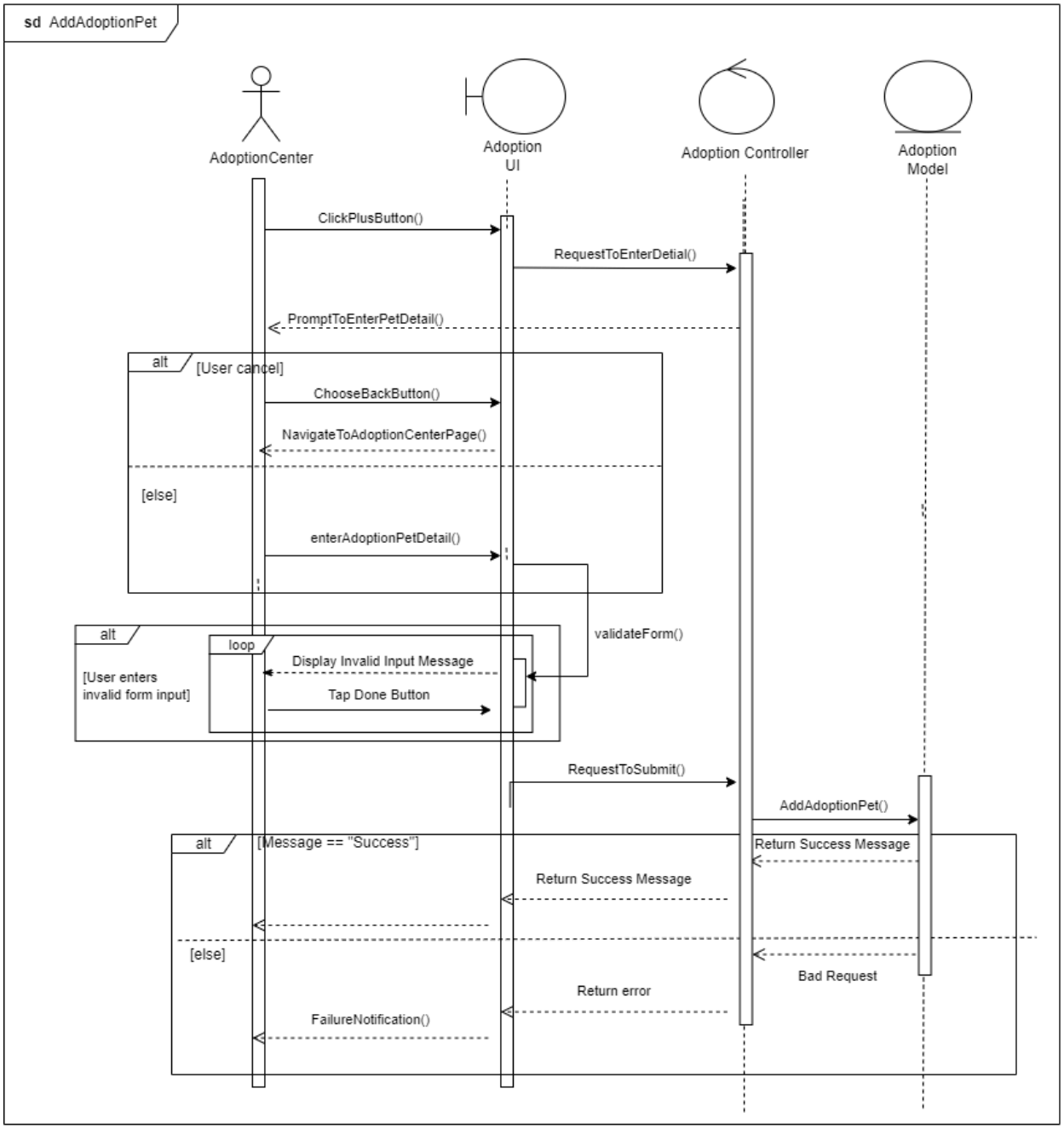

sd DeletePet
User
PetInformation UI
Pet Controller
Pet Model
Tap Delete Icon
Display Confirmation Message
Tap Confirm Button
deletePet(petId)
deletePet(petId)
Delete Successfully
Return Successful Deletion Message
Display Successful Deletion Message
Close Pop Up Page
Display Updated Pets on Homepage
alt
[User cancel deletePet]
Tap Cancel Button
Close Pop Up Page

Add Adoption Pet

Use Case ID:	UC_ADPT_01		
Use Case Name:	Add Adoption Pet		
Created By:	Lim Chun Wen	Last Updated By:	Lim Chun Wen
Date Created:	26th August 2024	Date Last Updated:	10th October 2024

Actor:	Adoption Center
Description:	Allows the adoption center to upload pets that are available for adoption.
Preconditions:	1. The adoption center is logged in and authenticated in admin format 2. The adoption center has the pet and wants to publish an adoption.
Postconditions:	1. The system successfully saves and publishes the pets to the app. 2. Other users are able to view the available pets in listview format.
Priority:	High
Frequency of Use:	Low
Flow of Events:	1. The system displays a “Add” button at “Adoption” page for adoption center to post their listing on a particular pets to the “adoption center” 2. When adoption center selects the “Add Button”,the system will bring to another page to fill up the details on the pets 3. Adoption center will manually fills in the item form: a. Pets Name b. Description c. Age d. Breed e. Color f. Gender g. Upload Images through phone camera h. Type i. Adoption Center : Name j. Adoption Center : Phone Number k. Adoption Center : Email 4. If the adoption center selects the “Submit” button,the system will save this information into the database and display a prompt to inform the user that “pets have been successfully posted”.
Alternative Flows:	AF-S4: Missing or Incomplete form 1. The system displays a prompt to inform the user that there are missing fields / wrong data types that require attention. 2. Once everything is validated , the system will go to step 4.
Exceptions:	EX1: Form submission fails due to network issues or server errors, the system displays an error popup message asking the user to try again

	at a later time.
Includes:	None
Special Requirements:	System needs to validate input data.
Assumptions:	Pets are in good or proper condition for adoption.
Notes and Issues:	None

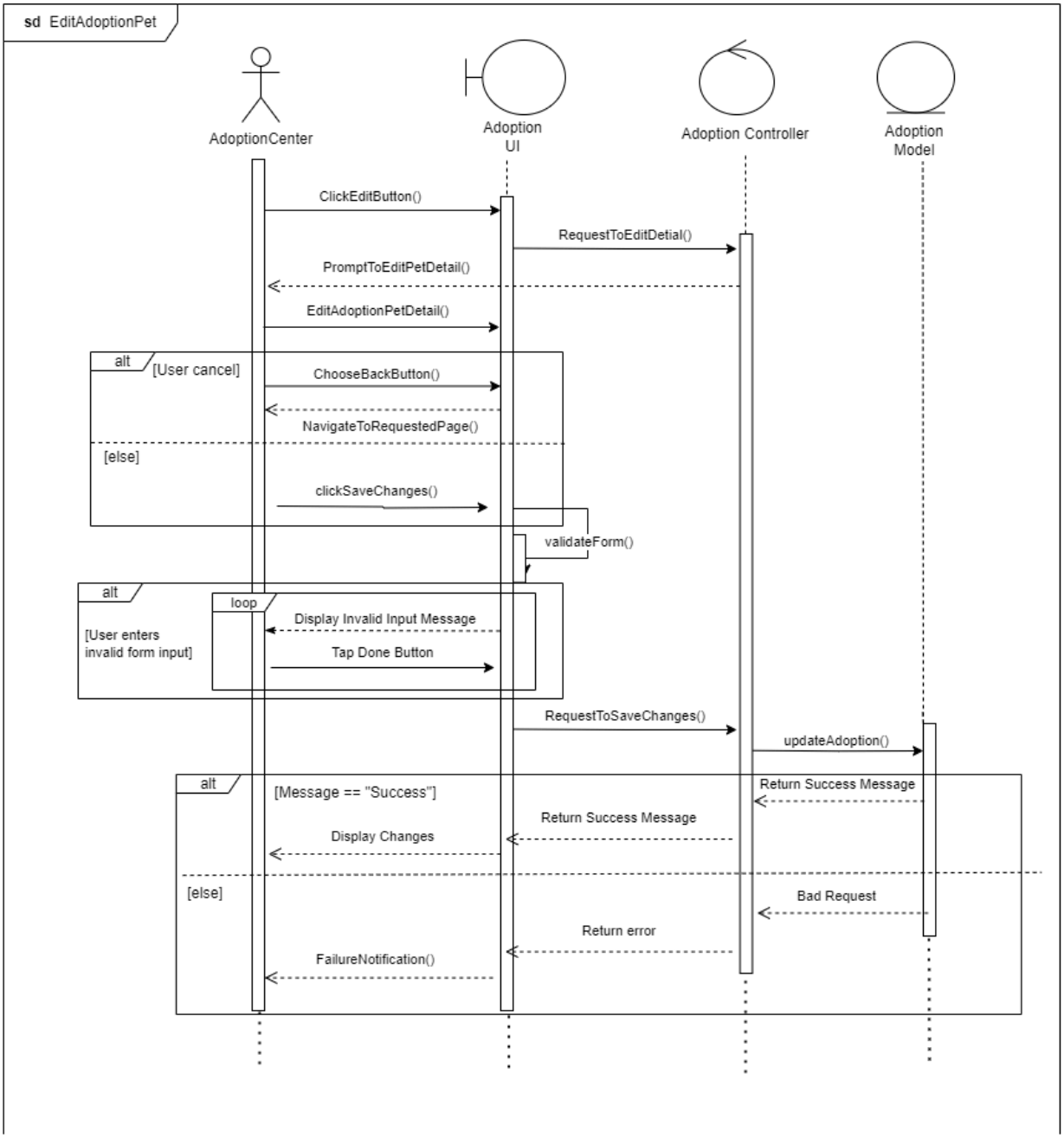


Edit Adoption Pet

Use Case ID:	UC_ADPT_02		
Use Case Name:	Edit Adoption Pet		
Created By:	Lim Chun Wen	Last Updated By:	Lim Chun Wen
Date Created:	26th August 2024	Date Last Updated:	10th October 2024

Actor:	Adoption Center
Description:	Allows the users to edit the pet detail that have uploaded for listings
Preconditions:	1. The user is logged in and authenticated in admin state. 2. The user has uploaded or created the pet posting in the database
Postconditions:	1. The system successfully saves and updates the information in the app. 2. Other users are able to view the updated listing in listview format.
Priority:	High
Frequency of Use:	Low
Flow of Events:	1. The system displays a “Edit ” button at page under a particular pets 2. When user selects the “Edit Button”,User will be able to modify the information in the form: a. Pets Name b. Price c. Description d. Age e. Breed f. Color g. Gender h. Upload Images via phone storage i. Type j. Adoption Center : Name k. Adoption Center : Phone Number l. Adoption Center : Email 3. If the user selects the “Submit” button,the system will save this information into the database and display a prompt to inform the user that “Pets have been successfully edited”.
Alternative Flows:	AF-S3: Missing or Incomplete form 3. The system displays a prompt to inform users that there are missing fields / wrong data types that require attention. 4. Once everything is validated , the system will go to step 3.
Exceptions:	EX1: Form submission fails due to network issues or server errors, the system displays an error popup message asking the user to try again at a later time.
Includes:	None

