Iowa State University



Final Report for VetGo

COM S 402C Spring 2024

Authored by Group 10: Andy Bruder, Elijah Brungardt, David Marvin, Josh Podlich, & Leo Stoyanov

Supervised by Dr. Mitra

Department of Computer Science

May 9, 2024

Chapter on Requirements

This document outlines the system requirements for a cross-platform mobile application designed for two distinct user types: veterinary doctors (vets) and pet owners.

General Requirements

- The application must support iOS, Android, and Web platforms.
- Both user types must undergo a signup and registration process.
- Integration with external systems such as Gmail and Outlook for calendar syncing is required.
- Utilization of AI and machine learning for enhancing user interactions and decision-making processes.

Requirements for Pet Owners

- Pet owners must be able to request appointments with vets in their area.
- The app should provide guidelines on appointment scheduling.
- Screening questions must be presented to determine the urgency of the pet's condition.
- In cases of health emergencies, the app should recommend nearby pet hospitals with emergency services.
- Integration of a speech-to-text engine to facilitate input via speech.
- The app should use the Pets WebMD database to suggest possible conditions/symptoms based on the pet owner's input.
- Billing information must be collected during the registration process.

Requirements for Veterinary Doctors

- Vets should receive notifications of appointment requests within their service area on their iPads.
- The app should allow vets to accept or reject appointments.
- The app must inform vets about the travel time to the appointment location.
- Calendar integration to check the vet's availability.
- Upon appointment confirmation, vets should be able to access the pet's medical records released by the owner.
- During appointments, the app should assist vets with a menu-driven approach to conducting wellness examinations.
- AI features should aid in diagnosing potential conditions based on observed symptoms during the exam.
- Vets must provide credentials verifying their qualification to practice in the area during registration.

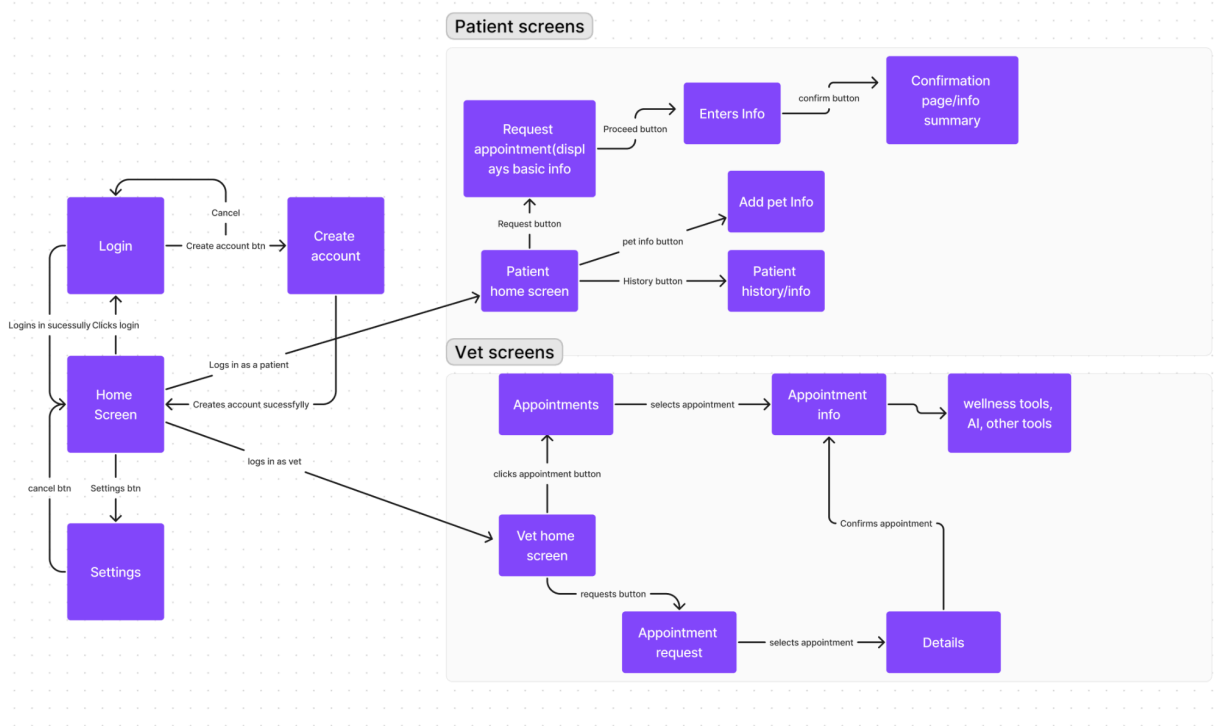
DESIGN OF THE PROJECT

Frontend

The original plan for the UI and design of frontend was a straightforward application. With a design based on the Uber application

- From the landing page, you can access every screen with 1-2 clicks
- Streamlined user experience to allow for a simple navigate-and-use application
- The style of the application keeps the user demographic in mind
    - Easy to read
    - “Friendly colors”
    - Rounded edges
- Input boxes are not overwhelming
- Very descriptive application

Initial Screen flow chart

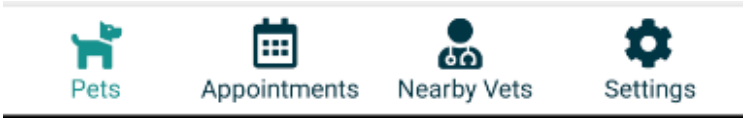


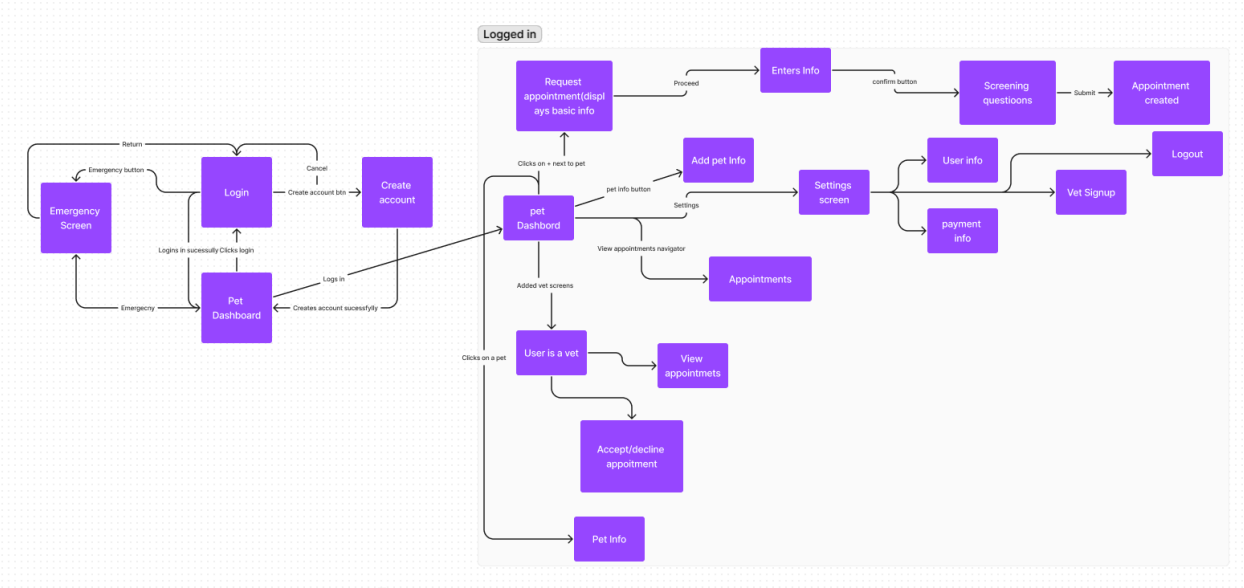
Initial Screen Sketches



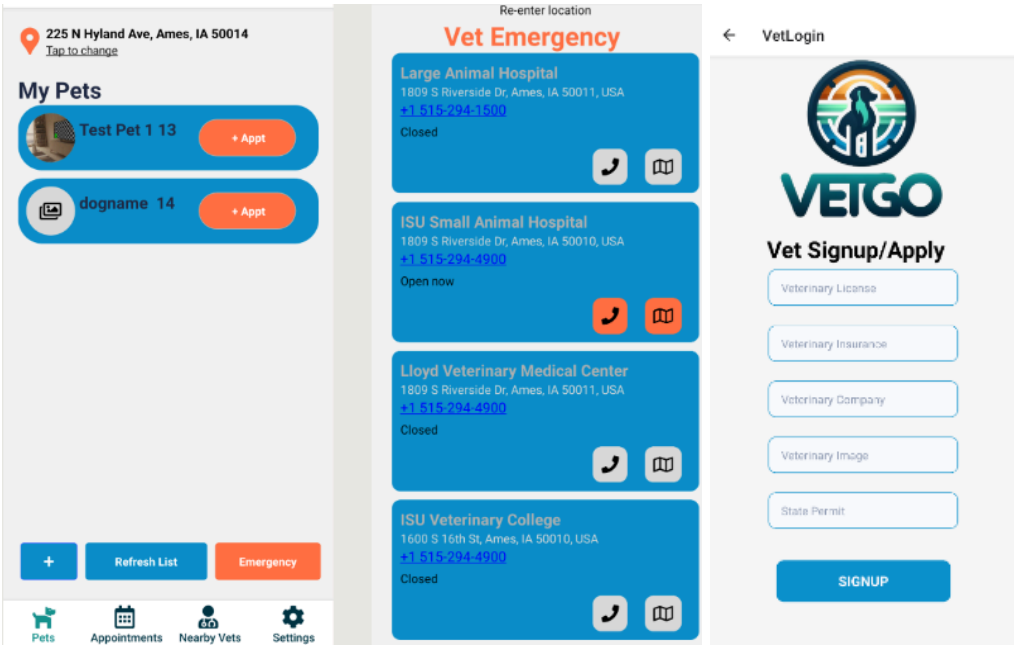
Finalized UI

The application changed as we continued to develop it and learned more about the requirements. We implemented more screens for user changes/information and removed some of the redundant screens. One of the main changes from the original design was implementing a navigation bar at the bottom of the screen. This allows us to easily navigate between screens and not have the button navigation as initially planned.



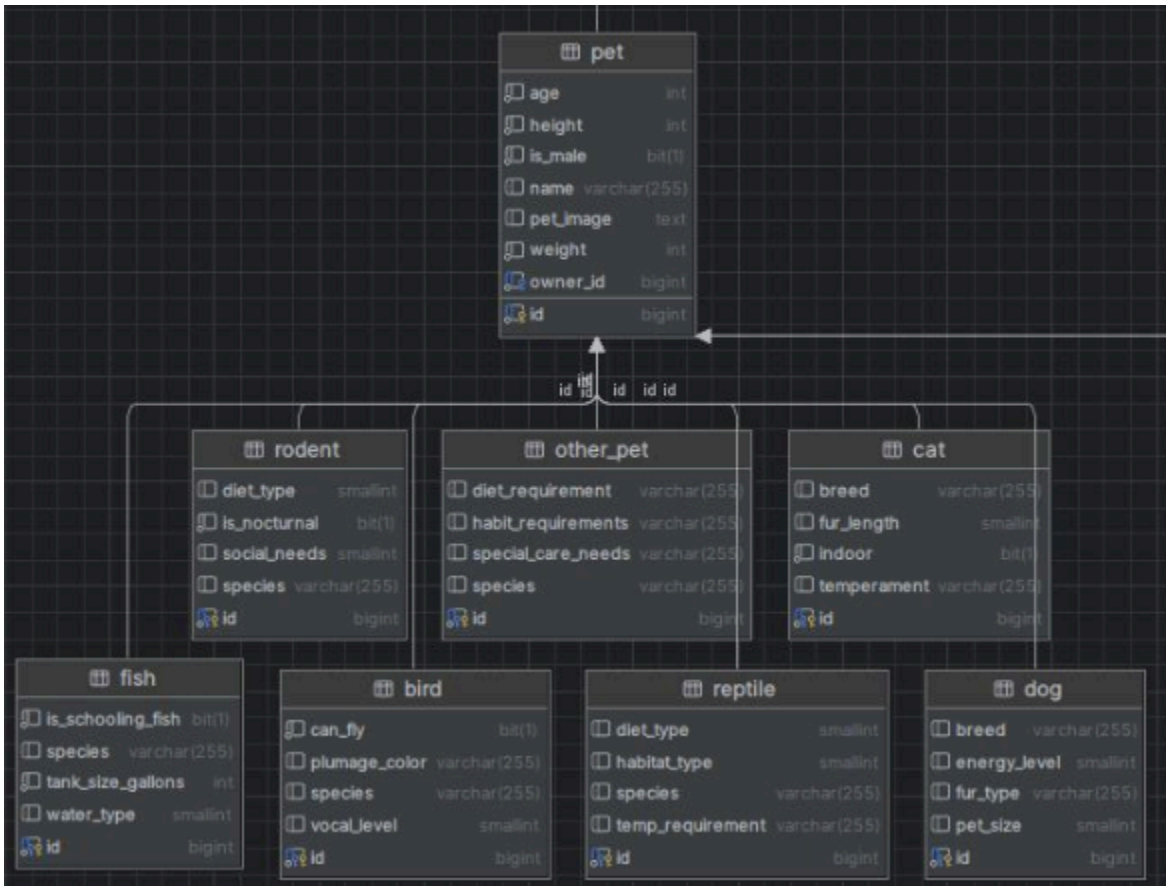


One of the largest factors that caused changes to the design was our creation of a logo. This logo gave the project a cohesive design and style, changing the screens' look to match. Although the Screens remain relatively similar to the initial sketches, they have been adjusted in the final project to contain more information and better match the project as a whole.



Backend

Since we took over a project that already existed we had some major changes to the backend. The total reconstruction allowed for much more information with added stability and scalability. There are many endpoints giving access to alot of different information and giving the ability for future development to continue if necessary.



Formatting of the backend also allowed us to break up many different tables with primary and secondary keys instead of having massive tables with all of the information stored. The backend also has error detection throughout the project so if something were to break, specific errors would tell you the exact issue. We also have Swagger UI available to view all of the backend documentation.

vet-controller

- PUT /vet/validate/{vid}
- PUT /vet/update/{vid}
- POST /vet/register
- GET /vet/nearby-vets/{uid}
- GET /vet/id/{vid}
- GET /vet/id/user/{uid}
- GET /vet/getUser/id/{vid}
- GET /vet/all
- DELETE /vet/delete/{vid}
- DELETE /vet/delete/all

user-controller

review-controller

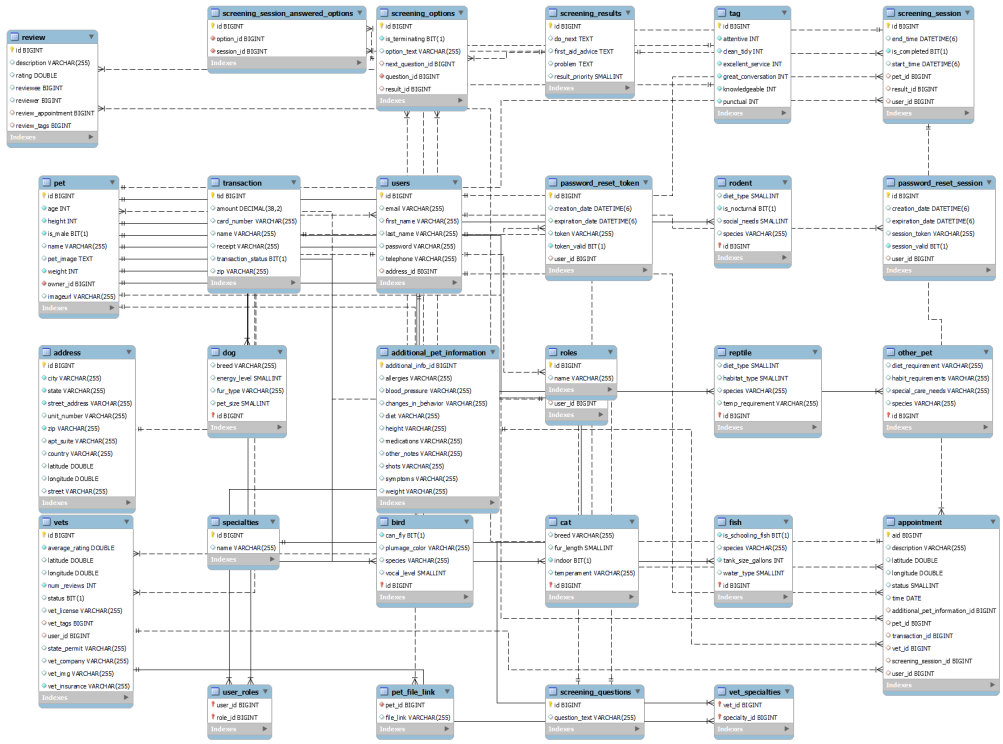
pet-controller

owner-controller

appointment-controller

transaction-controller

screening-controller



Overall design

The design of this project had a few key focuses, scalability, usability, and continued development. Since we were handed off this project, we implemented many features and logs to ensure that if another team takes over this project in the future, they can begin work with little adjustment period. We also ensured that the navigation and UI is able to be expanded upon if more features are to be added. As we conclude this project, as a team, we believe we were able to create a robust project capable of continued development or remaining as it is while maintaining our original design ideas.

Software Requirements Document for [VetGo]

TEAM: 10

AUTHORS: Andrew Bruder, David Marvin, Elijah Brungardt, Josh Podlich, & Leo Stoyanov

Version	Date	Author	Change
0.1	2/15/2024	Everyone	Initial Draft
1	5/8/2024	Everyone	Final Paper

Table of Contents

1 Introduction 3

1.1 Purpose 3

1.2 Scope 3

1.3 Definitions, acronyms, abbreviations 3

1.4 References 3

1.5 Overview 3

2 Overall Description 4

2.1 Product Perspective 4

2.2 Product functions 6

2.3 User characteristics 19

2.4 Constraints 19

2.5 Assumptions and Dependencies 19

3 Specific Requirements 20

3.1 External Interface Requirements 20

3.2 FEATURES 20

3.3 Performance requirements 20

3.4 Design Constraints 20

3.5 Software System Attributes 20

3.6 Other Requirements 21

Appendix 22

1 Introduction

1.1 Purpose

The purpose of this document is to establish how the application should interact with the end user, and establish all application requirements functional, and non functional. Once finalized, this document will state what must be accomplished for the application to be considered finished.

1.2 Scope

This SRS covers a number of potential use cases that users may encounter, as well as an overview of the project and its intended uses. It also includes information on the project's UI sketches, but the primary purpose is to give detailed descriptions of anticipated use cases.

1.3 Definitions, acronyms, abbreviations

Term	Description
Veterinary Doctor	Professional who deals with treating animals
Vet	Veterinary Doctor
Patient	The patient refers to the pet owner, not the pet

1.4 References

[OMIT]

1.5 Overview

[OMIT]

Overall Description

Our project is the creation/advancement of the VetGo project. It is an application that allows a user/pet owner to quickly get access to a veterinarian, who will be able to travel to the pets location and treat/diagnose the pet. The application focuses on quickly setting up an appointment, as well as allowing the veterinarian to accept/reject any appointments.

2.1 Product Perspective

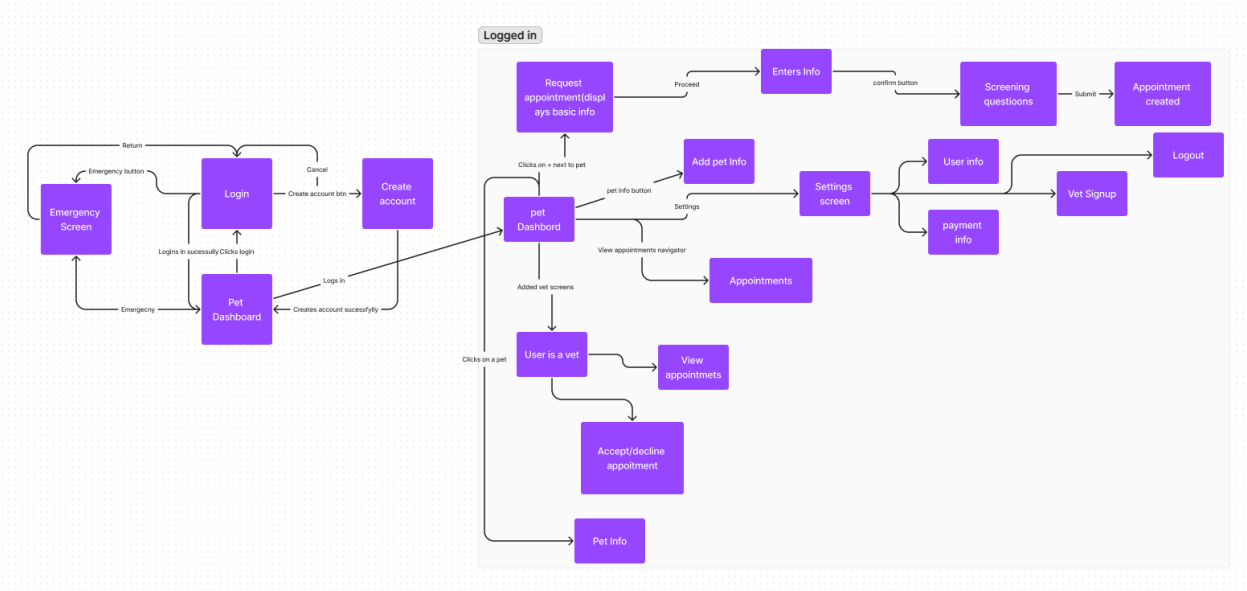
Similar applications include: Vetpocket, Uber

2.1.1 Concept of Operations

VetGo will be a cross-platform application for use on Android, iOS, and the web, to allow pet owners to create appointments with registered veterinarians. Additionally, the application will provide pet owners with information based on their pet's symptoms, suggesting emergency medical options if necessary or possible underlying conditions.
Veterinarians can manage incoming requests for appointments, sync requests with their calendars, view patient information and update medical records for veterinary visits.

With the support of a web server to handle requests for information from all front-end applications, a database to store user information, and a machine learning component, VetGo will provide accurate and timely information for pet owners and vets alike. One back-end component shared by all front-end applications will reduce the overhead of managing this application.

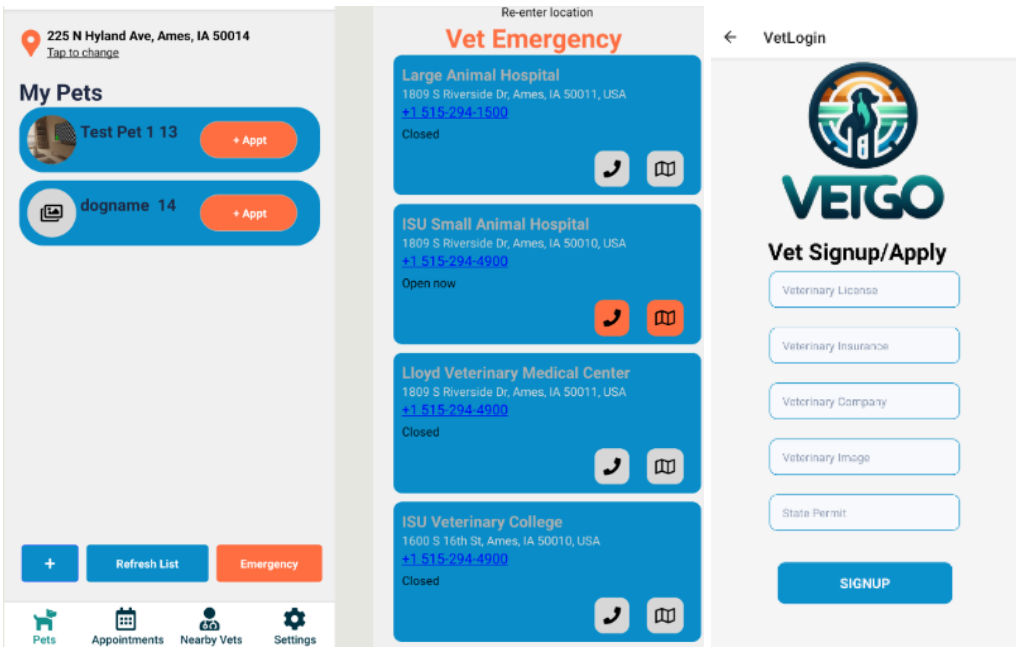
2.1.2 Major User Interfaces



2.1.2.1 Screen sketches for core features of VetGo



Updated Screens after development



2.1.3 Hardware Interfaces

Any device that supports a web browser and current standards of HTML, CSS, as well as Android and iOS devices either through app or on web.

