Software Requirements Specification for VetCare

Version 1.04

Christian Nieves, Seanghai Heng, Keenan Phillips, William Dash, Heethasha Sandeep Kumar, Paul Johny Mampily
RMIT University

10/9/24

Table of Contents

Revision History

Name	Date	Reason for Changes	Version
Keenan Phillips	16/09/24	• Adjusted dependencies	1.01
Keenan Phillips	20/09/24	• Removed "Leaving feedback and reviews" from project scope • Removed functional requirement of system sending confirmation emails upon booking an appointment and registering an account • Removed functional requirement of searching for veterinary services • Adjusted functional requirements for searching	1.02
Keenan Phillips	27/09/24	• Removed notification assumption in 6.1 • Removed leaving feedback/review assumption in 6.1	1.03
Keenan Phillips	09/10/24	• Removed text about integration with local clinics and pet stores (out of scope for this release) from 1.1 • Removed sharing records with other veterinary professionals from 1.2 • Removed comprehensive library of articles from 1.2 • Removed selection of preferred veterinarian from 2.1 • Added functional requirement in 2.1 • Removed "stay informed with the latest trends" in section 2 • Re-wrote section 4 • Added Milestone 3 documentation • Added 'Database' section to section 4	1.04

1 Introduction

1.1 Overview and Purpose

For pet owners, sometimes it isn't easy to find a vet clinic when they are sick or cannot find valuable information regarding their behaviour and welfare. For these reasons, The VetCare project aims to provide a convenient way for pet owners to ensure their pets receive pet care services with just a few clicks of a button. The VetCare project is an innovative online web application that utilises modern technologies to meet the needs of pet owners.

1.2 Scope

The primary purpose of VetCare is to deliver an accessible solution for managing various aspects of pet care. By making use of cutting-edge technology, the VetCare aims to:

- Allow pet owners to easily book, reschedule, or cancel appointments, view and manage medical records, and request prescription refills with minimal effort.
- Offer secure and convenient access to detailed medical histories, vaccination records, and treatment plans.
- Allow users to export their pet's medical records as well as their appointment history.
- Allow pet owners to request their prescriptions with delivery options.
- Provide users with a user-friendly platform that provides a smooth user experience.

2 Functional Requirements

2.1 Booking Appointments

- The system shall allow the user to book an appointment.
- The system must provide an interface where pet owners can select a date and time for their appointment.
- The system shall allow users to view upcoming appointments.
- The system must allow users to reschedule or cancel appointments.
- The system must allow users to book an appointment within operating hours (8 am – 7 pm).

2.2 Account Creation

- The system shall allow the user to create an account using an email.
- The system must provide a registration form where users can enter their details.
- The system must check if the account already exists.
- The system must verify the password and confirmation password before proceeding to account creation.

2.3 Update Pet's Medical Data

- The system must allow veterinarians to access the pet's medical information.
- The system must allow veterinarians to update pet medical data such as vaccination records and medical history.
- The veterinarian must be able to confirm or cancel the changes.
- The system must display a status message when updating the pet's information (e.g., "Update successful").
- The system must log all changes made to a pet's medical data for audit purposes.

2.4 Payments

- The system must integrate with a secure payment gateway (Stripe).
- The system must support various payment methods, including credit/debit cards.

- The system must send a confirmation message when payment is received.
- The system must send an error message when the payment fails.
- The system must allow users to save payment methods securely for future use.
- The system must provide an option for users for request refunds and display the status of refund requests.

2.5 Search and Filters

- The system must allow users to search for educational and latest-trend articles.
- The system must provide sorting options to allow users to order search results by date.
- The system must provide relevant search results even if no exact matches are found (e.g., fuzzy matching).
- The system must handle cases where no search results are found and provide suggestions.

2.6 Updating Personal Details

- The system must ensure that only authenticated users can access and update their details.
- The system must provide an interface where users can view and update their details.
- The system must validate the input for each field (e.g., email format, phone number format).
- The system must allow the user to delete their account.
- The system must log all updates to personal details, including the date and time of changes.
- The system must enforce a cooldown period between account deletions and re-registration.

3 Non-Functional Requirements

3.1. Usability Requirements

- The system interface must be intuitive and follow standard web design conventions.
- The system must provide help documentation or tooltips to assist users with complex tasks.
- The system must allow new users to complete key tasks (e.g., booking an appointment) within 5 minutes of first use.

3.2. Performance Requirements

- The system must load the homepage within 3 seconds under normal conditions.
- The system must be able to handle at least 300 concurrent users without significant degradation in performance.
- The system must aim for a query response times of under 500 milliseconds to ensure quick retrieval of data.
- The system must remain responsive during a high concurrent user count, with no more than a 5% increase in page load times.

3.3. Security Requirements

- The system must require users to authenticate using a secure login mechanism, including email and password, with two-factor authentication as an option.

3.4 Compatibility Requirements

- The system must be compatible with major web browsers, including Chrome, Firefox, Safari, and Edge.
- The system must display correctly on a most desktop and laptop screen sizes with a 16:9 aspect ratio.
- The system must support a range of operating systems, including Windows, macOS, Linux, iOS, and Android.

3.5 Reliability Requirements

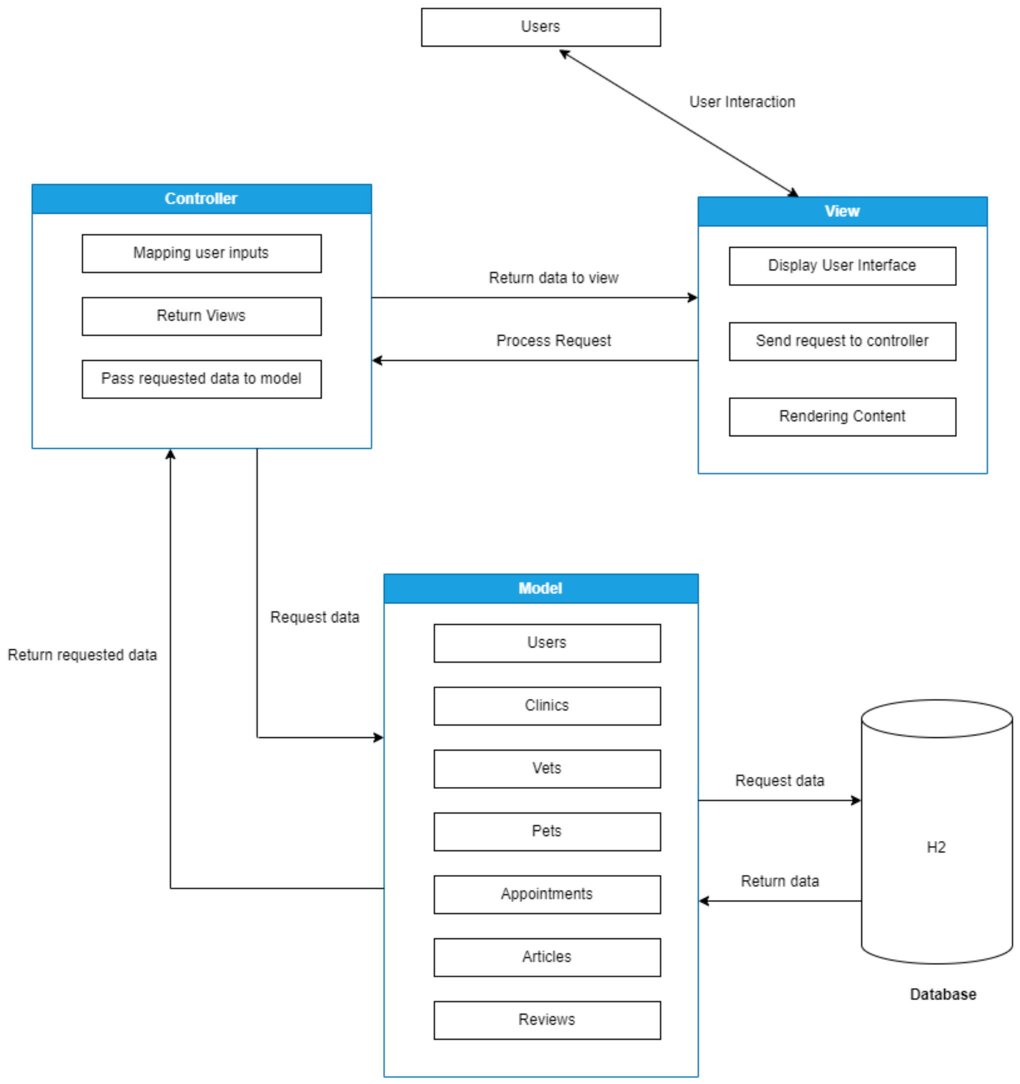
- The system must have an uptime of 99.9% or higher.
- The system must automatically back up data daily.