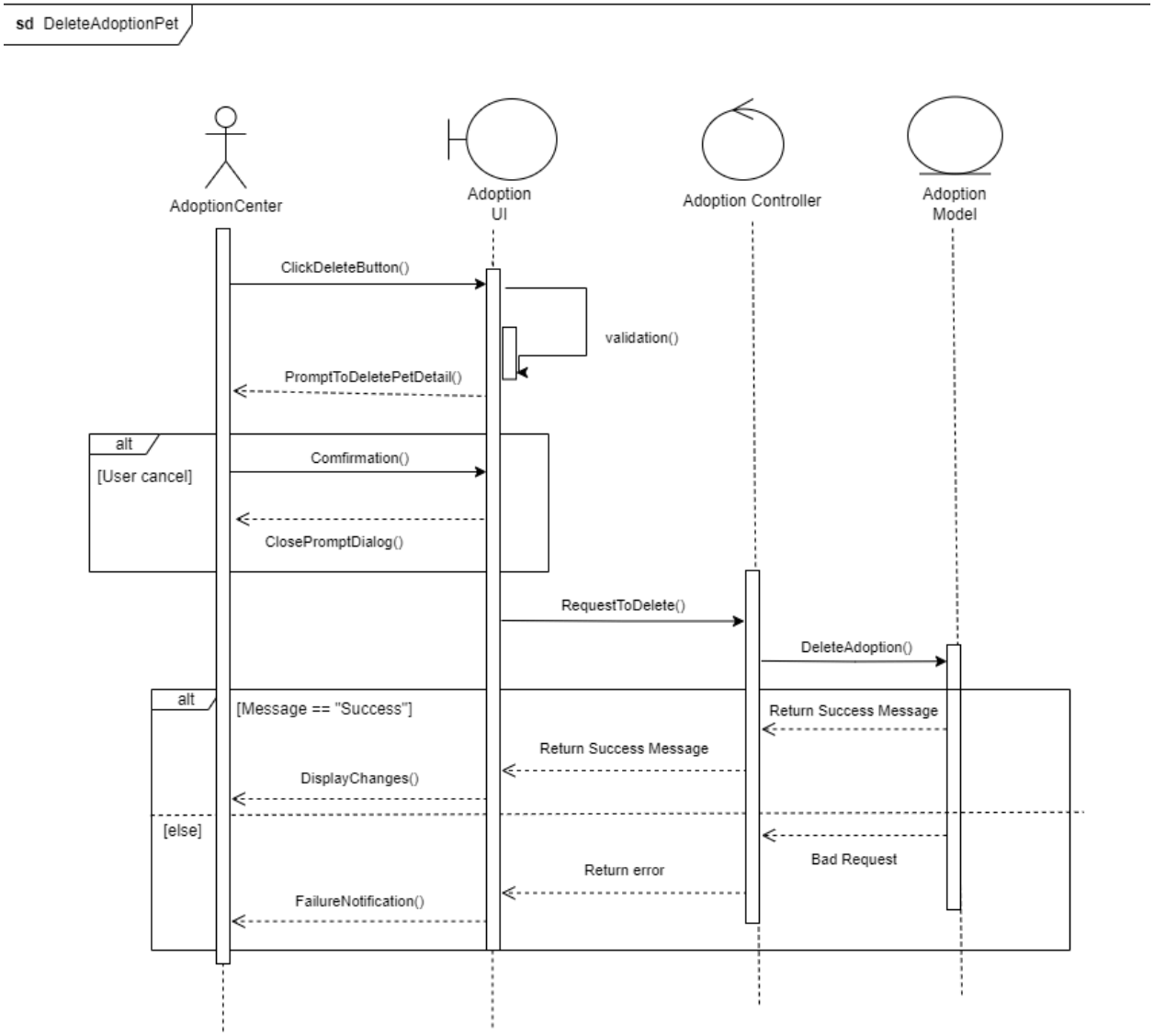
Special Requirements:	System needs to validate user input data
Assumptions:	User have uploaded information of the pets and the data are captured in the database.
Notes and Issues:	None.



Delete Pet Adoption

Use Case ID:	UC_ADPT_03		
Use Case Name:	Delete Pet Adoption		
Created By:	Lim Chun Wen	Last Updated By:	Lim Chun Wen
Date Created:	26th August 2024	Date Last Updated:	26th August 2024

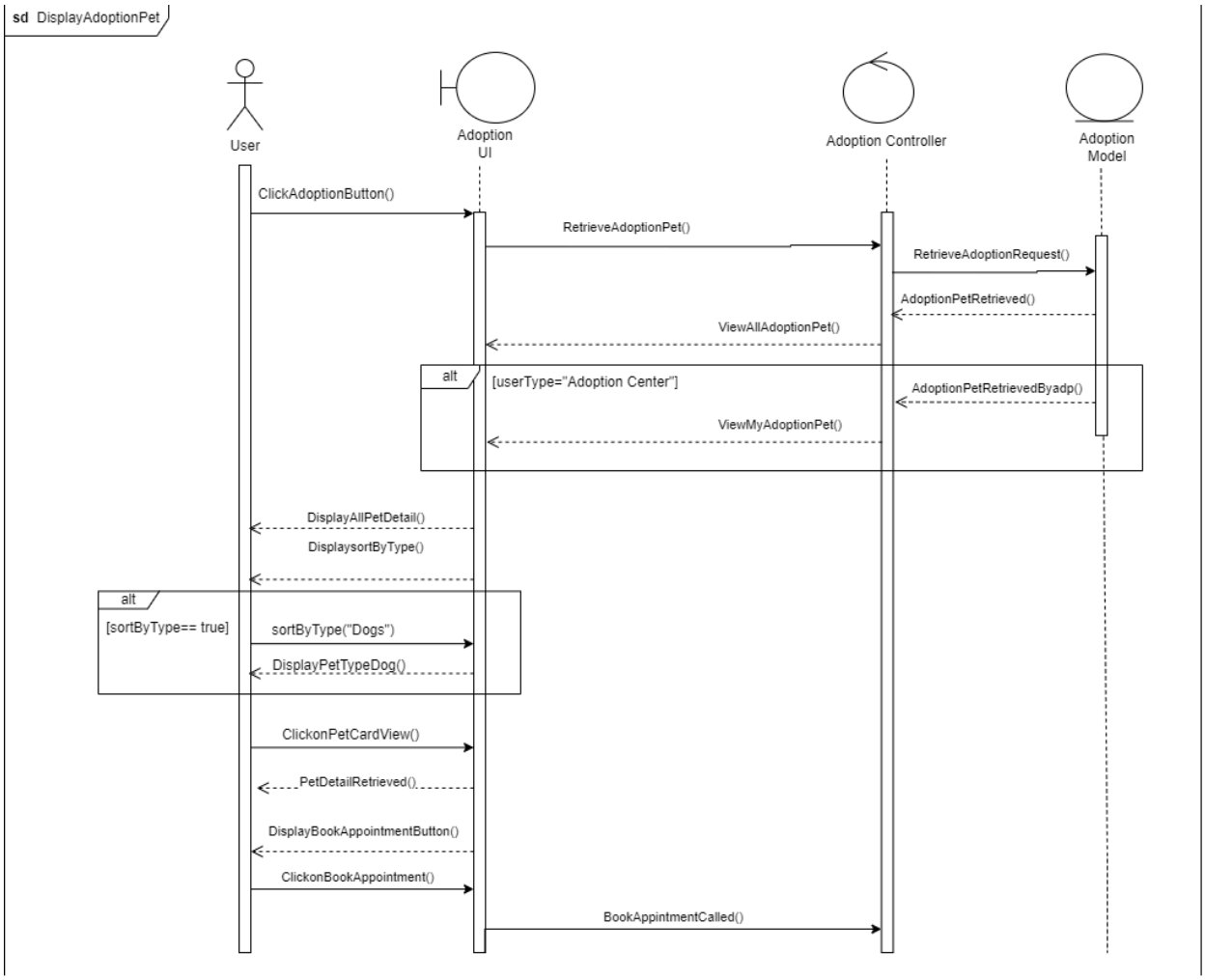
Actor:	Adoption Center
Description:	Allows adoption center to delete the detail of the pets once it has been adopted or not for adoption anymore
Preconditions:	1. The user is logged in and authenticated in the adoption center account. 2. The user has created the pet posting in the database
Postconditions:	1. The system successfully deletes the pet's details and updates the information in the database. 2. Item is removed from the "my pets house" page.
Priority:	High
Frequency of Use:	Low
Flow of Events:	1. The system displays a "Delete" button at individual pet page under individual page(adoption user) 2. When the user selects the "Delete Button",the system will prompt the user for confirmation. 3. When confirmation is clicked ,the system will display a dialogue box informing the user that "Pet has been successfully deleted".
Alternative Flows:	AF-S2 : User cancel the deletion 1. System will go back to individual pet page
Exceptions:	None
Includes:	None
Special Requirements:	None
Assumptions:	User want to remove the pet from the adoption listings
Notes and Issues:	None


sd DeleteAdoptionPet
AdoptionCenter
Adoption UI
Adoption Controller
Adoption Model
ClickDeleteButton()
validation()
PromptToDeletePetDetail()
alt
[User cancel]
Comfirmation()
ClosePromptDialog()
RequestToDelete()
DeleteAdoption()
alt
[Message == "Success"]
Return Success Message
Return Success Message
DisplayChanges()
[else]
Bad Request
Return error
FailureNotification()

Display Adoption Pet

Use Case ID:	UC_ADPT_04		
Use Case Name:	Display Adoption Pet		
Created By:	Lim Chun Wen	Last Updated By:	Lim Chun Wen
Date Created:	26th August 2024	Date Last Updated:	10th October 2024

Actor:	User
Description:	Allows the users to view the available pets uploaded by the adoption center and make contact with them.
Preconditions:	1. The user is logged in and authenticated. 2. The adoption center has the information of pets in the database.
Postconditions:	1. The user successfully makes contact with the adoption and makes an appointment.
Priority:	High
Frequency of Use:	Low
Flow of Events:	1. User navigates to the Adoption page. 2. System retrieves and display all the currently available pets for adoption 3. System will have a choice of filter by type Dog,Cats and Others 4. User click on one of the pet listing and a pop out modal will be shown to view the detail of the pets 5. User clicks on “Book appointment”,the system will open WhatsApp and make contact with the adoption center. 6. Buyer successfully booked an appointment with the adoption center.
Alternative Flows:	AF-S3 : User filter pet by type 1. System will filter out and only display the type of pet selected by the user at the filter section 2. Flow continue back to step 4
Exceptions:	EX1: The user does not have WhatsApp installed on their mobile device.
Includes:	
Special Requirements:	None
Assumptions:	User do not misused the system
Notes and Issues:	None