2.1.4 Software Interfaces

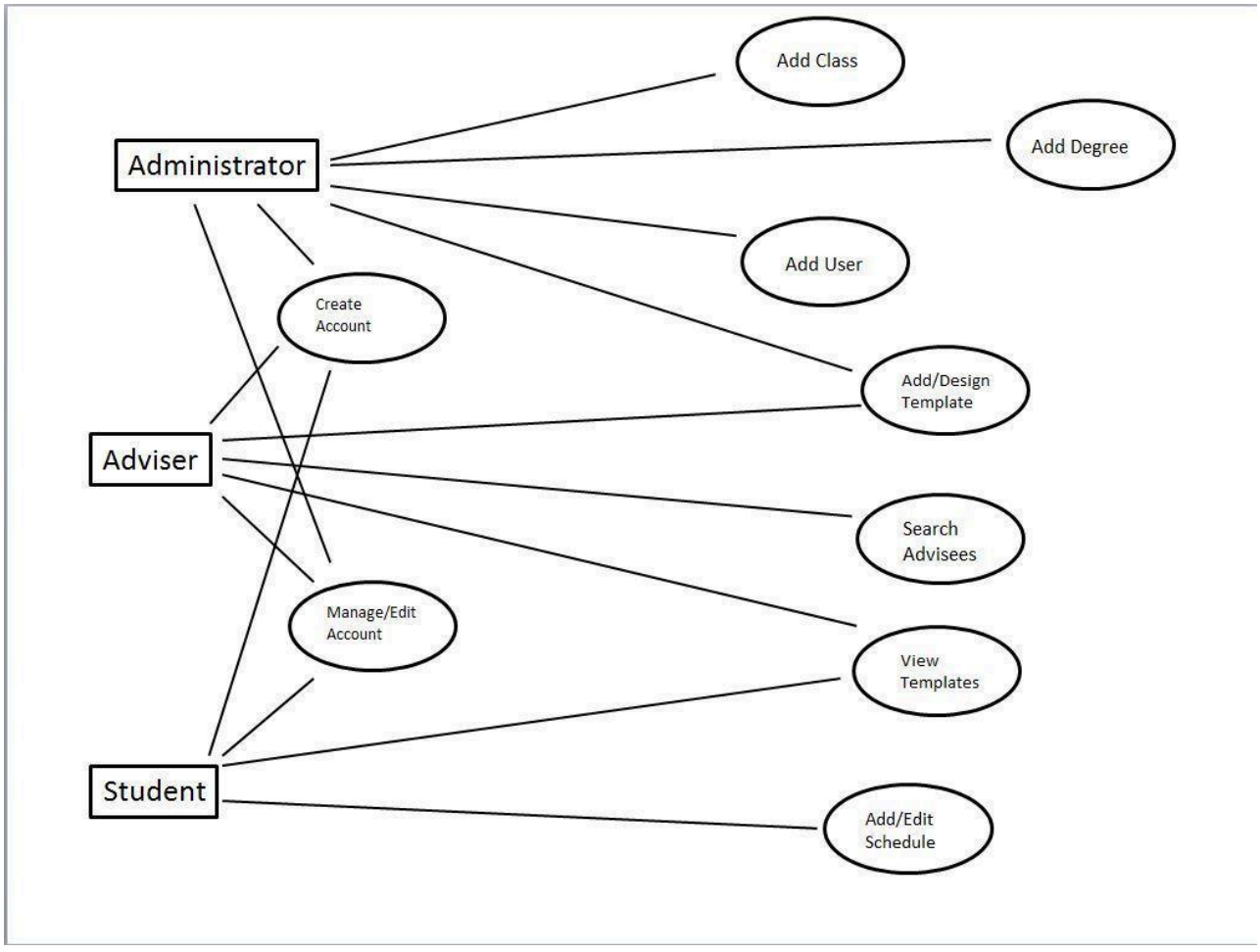
2.1.5 Communication Interfaces

2.1.6 Memory Constraints

2.1.7 Operations

2.1.8 Site Adaptation Requirements

2.2 PRODUCT FUNCTIONS



2.2.1 UC-0

User Registers on VetGo (User goal / New Users)

Main Scenario:

1. User downloads the VetGo app and opens it.
2. The system displays the welcome screen with options to "Log in" or "Register".
3. The user selects "Register".
4. The system presents a registration form.
5. The user fills in personal details, including name, email, password, and selects the account type (Pet Owner/Veterinary Doctor).
6. The user submits the form.
7. The system verifies the details and creates the new account.
8. The user receives a confirmation email or message.
9. The user logs in with the new credentials to access VetGo services.

Extensions:

5a1. The user uploads the necessary documents.

7a. The email is already in use.

7a1. The system alerts the user and requests a different email.

2.2.2 UC-1

User Logs Into VetGo Application (User goal / General)

Main scenario:

1. User launches the VetGo mobile app or accesses the VetGo website.
2. The system displays the login page, prompting for email and password.
3. User enters his or her email address and password and selects the "Log In" button.

4. The system verifies the credentials against the stored user database.
5. Upon successful authentication, the system directs the user to his or her respective home page (e.g., pet owner's home page, location page).
6. The user can now access the functionalities available to his or her user role, such as managing appointments, updating pet medical records, reviewing schedules, etc.

Extensions:

3a. User enters incorrect login credentials.

3a1. The system displays an error message indicating incorrect email or password.

3a2. The user is prompted to try again or use the "Forgot Password" option to reset their password.

6a. User logs in for the first time and needs to complete a profile setup.

6a1. The system directs the user to the profile setup page before proceeding to the home page.

6a2. User completes the necessary profile information and proceeds to their home page.

i.

ii. 2.2.3 UC-2

Pet Owner Requests an Appointment (User goal / Pet Owner)

Main Scenario:

1. Pet owner opens the VetGo mobile app.
2. The system displays the login page.
3. The pet owner enters login credentials and hits log in.
4. The system authenticates the pet owner and displays the home page.
5. Pet owner selects the pet with which he would like to request a new appointment.
6. Pet owner inputs the date of their appointment
7. Pet owner answers screening questions to determine the urgency of the pet's needs.
8. The system broadcasts the request to nearby veterinary doctors.

9. If a veterinary doctor accepts the appointment request, it is no longer broadcast and will only appear on the homepages of the requester (i.e., pet owner) and the accepter (i.e., veterinary doctor).
10. The pet owner logs out or continues to use the app for other functionalities.

Extensions:

3a. Pet owner's login credentials are incorrect.

3a1. The system displays an error message and allows the pet owner to try again or reset the password.

8a. If the appointment has not been accepted yet, the pet owner may cancel it.

8b. If the appointment has been accepted, the pet owner may cancel it along with a notification of the cancellation to the veterinary doctor.

iii. 2.2.4 UC-3

Veterinarian Accepts an Appointment (User goal / Veterinarian)

Main scenario:

1. Veterinarian opens the VetGo mobile app.
2. The system displays the login page.
3. The veterinarian enters login credentials and hits log in.
4. The system authenticates the veterinarian and displays the home page.
5. Veterinarian can accept appointments from pet owners that show up on the home page.
6. Pet Owner’s request for an appointment becomes accepted and stops being broadcasted to other potential veterinarians.
7. The appointment gets added to the Veterinarian’s calendar.
8. The Veterinarian can log out or continue using the app.

Extensions:

3a. Veterinarian's login credentials are incorrect.

3a1. The system displays an error message and allows the veterinarian to try again or reset the password.

5a. If the veterinarian has not accepted the appointment in the timeframe then the appointment request goes to another possible vet.

5b. Veterinarians can also reject the request at this time, and have the appointment request go to another possible vet.

6a. If the appointment has been accepted, the veterinarian may cancel it, and make the appointment request go to another possible vet.

iv.

v. 2.2.5 UC-4

Pet Owner Releases Medical Records of Pet to Veterinarian (User goal / Pet Owner)

Main Scenario:

1. Pet Owner requests an appointment with a veterinarian, which is accepted by a veterinarian.
2. On the appointment page, pet owners can click a button that releases the medical records for the pet.
3. Veterinarians then get a notification telling them that the pet's medical records have been released to them.

Extensions

3a. Veterinarians can then see the pet's medical records after opening the app.

2.2.6 UC-5

Wellness Examination (User goal / Veterinarian)

2.2.7 UC-6

Veterinary Doctor Reviews Pet Medical Records (User goal / Veterinary Doctor)

Main Scenario:

1. Veterinary doctor logs into the VetGo app before an appointment.
2. The system shows the vet's schedule on the homepage (i.e., upcoming accepted appointments and potential appointments near him or her).
3. The vet selects an upcoming appointment to prepare.
4. The system displays detailed information, including the pet's medical records.
5. The vet reviews the pet's medical history and any recent updates made by the pet owner.
6. The vet prepares for the appointment based on the pet's medical history.
7. After reviewing, the vet returns to the homepage or logs out.

Extensions:

4a. Medical records for the pet are incomplete or not updated.

4a1. The vet may request additional information from the pet owner by checking the pet owner's profile for his or her contact information (i.e., email and phone number).

2.2.8 UC-7

Pet Owner Completes Screening Questions for Emergency Assessment (User goal / Pet Owner)

Main Scenario:

2. Pet owner observes unusual symptoms in his or her pet and opens the VetGo app for guidance.
3. The system presents the options to "Assess Pet's Symptoms" on the homepage.
4. Pet owner selects the pet whose symptoms he or she would like to assess.

5. The system displays a series of screening questions designed to assess the urgency of the pet's condition.
6. Pet owner answers the screening questions, providing details about the pet's symptoms and behavior.
7. Upon completion, the system analyzes the responses
8. Based on the analysis, the system categorizes the situation as either an emergency or non-emergency.
9. If an emergency is detected, the system recommends immediate action, such as visiting the nearest pet emergency center, and provides contact details and directions.
10. If a non-emergency, the system suggests monitoring the pet and possibly scheduling a regular vet appointment through the app.

2.3 User characteristics

Users will be adults searching for veterinarian appointments. Users will likely not be using the app frequently or multiple times in quick succession, as there is likely little need for repeat appointments shortly after one another.

2.4 Constraints

The application must be cross-platform, requiring compatibility across multiple operating systems.

2.5 Assumptions and Dependencies

Users are assumed to have access to an Android or Apple device or a web browser. In addition, they are expected to have internet access and approved access to location services. They also have optional access to camera. Vets have access to their id/information at their register time

3 Specific Requirements

3.1 External Interface Requirements [OMIT THIS SECTION]

3.1.1 User Interfaces

3.1.2 Hardware Interfaces

3.1.3 Software Interfaces

3.1.4 Communications Interfaces

3.2 FEATURES

3.2.1 Cross-Platform

3.2.1.1 The system can be accessed on multiple platforms, including IOS, Android and the Web.

i. 3.2.2 Database

1. 3.2.2.1 The system contains a database that stores all the users' relevant information. It will be using mySQL for the server.

3.3 Performance requirements

1. Information will be updated within 1 second of submission to ensure no delay in appointments
2. Vet accepting appointment happens on the backend within 1 second to ensure no double bookings
3. Emergency information is up to date to ensure a pet owner can get the correct information

3.4 Design Constraints

1. UI must be easy to use
2. UI is accessible to all user demographics
3. Project designed with scalability
4. *for further information, look to the design section

3.5 Software System Attributes

3.5.1 Reliability

App must be very reliable since the health of pets is dependent on how well the app works. If the app breaks or stops working during a pet emergency then the health of the pet could be in serious danger.

3.5.2 Availability

The app must be available to users 24/7 since pet emergencies can occur at any time of the day. If a pet is having an emergency and the app is offline then the health of the pet could be in serious danger.

3.5.3 Security

Application security is extremely important considering the sensitive nature of the information contained. Libraries to execute SQL queries will be used to prevent injection, and credential checking will be strictly enforced to ensure that information is only accessible to the proper individuals.

3.5.4 Maintainability

The application will be developed in a way that it can be easily maintained and extended by another team, including integration with other APIs in the future.

3.5.5 Portability

[None]

3.6 Other Requirements

[None]

Contributions and Learnings

Member 1: Andy Bruder

Contributions

- Setup CI/CD pipelines for the project through GitLab.
- Setup web server and configurations for database.
- Set up Spring server to work with HTTPS, including key generation and deployment.
- Implemented Authentication controllers for sign-in and refresh-tokens.
- Implemented token generation features and scope based permissions for server.
- Implemented role-granting capabilities for the backend.
- Handled storing of authentication tokens and refresh tokens on client device.
- Refactored frontend requests to the newly secured server using Authentication Basic and Authentication Bearer. Essentially, every access to the server had to be rewritten to provide authorization and handle the data.
- Created calendar screen to view appointments for a user.

Learnings

- Learned React Native, TypeScript, Expo framework, SpringBoot, Spring Security, OAuth2.0 and JWT, and Authentication framework basics.

Member 2: Elijah Brungardt

Contributions

- Developed Broadcast Appointment Algorithm for backend.
- Created MyAppointments Screen that shows appointments for vets and pet owners.
- Created Request Payment Screen that allows vets to request payments.
- Created Send Payment Screen that allows pet owners to send payments to vets using Stripe.
- Created Review Screen that allows pet owners to leave a review for a vet.
- Setup Stripe for payment handling.

Learnings

- Learned React Native, Expo, TypeScript, REST APIs, and Stripe.
- Learned how to use Git to create and merge feature branches.

Member 3: David Marvin

Contributions

- Assisted in developing the frontend of the application.
- Worked on the following screens: Login, emergency, register, location, landing page, create pet, view pet, pets home screen, create appointment, settings, nearby vets, user info, and vet register.
- Styled the application's frontend and created the CSS, color, and formats for the application.
- Implemented Google maps API to query for locations and display important information.
- Implemented error handling on frontend prior to and after API requests are sent.
- Set up and usage of requests to backend throughout application.
- Handled the general creation of frontend pages.
- Constructed the screen flow diagrams.

Learnings

- Learned React Native, TypeScript, Expo framework, and Google Maps API.
- Improved practices with using Git issues to create clear tasks.

Member 4: Josh Podlich

Contributions

- Created screening questions screens.
- Gave screening questions a list view that gets updated every time a question's answer is changed.
- Refactored screening questions screens multiple times to better meet our client's expectations.
- Set up the creation of an appointment at the end of screening questions.

Learnings

- Learned React Native, TypeScript, and Expo framework.
- Learned how to manage and maintain multiple branches in GitLab.

Member 5: Leo Stoyanov

Contributions

- Restructured the database, specifically for User and Pet models, for maintainability and future functionalities.
- Implemented screening questions for pet owners to answer a set of questions concerning their pet's symptoms and produce a result containing an urgency level, what the problem could be, first-aid advice, and recommendations for the pet owner on what to do next.
- Manually populated the database with screening questions, options, and results.
- Implemented a forgot-password feature on the backend, which includes token generation and email confirmation mechanisms for a secure password-reset process.

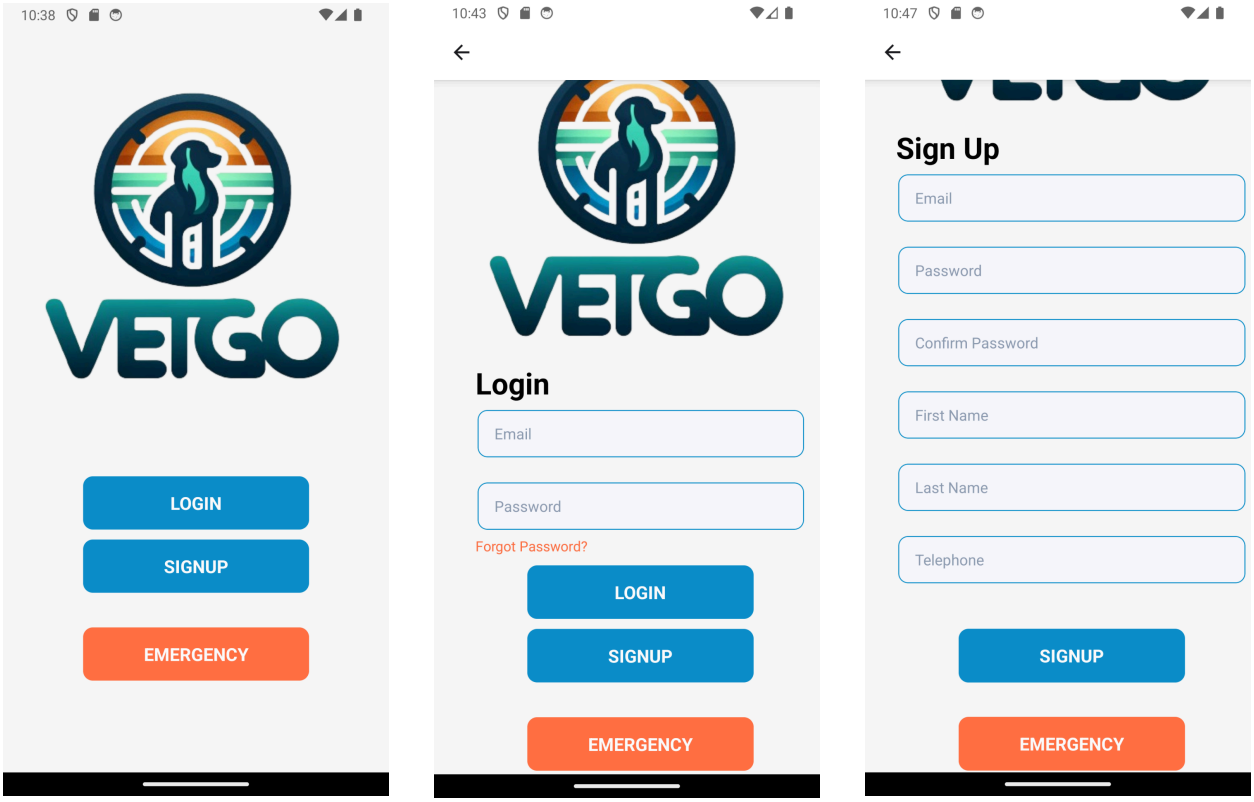
- Fixed creating and editing a pet's information on the backend to match the new database structure.
- Fixed the ability for users to save their pet's images; images are stored in a local directory, but in production, they would be stored in a server's directory.
- Implemented location services using Google's APIs: Pet owners may set their preferred appointment location, veterinarians may set their current location from which they will travel, and pet owners or veterinarians may preset their preferred home address or office address, respectively.
- Improved and fixed the frontend's navigation structure and bottom tab navigation bar.
- Developed a feature that calculates the haversine distance to find nearby vets for an accurate distance calculation from pet owners to nearby vets.

Learnings

- Learned React Native, TypeScript, and the Expo framework on the frontend.
- Learned SpringBoot and gained more experience with REST APIs on the backend.
- Applied previous skills and learned new skills in MySQL.

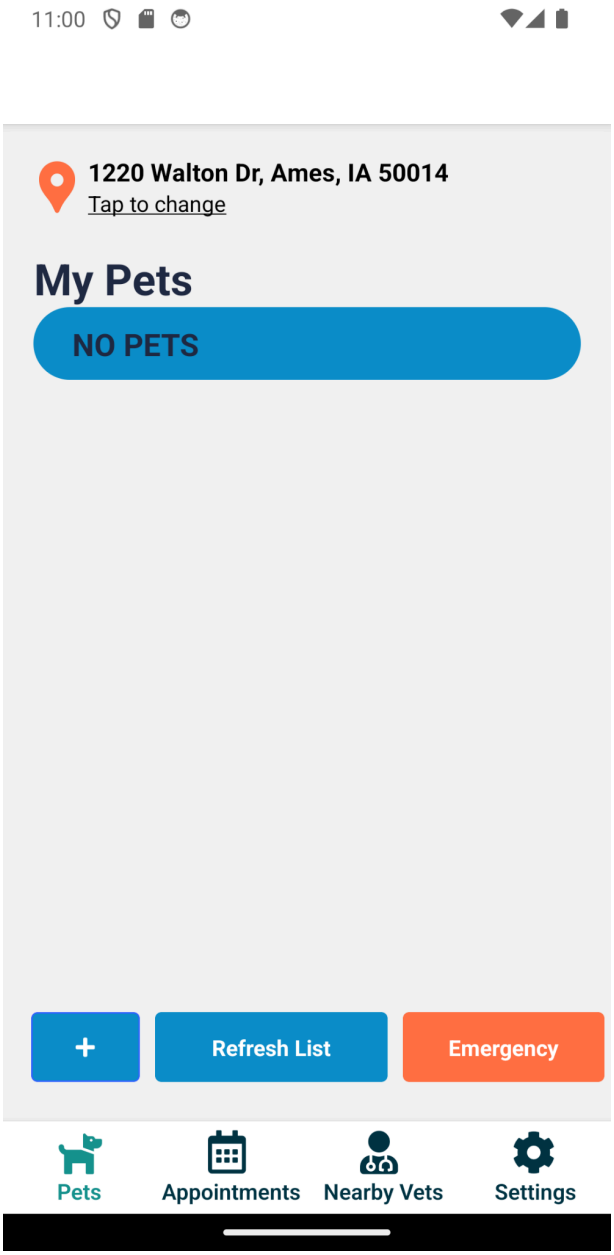
Results

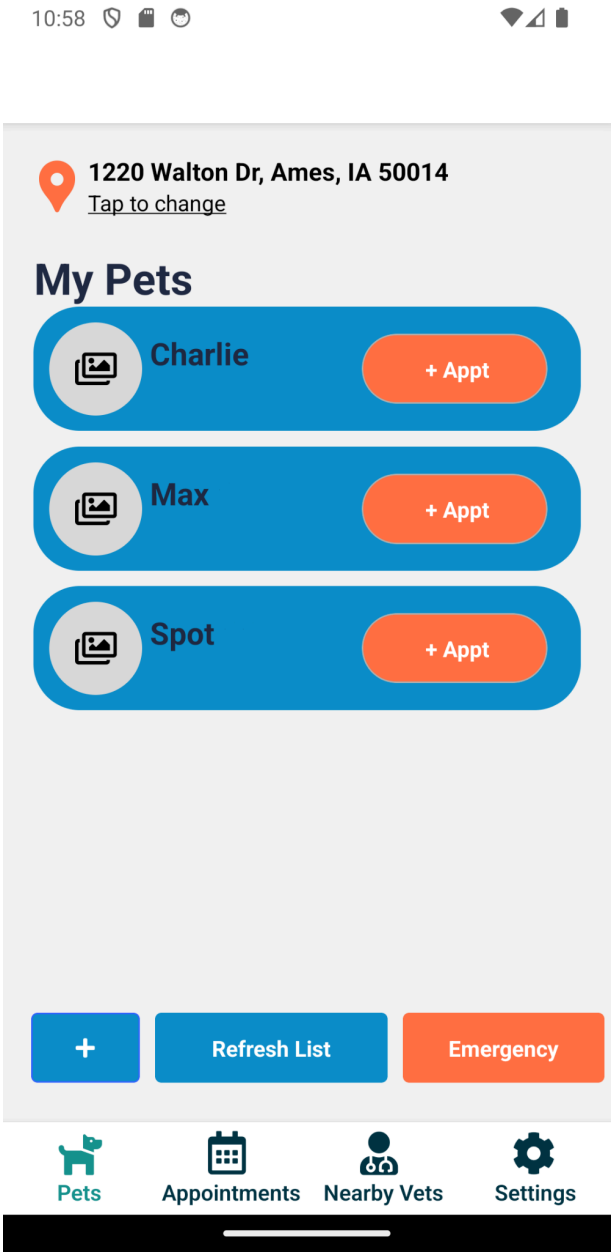
Login and Signup



Above is the VetGo landing page which opens upon the launch of the app, along with the login and sign up page. Existing users can login to access their account and new users can use the signup page to register. All users, owners and vets, must first use this initial signup page. This also provides access to the emergency information tab, displaying all nearby pet hospitals.