3.6 Maintainability Requirements

- The system's codebase must be well-documented.
- The system must use version control (e.g., Git) to manage code changes and ensure that all updates are tracked and reversible.
- The system must be designed with scalability in mind.
- The system must include automated testing options to verify the functionality of new code before it is deployed.

4 System Architecture

4.1 System Architecture Diagram



4.2 Model

The Model component in VetCare represents the core data of the application and encapsulates the business logic. It includes various entity classes representing real-world objects such as `Pet`, `Appointment`, `User`, and others. These models define how data is stored in the database.

Each model may have a corresponding Repository and Service that handle the actual interaction between the model and the database, as well as business logic specific to that model.

For example, the `Pet` model will have a corresponding `PetRepository` and `PetService`. The `PetRepository` interfaces with the database, while the PetService processes any business logic related to pet management, such as registering a new pet or retrieving a list of pets for a user.

4.3 Repository

The Repository layer is responsible for data persistence and retrieval. It is as an abstraction layer between the database and the business logic. Repositories are typically interfaces that extend the `JpaRepository` (or another persistence framework interface), allowing for standard CRUD operations as well as custom queries, without having to write raw SQL queries.

For instance:

- `PetRepository` would provide methods for retrieving pets from the database, such as `findByOwnerId(Long ownerId)` to get all pets owned by a specific user.
- `AppointmentRepository` would allow for fetching appointments based on `userID`.

Repositories are used by the Service layer to perform database operations.

4.4 Service

The Service layer handles the core business logic of the application. It sits between the Controller and the Repository, processing data before sending it to the database or returning it to the Controller.

For example:

- `PetService` handles tasks such as verifying if a pet's data is valid before saving it to the database or checking if a user has necessary permissions to manage a pet's information.

- The `AppointmentService` manages operations like appointment creation, finding all appointments for a given user, or editing an appointment's details.

Services allow for a separation of concerns by isolating business logic from data persistence and user interface logic.

4.5 Controller

The Controller is responsible for handling user requests and directing them to the appropriate Service layer. Controllers receive inputs from the user interface, process them with the help of services, and return the appropriate response. In VetCare's instance, the controller returns a Thymeleaf template view.

For example:

- A `PetController` handles requests related to pet management, such as registering a new pet or viewing a list of a user's pets. It uses the `PetService` to process these requests and then returns a view or data to the user.
- An `AppointmentController` handles requests related to booking or viewing appointments, letting the `AppointmentService` handle the appropriate logic.

4.6 View

The View component in VetCare is responsible for displaying data to the user and handling user interactions through the user interface. The view is built using Thymeleaf templates combined with HTML, CSS, and JavaScript.

Each view is integrated with the Controller, which provides the data needed to populate the page. When a user requests a page, the Controller sends a model with the necessary data to the View, which is then rendered dynamically.

For example:

- The Pets view will display a list of the user's registered pets in a table. The data for this table comes from the Controller, which receives it from the `PetService`.

- The Appointment Booking view allows the user to schedule an appointment by selecting a date, time, and pet. The data, such as available appointment times or pets, is passed to the View by the `AppointmentController`.

Example flow for rendering the Pets page:

1. A user navigates to the "View Pets" page.
2. The `PetController` fetches the list of pets from the `PetService`, which in turn calls the `PetRepository` to retrieve the data from the database.
3. The `PetController` sends this list of pets to the View.
4. The View (Thymeleaf template) dynamically displays the pets in a table, where each row represents a pet owned by the user.

VetCare's View components are also responsible for error basic handling, such as selecting a valid time to book an appointment. JavaScript is also utilised to export an appointment history to PDF format. CSS and Bootstrap are used to ensure that the layout is responsive and consistent across different devices.

4.7 Database

4.7.1 Overview of Entities

User: Represents users of the system, including pet owners

- UserID (Primary Key)
- Name
- Password
- Email
- PhoneNumber
- Address

Vet: Represents veterinarians working at various clinics

- VetID (Primary Key)
- Name
- Password

- Email
- ClinicID (Foreign Key to **Clinic**)
- PhoneNumber
- Address

Pet: Represents pets owned by users

- PetID (Primary Key)
- OwnerID (Foreign Key to **User**)
- Name
- Species
- Breed
- Age
- MedicalHistoryID (Foreign Key to **MedicalHistory**)

Appointment: Tracks appointments scheduled for pets with veterinarians

- AppointmentID (Primary Key)
- UserID (Foreign Key to **User**)
- PetID (Foreign Key to **Pet**)
- ClinicID (Foreign Key to **Clinic**)
- AppointmentDate
- Status (scheduled, completed, or cancelled)
- GeneralNotes
- Fees (base fixed fee, e.g. $84.95 for one appointment)
- VetID (Foreign Key to **Vet**)

MedicalHistory: Stores the medical history of pets

- MedicalHistoryID (Primary Key)
- PetID (Foreign Key to **Pet**)
- ChronicConditions
- Allergies
- Notes (Updated by VetInCharge)
- LastVaccinationDate
- LastTreatmentDate
- LastPrescriptionDate

VaccinationRecord: Contains records of vaccinations administered to pets

- VaccinationID (Primary Key)