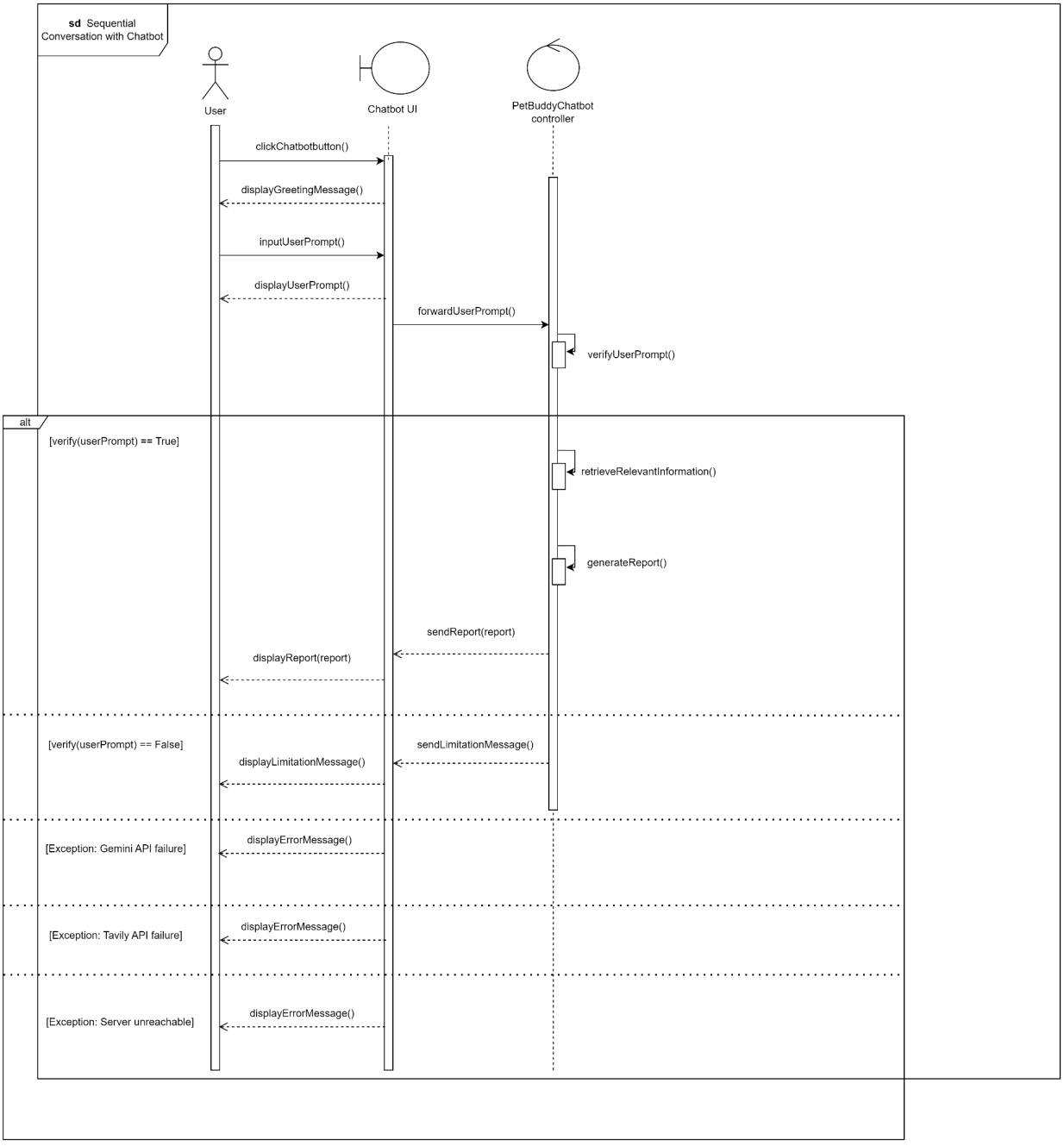

sd DisplayAdoptionPet
User
Adoption UI
Adoption Controller
Adoption Model
ClickAdoptionButton()
RetrieveAdoptionPet()
RetrieveAdoptionRequest()
AdoptionPetRetrieved()
ViewAllAdoptionPet()
alt
[userType="Adoption Center"]
AdoptionPetRetrievedByadp()
ViewMyAdoptionPet()
DisplayAllPetDetail()
DisplaysortByType()
alt
[sortByType== true]
sortByType("Dogs")
DisplayPetTypeDog()
ClickonPetCardView()
PetDetailRetrieved()
DisplayBookAppointmentButton()
ClickonBookAppointment()
BookAppintmentCalled()

Sequential Conversation With PetBuddy Chatbot

Use Case ID:	UC_CB_01		
Use Case Name:	Sequential Conversation With Chatbot		
Created By:	Hng Cherng Khai	Last Updated By:	Hng Cherng Khai
Date Created:	26th August 2024	Date Last Updated:	26th August 2024

Actor:	User
Description:	This use case outlines the interaction between the user and the PetBuddy chatbot, focusing on a structured conversation where the chatbot assists with pet care inquiries. The chatbot begins with a greeting and waits for the user's input. Based on the prompt, the chatbot processes the request using Google Gemini and Tavily APIs to generate and deliver relevant responses in a report format.
Preconditions:	1. The user has accessed the PetCare app and opened the chatbot interface. 2. Tavily API is fully functional and integrated into the system. 3. Google Gemini API is fully functional and integrated into the system.
Postconditions:	1. The user receives a detailed response or report based on their pet care-related prompt and online sources. 2. The conversation flow follows a structured sequence of input and response, ensuring that only relevant prompts are processed. 3. If the prompt is unrelated to pet care or involves map locations, the chatbot informs the user of its limitations.
Priority:	High
Frequency of Use:	High
Flow of Events:	1. User clicks on the PetBuddy Chatbot icon. 2. The user is navigated to the PetBuddy Chatbot page. 3. PetBuddy Chatbot automatically initiates the conversation with a greeting: ● "Hi there! 🐾 I'm PetBuddy, your pet care research assistant. How can I help you and your pet today?" 4. User inputs a text prompt into the chat box. 5. PetBuddy chatbot displays the user's input in the chat interface. 6. PetBuddy chatbot forwards the user's input to the Google Gemini API. 7. Google Gemini API verifies if the user's prompt is related to a pet care topic and does not involve a map location search. 8. Google Gemini API sends the verification outcome to the PetBuddy chatbot. 9. If the user's prompt is related to a pet care topic and does not involve a map location search, the PetBuddy chatbot forwards the user's prompt to the Tavily API for information retrieval. 10. Tavily API searches online sources, scrapes and filters relevant information based on the prompt. 11. PetBuddy chatbot receives the relevant information from Tavily API. 12. PetBuddy chatbot sends a system prompt to the Gemini API to generate a report based on the received information. 13. Gemini API generates the report, including reference links, and sends it to the PetBuddy chatbot. 14. The PetBuddy chatbot displays the report in the chat interface. 15. Step 4 to Step 11 repeats until the user clicks on the "back" button to close the chat.

	16. The user navigates back to the home screen.
Alternative Flows:	AF-S8: If the prompt is unrelated to pet care 1. The PetBuddy chatbot displays a limitation message clarifying its limitations: - "I'm sorry, I can only assist with pet care-related questions. Please ask me something about pets, their care, or related topics." 2. The system returns to step 4. AF-S8: If the prompt involves map location search 1. The PetBuddy chatbot displays a limitation message clarifying its limitations: - "I'm sorry, I cannot assist in searching map locations. Please ask me something about pets, their care, or related topics that are not related to map location search." 2. The system returns to step 4.
Exceptions:	EX1: APIs Unavailable: If either the Tavily API or Gemini API is unavailable, the PetBuddy chatbot must respond with a fallback message: - "Error occurred. Please try again later." EX2: Server Unreachable: If the PetBuddy chatbot cannot connect to the backend server, it must respond with the following message: - "Error occurred. Please try again later."
Includes:	xxx- Retrieve Information xxx- Generate Report
Special Requirements:	None
Assumptions:	1. Gemini API verifies correctly if the user's prompt is related to a pet care topic and does not involve a map location search. 2. The report generated by Gemini API includes reference links, and sends it to the PetBuddy chatbot. 3. The Gemini API can generate a detailed and accurate report based on the data received.
Notes and Issues:	None


sd Sequential Conversation with Chatbot
User
Chatbot UI
PetBuddyChatbot controller
clickChatbotbutton()
displayGreetingMessage()
inputUserPrompt()
displayUserPrompt()
forwardUserPrompt()
verifyUserPrompt()
alt
[verify(userPrompt) == True]
retrieveRelevantInformation()
generateReport()
sendReport(report)
displayReport(report)
[verify(userPrompt) == False]
sendLimitationMessage()
displayLimitationMessage()
[Exception: Gemini API failure]
displayErrorMessage()
[Exception: Tavily API failure]
displayErrorMessage()
[Exception: Server unreachable]
displayErrorMessage()