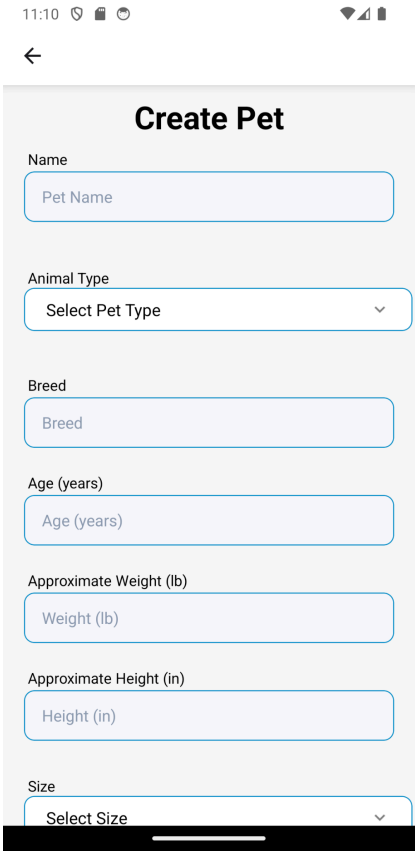
User and Vet Home

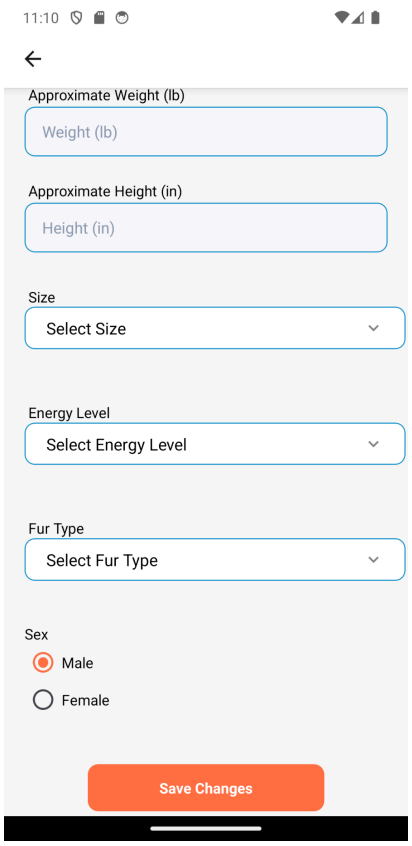




Above is the view presented upon the initial login, after the location has been either selected or bypassed by the user. The app appropriately displays the pets tied to a user's account. From here, users can create an appointment for a pet, upload an image of your pet, add a new pet, or navigate to the emergency screen, also available from the landing page. At this point, users are also able to navigate to the other tabs.

Pet Creation and Data

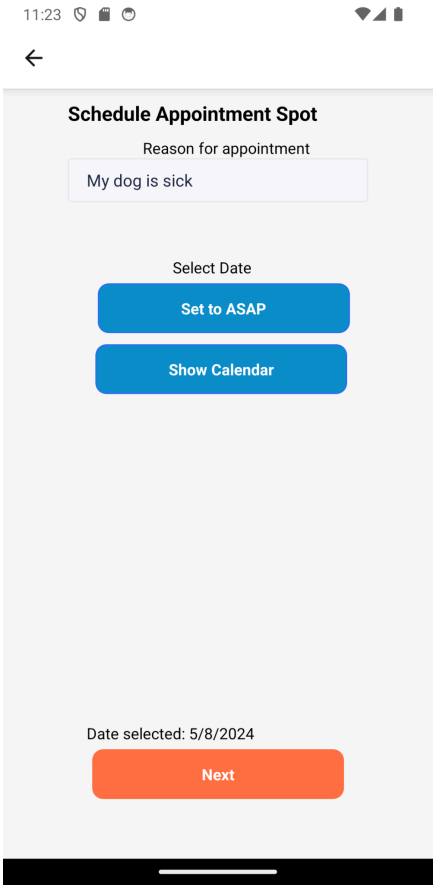


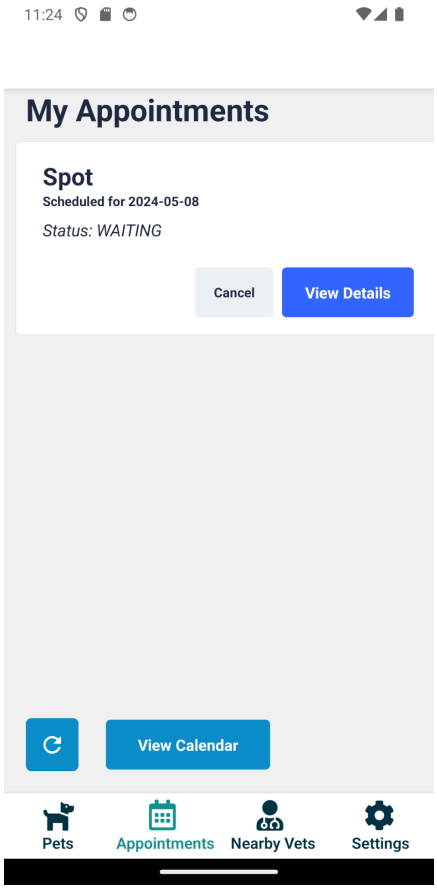


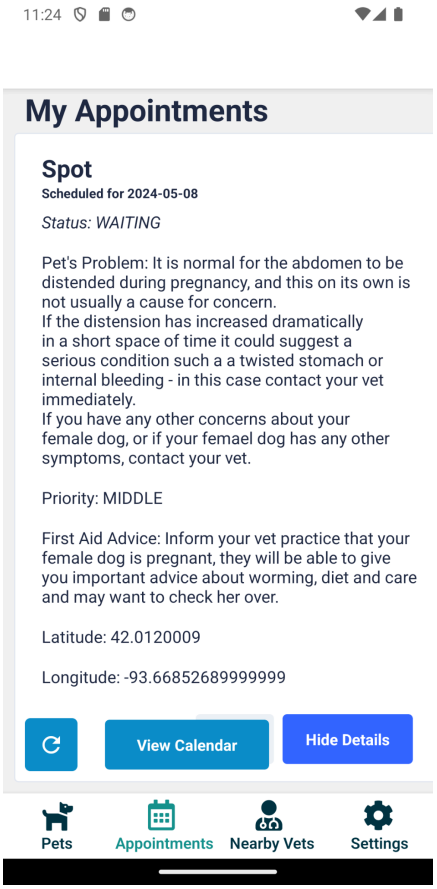


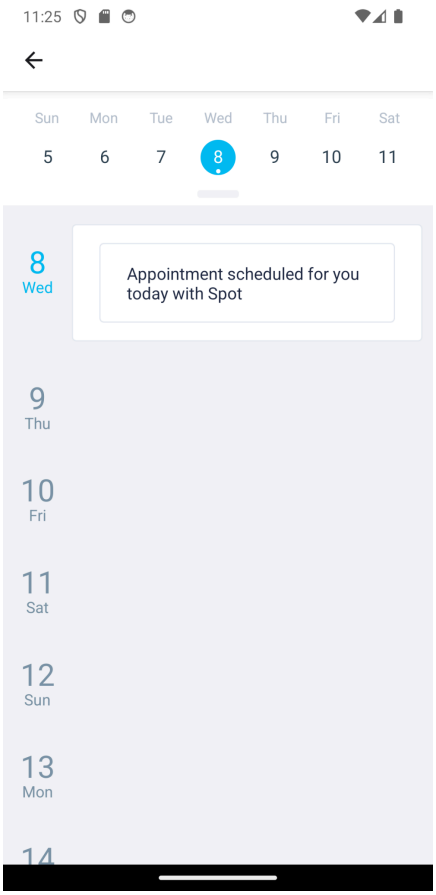
Once a user navigates to the pet creation page from their home screen, they will be able to add the necessary information to create a variety of serviceable pet types. These pets will show up on a user's pet list once they refresh the page. Then users may upload their image or view the pet information level as shown in the screen on the right.

Appointment Creation and Viewing



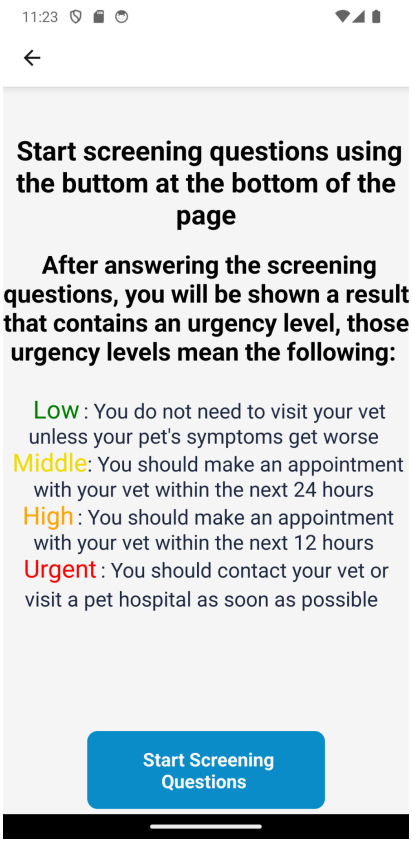


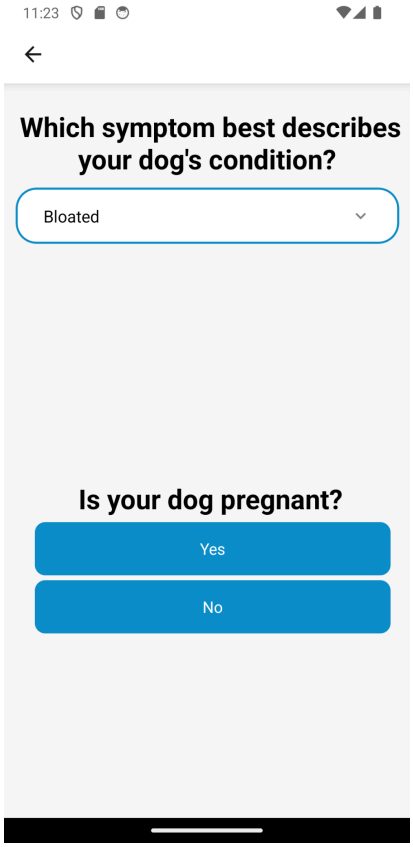


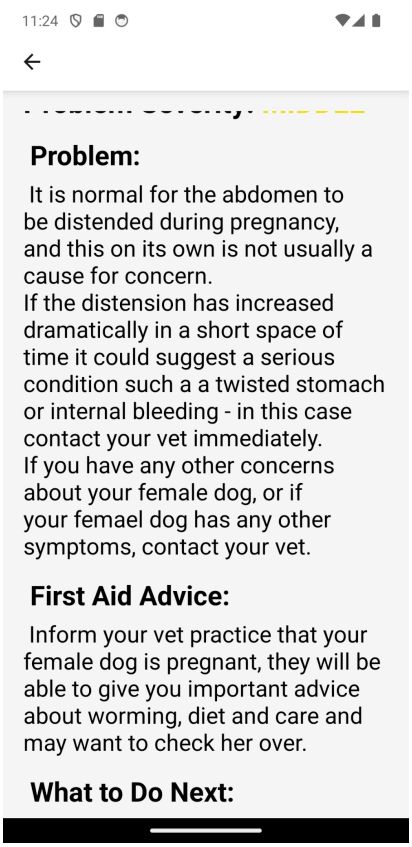


This page displays the process for creating an appointment for a particular pet. When creating an appointment, you can enter a description, set it to ASAP, and change the date via the calendar. Upon clicking next, you will be directed through screening questions (next section) before completing the process. Appointments will be viewable through the my appointments tab with appointment details and the calendar available on the my appointments tab.

Screening Questions

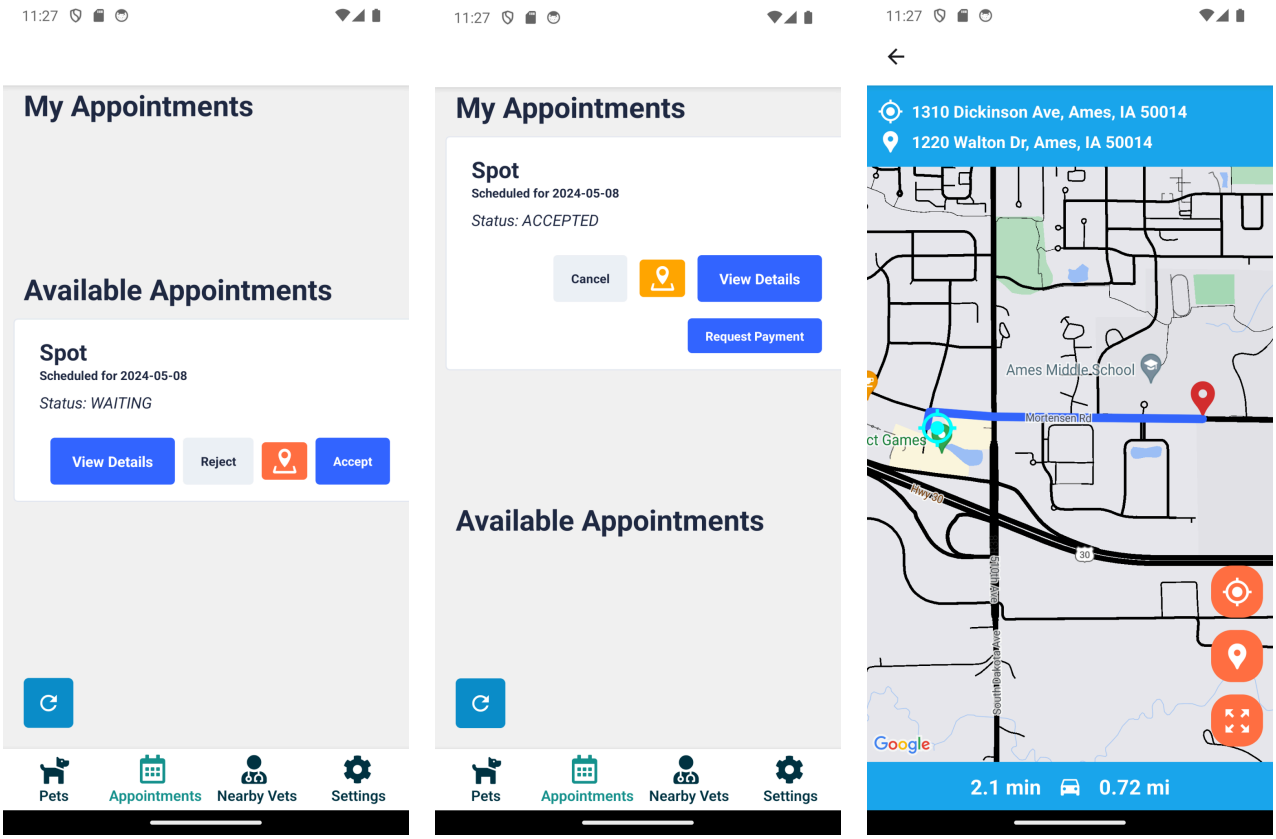






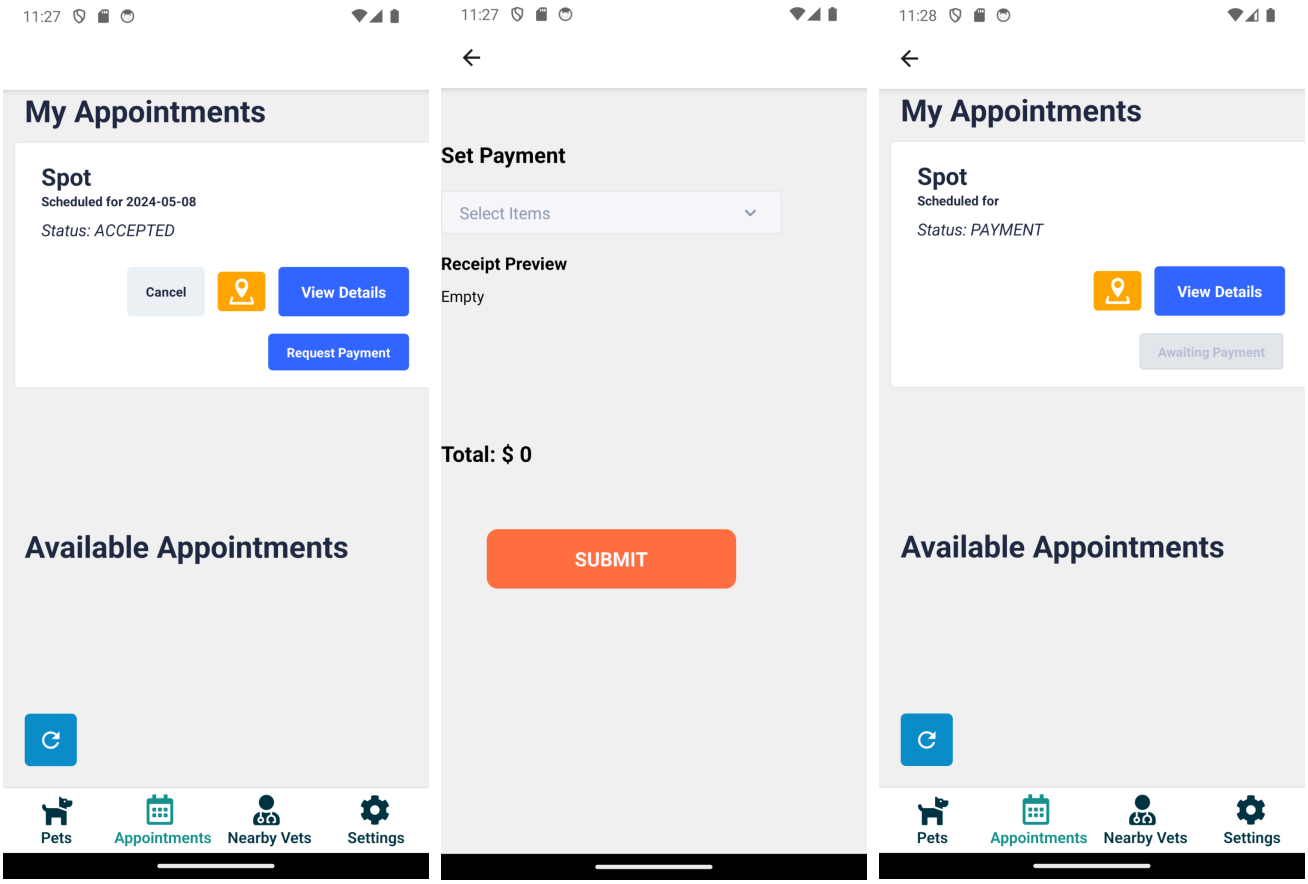
The screening questions above are initiated upon the creation of an appointment for a pet. Urgency levels are provided to give the user an idea of the problem's severity and the user is guided through answering questions based on the pet's condition. This information is saved along with the appointment upon completion.

Appointments - Vet Specific



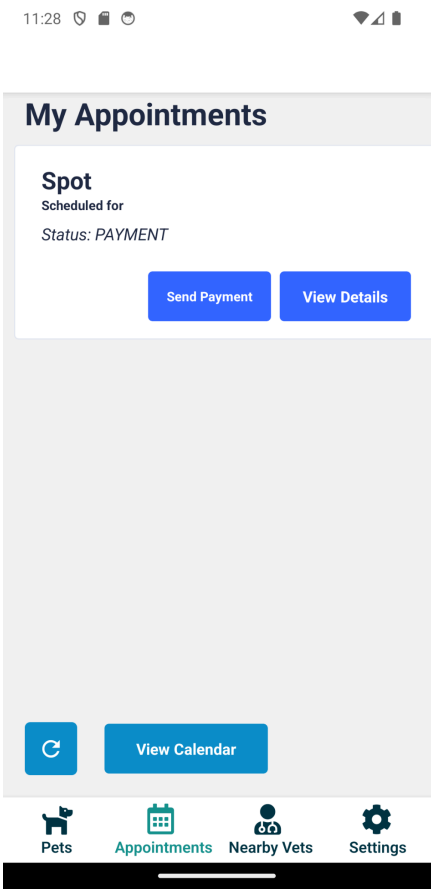
Vets have an additional view on their appointments screen, allowing them to view the details of the appointment and either accept or reject. The appointment will also allow the vet to get directions to the location provided by the owner. Vets can also request payment (payment) from the owner upon completion of the appointment.

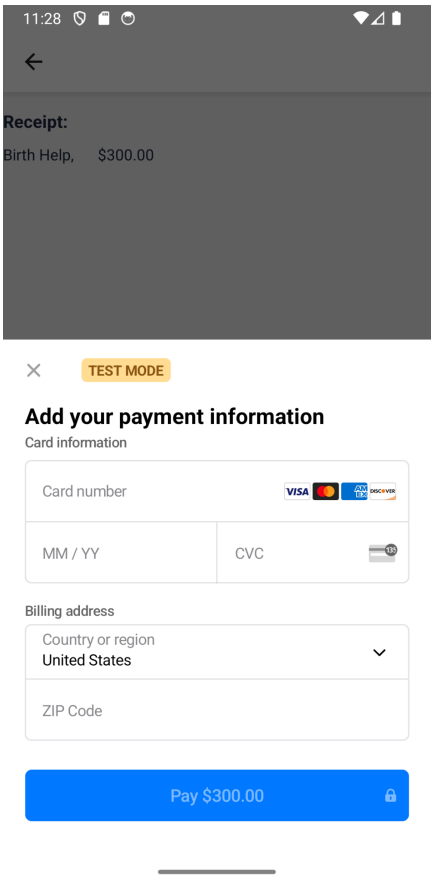
Payment - Vet Specific

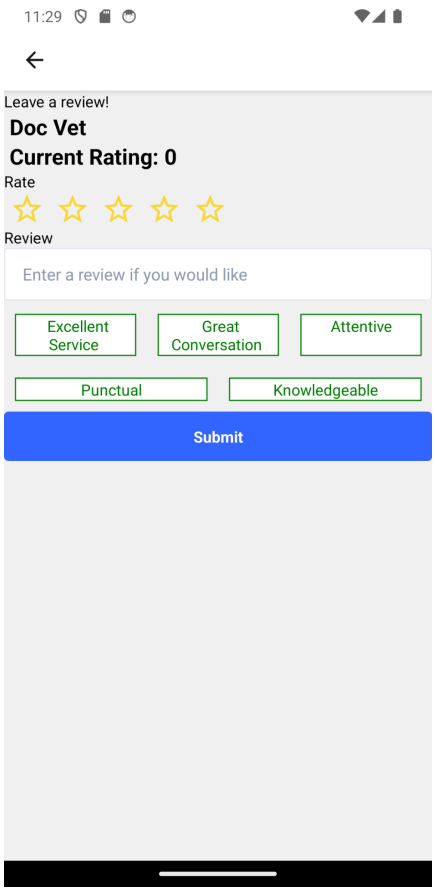


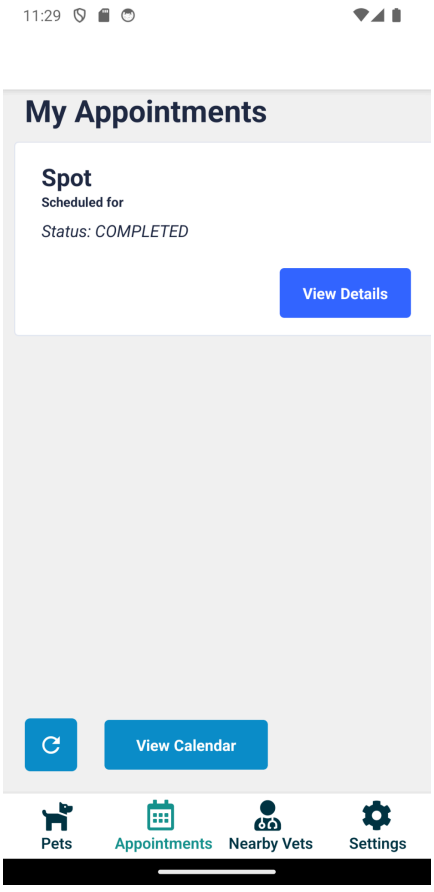
Vets are able to request payment from an owner after the appointment has completed. Vets select the services that they are charging and submit the request to the owner. Upon submission, the appointment awaits payment from the owner.