- PetID (Foreign Key to **Pet**)
- VaccineName
- DateAdministered
- NextDueDate
- Status
- Notes
- VetID (Foreign Key to **Vet**)

TreatmentPlan: Stores treatment plans for pets

- TreatmentPlanID (Primary Key)
- PetID (Foreign Key to **Pet**)
- Diagnosis
- Description
- DateAdministered
- EndDate
- Notes
- VetID (Foreign Key to **Vet**)

Prescription: Records prescriptions given to pets

- PrescriptionID (Primary Key)
- PetID (Foreign Key to **Pet**)
- VetID (Foreign Key to **Vet**)
- MedicineID (Foreign Key to **Medicine**)
- Instructions
- DosageQuantity
- DateAdministered
- ExpiryDate
- RepeatsLeft
- RenewalDate

Medicine: Stores information about medicines used in prescriptions

- MedicineID (Primary Key)
- Name
- Description
- Strength
- SideEffects
- Cost

EducationalResource: Contains educational resources available to users

- ResourceID (Primary Key)
- Title
- ResourceType
- Author
- PublishDate
- Category
- Content
- Description

Clinic: Represents veterinary clinics.

- ClinicID (Primary Key)
- Name
- Address
- PhoneNumber
- Email

SavedResources: Tracks educational resources saved by users

- SavedResourceID (Primary Key)
- UserID (Foreign Key to **User**)
- ResourceID (Foreign Key to **EducationalResource**)
- SavedAt

LatestTrends: Contains the latest trends and information in veterinary care

- TrendID (Primary Key)
- Title
- Description
- Author
- PublishDate
- TrendCategory

4.7.2 Relationships

- **User ⇔ Pet**: One-to-Many
    - A user can own multiple pets, but each pet has only one owner.
- **Vet ⇔ Clinic**: Many-to-One

- A vet works at one clinic, but a clinic can have multiple vets.

- **Pet ↔ MedicalHistory**: One-to-One
    - Each pet has one instance of medical history, created when significant medical events occur, such as a treatment plan, vaccine, or prescription being administered.
- **Pet ↔ Appointment**: One-to-Many
    - A pet can have multiple appointments
- **MedicalHistory ↔ VaccinationRecord/TreatmentPlan/Prescription**: One-to-Many
    - A medical history can have multiple associated records.
- **User ↔ Appointment**: One-to-Many
    - A user can book multiple appointments for their pets.
- **Appointment ↔ Vet**: Many-to-One
    - Each appointment is associated with one vet.

4.8 Example Flow of Use

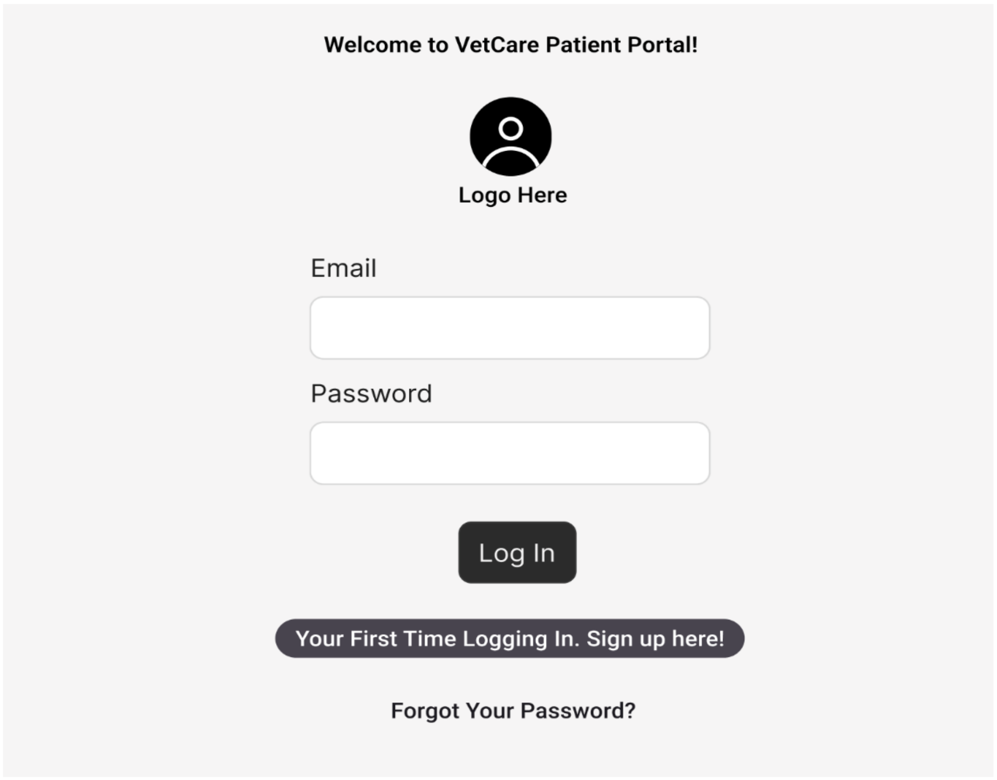
Here's an example of how the system components interact during a pet registration process:

1. The user accesses the "Register a Pet" page via the View, where they input the pet's information (name, species, breed, etc.).
2. The `PetController` receives the form submission from the user, extracts the data, and sends it to the `PetService`.
3. The `PetService` performs any necessary validations on the data (e.g., checking that the user doesn't already have a pet with the same name) and calls the `PetRepository` to save the new pet to the database.
4. The `PetRepository` uses the underlying database to persist the pet's information.

5. Once the pet is saved, the `PetService` returns a success message to the `PetController`, which then passes this back to the View for user feedback (e.g., displaying a confirmation message that the pet was successfully registered).

