
Software Requirements Specification

for

VetCare - Online Vet Clinic Management System

Version 1.0 approved

Prepared by Group-P09-03

Developers:

Aphisith Siphaxay [s3987059]

Ashmit Sachan [s3873827]

Henry Van Toledo [s3849054]

Kai Hei Kong [s3971187]

Kaiyang Zheng [s3992987]

Preeti Goel [s3879991]

20/08/2024

Table of Contents

1. Introduction	1
1.1 Purpose	1
1.2 Document Conventions	1
1.3 Intended Audience and Reading Suggestions	2
1.4 Product Scope	2
1.5 References	3
2. Overall Description	4
2.1 Product Perspective	4
2.2 Product Functions	6
2.3 User Classes and Characteristics	7
2.4 Operating Environment	8
2.5 Design and Implementation Constraints	9
2.6 User Documentation	11
2.7 Assumptions and Dependencies	12
3. External Interface Requirements	13
3.1 User Interfaces	13
3.2 Hardware Interfaces	14
3.3 Software Interfaces	15
3.4 Communications Interfaces	17
4. Nonfunctional Requirements	18
4.1 Performance Requirements	18
4.2 Safety Requirements	19
4.3 Security Requirements	20
4.4 Software Quality Attributes	21
4.5 Business Rules	22
5. Other Requirements	22
6. System Architecture	25
6.1 Architecture Overview	27
6.2 Architectural Decisions	28
7. User Interface Design	
8. Testing	32

Revision History

Name	Date	Reason For Changes	Version
Milestone 1	20/08/2024	Initial Software Requirements Specification	V1.0.0
Milestone 2	22/09/2024	Sprint 1 Application Implementation	V2.0.0

1. Introduction

1.1 Purpose

This document specifies the software requirements for the VetCare Online Vet Clinic Management System, version 1.0. The VetCare system is designed to facilitate pet owners in managing their pets' health by providing features such as appointment scheduling, access to medical records, prescription management, and educational resources and real time appointment notifications.

The scope of this SRS covers the entire VetCare system, including all major features and functionalities that will be delivered in version 1.0. This includes the full integration with veterinary clinics and stores, user interface design, back-end services, and the database schema. The document outlines both functional and non-functional requirements, ensuring that the system meets the needs of end-users and stakeholders.

This SRS does not cover future enhancements or versions beyond 1.0, nor does it address any hardware or infrastructure specifics beyond those necessary to support the software application.

1.2 Document Conventions

This document follows standard SRS conventions and adheres to the following guidelines:

Headings and Subheadings: Key sections are highlighted using distinct headings and subheadings to ensure the document is easy to navigate.

Numbering: All requirements are numbered sequentially and organized according to their priority. All requirements have their own priority.

Font and Style: The document uses a consistent font (e.g., Times New Roman, 12 pt) for the main text, with bold used for section headings and italics used for emphasis or when referring to other documents, external systems, or components.

Requirements Inheritance: It is assumed that priorities for higher-level requirements are inherited by detailed sub-requirements unless otherwise specified.

Terminology: Technical terms and acronyms are defined in the glossary (Appendix A). First use of an acronym is followed by its full form in parentheses.

References: All external documents, standards, and resources are referenced in the "References" section and are italicized when mentioned in the text.

1.3 Intended Audience and Reading Suggestions

This document is intended for the following audiences:

- Developers
- Product Owner
- Product User
- Stakeholders

Reading Suggestions:

Readers are advised to begin with the "*Introduction*" to gain an understanding of the document's purpose, scope, and intended use. The "Overall Description" provides a comprehensive overview of the system, which is useful for all audiences. Depending on their specific role:

Product Owner should proceed to read through the whole document. Emphasis on Product Perspective is suggested

Developers should proceed to "*External Interface Requirements*", "*Non Functional Requirments*", "*Other Requirements*", "*System Architecture*" and "*User Interface Design*" to dive into the technical details.

Project Managers should focus on "*Nonfunctional Requirements*" and "*Other Requirements*" to understand the project's constraints.

Testers should prioritize "*External Interface Requirements*" and "*Nonfunctional Requirements*" and "*Other Requirements*" for detailed specifications that will guide their testing strategies.