Payment - Owner Specific



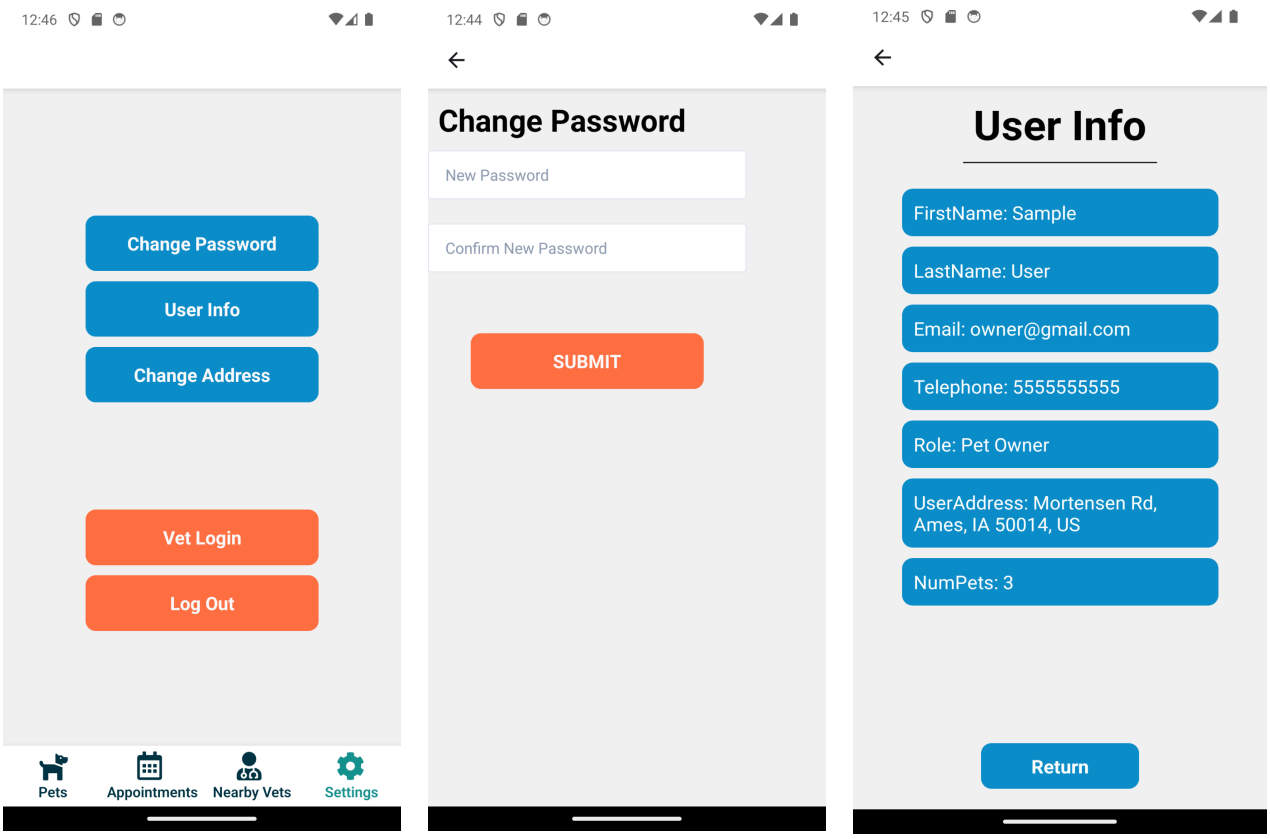




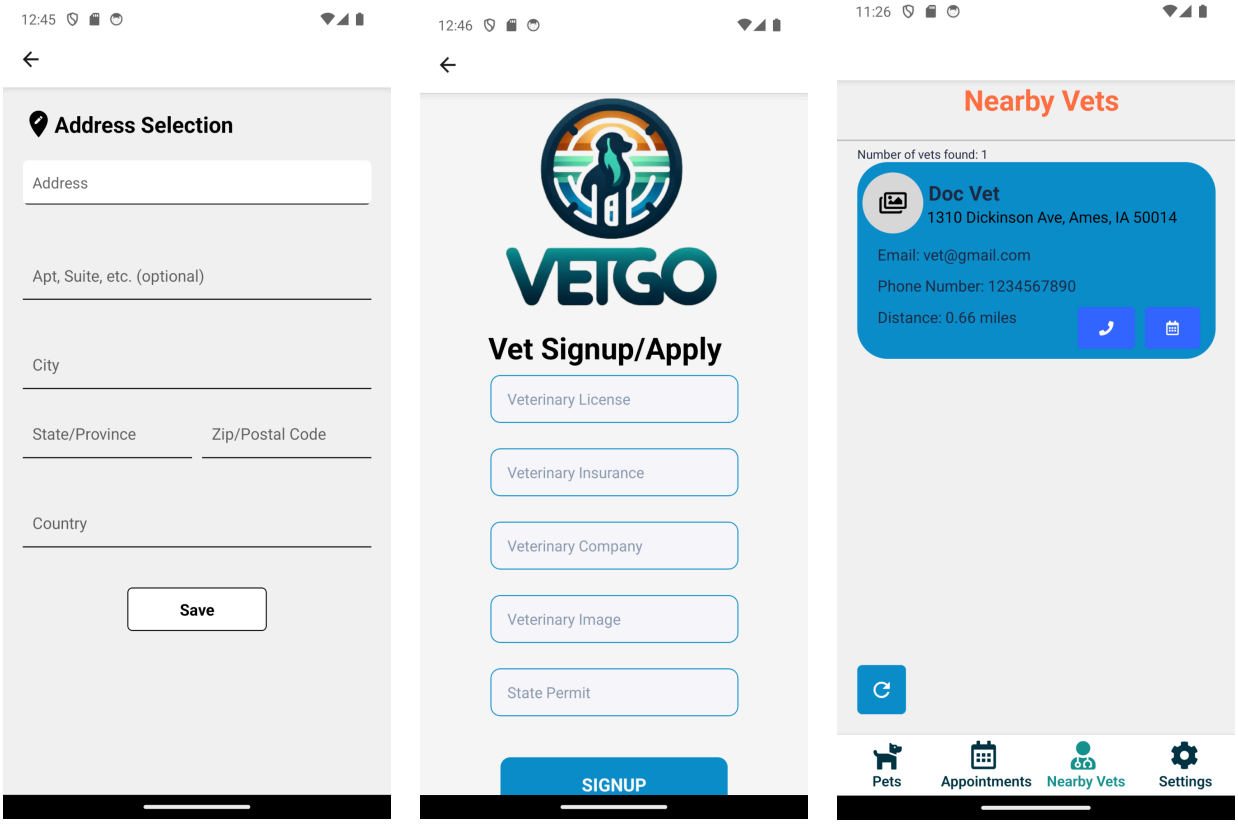


Once a vet has requested payment from an owner, their will be an option for the owner to submit their payment via credit card. Once information has been added and payment has been processed, the user will be directed to a review screen for their vet. After the review is either completed or skipped, the payment will be complete.

Miscellaneous



Some miscellaneous functions include those in the settings page, such as changing your password and viewing your user information.



Users are also able to change their address and sign up as a vet, so they can be a VetGo provider. Users are also able to view all registered vets in the area via the Nearby Vets tab.

VetGo - Group 10

Overview - What is VetGo?

- Uber like app for veterinarians
- Cross Platform
- Uses AI/Machine learning to help suggest conditions to the vet
- Allows pet owners to quickly find veterinarians
- Helps vets manage their appointments



Overview - How we plan to build VetGo

- Get previous years code working
- Find and solve problems in the code
- Add missing necessary features
- Using React Native, UI Kitten and Expo on frontend, MySQL and SpringBoot for backend



VetGo Demo



UI/UX Design

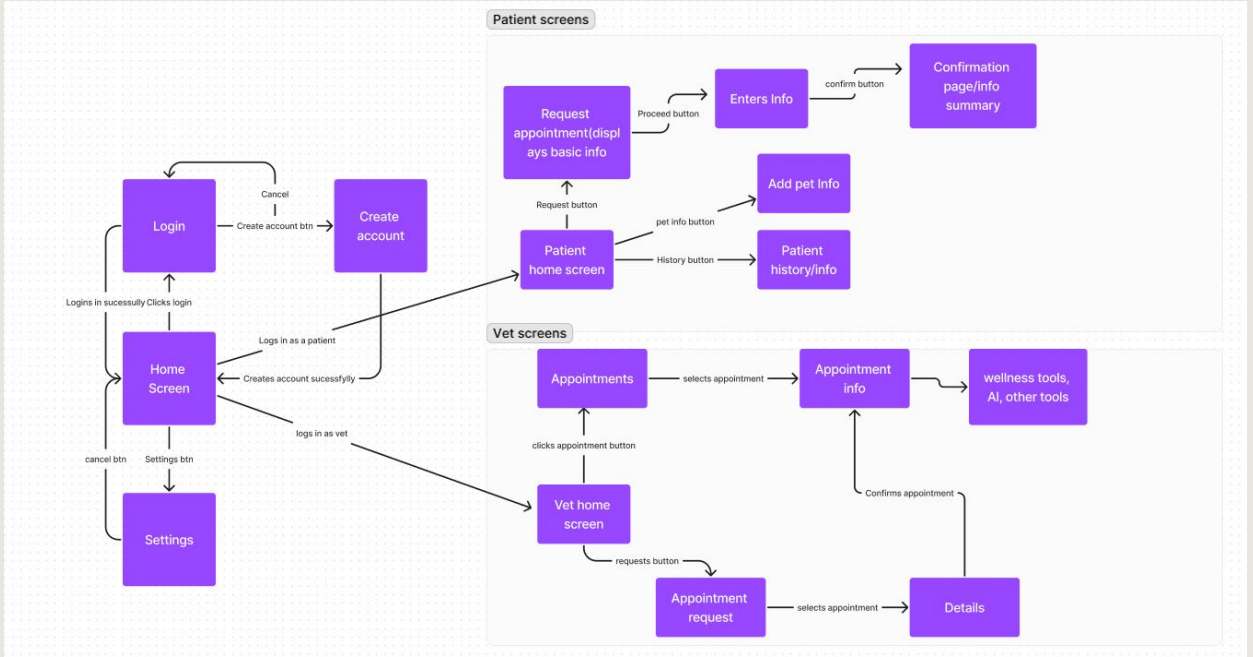
Where the project currently is

- UI/UX can be strengthen
- Increasing readability
- Changing the flow/accessibility of screens
- Increasing professional feel
- Creating better features for speed



Screenflow plans

- Difference between vet and user home screen
- Easier to access information
- Ensure it is quickly to navigate between screens
- Allows for addition of new screens/functionality



UI design plans

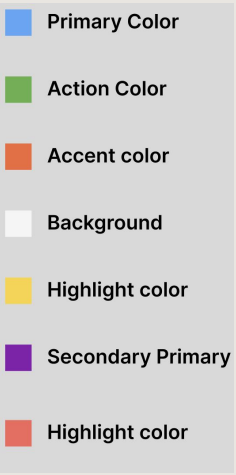
- Color Scheme
  - We want the color scheme to be to be high contrast to ensure readability to all users
  - We also want to color scheme to be eye catching and friendly feeling

Icon integration

- We want to incorporate intuitive icons to enhance user understanding
- This also allows users who may have difficulty reading to navigate through the pages

Text Readability

- Adding helpful information when hovering/pressing and holding on any button
- Increasing font size and changing fonts to allow for a more readable experience
- Speech to text







UI/UX plans for veterinarians

- Improving information Accessibility
    - Implementing quickaction shortcuts for all pages on the veterinarian portion of the program
- Information usability
    - Organization of data transformed to match industry standards
    - Offering customizable views, and filters to increase use
- Speed
    - Due to the nature of the application requiring veterinarians to quickly decide if they want to take a call
        - Allow veterinarians to view condensed specific information
        - Allow veterinarians to quickly sort information



UI security

- Current issues
  - The current program does not currently have a full level of security and checks for all of the pages
  - There may be ways to bypass the security by brute force
- Changes
  - Stress testing the current program
  - Adding timeout, incorrect password attempt
  - Extra checks on each page



Database Restructuring (1)

id	average_rating	email	latitude	longitude	num_reviewed	password	username	user_tags
1	0	never@gmail.com	1	1	0	$2a$10$Oc1TPIeTLBPWBzRwP0jgfOB238SDS8lj...	user1	NULL
2	0	gonna@gmail.com	1	1	0	$2a$10$Oc1TPIeTLBPWBzRwP0jgfOB238SDS8lj...	user2	NULL
3	0	give@gmail.com	42.0307812	-93.63191309999999	0	$2a$10$Oc1TPIeTLBPWBzRwP0jgfOB238SDS8lj...	user3	NULL
4	4.5	you@gmail.com	42.0307812	-93.63191309999999	1	$2a$10$Oc1TPIeTLBPWBzRwP0jgfOB238SDS8lj...	user4	4
5	0	up@gmail.com	1	1	0	$2a$10$Oc1TPIeTLBPWBzRwP0jgfOB238SDS8lj...	user5	NULL
6	0	leos1@iastate.edu	1	1	0	$2a$10$1lABg22BKYkpbUcYjJi4HuYH0Zmx0U1Q...	LeoKing	1
7	0	leos2@iastate.edu	1	1	0	$2a$10$mTD5tv1iISRO1FGgZm0lq.awLbUrdWY...	LeoStoy	2
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

- Remove Person model
- User Model and User Repository
  - Take on First Name, Last Name, Telephone, and Address fields/columns
  - Remove the Username; Email will be the only identifier.
  - Move average_rating to Vet model
  - Move num_reveived to Owner model
  - Give the User the option of whether to be a Pet Owner, Vet, or *both*
    - Move latitude and longitude to Vet model and Owner model



Leo

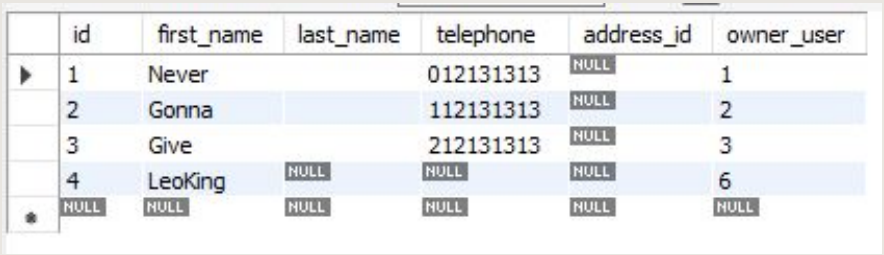
Database Restructuring (2)

- Pet Model, Pet Repository, and Pet Service
  - Pet Model with PetType field/column → Joined Inheritance architecture
    - Reptile Attributes Example:
      - Species: String (e.g., Turtle, Snake, Lizard)
      - HabitatType: Enum (AQUATIC, TERRESTRIAL, ARBOREAL)
      - DietType: Enum (HERBIVORE, CARNIVORE, OMNIVORE)
      - Temperature Requirement: String (e.g., "75-85˚F")
  - Additional Pet Information → Screening Questions Information



Database Restructuring (3)

- Vet Model and Vet Repository
  - Implement veterinarian specialties
    - Vet specialization → Pet owner's choice
- Owner Model and Owner Repository
  - Owner adds nothing new → Give it purpose



id	first_name	last_name	telephone	address_id	owner_user
1	Never		012131313	NULL	1
2	Gonna		112131313	NULL	2
3	Give		212131313	NULL	3
4	LeoKing	NULL	NULL	NULL	6
NULL	NULL	NULL	NULL	NULL	NULL

Missing, Broken, or Incomplete Features

- Billing Information
- Screening Questions + Machine Learning
- Time of Arrival
- Release of Medical Records
- Calendar Integration
- Pet Images
- Geolocation of Users
- Speech2Text



Security

The good

Encrypted passwords

password	username
$2a$10$2J.VJ1eCA.D0CAMEdebqPuUijbjW6kJlBxCaTJImWDtKKQf9sHxnK	user1
$2y$10$PTtXQ1h1JXGAbBJ7ItCH1.Bf2ITgnTNh1Ug95L9byXrKRpX8ZBSMS.Q5Nsi	user2
$2y$10$yG4A943e0jizgpS0nfJA5efNo6ivJtVTFzbyQcQpuJjdDpiapSr3i	user3

Sanitized inputs

```
User user;
if (username.contains("@")) {
    user = userRepository.findByEmail(username);
} else {
    user = userRepository.findByUsername(username);
}
```

```
@Param("radius") Double radius)
```

The bad

Poorly secured endpoints

```
@GetMapping(value = "/all")
public ResponseEntity<List<User>> getAllUser() {
    List<User> userList = new ArrayList<>();
    userRepository.findAll().forEach(userList::add);
    return ResponseEntity.status(HttpStatus.OK).body(userList);
}
```

Limited credential checking

```
// Delete specific user based on id
roy-kcj +1
@DeleteMapping(value = "/delete/{uid}")
public ResponseEntity<String> deleteUser(@PathVariable("uid") Long uid) {
    User user = userRepository.findById(uid).orElse( other: null);
```



Future Features - Administration

- There is no program oversight
  - New Veterinarians cannot be authenticated
  - There is no way to manage or monitor users on the application
- Current test users are inaccessible
- All user management is done through the back-end and manipulating SQL data

- TODO: Add an administrative role to do the following:
  - Confirm Vet status
  - Manage users and vets
  - Interact with admin functions through the application



Future Features - Authentication

- Veterinarian license is entered as a string during sign-up
  - Not a valid credential

- Submit the documents for a valid degree

- TODO:
  - Research:
    - Valid veterinarian degrees, certifications needed for practice, etc.
  - Implement:
    - Submit files. Uploading pdf, jpg, png, etc for confirmation by administrator
    - Leave room for the future: Degree certification APIs



Future Features - Emergency

- The “Emergency” function of the app is not working
- Directed to Emergency page based on screening or user choice
  - Not Implemented!
- Use Google Maps - Places API (New) to find local emergency pet services
- This is an integral part of the application which was left undeveloped by the previous team

- TODO:
  - Research Places API
  - Implement the emergency “choice” function, as well as the screening which both will direct you to our emergency pet services map



Next Demo!

- Refactoring existing code
- UI design
- Database restructuring
- Features:
  - Administrator functions
  - Vet authentication
  - Emergency clinics



VetGo Demo 2

Group 10

VetGo Purpose Recap

- Goal: Uber-like app for veterinarians and pet owners
  - Pet owner broadcasts appointments to vets
  - Each vet can decide which appointments to take
- Share medical records of a pet to a vet
- Emergency feature to provide a map of nearby vet hospitals
- Screening questions to quickly assess a pet's condition and provide an urgency level
- Use speech2text and machine learning to suggest conditions to the pet owner
- User-friendly interface