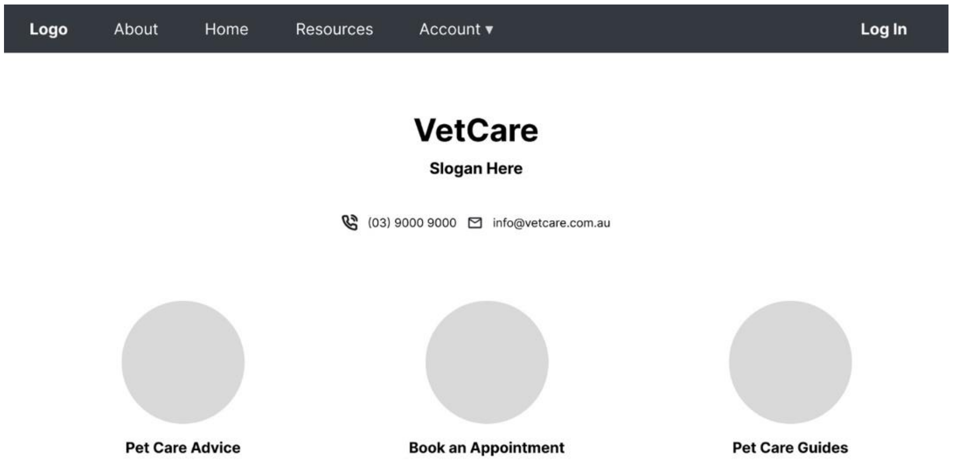
5. User Interface Design

5.1 Login page

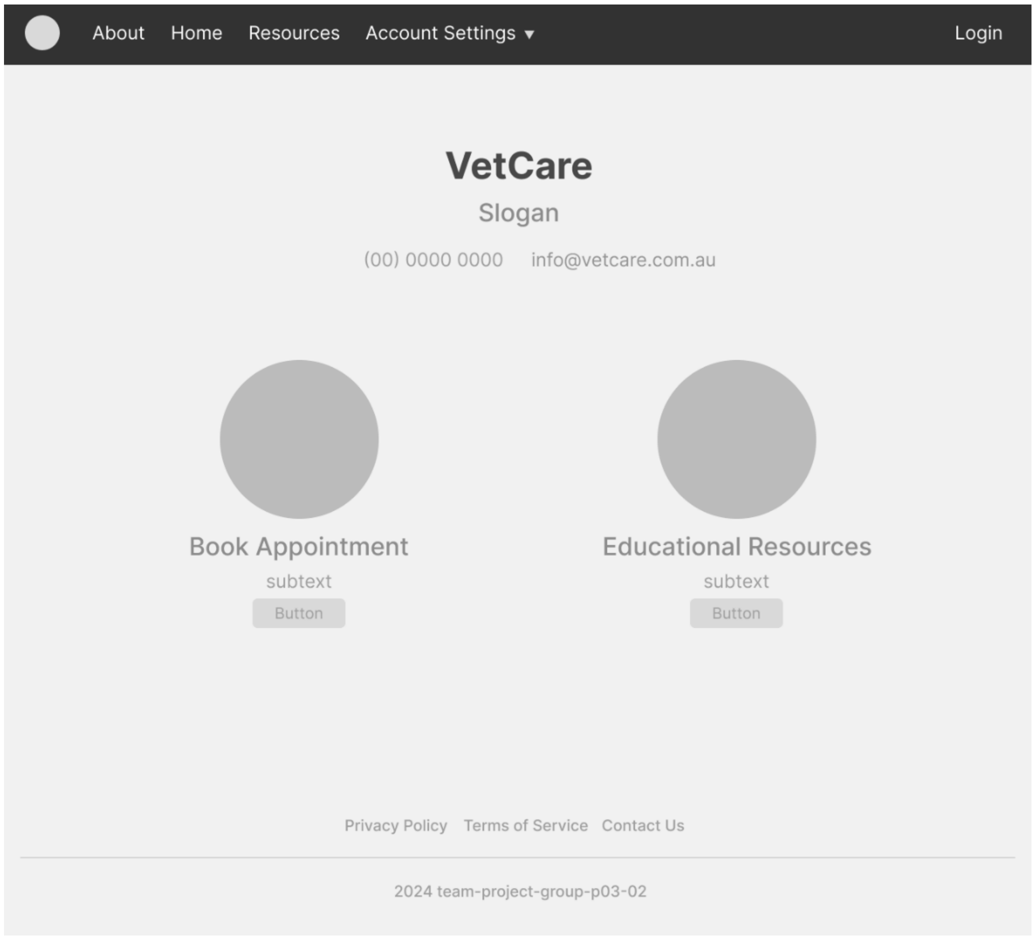

Welcome to VetCare Patient Portal!
Logo Here
Email
Password
Log In
Your First Time Logging In. Sign up here!
Forgot Your Password?