Dialog Map

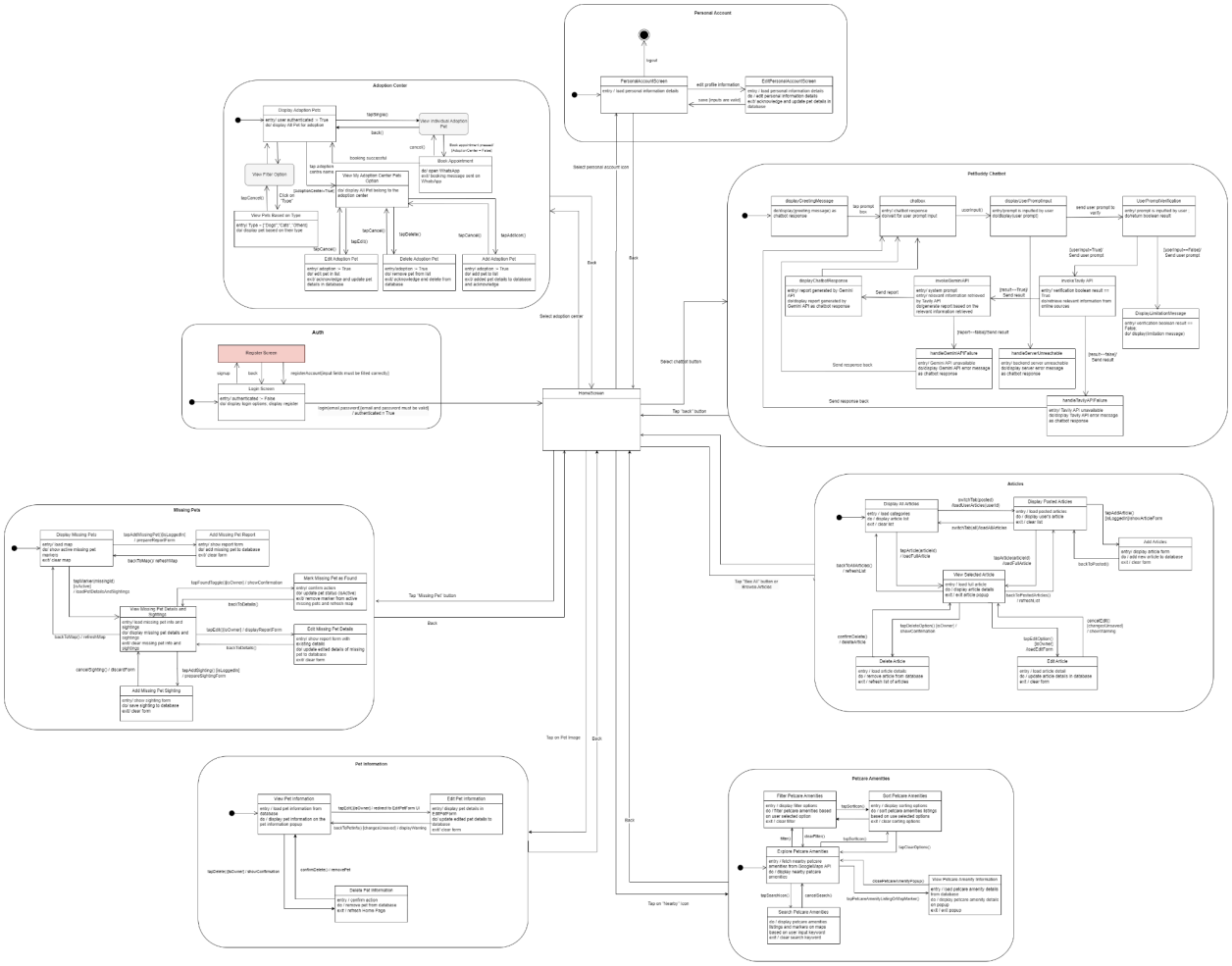


Figure 1: PetCare Dialog Map

System Architecture

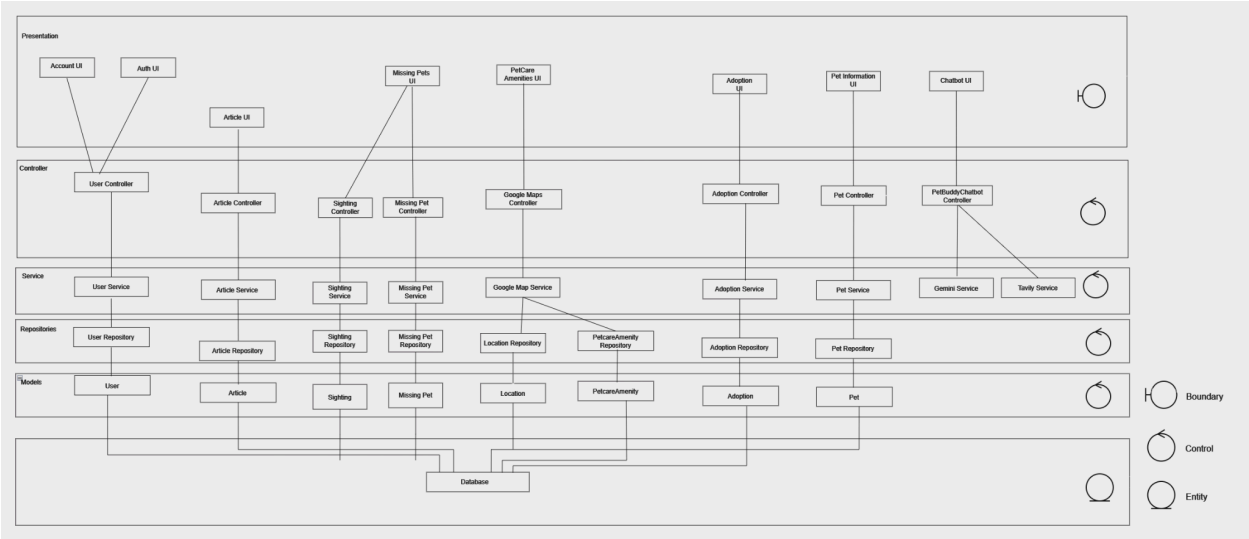


Figure 2. PetCare System Architecture Diagram

The layered architecture that includes controllers, models, repositories and the service layer is adopted in our application as it offers several benefits. First, it enhances modularity. Each layer handles a specific responsibility, which improves code modularity. Developers can work on different layers independently without introducing bugs in other parts of the system. In terms of maintainability, each layer handles its own logic. For instance, controllers handle HTTP requests, services handle business logic, repositories handle database queries. This separation reduces complexity and makes it easier to manage and update the codebase. It also offers flexibility as it allows us to swap out components with minimal impact on the rest of the system. For example, changing the database provider would not affect the business logic or controller layer. Overall, this design adheres to best practices like Separation of Concerns, modularity, and testability, which are crucial to make our application robust.

Presentation Layer

This layer is mainly responsible for the interaction between Users and PetCare. The different UIs will then call for the respective controllers to run the App Logic. This layer consist of:

UI	Description
Account	Account UI will allow users to view their personal account information and update their own profile information by calling User Controller.
Auth	Auth UI will allow users to register for an account, login to the PetCare account and reset their PetCare password if the user

	forgets their PetCare account password by calling User Controller.
Article	Article UI will allow the user to do its job by calling Article Controller.
MissingPet	MissingPet UI will allow the user to do its job by calling MissingPet Controller.
PetCare Amenities	PetCare Amenities UI will allow users to do its job by calling PetCare Amenities Controller.
Adoption	Adoption UI will allow user and adoption center to do its job by calling Adoption Controller
PetInformation	PetInformation UI will allow the user to do its job by calling the PetInformation Controller.
Chatbot	Chatbot UI will allow the user to do its job by calling the Chatbot controller.

Controller Layer

The Controller Layer acts as the intermediary between the Presentation Layer (UI) and the Service Layer. It receives requests from the UI, processes those requests by invoking the appropriate services, and then sends the results back to the UI. The Controller Layer is primarily responsible for handling user input and coordinating the flow of data between the frontend and backend.

Controller	**Description**
User	User controller is called by Account UI and Auth UI for account creation such as registration,login,reset password,view personal account details and edit personal details. It Interacts with UserService to process user related operations and return results.
Article	Article controller is called by Article UI to handle article related requests.It Interacts with Article Service to process Article related operations and return results.
MissingPet	MissingPet controller is called by MissingPet UI and GoogleMaps controller to handle any reporting of missing pet related request.It Interacts with MissingPet Service and GoogleMaps service to process reporting related operations and return results.
Sighting	Sighting Controller is called by MissingPet UI to handle any sighting of the missing pet the user found. It interacts with Sighting Service to process these related operations and return results.
PetcareAmenities	Petcare Amenities controller is called by Petcare Amenities UI and

	works together with GoogleMaps controller to handle nearby related location request.It Interacts with Petcare Amenities Service and Google Map service to process nearby related operations and return results.
GoogleMaps	Google Map controller works together with MissingPet and PetCare Amenities controller to handle requests for Google Map API.It interacts with GoogleMap Service to handle requests related to mapping and location based data.
Adoption	Adoption controller is called by Adoption UI to handle adoption related requests .It Interacts with AdoptionService to process adoption related operations and return results.
Pet	Pet controller is called by PetInformation UI to manage requests related to personal pet information. It Interacts with Pet Service to process pet related operations and return results.
PetBuddyChatbot	PetBuddy Chatbot Controller is called by Chatbot UI to handle the user's prompt. It serves as the central controller that coordinates the interactions between the Tavily Service and Gemini Service. Upon receiving a user's prompt, it first invokes the Gemini Service to verify if the prompt is related to pet care and does not involve a map location search. If the prompt is found to be unrelated to pet care or involves a map location search, the controller sends a limitation message to Chatbot UI. However, if the verification is successful, the controller then calls the Tavily Service to retrieve the most relevant information from online sources. Finally, it invokes the Gemini Service again to generate a report and then send the report to Chatbot UI.

Service Layer

The Service Layer acts as the intermediary between the Presentation Layer and the business logic. It contains the core functionality and application logic, handling the interactions between different components and implementing use cases.

Service	Description	Functions	Function Description
User	User service is called by User Controller for users to manage their own account.	`+registerUser(UserDetailsDTO userDTO): String`	Creates a new user using the provided input DTO, uploads the associated user information to Firebase Storage, and returns a success message.

		`+loginUser(String idToken): String`	Create the ID token for the client and fetch user details from Firebase Storage and enable the user to login to their account if login credentials are correct.
		`+getUserByUserId(String userId: UserDetailsDTO`	Fetches a specific user by its ID, populates an UserDetailsDTO with its details and profile picture information, returning the DTO.
		`+updateUser(UserDetailsUpdateDTO userDetailsUpdateDTO): String`	Updates an existing user details using the provided update DTO, uploads the new user information to Firebase Storage, and returns a success message.
		`+sendPasswordResetEmail(String email): void`	Send the email with the reset password link to the user provided email when the user wants to reset their password and display a success message.
Article	Article service is called by Article Controller for user. Logic within this controller includes creating,reading,updating and deleting of article related input.	`+getAllArticles(): List<ArticleDetailsDTO>`	Retrieves all articles from the repository, converts them into ArticleDetailsDTO objects, and fetches poster details, returning a list of article DTOs.
		`+getArticleByArticleId(String articleId): ArticleDetailsDTO`	Fetches a specific article by its ID, populates an ArticleDetailsDTO with its details and poster information, returning the DTO or an empty one if the article is not found.
		`+getArticleByPosterId(String posterId): List<ArticleDetailsDTO>`	Retrieves articles posted by a specific user identified by the poster ID, returning a list of populated ArticleDetailsDTO objects.
		`+addArticle(ArticleDetailsInputDTO`	Creates a new article using the provided input DTO,

		articleDetailsInputDTO): String	uploads the associated thumbnail image to Firebase Storage, and returns a success message.
		+deleteArticle(String articleId): String	Deletes an article identified by the article ID by calling the respective method in the repository and returning the result message.
		+editArticle(String articleId, ArticleDetailsUpdateDTO articleDetailsUpdateDTO): String	Updates an existing article using the provided update DTO, uploads the new thumbnail image to Firebase Storage, and returns a success message.
MissingPet	MissingPet service works together with MissingPet Controller for users to report their missing pet.Logic within this controller includes creation of missing pet and report sighting by any users.	+getAllMissingPets(): List<MissingPetDTO>	Retrieves all missing pets from the repository, converts them into MissingPetDTO objects, and fetches owner and pet details, returning a list of missing pet DTOs.
		+getMissingById(String missingId): MissingPetDTO	Retrieves a specific missing pet by its ID, converts it into a MissingPetDTO, fetches its owner and pet details, and returns the DTO.
		+addMissing(MissingPetInputDTO missingDTO): ResponseEntity<String>	Creates a new missing pet record using the provided input DTO, uploads the associated image to Firebase Storage, sets references for the owner and pet, and returns a success message.
		+markFound(MissingPetInputDTO missingDTO, String missingId): String	Marks a missing pet as found based on the provided missing ID, updates the pet's status, and returns a message indicating the result.
Sighting	Sighting Service works together with Sighting controller for user to add	+getAllSightings: List<SightingDTO>	Retrieves all sightings from the repository, converts them into SightingDTO objects, and fetches reporter details,

			returning a list of sighting DTOs.
		`+addSighting(SightingInputDTO sightingInputDTO): String`	Creates a new sighting using the provided input DTO, uploads the associated image to Firebase Storage, updates the corresponding missing pet's sighting list, and returns a success message.
GoogleMaps	GoogleMap service is called by the Petcare Amenities Controller for users to retrieve nearby pet related places.This logic includes sending and retrieving requests for nearby places based on the user locations.	`+searchNearbyPetCare(double latitude, double longitude, int radius, List<String> keywords): List<PlaceDetailsDTO>`	Calls GoogleMaps API to retrieve nearby locations based on the user's current location and a keyword. The results are passed into PlaceDetailsDTO. The function loops through the list of keywords and stores the PlaceDetailsDTOs into a list. The list is then passed in to removeDuplicateLocations to ensure that there are no repeated locations before sending the data back to the frontend.
		`+deleteOldPlaces(long timeThreshold): void`	Calls the deleteOldPlaces repository to remove locations that are more than 24 hours from the firebase database. This function is called in the searchNearbyPetcare service function.
		`+saveOrUpdatePlaceDTO: String`	This service is called by the savePlaceDTO service to save the PetcareAmenityModel to the firebase by calling the petcareAmenityRepository.
		`+savePlaceDTO(PlaceDetailsDTO placeDetailsDTO): String`	Retrieve data from PlaceDetailsDTO, format the data and store them into LocationModel and PetcareAmenityModel. The function then calls the

			location repository to save the details into the firebase database. The returned locationId is passed into the PetcareAmenityModel.
		`+filterLocations(Double minRating, Boolean openNow): List<PlaceDetailsDTO>`	Retrieve sorted list of PetcareAmenityModels by calling the filter function in the PetcareAmenity Repository. The data is then stored in a list of PlaceDetailsDTO before returning to the frontend.
		`searchLocationByKeyword(String keyword):`	Calls the GoogleMaps API search function by passing in the keyword, radius, and the user location. The results are stored in a list of PlaceDetailsDTO.
		`+removeDuplicateLocations(List<PlaceDetailsDTO> placeDTOList): List<PlaceDetailsDTO>`	Checks if the locations in the provided list are duplicated using a set and returns a list of unique places.
		`+fetchPhotoAsBase64((String photoReference: String`	Fetch photo from GoogleMaps API by passing in the photoReference string. The returned result is converted to a byte array then encoded into Base64 String.
Adoption	Adoption Service works together with Adoption Controller for User and Adoption Center. This logic includes viewing available pets for adoption for users and uploading listings for the Adoption Center.	`+getAllAdoption(): List<AdoptionModel>`	Retrieves all adoptions from the repository, converts them into AdoptionModel objects, and fetches adoption details, returning a list of adoption obj..
		`+getAdoptionByAdp(String userId): List<AdoptionModel>`	Retrieves adoption posted by a specific adoption center identified by the user ID, returning a list of populated Adoption Model objects.
		`+getIndividualAdoption`	Retrieves an adoption