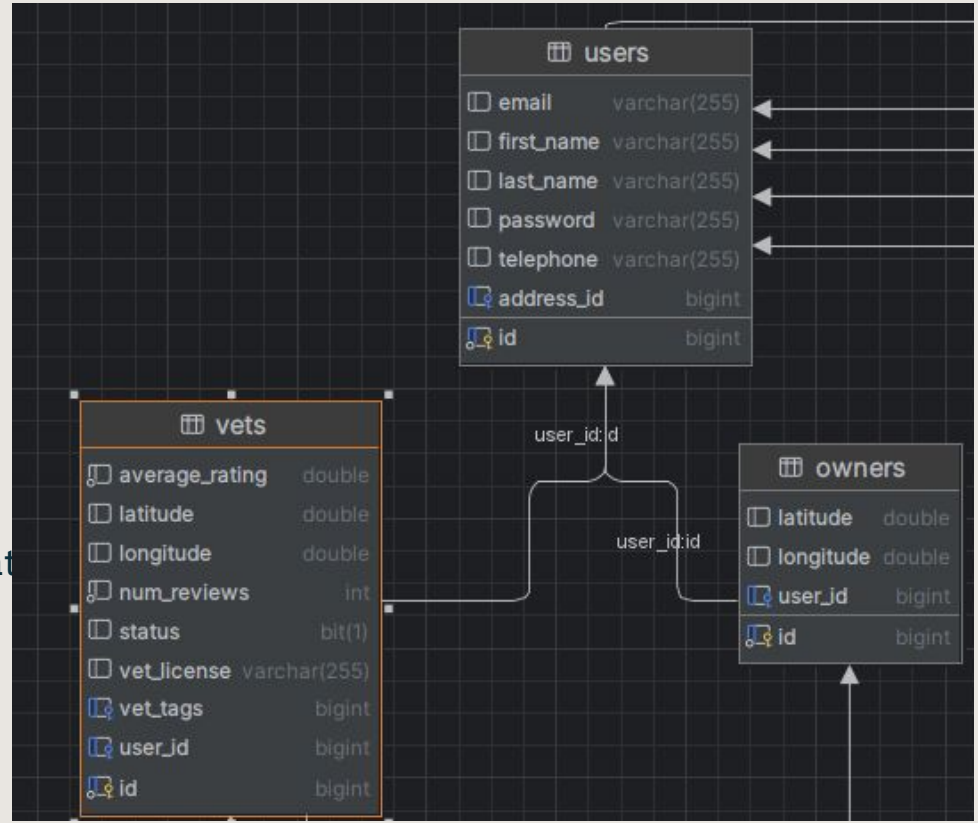




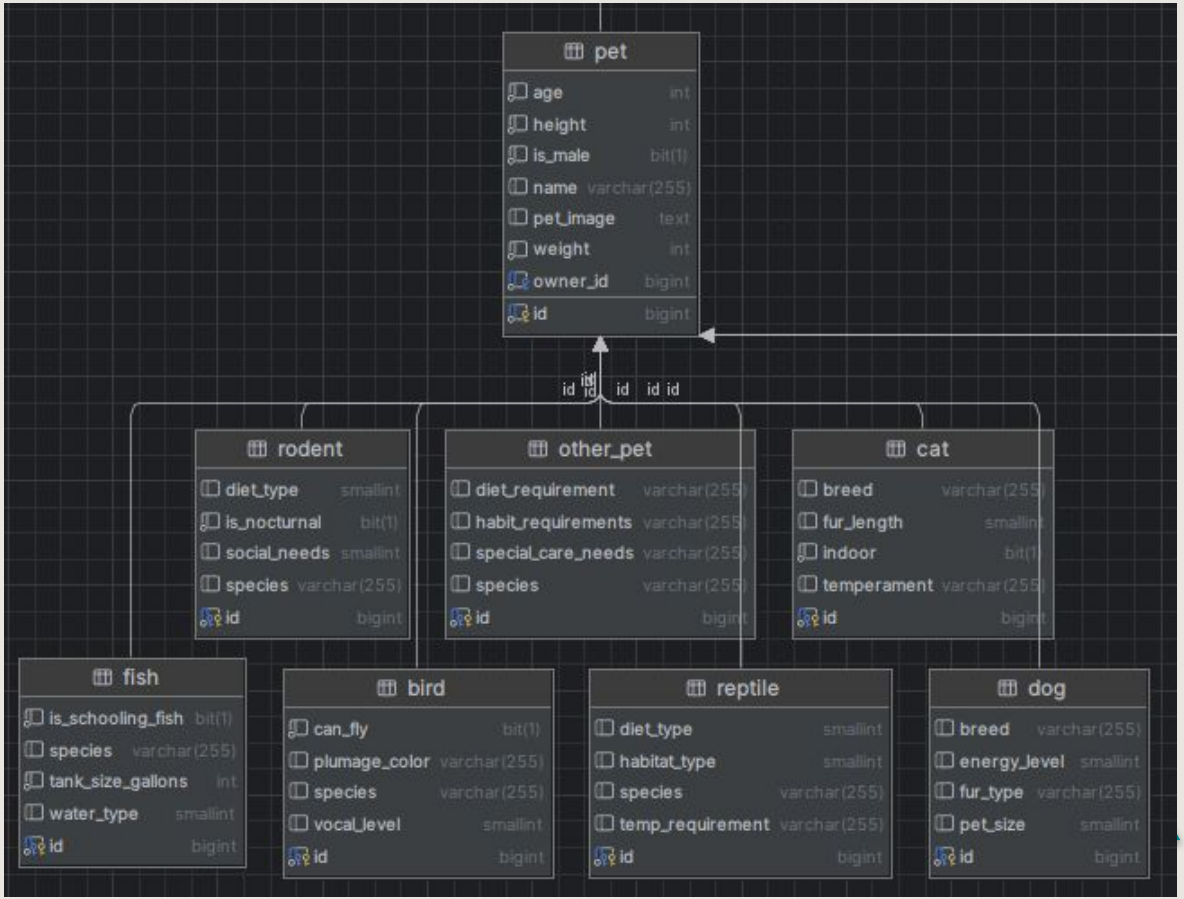
Leo

Recap of Database Restructuring (1)

- Remove Person model
- User
  - Email as only identifier
  - Vets have reviews, tags, and rat
  - Owners have latitude and longitude
  - Dynamic roles

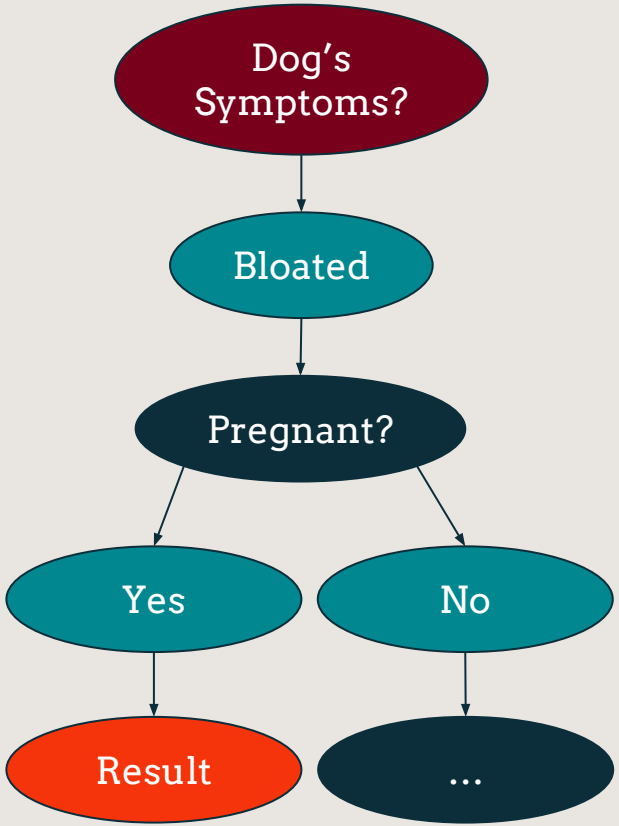


Recap of Database Restructuring (2)



Screening Questions Overview

- Choose to answer questions based on symptoms or when creating an appointment
- Determines the level of urgency in need of veterinary assistance
- Screening session → Root screening question (depending on pet type) → Choose an option → New screening question → … → Result
  - From a high level, decision tree data structure
  - Contents of the result: urgency level, what to do next, first aid advice, and the problem
  - Urgency levels: self-care, medium, high, and urgent



Leo

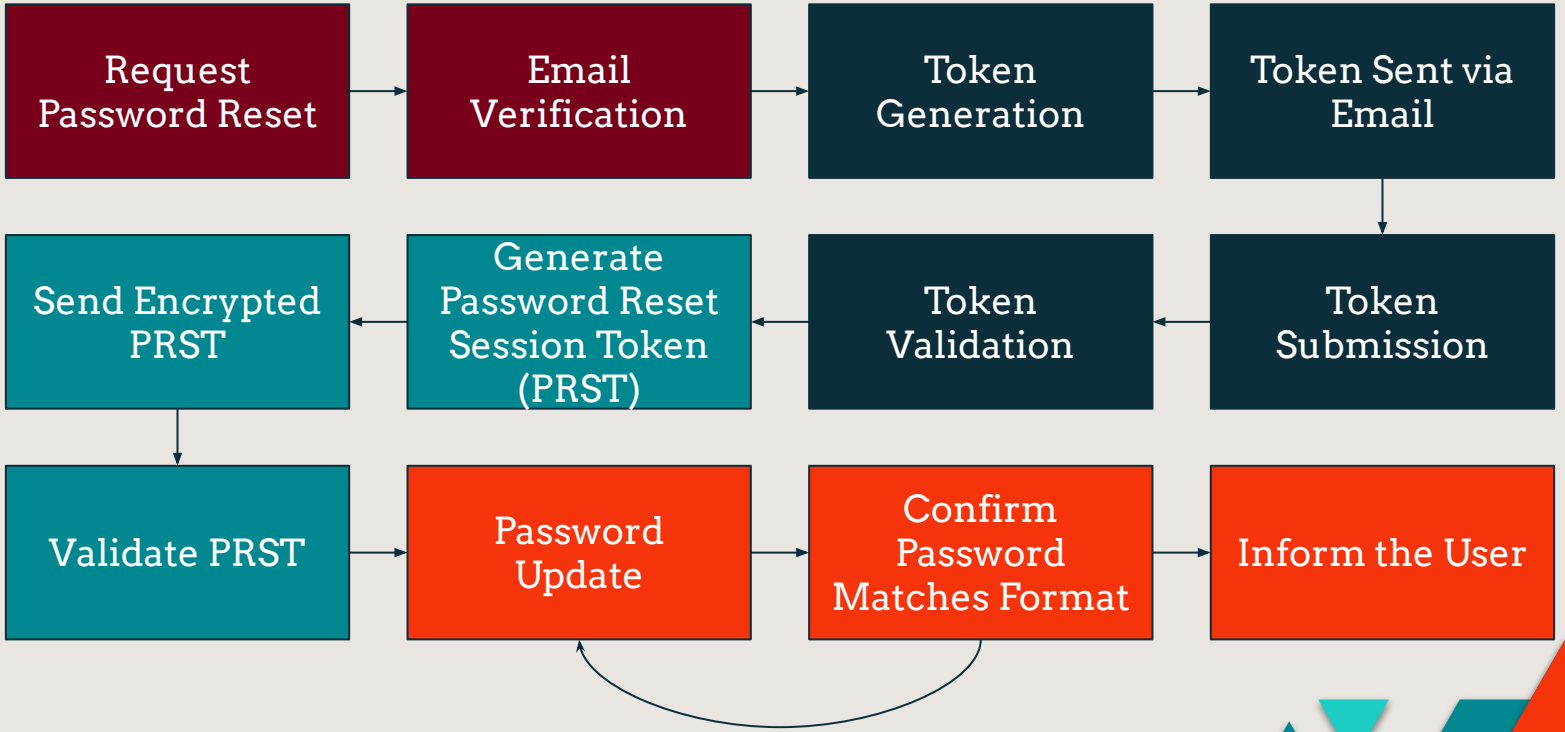
The Future of Screening Questions

- Revocation of access to a completed screening session after some time
  - Reasons: Urgency requires response and the dynamic nature of health issues
  - Editing window duration tailored to urgency level
- Administrative replacement and creation of:
  - Questions preceding or succeeding already existing questions
  - Options within already existing questions
  - Implemented using a date created and expiration date
- Completed screening sessions attached to broadcasted appointment or personally sent



Leo

Forgot Password Functionality (Backend)



Security Concerns

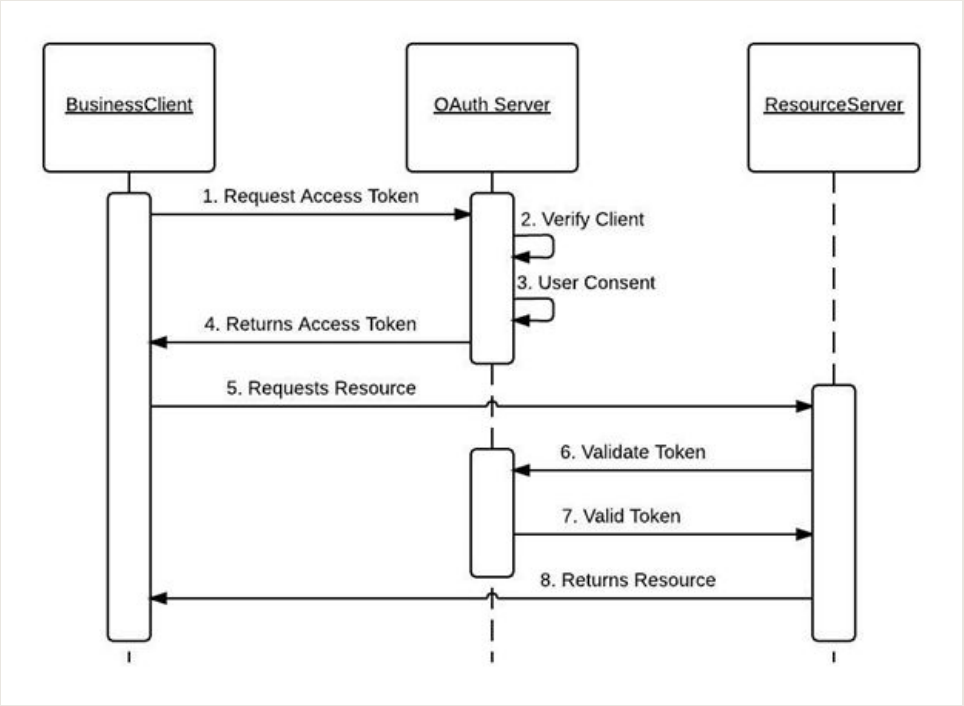
- Plaintext communication
    - http vs https (self-signed certificate)
    - Expo Bare Workflow vs Managed workflow (on hold)
- No credential checking
    - Data manipulation by malicious actors

```
// Delete specific user based on id
roy-kcj +1
@DeleteMapping(value = "/delete/{uid}")
public ResponseEntity<String> deleteUser(@PathVariable("uid") Long uid) {
    User user = userRepository.findById(uid).orElse( other: null);
```



Security Continued - OAuth

- OAuth 2.0 and JWT for server security



Authorization Server Components

```java
@Bean
@Order(1)
public SecurityFilterChain asSecurityFilterChain(HttpSecurity http) throws Exception {

    OAuth2AuthorizationServerConfiguration.applyDefaultSecurity(http);

    return http
            .getConfigurer(OAuth2AuthorizationServerConfigurer.class) OAuth2AuthorizationServerConfi...
            .tokenEndpoint(tokenEndpoint -> tokenEndpoint
                    .accessTokenRequestConverter(new JwtBearerGrantAuthenticationConverter())
                    .authenticationProvider(new JwtBearerGrantAuthenticationProvider(authorizationService(), tokenGenerator())))
            .oidc(withDefaults())
            .and() HttpSecurity
            .exceptionHandling(e -> e
                    .authenticationEntryPoint(new LoginUrlAuthenticationEntryPoint( loginFormUrl: "/login")))
            .oauth2ResourceServer(OAuth2ResourceServerConfigurer::jwt)
            .build();
}
```

Security Filter Chain - sets configurations and security rules for server, most importantly sets login form and the authentication provider and token generation settings

```java
@Bean
public AuthenticationProvider authenticationProvider(DataSource dataSource, PasswordEncoder passwordEncoder) {
    JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSource);

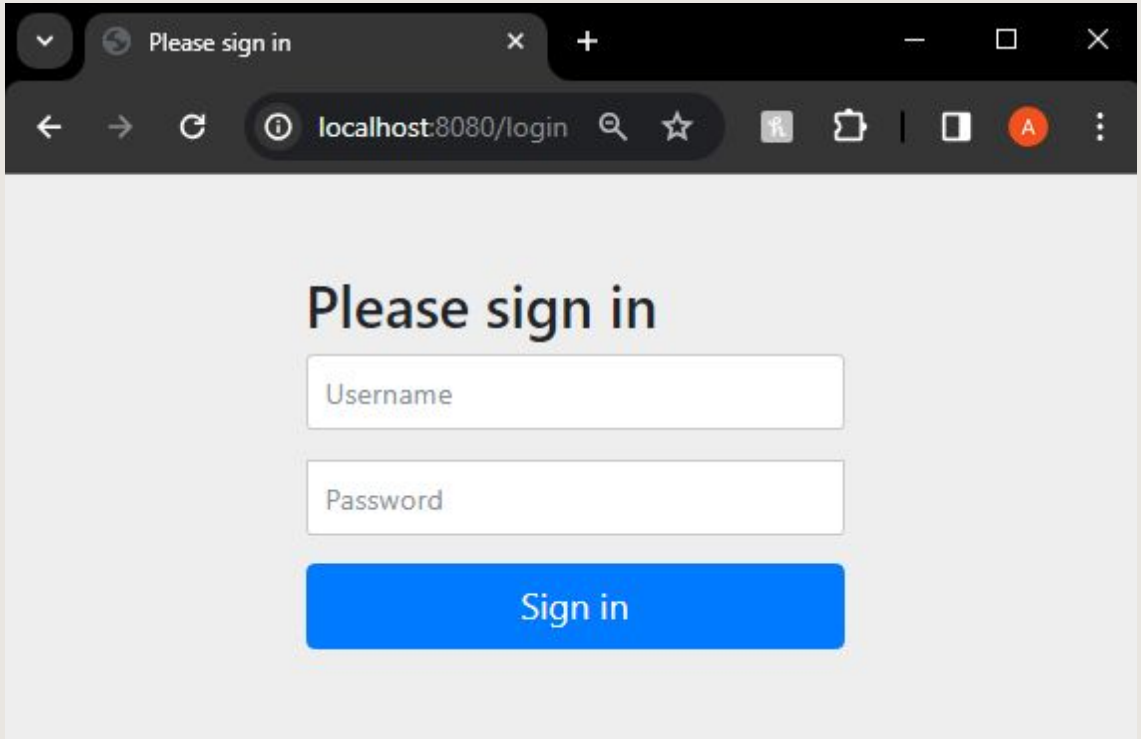
    Andy
    return new AuthenticationProvider() {
        Andy
        @Override
        public Authentication authenticate(Authentication authentication) throws AuthenticationException {
            String username = authentication.getName();
            String password = authentication.getCredentials().toString();

            String query = "SELECT password FROM users WHERE username = ?";
            try {
                String dbPassword = jdbcTemplate.queryForObject(query, new Object[]{username}, String.class);
                if (passwordEncoder.matches(password, dbPassword)) {
                    // Passwords match, return a fully authenticated Authentication object
                    return new UsernamePasswordAuthenticationToken(username, password, Collections.emptyList());
                } else {
                    // Passwords do not match, throw AuthenticationException
                    throw new BadCredentialsException("Invalid credentials");
                }
            } catch (EmptyResultDataAccessException e) {
                // User not found in the database
                throw new UsernameNotFoundException("User not found");
            }
        }
```

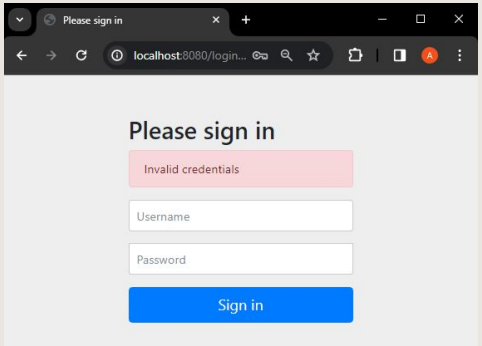
Authentication Provider - verifies credentials from user input

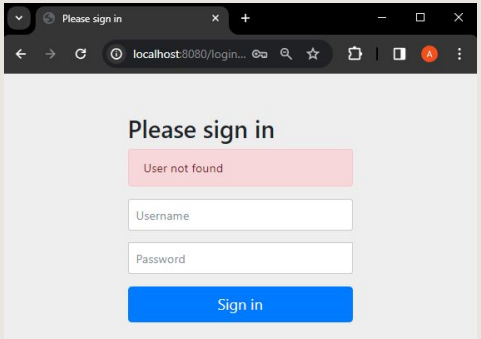


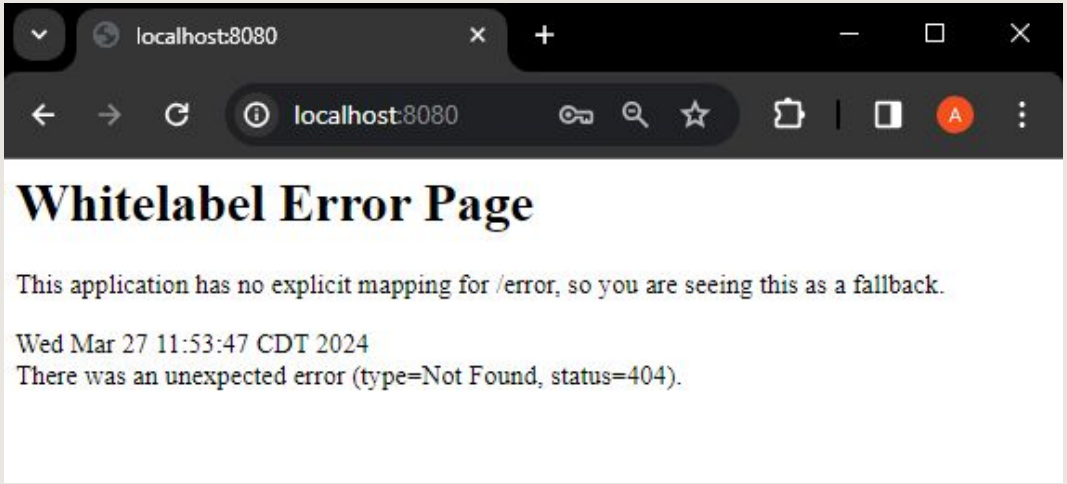
Login Form


Please sign in
localhost:8080/login
Please sign in
Username
Password
Sign in

Login Results





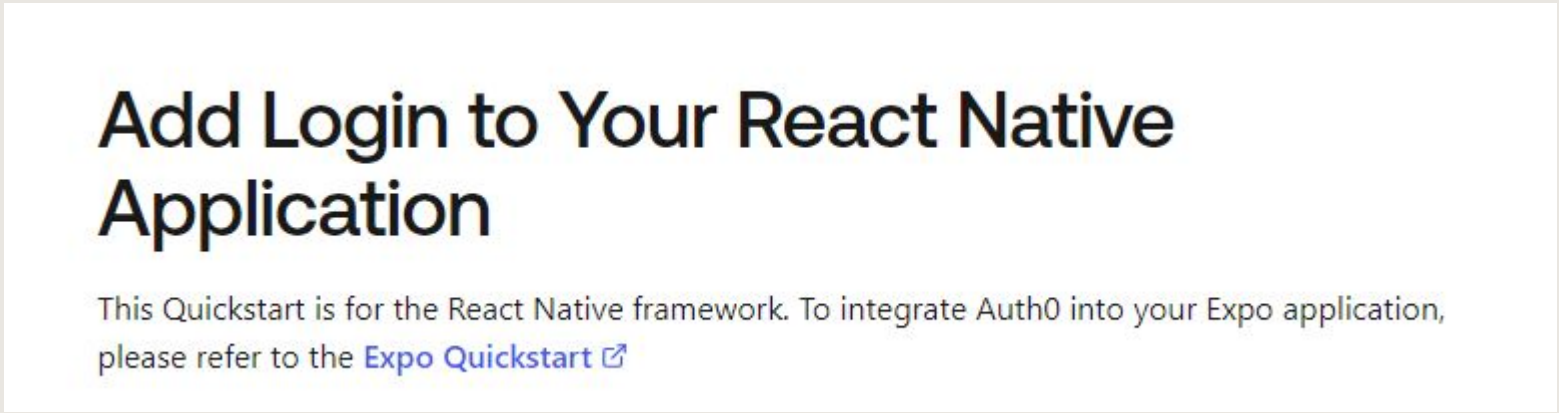


TODO: OAuth

- Integrate authorization server with resource server
    - Change existing Backend server to resource server
- Register client application with authorization server and migrate existing data to authorization database



TODO: OAuth Continued



Auth0 has several SDKs for various front-end frameworks. Expo GO is explicitly not supported, so we will be trying the React Native SDK, or separate Android/iOS/web SDKs (workflow)