5.2 Landing page / Home Page

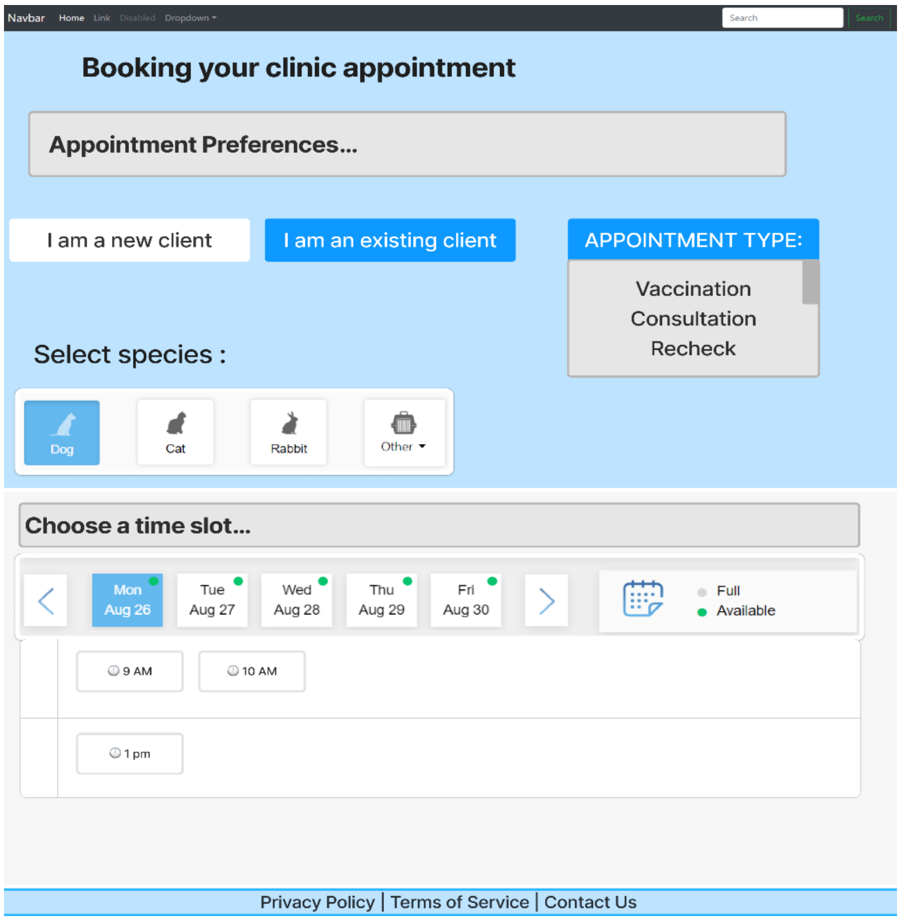
First iteration



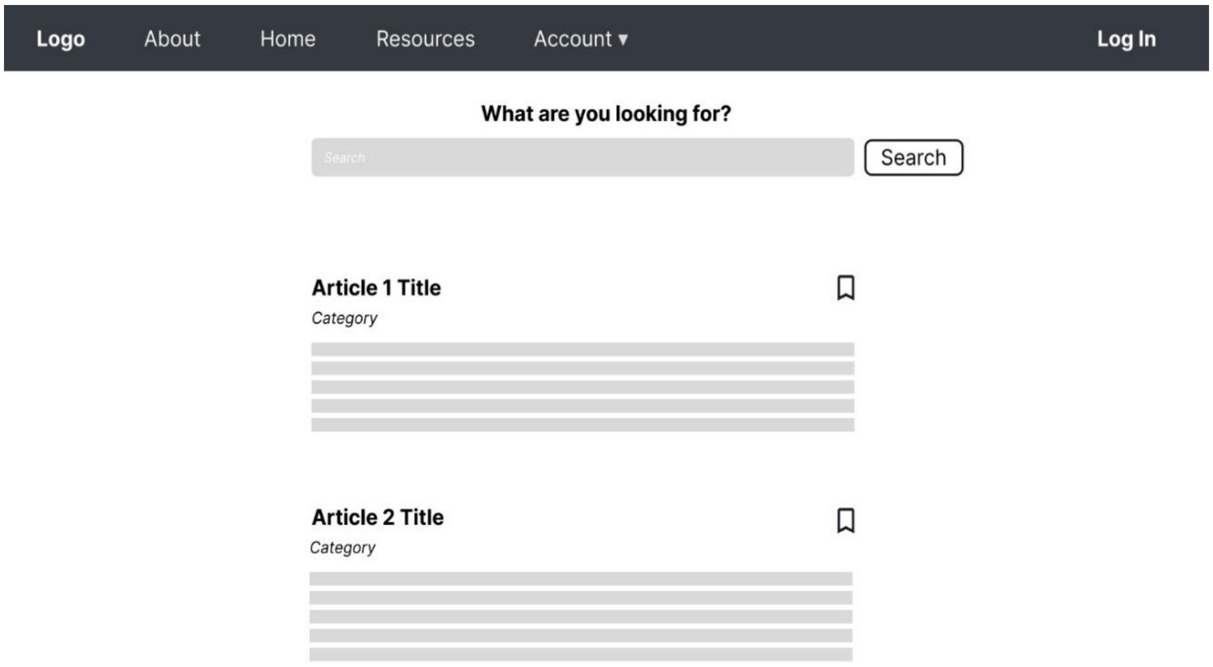
Second iteration



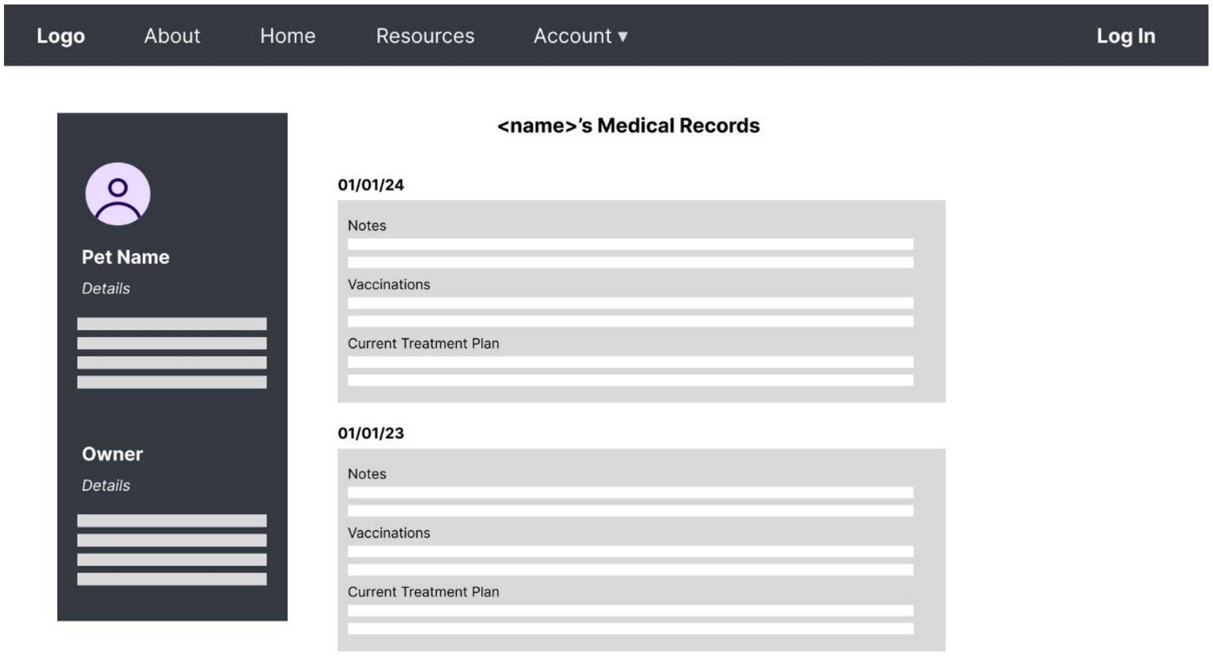
5.3 Appointments page



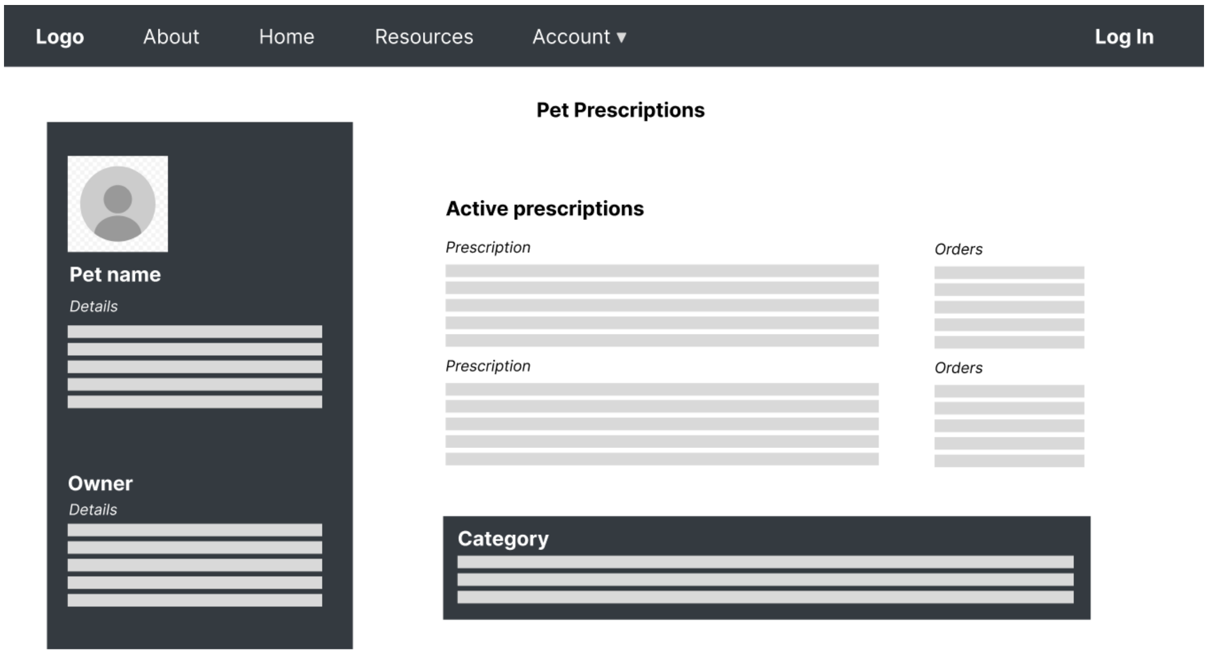
5.4 Educational Resources page



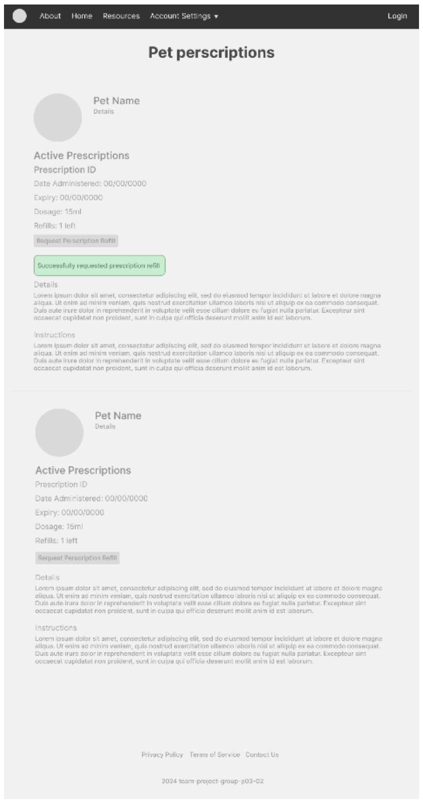
5.5 Medical Records page



5.6 Prescriptions page



Second Iteration



Pet perscriptions

Pet Name
Details

Active Prescriptions
Prescription ID
Date Administered: 00/00/0000
Expiry: 00/00/0000
Dosage: 15ml
Refills: 1 left

Successfully requested prescription refill

Details
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Instructions
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Pet Name
Details

Active Prescriptions
Prescription ID
Date Administered: 00/00/0000
Expiry: 00/00/0000
Dosage: 15ml
Refills: 1 left

Details
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Instructions
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

2024 team-project-group-p03-02

6 Assumptions and Constraints

6.1 Assumptions

- The system should store and access appointment and pet information from a database as opposed to any other storage format.
- Users can access their pet's information securely through an account system.
- Veterinarians should have access to pet medical data and can edit this data.
- Veterinarians should have their own accounts separate from the users.
- Users should pay during booking with a third-party payment system such as Stripe.
- Users should have the ability to delete their accounts.

6.2 Constraints

- We are limited by our resources, most notably time. This will impact the size of our scope for the system, reducing the quality of our implementations and the number of features we can include in our solution.
- Since we are students and are working with new technologies, our experience is limited. Due to this it will take a longer period for us to develop.
- To comply with local laws, we need to provide each user with the option to delete their account if they request. So, the feature for users to delete their account has been added.

7 Dependencies

VetCare's successful operation relies on the following external dependencies and libraries:

7.1 Payment Processing – Stripe API

- Stripe is a secure and reliable service for processing online payments. The system uses the Stripe API to handle all payment related processes, such as prescription ordering and online vet fee payments.
- The latest stable version of the API will be used.

7.2 Database – MySQL Connector

- MySQL is a popular and industry-standard database. The application will connect to it via a JBDC driver.

7.3 Application Framework – Spring Boot

- The system is built using the Spring framework, which uses the model-view-controller (MVC) architecture for the application. It controls user authentication capabilities, simplification of database interactions through the Java Persistence API, and allows the application to be run on Maven commands.

7.4 Templating Engine – Thymeleaf

- The system will use Thymeleaf as a templating engine for displaying and rendering all the website's HTML content.
- Thymeleaf 3.1.2 will be used.

7.5 Utilities – Apache Commons

- The system will make use of Apache Commons, which comprises of reusable Java components for handling common operations like file I/O and strings. Implementing these will reduce development time for common programming tasks.

7.6 Front-End Framework – Bootstrap

- Bootstrap provides simple and reusable components for user interfaces, which reduces the need for custom CSS.
- Components like navigation bars, forms, and buttons will be integrated into the application.

8 Glossary