		`(String petId): AdoptionModel`	posted by a specific Adoption Center identified by the pet ID, returning a list of populated Adoption Model objects.
		`+addAdoption(AdoptionDetailsDTO adp): String`	Creates a new adoption using the provided input DTO, uploads the associated images to Firebase Storage, and returns a success message.
		`+editAdoption(String petId, AdoptionDetailsDTO adp): String`	Updates an existing adoption using the provided update DTO, uploads the new images to Firebase Storage, and returns a success message.
		`+deleteAdoption(String petId): String`	Deletes an adoption identified by the petId by calling the respective method in the repository and returning the result message.
Pet	Pet Service includes logics to create, retrieve, update and delete personal pet information that belongs to a User.	`getPetsByUserId(String userId): List<PetDetailsDTO>`	Retrieve all the pets that belong to a user by calling the function in pet repository, which returns a list of PetModel. The data of each PetModel is formatted and stored in the PetDetailsDTO. The function returns a list of PetDetailsDTO.
		`addPet(PetDetailsInputDTO petDetailsInputDTO: String`	Create a PetModel by passing in the values stored in the PetDetailsInputDTO sent from the ReactNative frontend. Date object is converted to Timestamp to fit the data type of the Pet collection in Firebase.
		`updatePet(PetDetailsUpdateDTO petDetailsUpdateDTO): String`	Retrieve data from PetDetailsUpdateDTO which is sent from the frontend EditPetForm and store them in a PetModel. The PetModel

			is then passed to the updatePet function in the pet repository.
		`deletePet(String petId): String`	Call deletePet function in the pet repository and pass in the petId of the pet that needs to be removed from the firebase collection. No input formatting is required.
Gemini	Gemini Service works together with Tavily Service and is called by PetBuddy Chatbot Controller. It follows a logic where it first verifies whether the user's prompt is related to pet care and does not involve a map location search, then sends the verification outcome to the PetBuddy Chatbot Controller. Additionally, it follows another logic where it receives the user's prompt along with relevant data from the PetBuddy Chatbot Controller (sourced from the Tavily Service), invokes the Gemini API to generate a report, and forwards the report to the controller. Gemini service's logic also includes error-handling logic triggered if the Gemini API is unavailable.	`classify_input_with_gemini(String prompt): String`	Takes a parameter representing the user input to be classified. The function returns a classification result, which can be either 'location,' indicating a location-based query; 'pet care,' for inquiries related to pet care and well-being; or 'other,' for all other types of queries.
		`get_gemini_response(String msg,String content): String`	The function takes two parameters: msg, which is the user's query that the response should be tailored to, and content, which contains the information that will be referenced in generating the response. The function returns a detailed response generated based on the user query and the provided content. If the response is empty, it will return a message indicating that assistance is only available for pet care-related questions. In case of an error, the function will return an error message detailing the issue encountered during processing.

Tavily	Tavily Service works together with Gemini Service and is called by PetBuddy Chatbot Controller. It follows a logic where it receives user input from the PetBuddy Chatbot Controller, sends it to the Tavily API, applies searching, scraping, filtering, and extracting of the most relevant information from online sources, and forwards it to the PetBuddy Chatbot Controller for processing by the Gemini Service. Tavily Service's logic also includes error-handling logic is triggered if the Tavily API is unavailable.	`search_tavily(String query): String`	Performs a search using the Tavily Search API with the provided user query, which is a parameter of the function, and returns the search results. If no results are found, it returns a message indicating the absence of results. In case of an error, it returns an appropriate error message or informs the user if the Tavily service is unavailable.

Repositories Layer

The Repository Layer is responsible for directly interacting with the database or other data sources. It abstracts the data access logic and provides a clean API for the Service Layer to fetch or update data without worrying about how data is stored or retrieved.

Repository	Description	Functions	Function Description
User	User repository is called by User Service to manage all the data operation under "USER" in the database and returned in User Model.	`+saveUser(UserModel user): DocumentReference`	Save the new user information to Firebase storage with Firebase UID as the document ID and return the DocumentReference for the saved user.
		`+getUserDocReferenceByUserId(String userId): DocumentReference`	Fetches the user information saved in the Firebase storage using userId during login.
		`+getUserByUserId(String userId): UserModel`	Fetches an user information by its ID, returning the

			corresponding UserModel if it exists, or throwing an exception if the user does not exist.
		`+updateUser(UserModel userModel): String`	Updates an existing user's information based on the provided UserModel and user ID, returning a success message upon completion or throwing an exception if the update fails.
Article	Article repository is called by Article Service to manage all the data operation under “Article ” in the database and returned in the Article Model.	`+getAllArticles(): List<ArticleModel>`	Retrieves a list of all articles from the Article collection, returning them as a list of ArticleModel objects.
		`+getArticleByArticleId(String articleId): ArticleModel`	Fetches an article by its ID, returning the corresponding ArticleModel if it exists, or null if the document does not exist.
		`+getArticleByPosterId(String posterId): List<ArticleModel>`	Retrieves a list of articles posted by a specific user, identified by the poster ID, by querying the Article collection
		`+addArticle(ArticleModel articleModel): String`	Adds a new article to the Article collection and returns a success message if the operation is successful or an error message if an exception occurs.
		`+editArticle(ArticleModel articleModel, String articleId): String`	Updates an existing article's fields based on the provided ArticleModel and article ID, returning a success message upon completion or throwing an exception if the update fails.
		`+deleteArticle(String articleId):`	Deletes an article from the Article collection based on

		`String`	the specified article ID and returns a success message if successful or an error message if an exception occurs.
Location	Location repository is called by GoogleMaps Service to manage all the data operation under “Location” in the database and returned in LocationModel.	`saveOrUpdateLocation(LocationModel locationModel): String`	Save data stored in the LocationModel to the firebase collection. It checks if a location with the coordinates exists. If it exists, the function updates the record, else, adds a new location.
		`getLocationDocReferenceByLocationId(String locationId): DocumentReference`	Retrieve location document reference from the firestore Location collection by passing in locationId. This function is used by the SavePlaceDTO function in the GoogleMapsService to get the location reference which will be added to the PetcareAmenityModel.
PetcareAmenity	Petcare Amenities repository is called by PetcareAmenities Service and GoogleMap Service to manage all the data operation under “Petcare Amenities” in the database and returned in Petcare Amenities Model.	`+save(PetcareAmenityModel petcareAmenityModel): String`	Calls the firestore PetcareAmenity collection to add a new petcare amenity record. The PetcareAmenityModel that has the same data type and format is passed into the Firestore function.
		`+saveOrUpdatePlaceDetails(PetcareAmenityModel petcareAmenityModel): String`	Check if the Location collection contains the location using the amenityId. If it exists, update the location details, else, add the new place to the firestore.
		`+deleteOldPlaces(long timeThreshold: void`	Query firestore to find places older than the time threshold given. The places found are then removed from the collection to ensure the freshness of the data.

		`+filterLocations(Double minRating, Boolean openNow): List<PetcareAmenityModel>`	Query firestore based on a few filtering conditions, including both rating and openNow, rating only and openNow only. The returned results are stored in a list of PetcareAmenityModel before returning it to the service layer.
Pet	PetInformation repository is called by PetInformation Service to manage all the data operation under “PetInformation ” in the database and returned in the Pet Model.	`+getPetsByUserId(String userId): List<PetModel>`	The function first gets the document reference of the owner using the userId given. The returned owner reference is then used to match the owner field in the Pet collection to find the pets that belong to this user.
		`+addPet(PetModel petModel): String`	Write data into firestore Pet collection and return a success message if the data has been added successfully.
		`+updatePet(PetModel petModel): String`	Update the Pet collection with the pet details of a given pet which is stored in the PetModel.
		`+deletePet(String petId): String`	Remove the pet record from the Pet collection by petId.
Adoption	Adoption repository is called by Adoption Service to manage all the data operations under “Adoption ” in the database and returned in the Adoption Model.	`+getAllAdoption(): List<AdoptionModel>`	Retrieves a list of all adoption pets from the Adoption collection, returning them as a list of AdoptionModel objects.
		`+addAdoption(Adoption adoptionModel): String`	Adds a new adoption to the Adoption collection and returns a success message if the operation is successful or an error message if an exception occurs.

		`+editAdoption(AdoptionModel adoptionModel, String petId): String`	Updates an existing article's fields based on the provided AdoptionModel and article ID, returning a success message upon completion or throwing an exception if the update fails.
		`+deleteAdoption(String petId): String`	Deletes an adoption from the Adoption collection based on the specified adoption ID and returns a success message if successful or an error message if an exception occurs.
		`+getAdoptionByAdp(String userId) : List<AdoptionModel>`	Retrieves a list of adoption pet posted by a specific adoption center, identified by the user ID, by querying the User collection
		`+getIndividualAdoption(String petId): AdoptionModel`	Fetches an adoption by its ID, returning the corresponding Adoption Model if it exists, or null if the document does not exist.
MissingPet	MissingPet repository is called by MissingPet Service to manage all the data operation under “MissingPet” in the database and returned in the MissingPet Model.	`+existsActiveMissingPet(String petId): boolean`	Checks if there is an active missing pet associated with the specified pet ID by querying the MissingPet collection. Returns true if an active document is found, otherwise returns false.
		`+getMissingDocReferenceByMissingId(String missingId): DocumentReference`	Retrieves a reference to a specific missing pet document based on its missing ID.
		`+updateMissingPetSightingList(String missingId, DocumentReference sightingRef):String`	Updates the sightingList array of a missing pet document by adding a new sighting reference using Firestore's arrayUnion

			method. Returns a success message upon completion.
		`+getAllMissingPets(): List<MissingPetModel>`	Retrieves a list of all missing pets from the MissingPet collection, returning them as a list of MissingPetModel objects.
		`+getMissingById(String missingId): MissingPetModel`	Fetches and returns the MissingPetModel object corresponding to the specified missing ID. If the document does not exist, it logs a message and returns null.
		`+addMissingPet(MissingPetModel missingPetModel) : DocumentReference`	Adds a new missing pet to the MissingPet collection and returns the document ID if successful or an error message if an exception occurs.
		`+markFound(MissingPetModel missingModel, String missingId): String`	Marks a missing pet as found by updating its document in the MissingPet collection with the provided model. Returns the document ID along with a confirmation message if successful or an error message if an exception occurs.
Sighting	Sighting repository is called by Sighting Service to manage all the data operation under “MissingPetSighting” in the database and returned in the Sighting Model.	`+getSightingDocReferenceBySightingId(String sightingId): DocumentReference`	Retrieves a reference to a specific document in Firestore based on sightingId
		`+getAllSightings(): List<SightingModel>`	Retrieves all sighting documents from the Firestore MissingPetSighting collection, converts them into SightingModel objects, and returns them as a list.
		`+addSighting(SightingModel sightingModel):`	Adds a new sighting to the Firestore MissingPetSighting