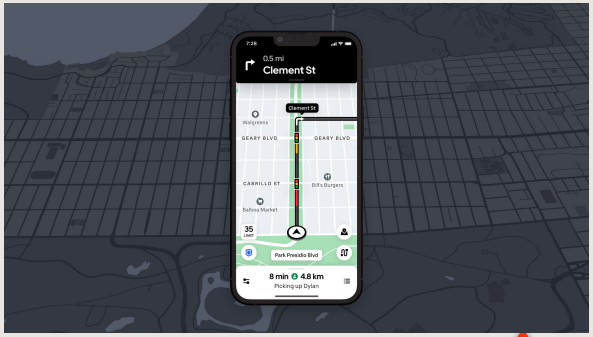


Broadcast Appointment Feature

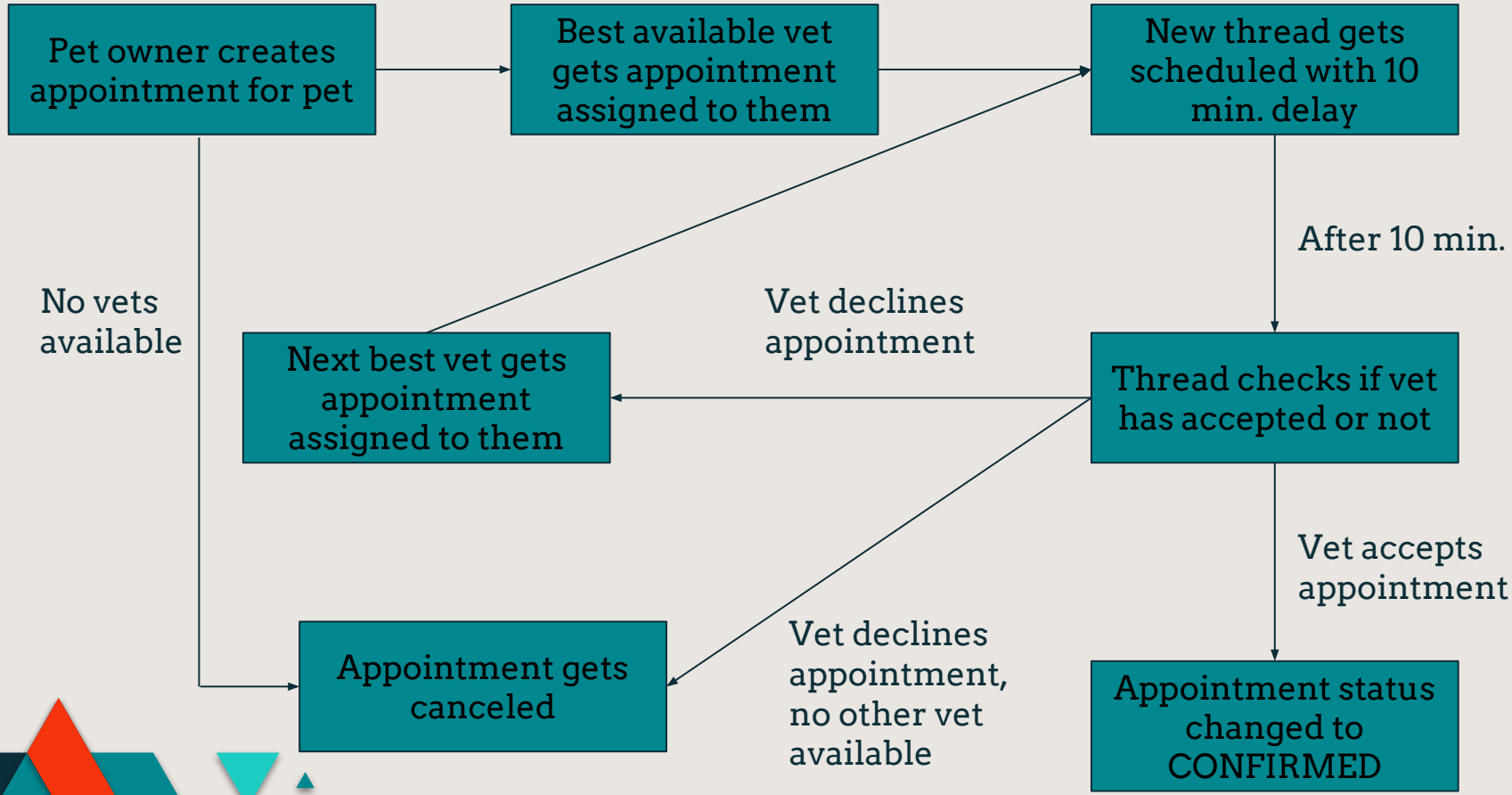
Elijah Brungardt

Broadcast Appointment Overview

- Based off Uber's algorithm
- Broadcast algorithm chooses best vet based on location
- Java thread is scheduled with 10 min. delay to check if vet has accepted appointment



Flowchart


Pet owner creates appointment for pet
Best available vet gets appointment assigned to them
New thread gets scheduled with 10 min. delay
After 10 min.
Thread checks if vet has accepted or not
Vet declines appointment
Next best vet gets appointment assigned to them
No vets available
Appointment gets canceled
Vet declines appointment, no other vet available
Vet accepts appointment
Appointment status changed to CONFIRMED

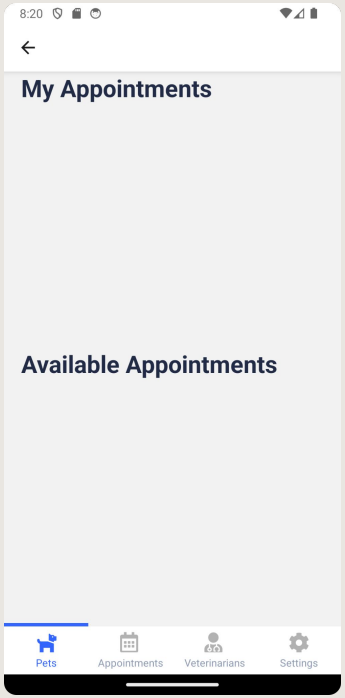
Appointment Backend

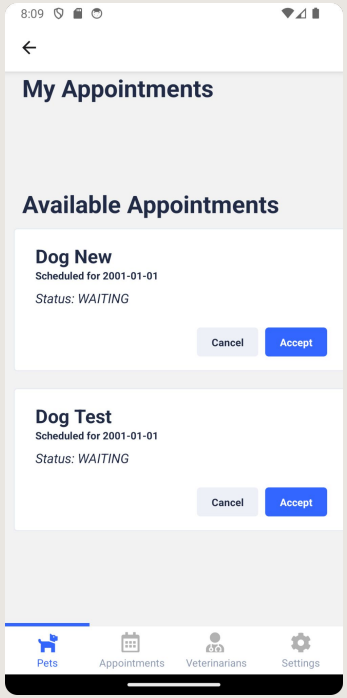
- Uses Java ScheduledExecutorService class
- Can schedule commands to run after a given delay, or to execute periodically.
- Allows for concurrency programming in Java
- Task determining if the vet has accepted or not is scheduled to execute after a certain amount of time

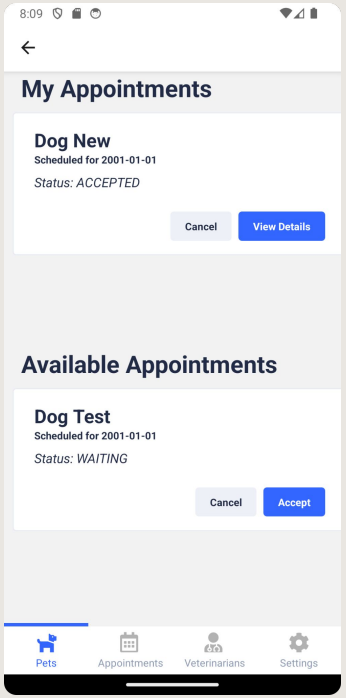
```
scheduler.schedule(() -> checkVetResponse(appointment, vets), TIMERDEADLINE, TimeUnit.MINUTES);
```

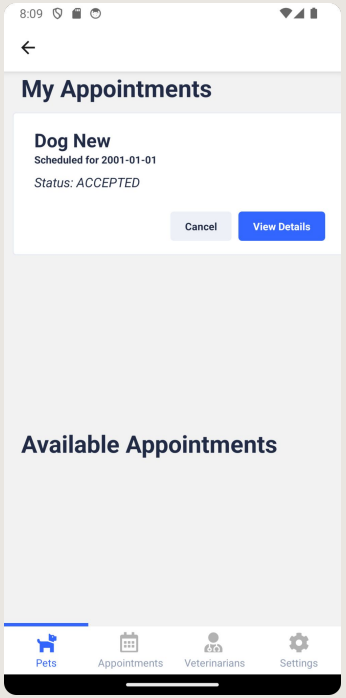


Appointment Frontend for Vet

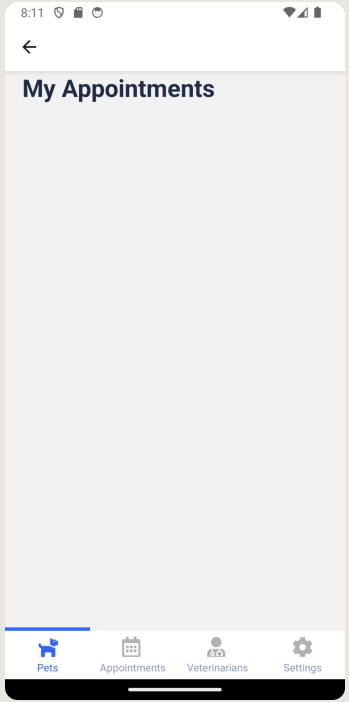


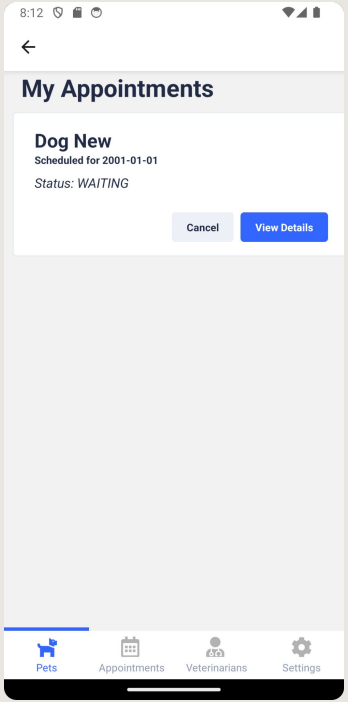


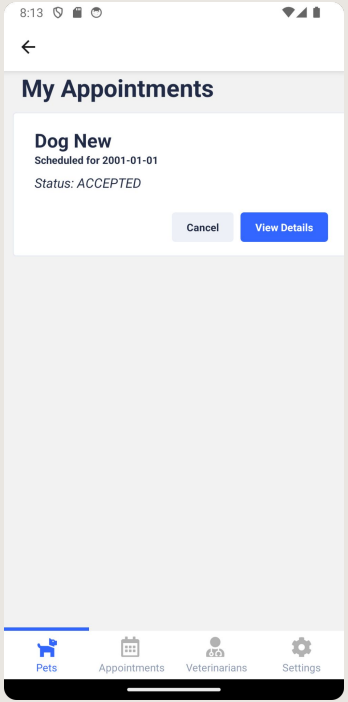




Appointment Frontend for Owner





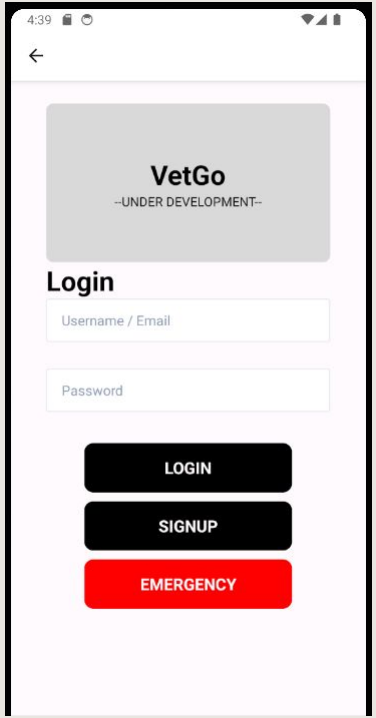


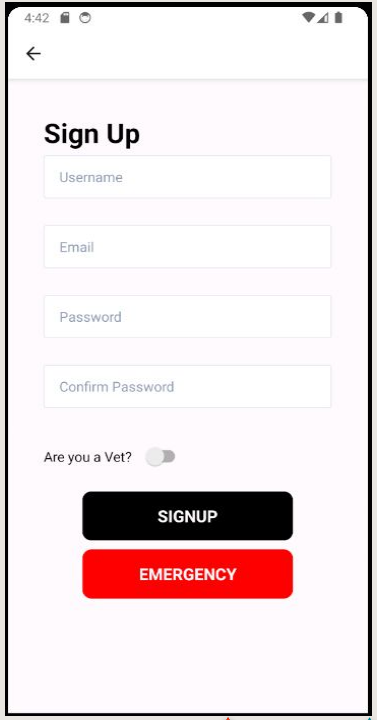
Front End Overhaul

David Marvin

Where the Project Started

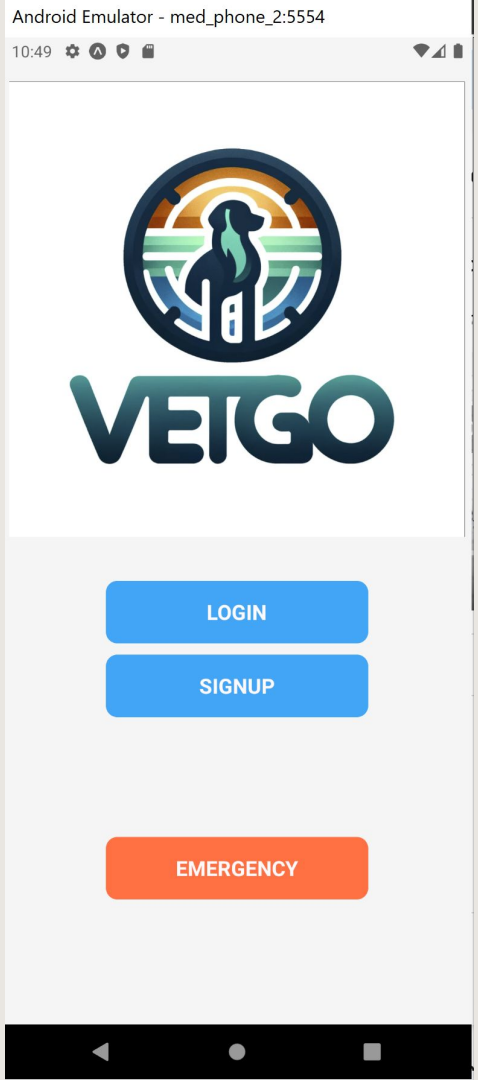
- Nearly all of the pages were inaccessible, inconsistent styles and formatting
- Most buttons did not have pages linked
- Existing pages were incomplete and not accessible





Front End Overhaul

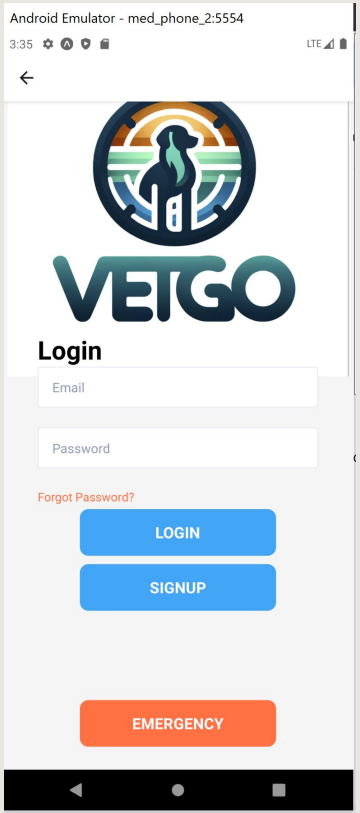
- Landing Page
  - Logo
  - Spacing
- Navigation Bar
  - Formatting
- Buttons
  - Functionality
  - Style
- Color scheme
- Readability

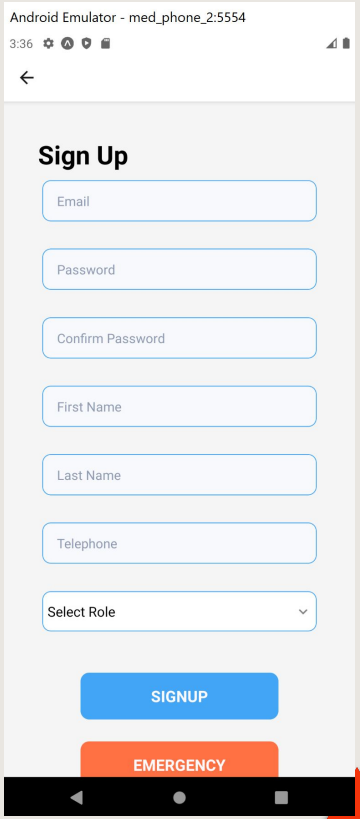


Login/Register

- Database changes
- Login
  - Sent a lot of extra information
  - Mishandled returning information
  - Easy to error
- Register
  - Did not ask for necessary information
  - Didn't fully work (would break if registered with certain information

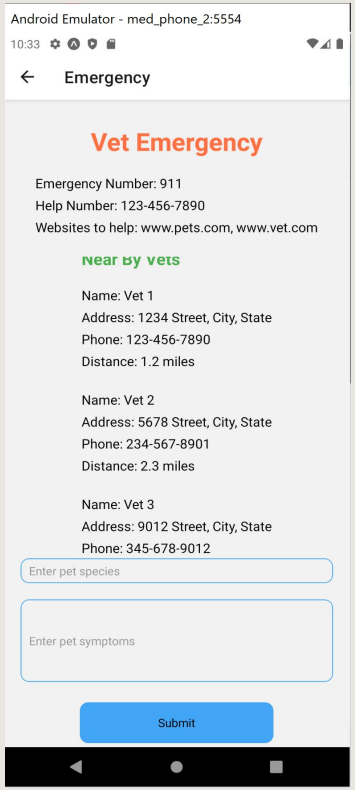


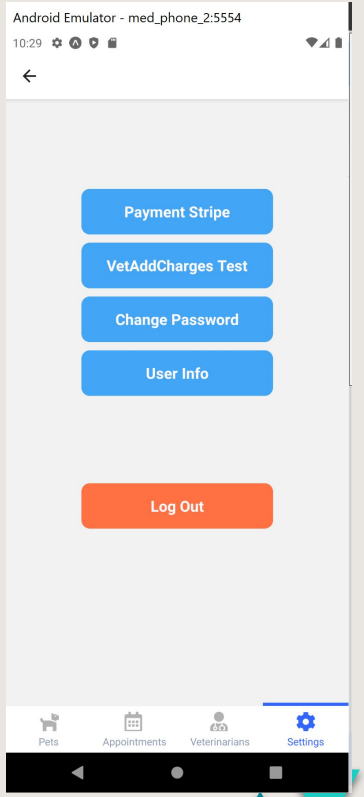


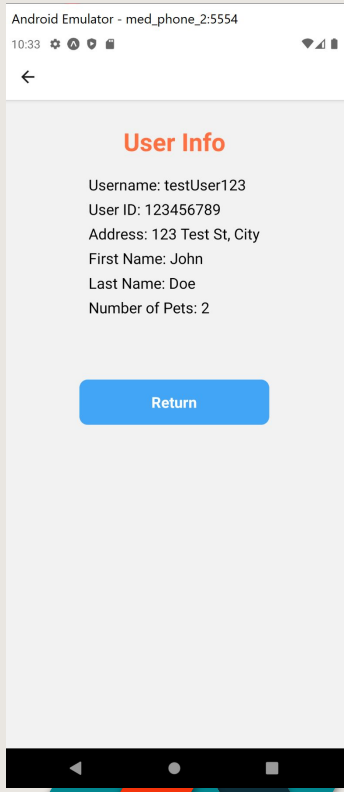


Emergency/ Settings/ User Info

- Emergency
  - Button was not clickable
- Settings Screen
  - Style
  - Connecting buttons
  - Functionality
- User Info
  - Return button
  - Display info



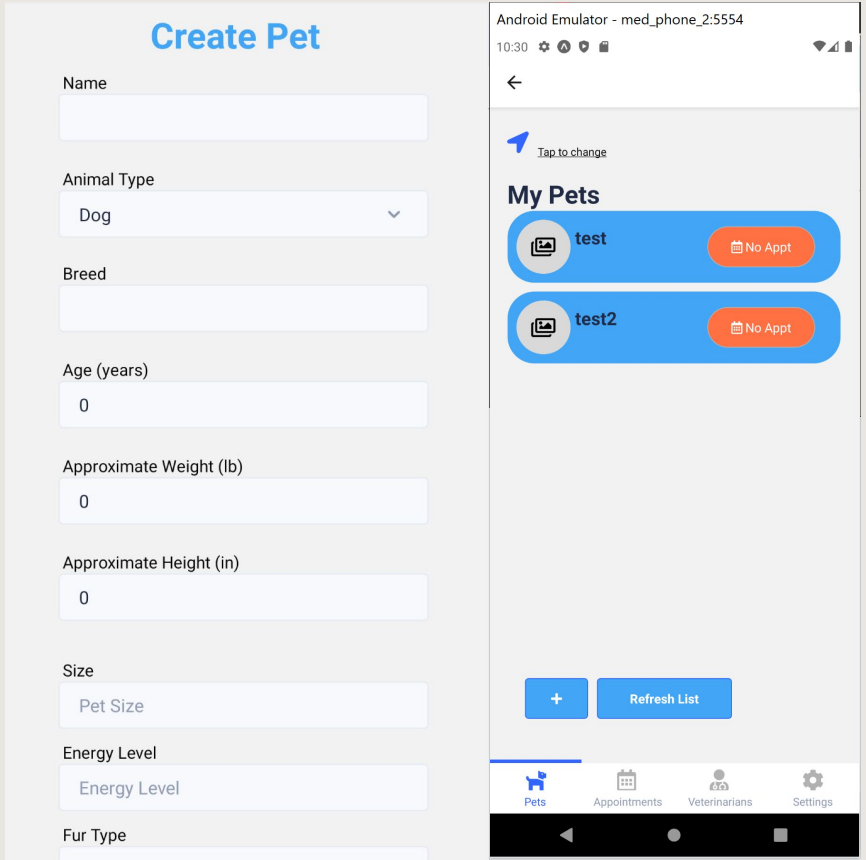




Create Pet/Misc Changes

- Create Pet
  - New database format
  - Formatting page
  - Changing requests
  - Editing logic
- Miscellaneous
  - Location Setup
  - Removal of redundant values





Demo of Screening Questions



Missing, Broken, or Incomplete Features

- Billing Information
- Screening Questions + Machine Learning
- Time of Arrival
- Release of Medical Records
- Calendar Integration
- Pet Images
- Geolocation of Users
- Speech2Text



What we plan to do next

David

- Location API
- View Appointments page
- Create Appointment

Elijah & Leo

- Vet Authentication
- Calendar Integration
- Pet Images

Andy

- Migrate user data
- Resource server setup
- Auth0 frontend framework



VetGo Demo 3

Group 10

VetGo Purpose Recap



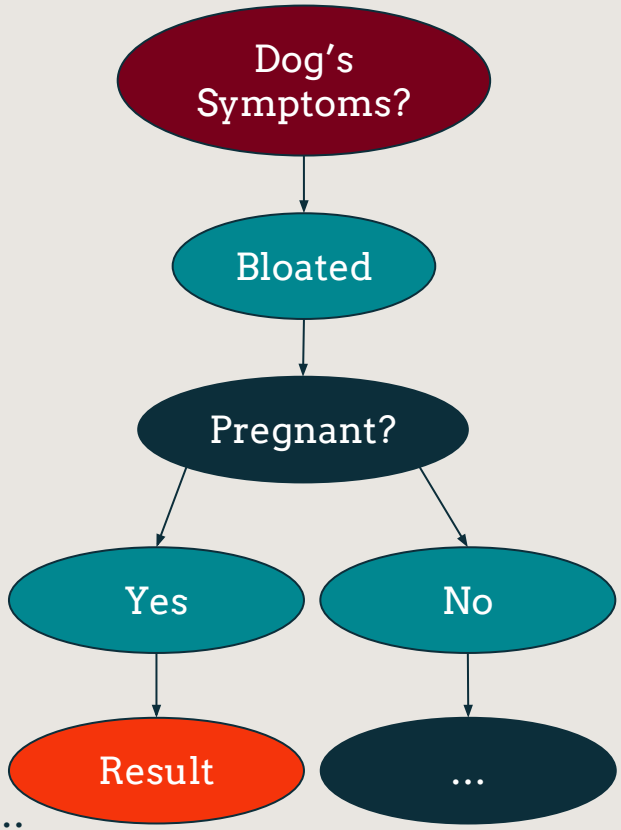
- Goal: Uber-like app for veterinarians and pet owners
  - Pet owner sequentially unicasts appointments to vets
  - Each vet can decide which appointments to take
- Emergency feature to provide information on nearby vet hospitals
- Screening questions to quickly assess a pet's condition and provide an urgency level
- Security through authentication and authorization protocols
- User-friendly interface