- **API (Application Programming Interface)**: A set of functions and instructions that allow different software applications to communicate with each other. In this project, the Stripe API are used for payments, while OAuth 2.0 is used for user authentication.
- **Authentication**: The process of verifying the identity of a user or system. In VetCare, authentication ensures that only registered users and veterinarians can access their accounts.
- **CRUD Operations**: An acronym for Create, Read, Update, and Delete operations. These are the basic functions of persistent storage and are used throughout the VetCare system to manage data in the database.
- **Database Management System (DBMS)**: A tool that uses a database to store, retrieve, and manage data. VetCare uses the H2 database, a lightweight relational DBMS, for storing data related to users, pets, appointments, and more.
- **Entity-Relationship Diagram (ERD)**: A visual representation of the database schema, showing how different entities (e.g., Users, Pets, Appointments) are related to each other.
- **PDF (Portable Document Format)**: A file format used to present documents consistently across different devices and platforms. VetCare uses Apache PDFBox to generate PDF documents for appointment histories.
- **Thymeleaf**: A modern Java-based templating engine used to process and generate HTML views in the VetCare application.
- **Two-Factor Authentication (2FA)**: An additional layer of security that requires not only a username and password but also something that only the user has access to, such as a mobile device, to verify their identity.
- **Unit Testing**: A software testing method where individual units or components of a software are tested by themselves. VetCare uses JUnit for unit testing to ensure that individual parts of the code are correct.

- **User Story**: A simple description of a feature from the perspective of the user or customer. User stories in VetCare guide the development of features, such as account creation or booking appointments.
- **Version Control**: A system that records changes to files over time so that specific versions can be used later. VetCare uses Git for version control to manage changes to the codebase.
- **Vet**: Short for veterinarian, a medical professional who treats animals. In VetCare, vets are users with specific roles that allow them to access and update pet medical data.
- **Web Framework**: A software framework that is designed to support the development of web applications including web services, web resources, and web APIs. VetCare uses Spring Boot as its web framework.
- **Wireframe**: A basic visual guide used to suggest the layout and structure of a web page or app interface without focusing on smaller design details, such as colours.
- **User Interface (UI)**: The means by which the user and a computer interact. In VetCare, UI components are designed to be intuitive and user-friendly to enhance user experience.
- **User Experience (UX)**: The experience of a user when interacting with the application, mostly in terms of how easy it is to use.

Appendix: Milestone 2

New/Adjusted Requirements

- User Story – Create Skeleton Sitemap for VetCare
    - To help colleagues complete their tasks more effectively, a sitemap was created to avoid confusion about the linkage of webpages to each other.
- **Removed 'email verification' Task from** User Story #7
    - Email verification is currently out of scope due to time constraints.
- **Removal of Continuous Integration (CI) from Current Sprint**
    - Due to time constraints, CI has been removed from the current sprint and moved onto the next.
- User Story #22 **and** User Story #23 **Removed from Current Sprint**
    - Due to time constraints, these stories have been moved from the current sprint to the final sprint.

Minutes of Meetings

- Refer to Meeting Minutes PDF located in /docs/milestone2 of GitHub repository.