		`String`	collection, returning the document ID

Model Layer

The Model Layer defines the structure of the data used across the application. It represents the objects that will be called by the Controller ,Service and Repositories layer which defines the rules and relationships for this data.The entity classes are stored in the database of the persistent layer. This layer consists of:

Model	Data Type	Field
User	String String String Integer String String String	address email password phoneNumber profilePicUrl status userName
Article	String String String User Date String	articleBody articleCategory articleTitle poster publishedTime thumbnailImage
Location	String double double String	locationId locationLatitude locationLongitude locationAddress
PetcareAmenity	String String boolean List<PlaceOpeningHoursDTO> String String double String Date	amenityID amenityName openNow openingHours contactNumber websiteURL rating Location timestamp
Pet	String String String float String	petName sex breed weight coatColor

	Timestamp Long Long User String	dateOfBirth markings medicCondition Owner petImageUrl
Adoption	 String String String String String String List<MultipartFile>	**Inherit:** Attributes from Pet **Additional:** adptEmail adptName adptNumber age description Type imagesList
MissingPet	String Boolean Timestamp Timestamp String String GeoPoint Pet User List <Sighting>	id active lastSeenDateTime publishedTime lastSeenImage lastSeenDescription lastSeenLocation missingPet owner sightingList;
Sighting	String String String GeoPoint Timestamp User Pet	Id sightingDescription sightingImage sightingLocation sightingDateTime reporterContact missingPet

Tech Stack

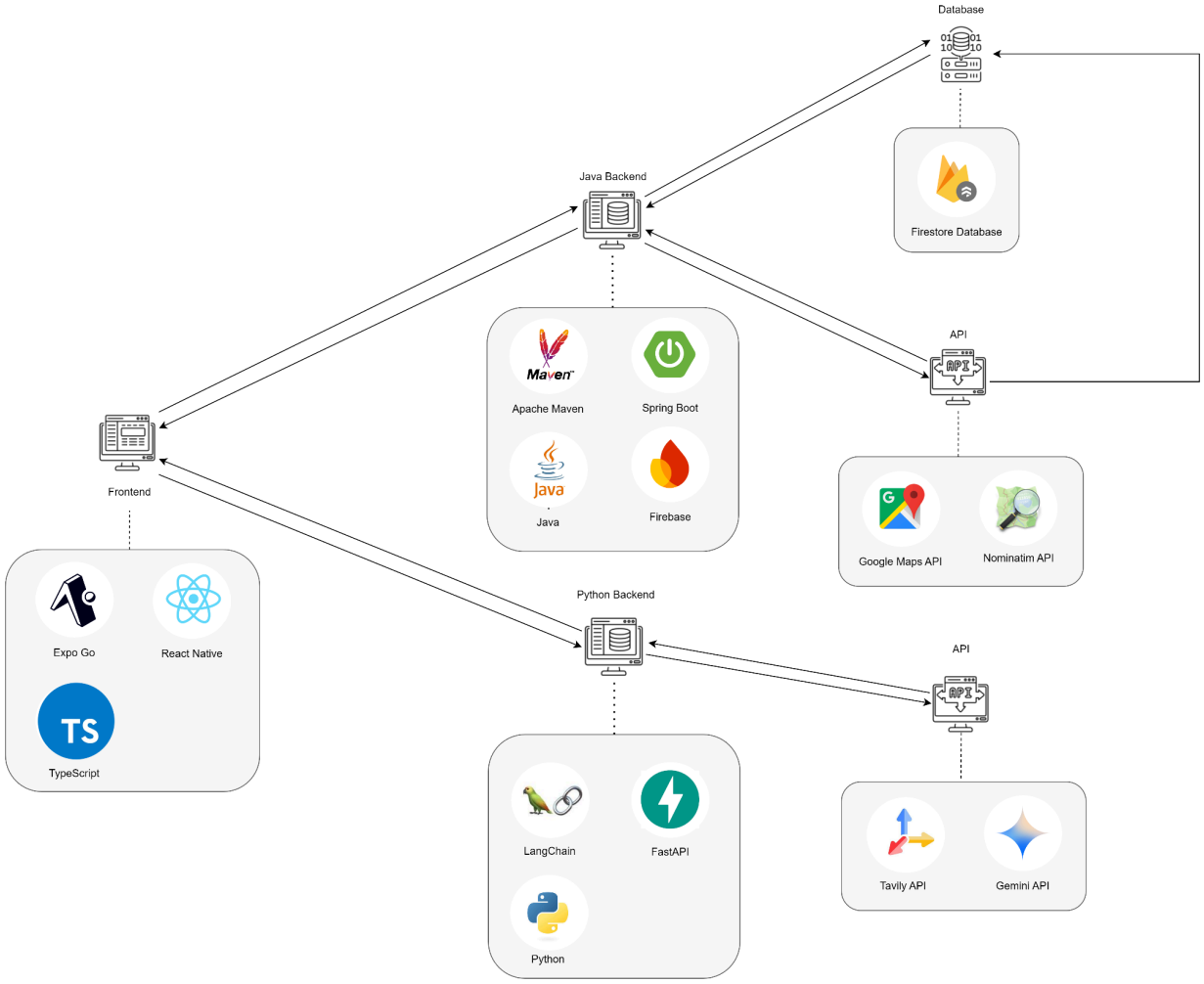


Figure 3. Tech Stack Diagram

Expo Framework

Expo is a framework and platform built on top of React Native. It simplifies and enhances the process of building and deploying React Native apps by offering additional tools, libraries, and services that eliminate much of the complexity that comes with native mobile app development. Expo is adopted to view the deployment of our application across both Android and iOS platforms, allowing us to easily develop and test the application.

Figure 4. Expo Go Running Screen

React Native

The frontend of our application is developed using JavaScript and React Native, providing a dynamic and responsive user interface across both iOS and Android. The codebase is written in TypeScript which enhances development with type safety and better tooling support.

React Native enables us to create user interfaces that render as native components, ensuring a smooth and highly performant experience on both platforms. The JavaScript layer handles the application's logic and UI interactions, while the native side efficiently manages rendering and performance-intensive tasks.

Maven

We created a Maven project for the Java backend, which is responsible for managing the application's dependencies and project structure. Maven is a build automation tool used in Java projects that simplifies the management of external libraries and dependencies. All the required packages are stored in the pom.xml file. Maven will automatically download these dependencies before running the project.

Spring Framework

We integrated the Spring Framework into our Java backend to simplify the development of enterprise-level applications. By leveraging Spring’s powerful ecosystem, including Spring Boot, we can easily configure and manage the application’s infrastructure, making development faster and more efficient.

Firebase Integration

In our mobile application, we integrated Firebase to provide backend services such as authentication, firestore database, and cloud storage. The Firebase SDK connects the Expo React Native App running on the client device to Firebase's powerful cloud services. The integration enables seamless data management with **Cloud Firestore** for real-time data syncing and querying, **Cloud Storage** for handling file uploads and downloads, and **Firebase Authentication** for user management.