Leo

Previous Demos Recap

- Database Restructuring
  - Vets and Owners map to Users
  - Each now have relevant attributes
- NEW: Logical Restructuring (Client's Request)
  - Default role is Owner
  - Register and become authorized to be a Vet after initial registration
  - No need for Vet-Owner role
- Screening Questions
  - Screening session → Root screening question (depending on pet type) → Choose an option → New screening question → … → Result
- Forgot Password

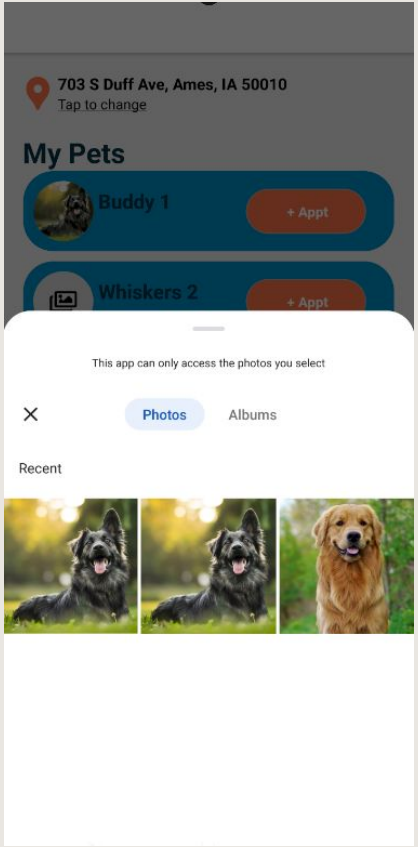


Leo

Pet Image Picking

- Originally saved as Binary Large Objects (BLOBS)
  - Slow performance and poor scalability
- Save the path of the image to storage as a URL
  - Send multipart data → Use as MultipartFile

    → Save the file to directory and image URL to database
    - Local storage for development
  - Simply accessed with {BASE_URL}/{imageURL}



Leo

Navigation Improvements

- Native stack navigator with bottom tab component

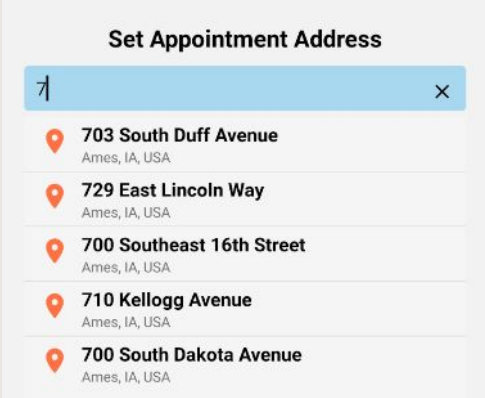
  → React Native bottom tab bar navigator

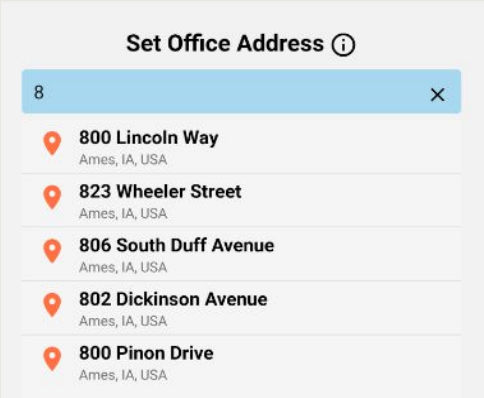
  - Component lifecycle handling
  - Logical flow
- Updated navigation across the app to reflect the restructuring
- Implemented global React Native context with user data



GeoLocation (1)

- Fixed Geolocation API usage
  - Optionally set location upon logging in
  - Owners set “Appointment Location”
  - Vets set “Office Location”

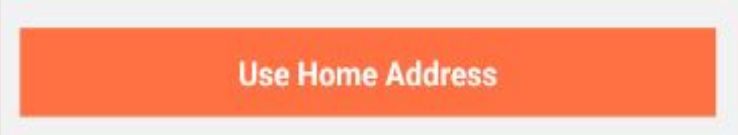




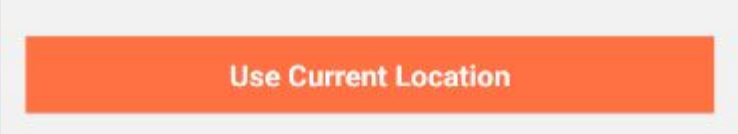
Leo

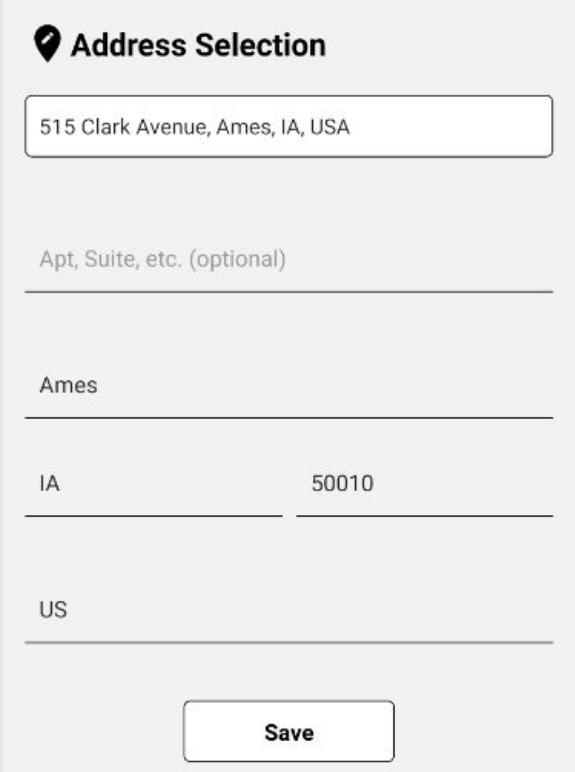
GeoLocation (2)

- Set address
  - Fields autocomplete
  - Owners can use for location to be set



  - Vets can use current location





Leo

Two Challenges

- Decision Rules Screening Questions
  - Set of if-then statements
  - Allow for additional inputs and flexibility though probability (e.g., probable diagnosis)
  - Difficult to compact the decision tree into rules
    - Need a subject matter expert (e.g., veterinarian)
- Diagnostic Machine Learning Model
  - Deterministic decision tree ≠ robust dataset
    - Severe overfitting
  - Model usage (e.g., LLM)
  - No non-proprietary dataset



Leo

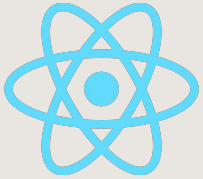
Software Development Practices and Tools

Speaker: Elijah Brungardt

Frontend Tools

React Native

Cross-platform frontend framework





TypeScript

Adds static typing to Javascript

Expo

Builds Native IOS and Android Apps from a Single React Codebase





Elijah

Backend Tools



Spring Boot

Application framework based on Java



MySQL Server

Relational Database Management System that uses SQL



Stripe

API for handling payments



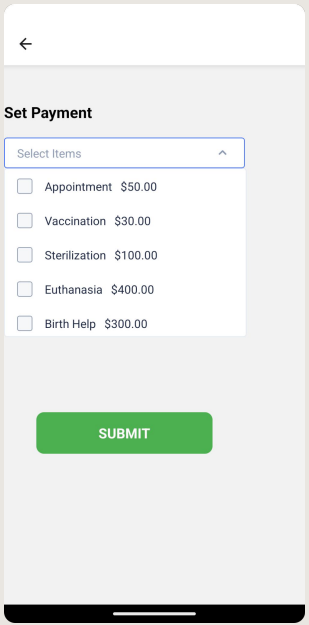
Google Maps API

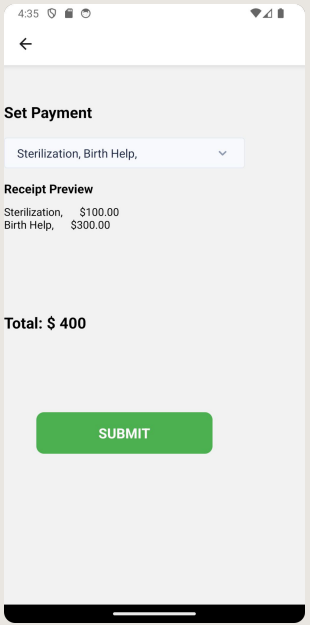
API for handling maps and locations



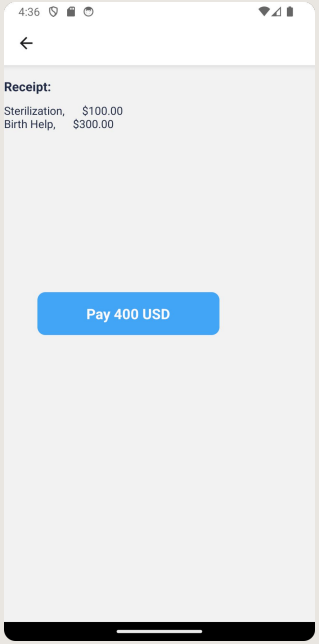
Elijah

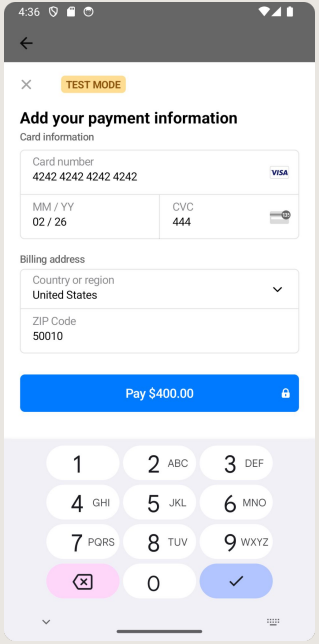
Payment Screens





Request Payment Screen for Vet

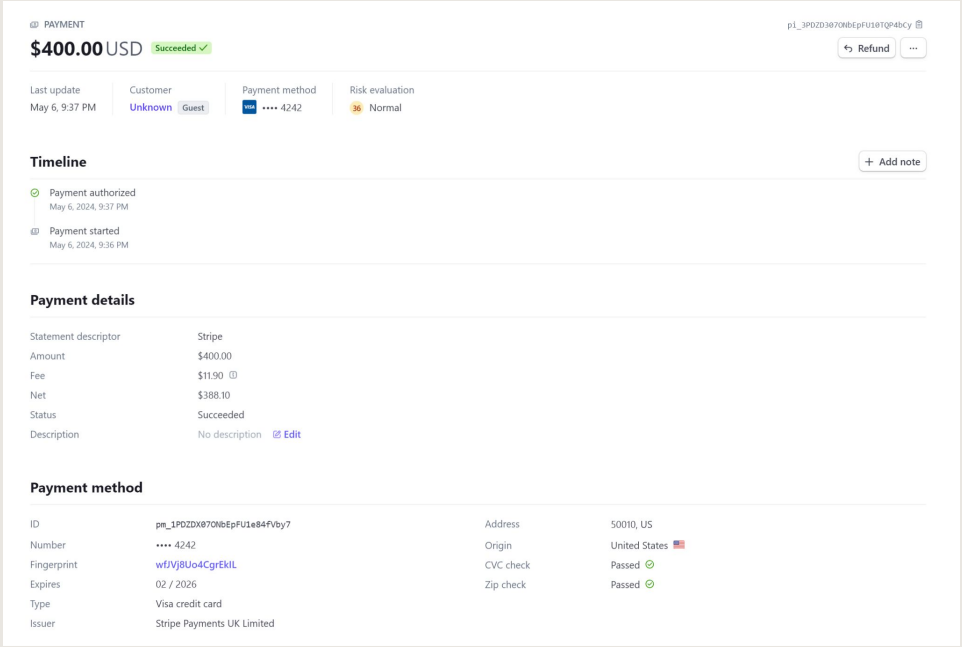




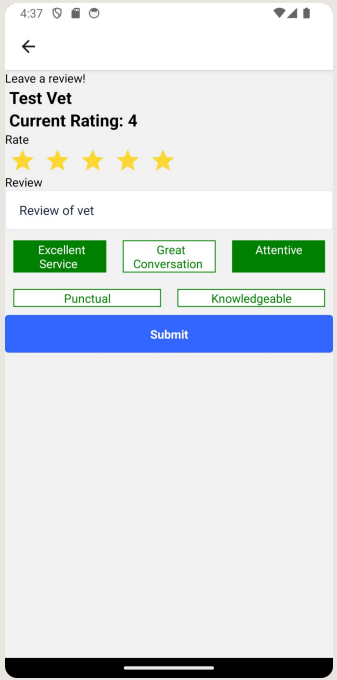
Send Payment Screen for Pet Owner

Elijah

Payment shows up on Stripe Dashboard



Pet Owner can add review for vet afterwards



Elijah

Project Growth and Development

Speaker: David Marvin

What we as a team feel proud of

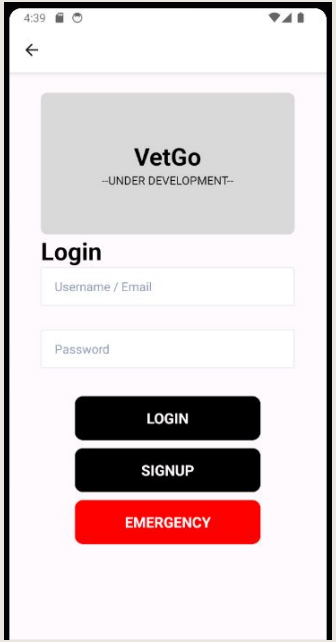
- Picking up the project from where the other team started
- Creation of a robust application
- UI is easy to use for the large user demographics
  - Easy access all screens
  - Information is displayed in a very easy yet descriptive way way
- Project was made with scalability in mind
  - Database
  - Frontend
- Error checking

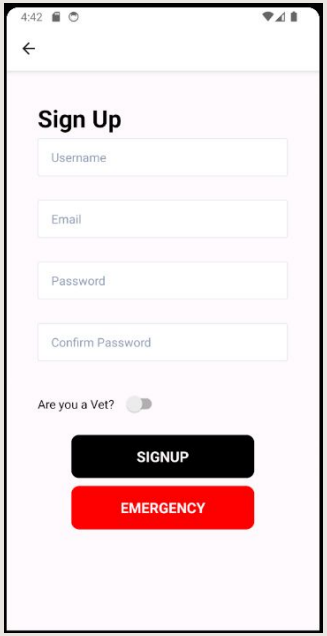


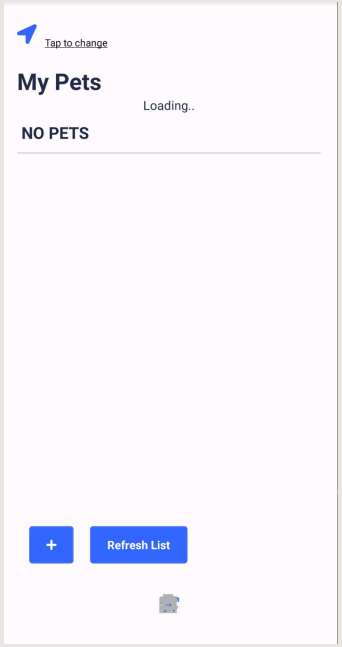


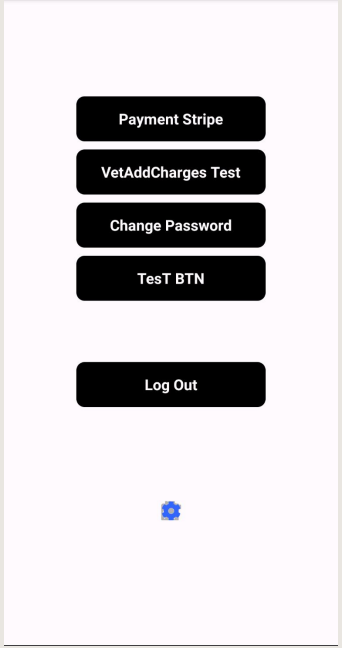
David

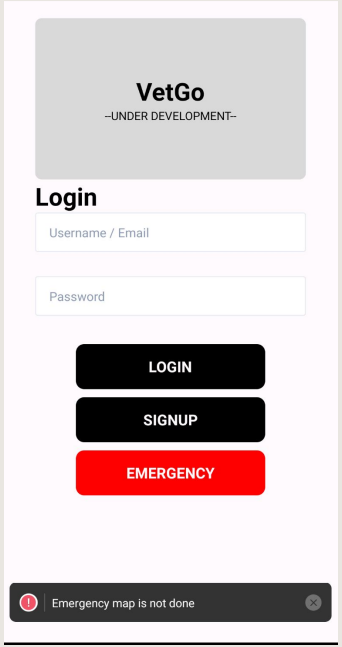
How the project has grown





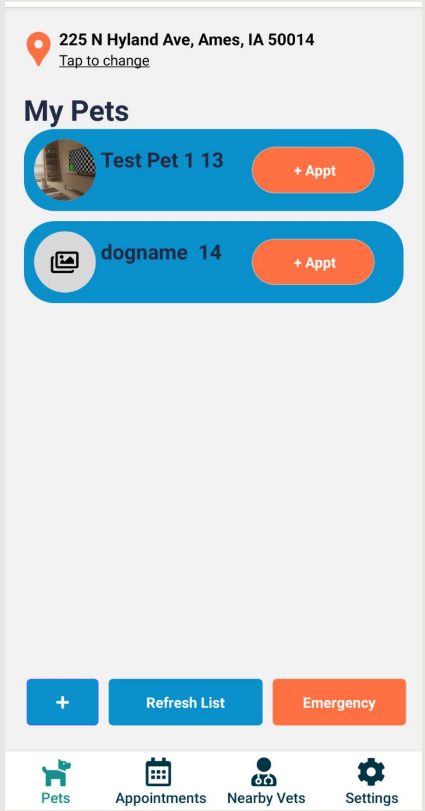


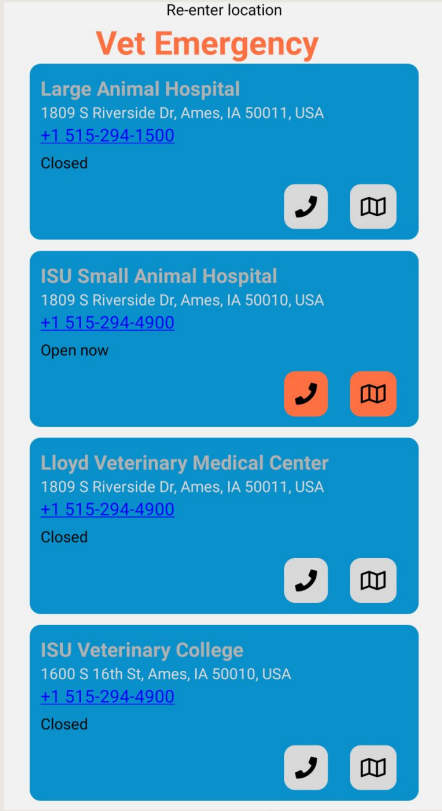




VetGo Emergency Screen

- Ease of Use
  - Emergency button is 2 press away
- Uses your location and queries google maps API
  - Location
  - Hours
  - Phone Number
    - Calling on demand
    - Routs you to maps

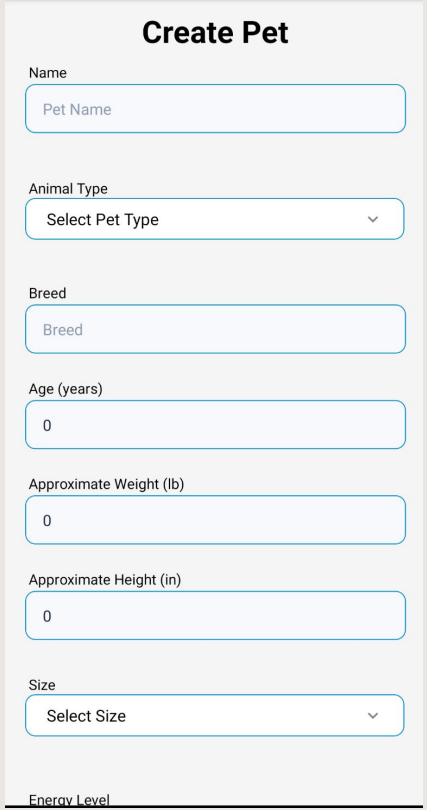


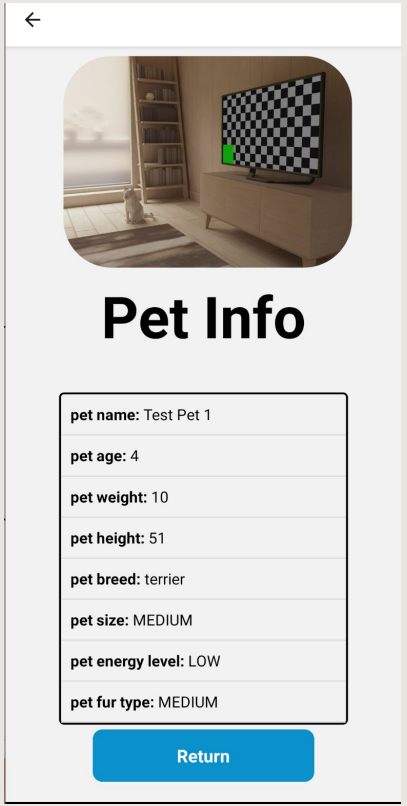


David

Create Pet Updates

- Extremely important page to be kept simple
- Displaying errors
- UI was not the most readable
- Challenging to create a UI that gets a ton of information without being overwhelming

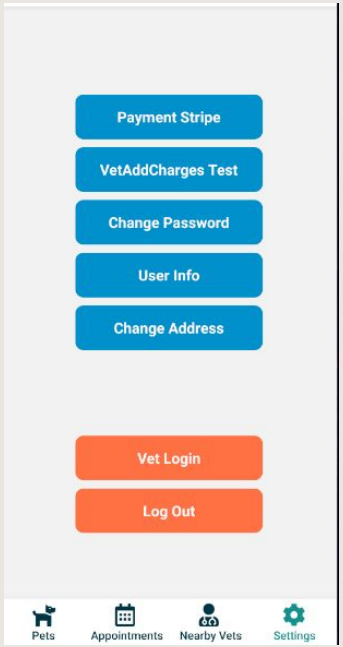


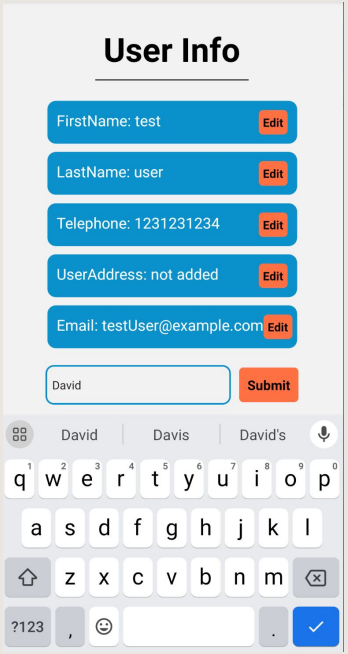


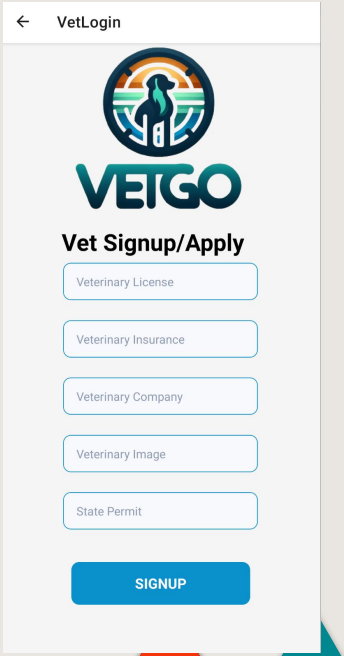
David

VetGo User Info / Vet Reseristration

- Easy updates to user info
- Ability to apply to be a vet
  - Similar to uber UI
- Challenging to find what information is required to sign up