Appendix: Milestone 3

Adjusted Requirements

- **Due to time constraints, the following user stories have been removed from this sprint for release:**
  - Automatic Prescription Ordering
  - Manage Payments
  - Update Pet Medical Data
  - Stay informed with the latest trends
- **Automatic future-to-past appointment conversion**
  - Transferred from Seanghai to Keenan
- **Use Docker to Host VetCare**
  - Transferred from Keenan to Seanghai
- Booking Appointments feature re-done by Keenan due to buggy and incomplete implementation during previous sprint

New Requirements

- **View All Registered Pets**
  - Identified and assigned to Keenan
- **Register a new pet**
  - Identified and assigned to Keenan
- **Remove "Register as a vet" from the login page**
  - Identified by William
  - Assigned to William
- **Clean up login and register pages**
  - Identified by William
  - Assigned to William
- **Login and Logout buttons appear correctly**
  - Identified by William
  - Assigned to Keenan

- **Appointments should be a separate tab in the navbar**
  - Identified by William
  - Assigned to Keenan
- **Current appointment details should autofill when editing an appointment**
  - Identified by William
  - Assigned to Keenan. Postponed until next sprint
- **Edit user stories to remove “as a developer”**
  - Identified by William
  - Assigned to nobody. Postponed until next sprint

Agreed and Delivered Functionality

The below points are indicative of functionality that was agreed between the Product Owner (PO) and the Scrum Master (Keenan). An agreement of this can be found in the /docs/Milestone3/ folders of the GitHub repository.

- Users can sign up and log in securely
- Users can book an appointment
- Users can manage their appointments (view/edit/cancel)
- Users can register pets
- Users can view medical information (prescription, vaccines, etc.)
- Users can have an appointment history and medical history exported in PDF form
- Users can request prescriptions
- Users can view medicine dosage information
- VetCare will be containerised via Docker
- CI/CD
- Responsive mobile and desktop design
- Pet care education resources [revised]

Test Case Documentation

AccountControllerTest.java

ID	Description	Tested Method	Expected Result
1	View account settings. Tests if the account settings view is displayed.	AccountController.viewAccountSettings	HTTP status 200 OK, view name account/settings, and user attribute is present in the model.
2	Tests if the contact information is updated successfully when the user submits the form with new contact information.	AccountController.updateContactInfo	The HTTP response status is 200 OK, the view name is account/settings, and the model contains a successMessage indicating successful update.
3	Tests if the password is changed successfully when the user provides the correct old password and a new password.	AccountController.changePassword	The HTTP response status is 200 OK, the view name is account/settings, and the model contains a successMessage indicating the password change was successful.
4	Tests if the password change fails when the user provides an incorrect old password.	AccountController.changePassword	The HTTP response status is 200 OK, the view name is account/settings, and the model contains an errorMessage indicating the password change failed.
5	Tests if the account is deleted successfully when the user provides	AccountController.deleteAccount	The HTTP response status is 3xx Redirect, and the user is

	the correct password.		redirected to the home page (/).
6	Tests if account deletion fails when the user provides the wrong password.	AccountController.deleteAccount	The HTTP response status is 200 OK, the view name is account/settings, and the model contains an errorMessage indicating account deletion failed.

AppointmentControllerTest.java

ID	Description	Tested Method	Expected Result
1	Simulates booking an appointment for a pet.	AppointmentController.bookAppointment	User is redirected to /appointments, flash message "Appointment booked successfully!" is present.
2	Verifies if a specific appointment is displayed correctly.	AppointmentController.viewSpecificAppointment	HTTP status 200 OK, view name appointments/view, model contains the specific appointment.
3	Checks if all appointments for the current user are retrieved and displayed.	AppointmentController.viewAppointments	HTTP status 200 OK, view name appointments/index, model contains the list of appointments.
4	Verifies if the edit form for a specific	AppointmentController.showEditForm	HTTP status 200 OK, view name appointments/edit,

	appointment is shown correctly.		model contains the appointment to be edited.
5	Simulates updating an appointment's details.	AppointmentController.updateAppointment	User is redirected to /appointments, flash message "Appointment updated successfully!" is present.
6	Verifies if an appointment can be successfully canceled.	AppointmentController.deleteAppointment	User is redirected to /appointments, flash message "Appointment cancelled successfully!" is present.

HomeControllerTest.java

ID	Description	Tested Method	Expected Result
1	Checks if the home page is displayed correctly.	HomeController.index	HTTP status 200 OK, view name home/index.html.
2	Checks if the vet dashboard page is displayed correctly.	HomeController.userHome	HTTP status 200 OK, view name vet/index.
3	Verifies if the admin dashboard page is displayed correctly.	HomeController.adminHome	HTTP status 200 OK, view name admin/index.

LoginControllerTest.java

ID	Description	Tested Method	Expected Result
1	Checks if login page is displayed correctly.	LoginController.login	HTTP status 200 OK, view name authentication/login.

PetControllerTest.java

ID	Description	Tested Method	Expected Result
1	Checks if the view for registered pets is returned correctly when pets are available.	PetController.viewRegisteredPets	HTTP status 200 OK, view name pets/index, model contains a list of pets.
2	Checks if the view for registered pets shows the 'no pets' message when there are no pets.	PetController.viewRegisteredPets	HTTP status 200 OK, view name pets/index, model contains noPetsMessage attribute with the message: "You have no registered pets."
3	Checks if the pet registration form is displayed correctly.	PetController.showPetRegistrationForm	HTTP status 200 OK, view name pets/new.
4	Checks if the medical history of a specific pet is shown correctly when the pet exists.	PetController.viewPetMedicalHistory	HTTP status 200 OK, view name pets/view, model contains attributes for the pet, vaccinations, treatment plans, and prescriptions.
5	Checks if the error message is shown when the pet does not exist.	PetController.viewPetMedicalHistory	HTTP status 200 OK, view name pets/error, model contains errorMessage attribute with the message: "Pet not found."

RegistrationControllerTest.java

ID	Description	Tested Method	Expected Result
1	Checks if the registration page is displayed correctly with a user attribute in the model.	RegistrationController.register	HTTP status 200 OK, view name authentication/register, model contains the user attribute.
2	Checks if a new user is registered successfully when provided with valid email and password.	RegistrationController.register	HTTP status 3xx Redirection, no errors encountered during registration.

ClinicTest.java

ID	Description	Assertions	Expected Result
1	This test verifies that the getter and setter methods for the Clinic class work as expected.	• clinic.getClinicID() returns 1L. • clinic.getName() returns "Health Clinic". • clinic.getAddress() returns "123 Main St". • clinic.getPhoneNumber() returns "123-456-7890". • clinic.getEmail() returns "info@healthclinic.com".	All getter methods return the correct values that were set using the corresponding setter methods.