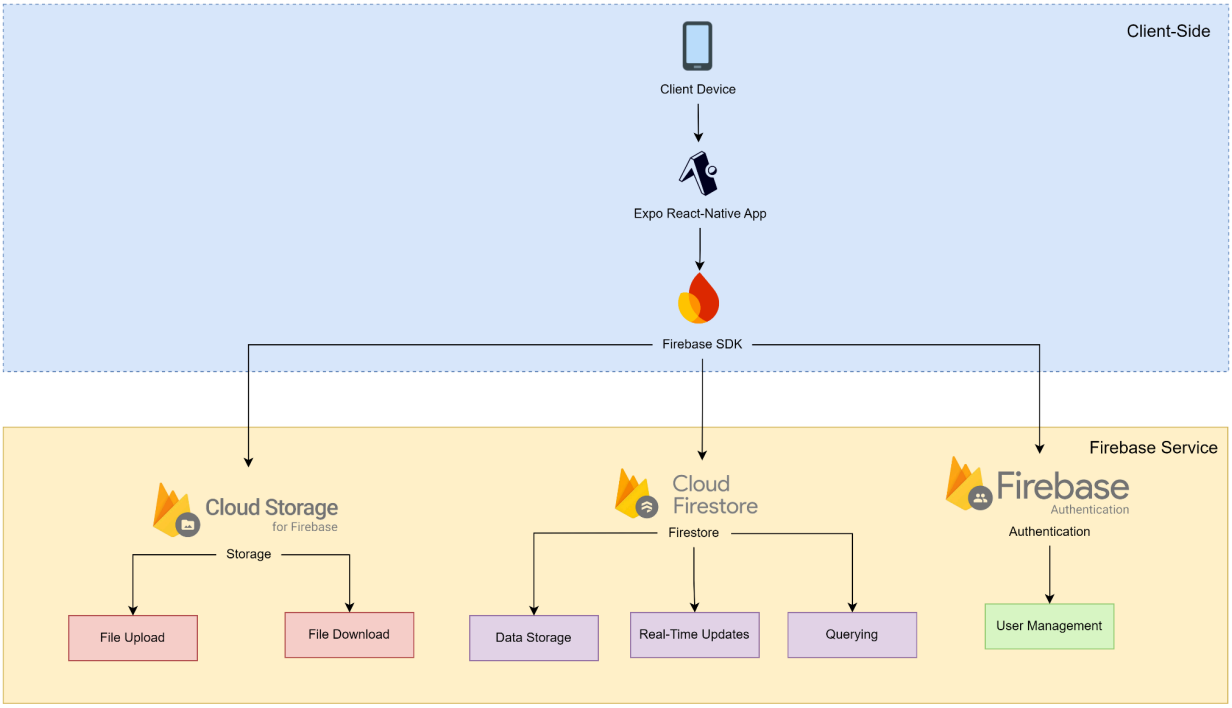


Figure 5. Firebase Architecture Diagram

Application Skeleton

Frontend

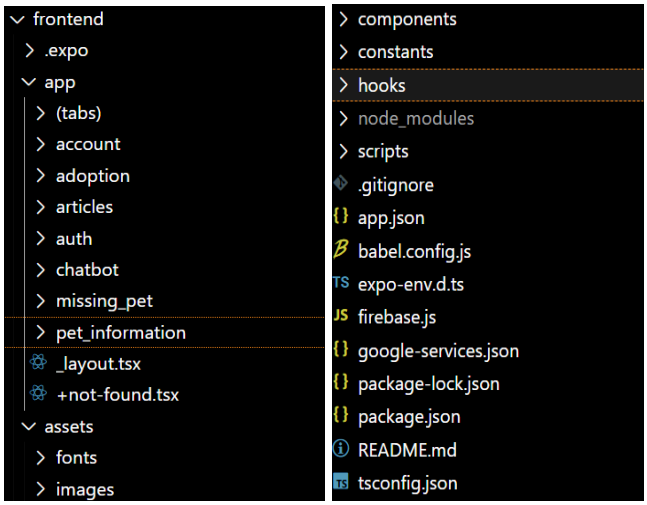
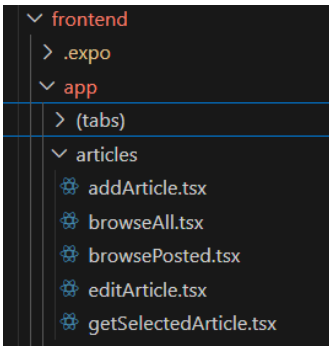
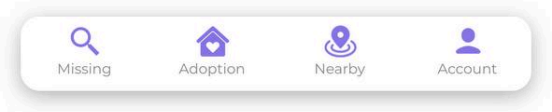
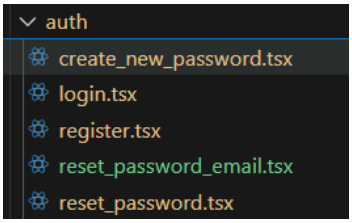
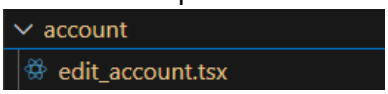
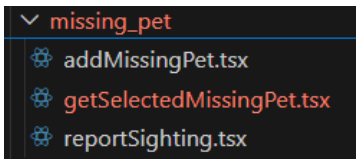
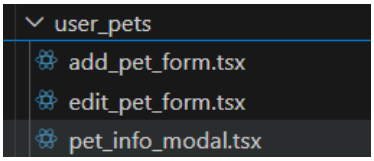
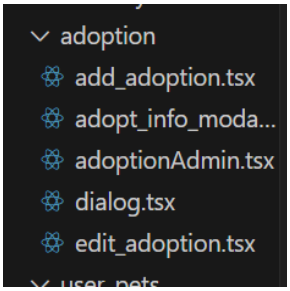
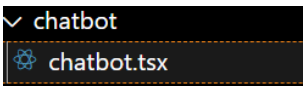
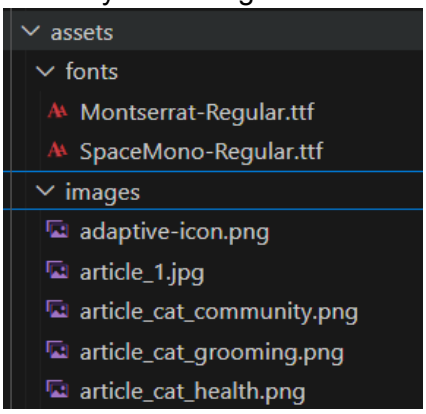


Figure 6. PetCare Frontend Folder Structure

Directory/ File	Description
.expo/	Expo-related configurations and cached data for running the app locally.
app/	Main application directory containing various app-related screens and components.
app/_layout.tsx	Main layout component for structuring the app's screens.
app/+not-found.tsx	Handles display when a page or resource is not found (404 error).
app/(tabs)/	Manages bottom tab bar navigation within the app.  
app/articles/	Contains components/screens related to PetCare articles.

	
app/auth/	Contains authentication-related screens/components (e.g. login, signup and reset password). 
app/account	Contains components/screens related to the user account. 
app/missing_pet/	Contains components/screens related to PetCare articles. 
app/pets_information/	Contains components/screens related to the user's pets. 
app/adoption	Contains components/screens related to adoption 
app/chatbot	Contains components and screens related to the chatbot functionality.

	
assets/	Directory for storing static assets like images and fonts. 
components/	Contains reusable components like markers, modals, and themed views.
constants/	Stores constant values to be reused in the application.
hooks/	Contains custom React hooks for managing reusable logic in the app.
google-service.json	Firebase configuration for Android, linking the app to Firebase.
firebase.js	Configuration for Firebase integration.

Backend

Java Backend

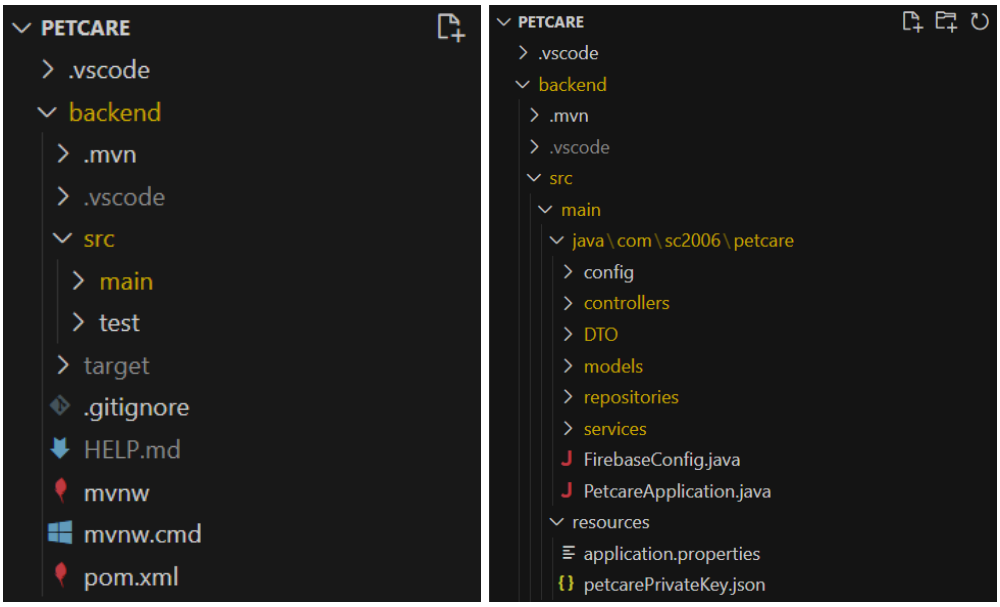


Figure 7. PetCare Backend Folder Structure

The main folders of the backend application server consists of three folders: the target folder, the Java folder and the resources folder. The target folder of the Java backend contains all the output files that will be generated when the project is built. A notable file that can be found in the target folder will be the jar file which will be used to execute and run the backend application.

The "java" folder contains the backend code for the PetCare mobile application, responsible for handling the application's core logic and processing functions. Key components within this folder include the controller, service, repositories, and model packages, each playing a crucial role in the application's architecture which will be further elaborated in the section below.

Figure 8. DTOs

We have utilized Data Transfer Objects (DTOs) to format the data retrieved from the database or received from the frontend into the desired structure before further processing. DTOs ensure that the data passed to the processing functions or sent to the user interface is in the correct data type and format. This approach promotes clean data handling by isolating the application logic from the specific details of how the data is stored or represented in the UI, reducing the risk of errors and improving code maintainability. By using DTOs, we ensure that data flowing between layers of the application is consistent and well-structured.

Figure 9. Controllers

The controller package contains the endpoints that enable the PetCare frontend to interact with the backend via a RESTful API. These endpoints are defined using Spring's @RequestMapping annotation, which specifies the URL paths that trigger specific functions. When a request is made to an endpoint, parameters are passed using annotations like @PathVariable and @PathParam, which retrieves data from the request. This data is then stored in Data Transfer Objects (DTO), ensuring that the data is organized and ready for further processing. The

controllers then invoke their respective service classes, which handle the core processing functions of the application.

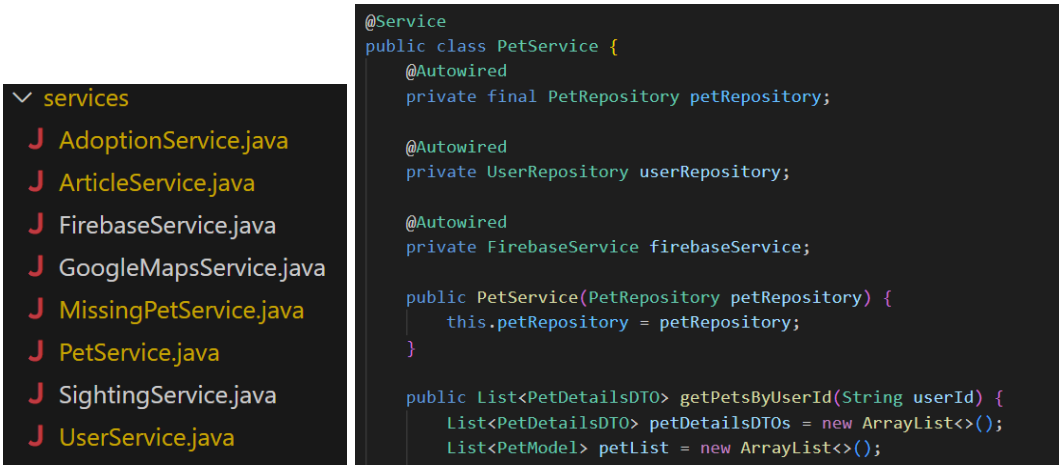


Figure 10. Services

The service package contains classes that implement the application's business logic. These classes perform tasks such as interacting with external apis (eg. calling the Google Maps API) and processing the data received from the controllers. The services act as the middle layer between the controllers and the repositories, ensuring that the application's logic is applied before interacting with the database or other external services.

The service class can be implemented by first annotating the class as a service with the “@Service” annotation. Next the developers just need to define and annotate the model class and repositories interfaces with the “@Autowired” annotation which will auto inject the model and repositories when the application is built and running.

Figure 11. Repositories

The classes in the repositories folder are responsible for handling the interaction between the application and the Firestore database. These repository classes provide functionality to read and write data to Firestore, allowing the application to query and persist data effectively.

For example, the PetRepository class manages pet-related data by interacting with the Pet collection stored in Firebase using methods like getAllPets() to retrieve all pet documents and getPetsByUserId() to fetch pets that belong to a specific user. The returned data is then mapped to Java objects (PetModel), enabling smooth integration between the Firestore database and the application's data model.

By leveraging repository classes, the application's service layer can focus on business logic, leaving the repository classes to manage database interactions. This separation of concerns improves the maintainability and scalability of the code.

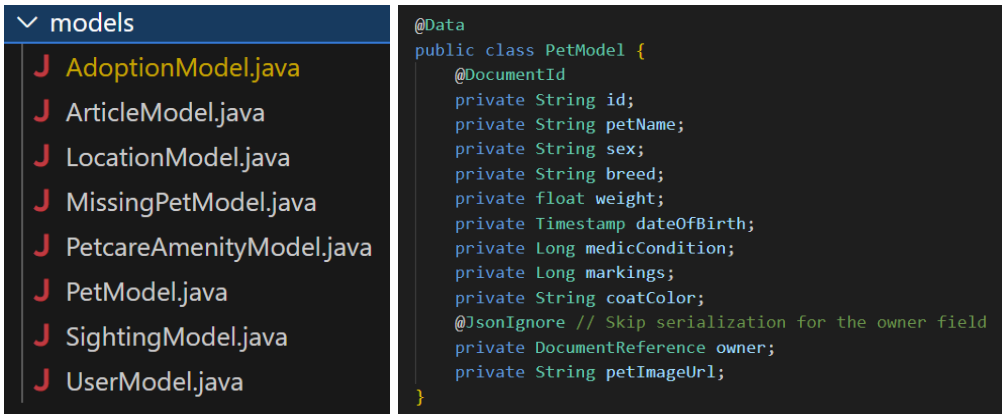


Figure 12. Models

The model folder stores the data collections representing the application's core entities. We use the @Data annotation from Lombok to automatically generate essential methods such as getters and setters. This eliminates the need for boilerplate code, improving readability and maintainability. When a Pet document is retrieved from Firestore, the field "id" (annotated with @DocumentId) will be automatically populated with the Firestore document's ID.

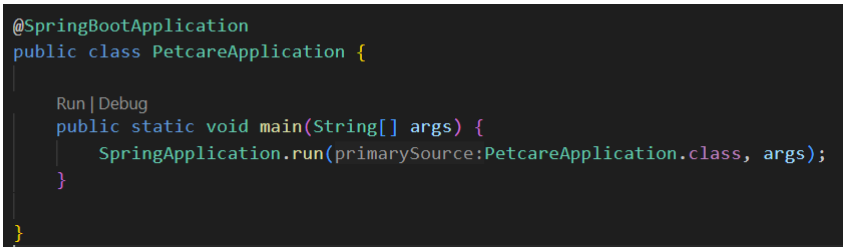


Figure 13. PetCare Java Backend Application

The PetcareApplication.java file is the main entry point for the PetCare application, which is built using the Spring Boot framework. The file is annotated with @SpringBootApplication, indicating

that it is a Spring Boot application and will auto-configure components when the application starts.