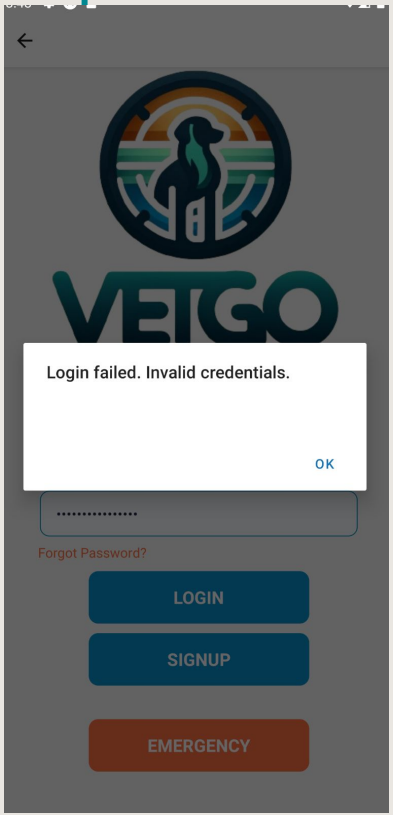


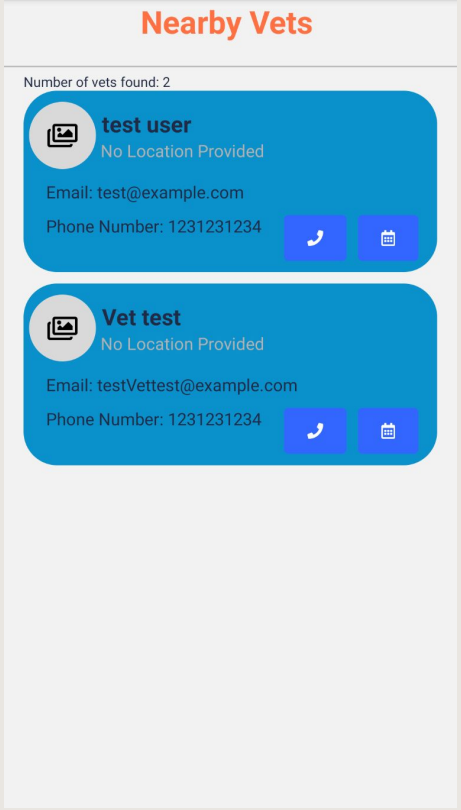


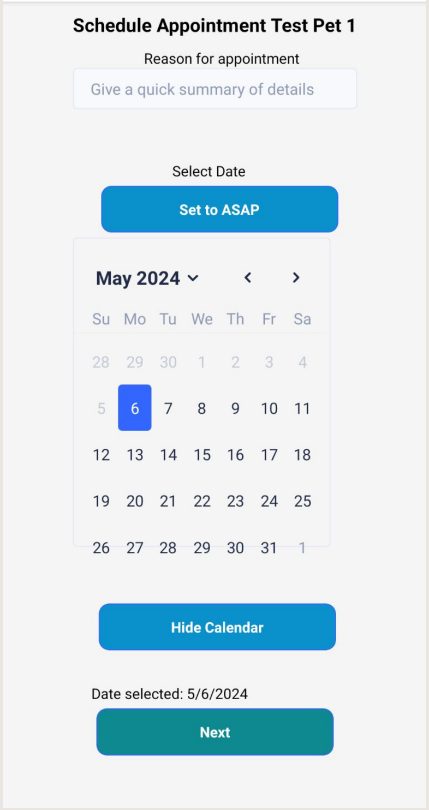


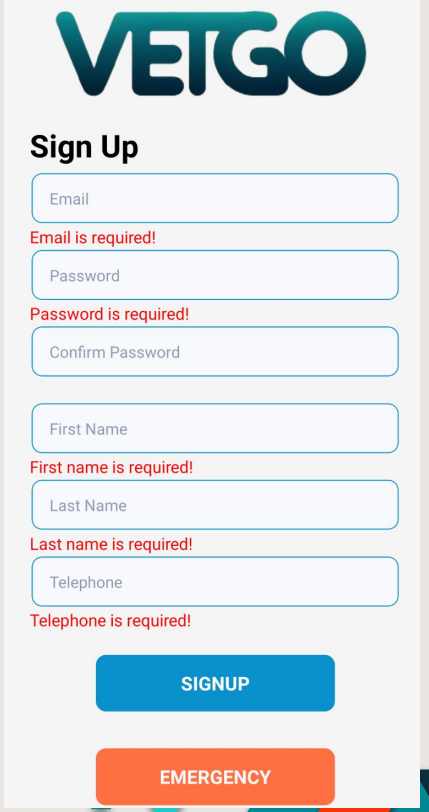
David

VetGo UI Ease of use



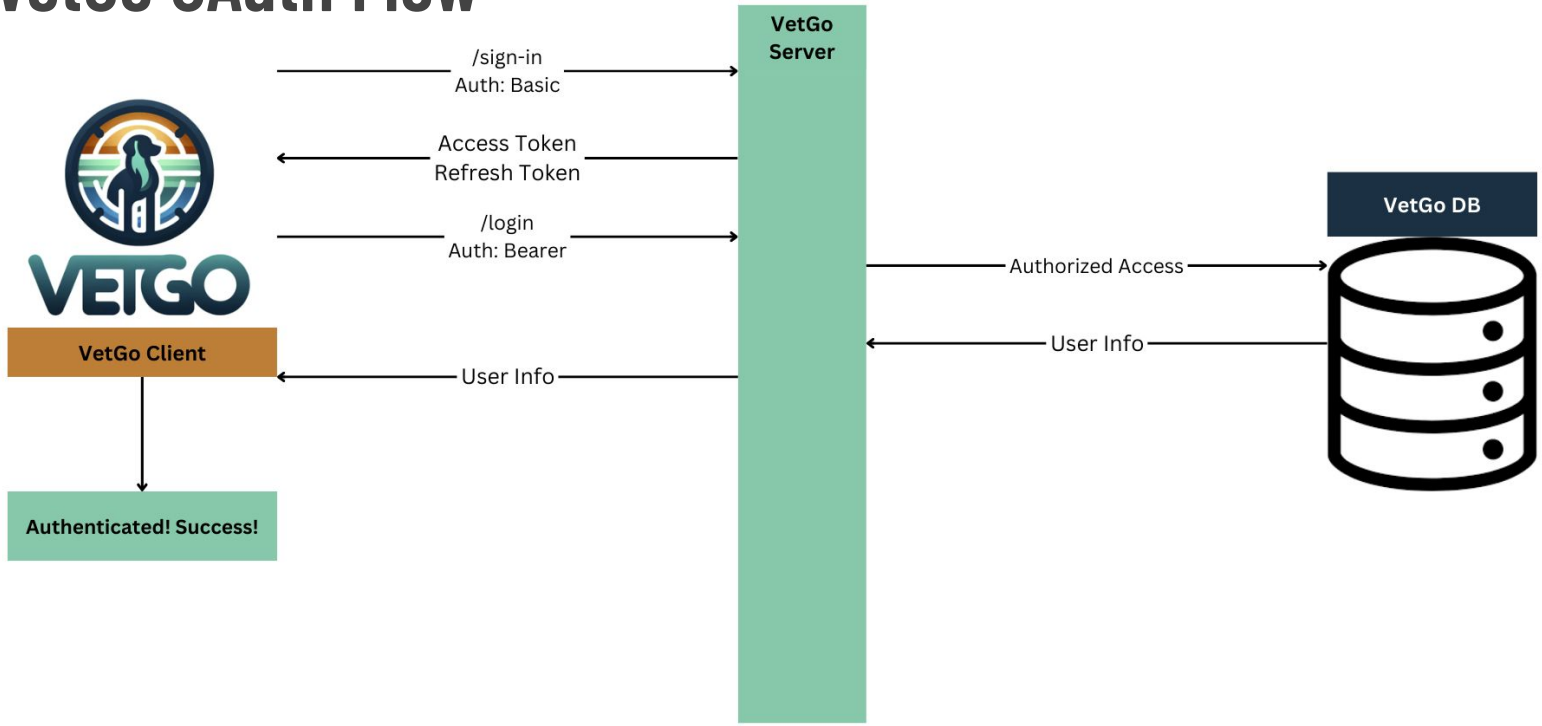






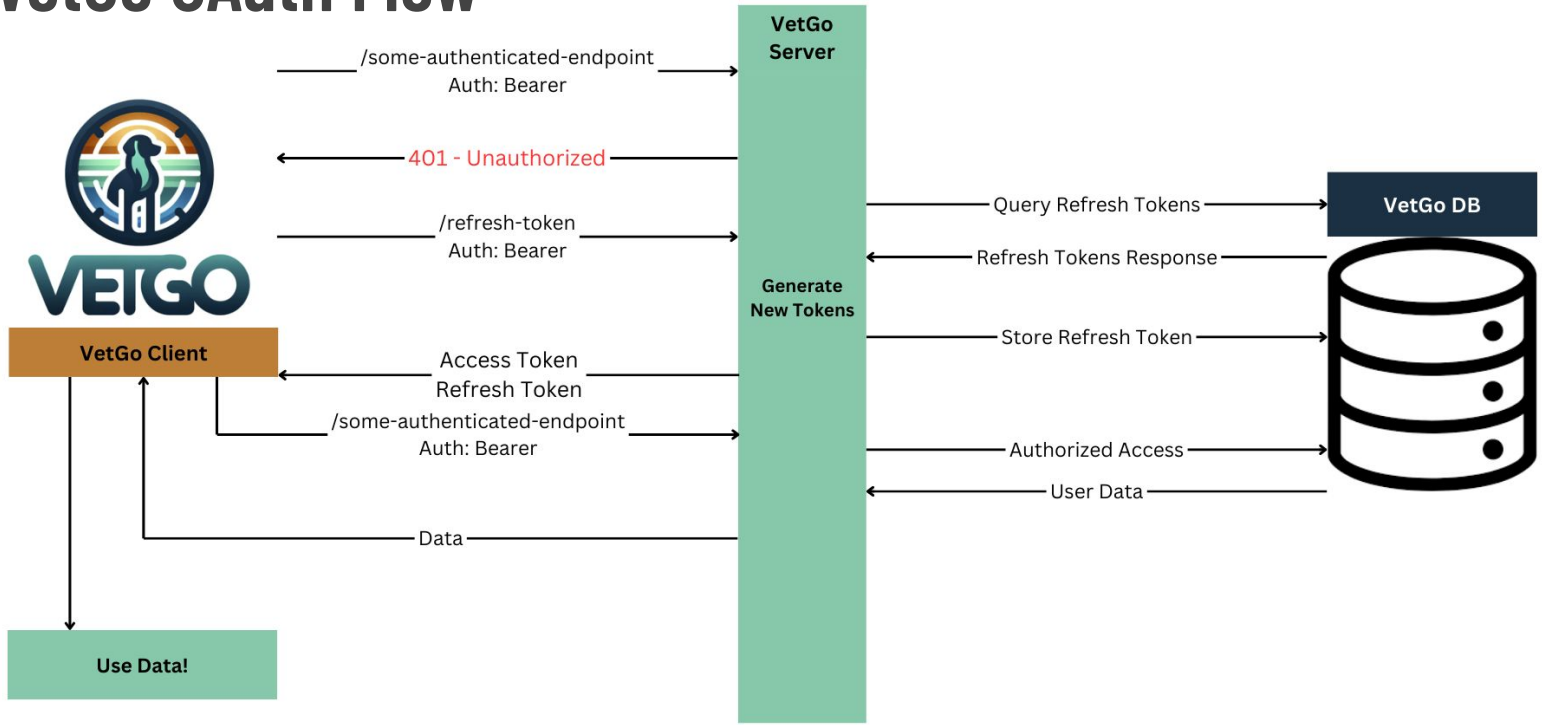
David

VetGo OAuth Flow



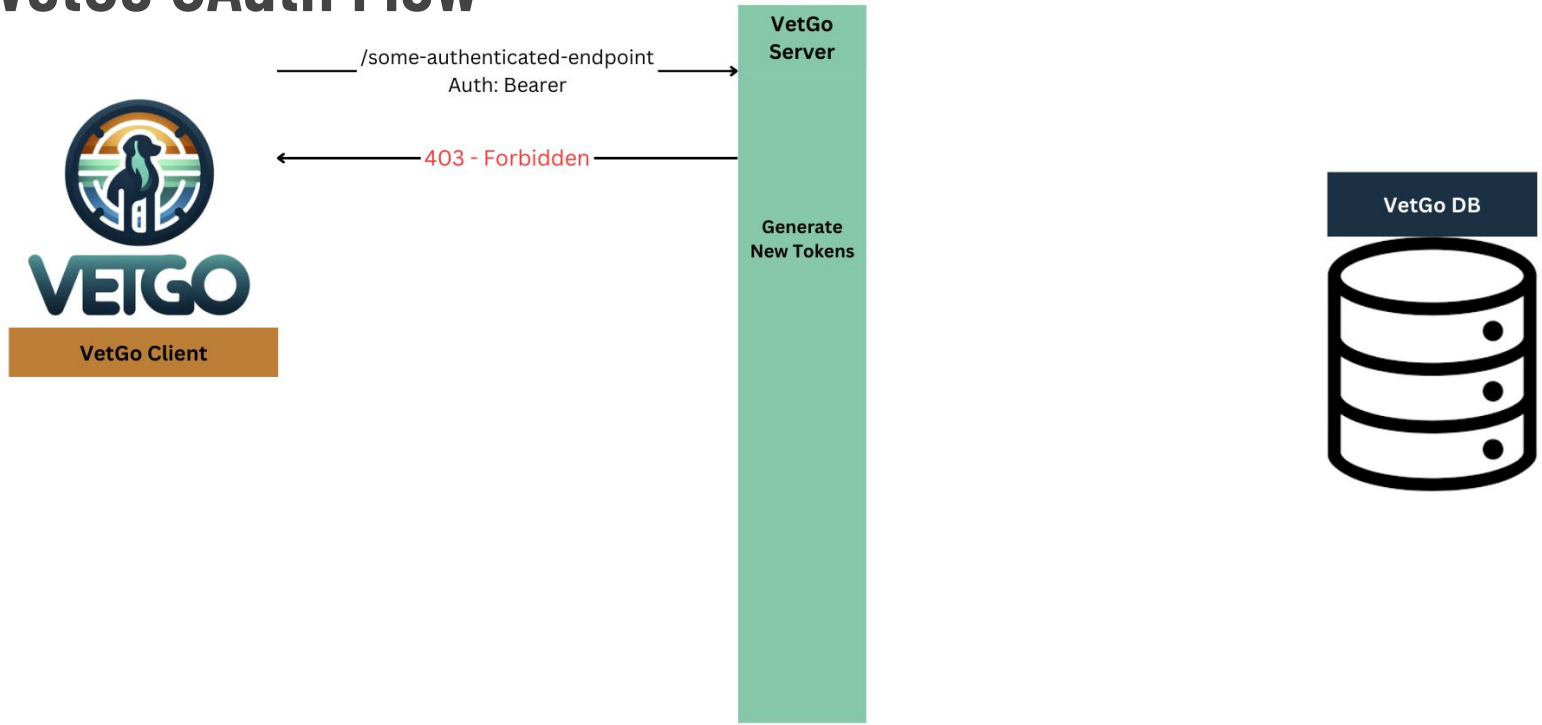
Andy

VetGo OAuth Flow



Andy

VetGo OAuth Flow



Andy

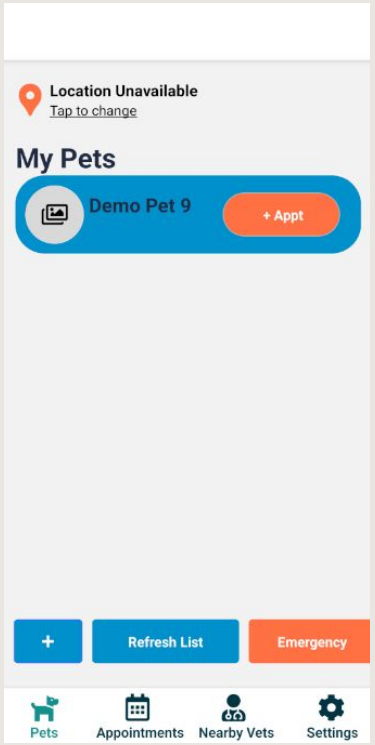
Notes On OAuth

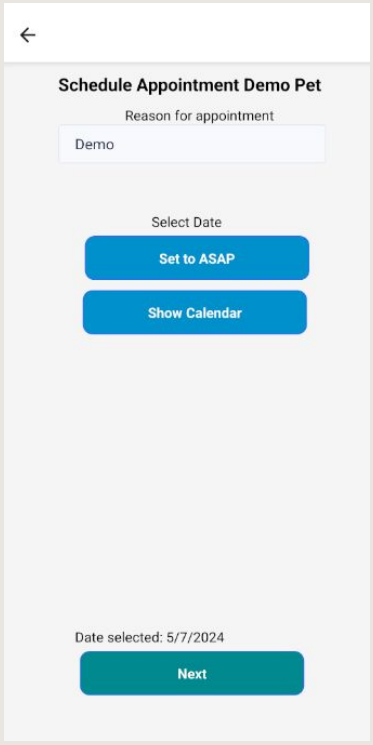
- Expo Secure Store
- Prevents unauthorized access, not sniffing
- Application scale & refresh tokens

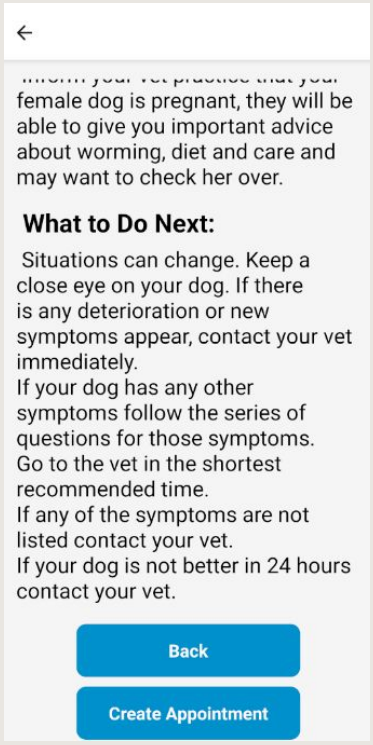


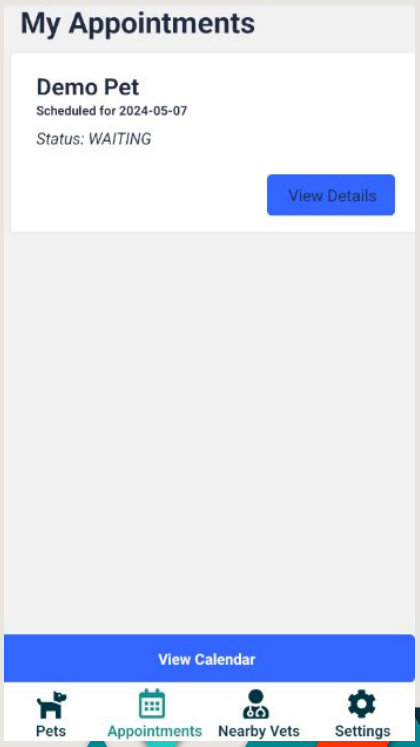
Andy

Calendar

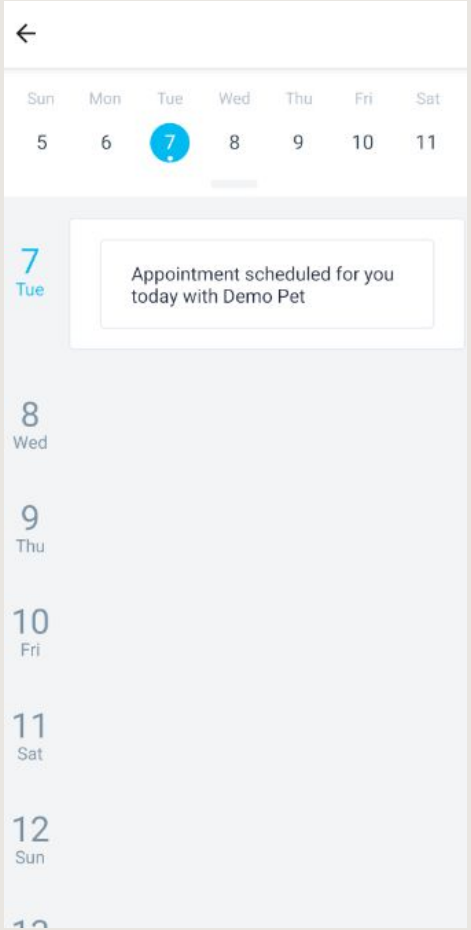

Location Unavailable
Tap to change
My Pets
Demo Pet 9
+ Appt
+
Refresh List
Emergency
Pets
Appointments
Nearby Vets
Settings


Schedule Appointment Demo Pet
Reason for appointment
Demo
Select Date
Set to ASAP
Show Calendar
Date selected: 5/7/2024
Next


female dog is pregnant, they will be able to give you important advice about worming, diet and care and may want to check her over.
What to Do Next:
Situations can change. Keep a close eye on your dog. If there is any deterioration or new symptoms appear, contact your vet immediately.
If your dog has any other symptoms follow the series of questions for those symptoms.
Go to the vet in the shortest recommended time.
If any of the symptoms are not listed contact your vet.
If your dog is not better in 24 hours contact your vet.
Back
Create Appointment


My Appointments
Demo Pet
Scheduled for 2024-05-07
Status: WAITING
View Details
View Calendar
Pets
Appointments
Nearby Vets
Settings

Calendar View



Calendar Future

- Integrate with Google and Outlook Calendar APIs
- Expand functionality



Challenges

- Learning React Native
- Figuring Out Screen Flow of App
- Making the app to be easy to use for users
- Security and Reliability
- OAuth integration
- Expo compatibilities



Andy

Contributions

- Andy: Security OAuth - Access Tokens, Refresh Tokens, Grant User Roles, HTTPS, CI/CD Setup, Server Setup, Calendar
- David: Login, Signup, Emergency, Create Pets, Create Appointment, User Info, pet Info, Landing Page, NearBy Vets, register as a vet, Styling, Settings Page
- Elijah: Broadcast Appointment Algorithm, MyAppointments Screen, Payment Screens, Review Screen, Stripe Setup
- Josh: Screening Questions Screens
- Leo: Backend Screening Questions, Backend Nearby Vets, Database Restructuring, Navigation Restructuring, Backend Forgot Password, Pet Image Picker Fixes, and GeoLocation



Demonstration

- https://drive.google.com/file/d/17PBEd_ojE8fHElYQdydWBhoE6TSg_iAL/view?usp=drive_link