CustomUserTest.java

ID	Description	Assertions	Expected Result
1	This test verifies that the getter and setter methods for the CustomUser class work as expected.	⍰ user.getUserId() returns 1L. ⍰ user.getName() returns "John Doe". ⍰ user.getEmail() returns "john.doe@example.com". • user.getPassword() returns "password123".	All getter methods return the correct values that were set using the corresponding setter methods.

		• user.getPhoneNumber() returns "123-456-7890". • user.getAddress() returns "123 Main St". • user.getUserType() returns UserType.PetOwner.	

AppointmentServiceImplTest.java

ID	Description	Tested Method	Expected Result
1	Ensures that when an appointment is created and saved using the appointmentService, it returns the correct appointment object.	createAppointment(Appointment appointment)	The appointment returned matches the appointment passed to the createAppointment method.
2	Ensures that when appointments are retrieved by user ID, the correct list of appointments is returned.	getAppointmentsForUser(Long userId)	The list contains the correct number of appointments associated with the user.
3	Ensures that an appointment can be retrieved by its ID, and the correct appointment object is returned.	findAppointmentById(Long appointmentId)	The appointment returned matches the expected appointment associated with the given ID.
4	Verifies that when an appointment is saved, the repository's save method is correctly invoked.	saveAppointment(Appointment appointment)	The save method of appointmentRepository is invoked with the correct appointment object.

PetMedicalHistoryServiceTest.java

ID	Description	Tested Methods	Expected Results
1	Verifies that the correct MedicalHistory is returned for a valid pet ID.	getMedicalHistoryBypetId(Long petId)	Returns an Optional containing the MedicalHistory.
2	Ensures that an empty Optional is returned when no medical history exists for a pet.	getMedicalHistoryBypetId(Long petId)	Returns an empty Optional.
3	Checks if vaccination records are correctly returned for a pet.	getVaccinationRecordsBypetId(Long petId)	Returns a list of vaccination records.
4	Verifies that treatment plans are returned for the specified pet.	getTreatmentPlansBypetId(Long petId)	Returns a list of treatment plans.
5	Ensures prescription records are retrieved for a pet.	getPrescriptionsBypetId(Long petId)	Returns a list of prescriptions.

PetServiceImplTest.java

ID	Description	Tested Methods	Expected Results
1	Ensures that the correct Pet object is returned when it is found by ID.	getPetById(Long petId)	Returns the Pet object with the expected name "Buddy".
2	Verifies that an exception is thrown when the pet is not found by ID.	getPetById(Long petId)	Throws an EntityNotFoundException with the message "Pet not found with ID: 1".
3	Verifies that the savePet method calls the repository's save method exactly once.	savePet(Pet pet)	The repository's save() method is called one time.
4	Ensures that the correct Pet object is found by ID and returned in an Optional.	findPetBypetId(Long petId)	Returns an Optional containing the Pet with the expected petId.

5	Verifies that an empty Optional is returned when the pet is not found by ID.	findPetBypetId(Long petId)	Returns an empty Optional when no Pet is found.

PrescriptionServiceImplTest.java

ID	Description	Tested Method	Expected Result
1	Verifies that a prescription with a future expiry date (tomorrow) and repeats left is considered valid.	checkPrescription(Prescription prescription)	Returns "valid".
2	Ensures that a prescription with today's expiry date and repeats left is considered valid.	checkPrescription(Prescription prescription)	Returns "valid".
3	Ensures that a prescription with an expiry date in the past (yesterday) is considered invalid.	checkPrescription(Prescription prescription)	Returns "out of date prescription".
4	Ensures that a prescription with zero repeats left is considered invalid.	checkPrescription(Prescription prescription)	Returns "no repeats remaining".
5	Ensures that a prescription with negative repeats left is considered invalid.	checkPrescription(Prescription prescription)	Returns "no repeats remaining".
6	Verifies that the number of repeats for a prescription is correctly decremented.	decrementPrescription(Prescription prescription)	The repeats left in the prescription should be decremented by 1.

UserServiceTest.java

ID	Description	Tested Methods	Expected Result
1	Verifies that an exception is thrown if a user is not found by the provided username.	loadUserByUsername(String username)	UsernameNotFoundException.
2	Ensures that the correct UserDetails are returned when the user exists in the repository.	loadUserByUsername(String username)	Returns valid UserDetails with username and password.
3	Verifies that the user is correctly retrieved by ID when found in the repository.	getUserById(Long userId)	Returns the correct user object.
4	Verifies that an exception is thrown if the user is not found by username.	getUserIdByUsername(String username)	Throws RuntimeException.
5	Ensures the user ID is returned when the user is found by username.	getUserIdByUsername(String username)	Returns the correct user ID.
6	Verifies that the contact information is updated successfully.	updateContactInfo(Long userId, User updatedUser)	Updates the user's contact information correctly.
7	Verifies that an exception is thrown if the user	updateContactInfo(Long userId, User updatedUser)	Throws RuntimeException.

	is not found when trying to update contact information.		
8	Ensures that the password is successfully changed when the old password matches.	changePassword(Long userId, String oldPassword, String newPassword)	Returns true and updates the password.
9	Verifies that the password change fails if the old password does not match.	changePassword(Long userId, String oldPassword, String newPassword)	Returns false when the old password is incorrect.
10	Verifies that the user is deleted if the provided password matches.	deleteAccount(Long userId, String password)	Deletes the user and returns true.
11	Verifies that the account deletion fails if the password does not match.	deleteAccount(Long userId, String password)	Returns false, and the user is not deleted.

PrescriptionServiceImplTest.java

ID	Description	Tested Methods	Expected Result
1	Verifies that a prescription that's due tomorrow with 2 repeats is valid.	checkPrescription(prescription)	"valid"
2	Verifies that a prescription that's due today is valid.	checkPrescription(prescription)	"valid"

3	Verifies that a prescription that was due yesterday is invalid.	checkPrescription(prescription)	"out of date prescription"
4	Verifies that prescription with no repeats left is invalid	checkPrescription(prescription)	"no repeats remaining"
5	Verifies that a prescription with negative repeats is invalid.	checkPrescription(prescription)	"no repeats remaining"
6	Verifies that a prescription with 2 repeats decrements to 1 after this method is used.	decrementPrescription(prescription)	1

EduResourcesControllerTest.java

ID	Description	Tested Methods	Expected Result
1	Checks if the home page is displayed correctly.	eduResources.index	HTTP status 200 OK, view name eduResources /index.html.

EduResourcesServiceImplTest.java

ID	Description	Tested Methods	Expected Result
1	Checks that calling service function to find all resources by a type/category will return a list of resources	findAllResourcesByType(String category)	Returns a list of resources

2	Checks that the function retrieves the correct resource when finding it by its ID	FindResourceById(Long id)	Returns the resource retrieved

SavedResourcesServiceImplTest.java

ID	Description	Tested Methods	Expected Result
1	Checks that saving a resource will update the database	saveResource(EduResources eduresource)	Saved resource is updated in the database table saved_resources
2	Checks that deleting a saved resource will delete it from the database	deleteSavedResource(CustomUser user, EduResources eduresource)	Deleted resource is updated and is no longer showing in saved_resources