Python Backend

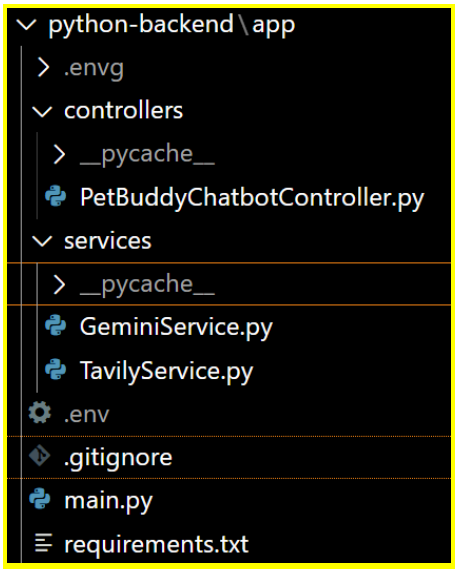


Figure 14. Python Backend Folder Structure

The Python backend is responsible for managing user input in the PetBuddy chatbot, a key feature of the PetCare mobile application. The python-backend folder contains the core backend logic specifically designed for the PetBuddy chatbot, handling essential processing and decision-making functions. Within this folder, the app directory serves as a Python package that organizes the main components of the backend. This includes the controllers and services directories, which manage different aspects of the chatbot's functionality. Additionally, the folder contains the main.py file, which is the entry point of the backend application, responsible for setting up the FastAPI server and routing requests. In the following section, we will provide a detailed explanation of main.py, as well as an overview of the services and controllers directories.

Figure 15. Icon of FastAPI Web Framework

To create the API endpoints for the PetBuddy chatbot, we adopted FastAPI due to its superior speed and performance, which is especially important for chatbot functionality. FastAPI's asynchronous capabilities allow the system to handle multiple concurrent requests efficiently,

ensuring real-time responses to user queries. In the PetBuddy system, this is crucial as it integrates external APIs like Tavily and Google Gemini, requiring fast processing of pet care-related queries. Additionally, FastAPI simplifies the inclusion of routes, such as those from the PetBuddyChatbotController, while enabling Cross-Origin Resource Sharing (CORS) to ensure smooth communication between the frontend and backend. The framework's speed and ease of use make it an ideal choice for the scalable and responsive chatbot backend that PetBuddy chatbot requires.

app > main.py > ...

```python
from fastapi import FastAPI,APIRouter
from fastapi.middleware.cors import CORSMiddleware
from controllers.PetBuddyChatbotController import router as chatbot_router
import uvicorn
app = FastAPI()
router = APIRouter()
origins = [
    "http://localhost:3000",
    "localhost:3000"
]

app.add_middleware(
    CORSMiddleware,
    allow_origins=origins,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# Include the routers from different controllers
app.include_router(chatbot_router)
app.include_router(router)
# Define root route
@app.get("/")
def read_root():
    return {"message": "Hello, world!"}

if __name__ == "__main__":
    uvicorn.run(app, host="0.0.0.0", port=8000)

```

Figure 16. main.py File

The main.py file is to set up the FastAPI application, enable Cross-Origin Resource Sharing (CORS) to allow requests from the frontend, and include routes from the PetBuddyChatbot Controller. The CORS middleware is configured to accept requests from the specified origins and allows all HTTP methods and headers. Additionally, the script defines a root route that returns a basic message for testing purposes. Finally, it starts the application using Uvicorn, making it the main entry point for initializing and running the backend server, while integrating chatbot functionality and ensuring proper communication between the frontend and backend.

Figure 17. Icon of LangChain Framework

Langchain framework is designed to facilitate the development of applications that use large language models (GPT-4, Google Gemini) in combination with other tools or data sources. In

our case, we adopted the LangChain framework to orchestrate the flow between Tavily and Gemini, making it easier to manage the interaction between the data retrieval and language processing components.

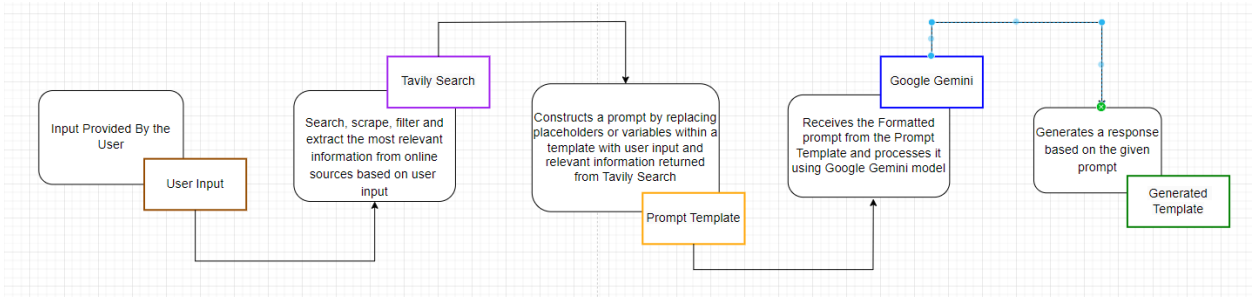


Figure 18. The LangChain-Driven AI Workflow: Tavily Search and Google Gemini Integration

app > controllers > PetBuddyChatbotController.py > ...

```python
from fastapi import APIRouter, HTTPException
from fastapi.responses import JSONResponse
from services.GeminiService import GeminiService
from services.TavilyService import TavilyService
from pydantic import BaseModel
import logging

router = APIRouter()

class Message(BaseModel):
    msg: str

gemini_service = GeminiService()
tavily_service = TavilyService()

@router.post("/get_response")
async def get_response(message: Message):
    try:
        # Call the Gemini service for classification
        category = gemini_service.classify_input_with_gemini(message.msg)

        if category == "location":
            return JSONResponse(content={"response": "I'm sorry, I cannot assist in searching map locations. Please ask me something about pets, their care, or re
        elif category != "pet care":
            return JSONResponse(content={"response": "I'm sorry, I can only assist with pet care-related questions. Please ask me something about pets, their care

        # Call Gemini API to get the pet care response
        content = tavily_service.search_tavily(message.msg)
        if (content == "I'm sorry, but the Tavily service is currently unavailable. Please try again later."):
            return JSONResponse(content={"response": content})
        response = gemini_service.get_gemini_response(message.msg, content)
        return JSONResponse(content={"response": response})
    except Exception as e:
        logging.error(f"Error while processing the chat message: {e}")
        raise HTTPException(status_code=500, detail="Error processing request.")
```

Figure 19. PetBuddyChatbot Controller

PetBuddy Chatbot Controller handles chat messages for the PetBuddy chatbot, integrating both the Gemini Service and Tavily Service to process and respond to user queries. The controller handles exceptions by providing fallback responses when services are unavailable and ensures proper error logging for troubleshooting. This setup ensures an organized flow for managing user input and generating appropriate responses.

app > services > TavilyService.py > ...

```python
import logging
from langchain_community.utilities.tavily_search import TavilySearchAPIWrapper
from langchain_community.tools.tavily_search import TavilySearchResults
from dotenv import load_dotenv
import os
load_dotenv(dotenv_path='.env')
# API keys setup
TAVILY_API_KEY = os.getenv('TAVILY_API_KEY')
class TavilyService:
    def __init__(self):
        self.search_api = TavilySearchAPIWrapper(tavily_api_key=TAVILY_API_KEY)
        self.description =  ("A search engine optimized for comprehensive, accurate, "
                             "and trusted results. Useful for answering questions about "
                             "current events or recent information.")
        self.tavily_tool = TavilySearchResults(api_wrapper=self.search_api, description=self.description, search_depth="advanced")


    def search_tavily(self, query: str) -> str:
        try:
            result = self.tavily_tool.invoke({"query": query})
            if not result:
                return "No results returned from Tavily Search."
            return result
        except Exception as e:
            logging.error(f"Error searching Tavily: {e}")
            # Check for API unavailability based on the exception message
            if "unavailable" in str(e).lower() or "failed to connect" in str(e).lower():
                return "I'm sorry, but the Tavily service is currently unavailable. Please try again later."
            return "Error occurred. Please try again later."

```

Figure 20. TavilyService.py

TavilyService.py in the services folder of the python backend application contains a class that implements the application's business logic. The TavilyService class is designed to interact with the Tavily Search API, providing a robust search capability optimized for comprehensive and accurate results.

The class initializes by creating an instance of TavilySearchAPIWrapper using the API key obtained from environment variables. It sets up a description highlighting Tavily Search's suitability for answering questions related to current events and recent information.
The TavilySearchResults object is configured with an "advanced" search depth to ensure detailed search results.

The search_tavily method in the class takes a user's query as input and performs a search using Tavily Search. If no results are found from Tavily API, it returns a message indicating that no results were returned.

app > services > GeminiService.py > GeminiService > get_gemini_response

```python
import logging
from langchain_google_genai import ChatGoogleGenerativeAI
from langchain_community.adapters.openai import convert_openai_messages
from dotenv import load_dotenv
import os
load_dotenv(dotenv_path='.env')
GOOGLE_API_KEY = os.getenv('GOOGLE_API_KEY')
class GeminiService:
    def __init__(self):
        self.llm = ChatGoogleGenerativeAI(temperature=0, model="gemini-pro", convert_system_message_to_human=True)

    def classify_input_with_gemini(self, prompt: str) -> str:
        try:
            location_check_prompt = [
            {"role": "system", "content": "You are an intelligent assistant specialized in identifying location-based queries."},
            {"role": "user", "content": (
                f'Based on the following input, determine if it involves a map location search or a directions query: "{prompt}".\n'
                f'Respond strictly with one of the following:\n'
                f'1. "location" - if the input mentions places, directions, or phrases like "where is," "nearest," "location of," or "how to get to."\n'
                f'2. "none" - if the input does not relate to any location-based search or directions query.'
            )}
            ]


            lc_messages = convert_openai_messages(location_check_prompt)
            location_response = self.llm.invoke(lc_messages).content.strip().lower()
            print(location_response)

            # If the response indicates that it is a location search, return immediately
            if "location" in location_response or "map location search" in location_response:
                return "location"

            # Step 2: If not a location search, check if the input involves pet care or other topics
            classification_prompt = [
                {"role": "system", "content": "You are an intelligent assistant specialized in classifying user queries."},
                {"role": "user", "content": (
                    f'Classify the following input into one of these categories: "Pet Care", or "Other".\n\n'
```

Figure 21. GeminiService.py

GeminiService.py in the services folder of the python backend application contains a class that implements the application's business logic. The class initializes an instance of ChatGoogleGenerativeAI from the LangChain Google GenAI module, with the model set to "gemini-pro". This instance is used to process and generate responses.

The class contains two methods: classify_input_with_gemini and get_gemini_response. The classify_input_with_gemini method is responsible for determining whether a user's query relates to a "location"-based search or a "pet care" inquiry. On the other hand, the get_gemini_response method constructs a prompt by integrating user input and relevant information returned from Tavily Search into a predefined template. This prompt is then used to generate responses for pet care-related queries using the Google Gemini model. Both methods leverage the capabilities of Google Gemini to deliver accurate and context-specific responses.