Stakeholders should review "*Introduction*", "*Product Scope*" and "*Other Requirements*"

1.4 Product Scope

The "VetCare" project is an all-inclusive web-based veterinary clinic administration system created to greatly improve the efficiency and accessibility of pet care services. Its main goal is to make managing several facets of pet healthcare, like scheduling appointments, accessing medical records, managing prescriptions, and providing instructional materials, easier. The application integrates these services into an easy-to-use platform to make interactions between pet owners and veterinarian clinics simple. This guarantees prompt medical attention and adherence to treatment guidelines, which not only improves convenience but also leads to improved health outcomes. Additionally, by utilising digital solutions, we are able to increase the outreach of our platform, enhancing user experience within this industry. VetCare's goal is to encourage sustainable practices, through reduction of excessive paperwork and inefficient processes. Additionally, we aim to better veterinary care by increasing accessibility and making processes far more manageable and intuitive.

1.5 References

Ahirav, D (2024), *Real-Time Communication with WebSockets: A Complete Guide*, DEV, accessed 24 August 2024.

<https://dev.to/dipakahirav/real-time-communication-with-websockets-a-complete-guide-32g4>

Animal Welfare Standards and Guidelines 2023, State and Territory Legislation, Animal Welfare Standards, accessed 25 August 2024.

<https://animalwelfarestandards.net.au/welfare-standards-and-guidelines/state-and-territory-legislation/>

Australian Cyber Security Centre (ACSC) (2023), *Essential Eight Assessment Guidance Package*, Australian Signals Directorate, accessed 24 August 2024.

<https://www.cyber.gov.au/about-us/news/essential-eight-assessment-guidance-package>.

Australian Direct Marketing Association (ADMA) (n.d.) *3 things you need to know about the Spam Act*, accessed August 25, 2024.

<https://www.adma.com.au/resources/3-things-you-need-know-about-spam-act>

Barot, S (2023), *Spring vs Spring Boot: Technical Comparison of Both the Frameworks*, AGLOW ID, accessed 24 August 2024.

<https://aglowiditsolutions.com/blog/spring-vs-spring-boot/>

DNV (n.d.) *ISO/IEC 27001 - Information Security Management System (ISMS)*, DNV, accessed 24 August, 2024.

<https://www.dnv.com/services/iso-iec-27001-information-security-management-system-3327>

Doglio F (2023), *Monolith vs Microservice Architecture: A Comparison*. Camunda, accessed 24 August 2024.

<https://camunda.com/blog/2023/08/monolith-vs-microservice-architecture-comparison/>

International Organization for Standardization (n.d.), *ISO/IEC 27001 - Information security management*, accessed August 25, 2024. <https://www.iso.org/standard/27001>

Lumigo (n.d), *Containerized Applications: Benefits, Challenges & Best Practices*, accessed August 24, 2024.

<https://lumigo.io/container-monitoring/containerized-applications-benefits-challenges-best-practices/>

NNG(Nielsen Norman Group) (2024) *10 Usability Heuristics for User Interface Design*, NNG website, accessed 24 August 2024.

<https://www.nngroup.com/articles/ten-usability-heuristics/>

Office of Local Government, NSW (n.d.) *Microchipping and registration*, NSW Government, accessed August 25, 2024.

<https://olg.komosionstaging.com/public/dogs-cats/nsw-pet-registry/microchipping-and-registration/>

Office of the Australian Information Commissioner (n.d.), *About the Notifiable Data Breaches scheme*, accessed August 25, 2024.

<https://www.oaic.gov.au/privacy/notifiable-data-breaches/about-the-notifiable-data-breaches-scheme>

Office of the Australian Information Commissioner (n.d.), Australian Privacy Principles, accessed August 25, 2024. <https://www.oaic.gov.au/privacy/australian-privacy-principles>

Victorian Government (2018) Veterinary Practice Regulations 2018 (Statutory rule No. 11/2018), accessed 25 August 2024. <https://www.legislation.vic.gov.au/in-force/statutory-rules/veterinary-practice-regulations-2018/001>

Wikipedia contributors (n.d.), Terms of service, Wikipedia, The Free Encyclopedia, accessed August 25, 2024. https://en.wikipedia.org/wiki/Terms_of_service

World Intellectual Property Organization (n.d.), Trademarks, WIPO, accessed August 25, 2024. <https://www.wipo.int/trademarks/en/>

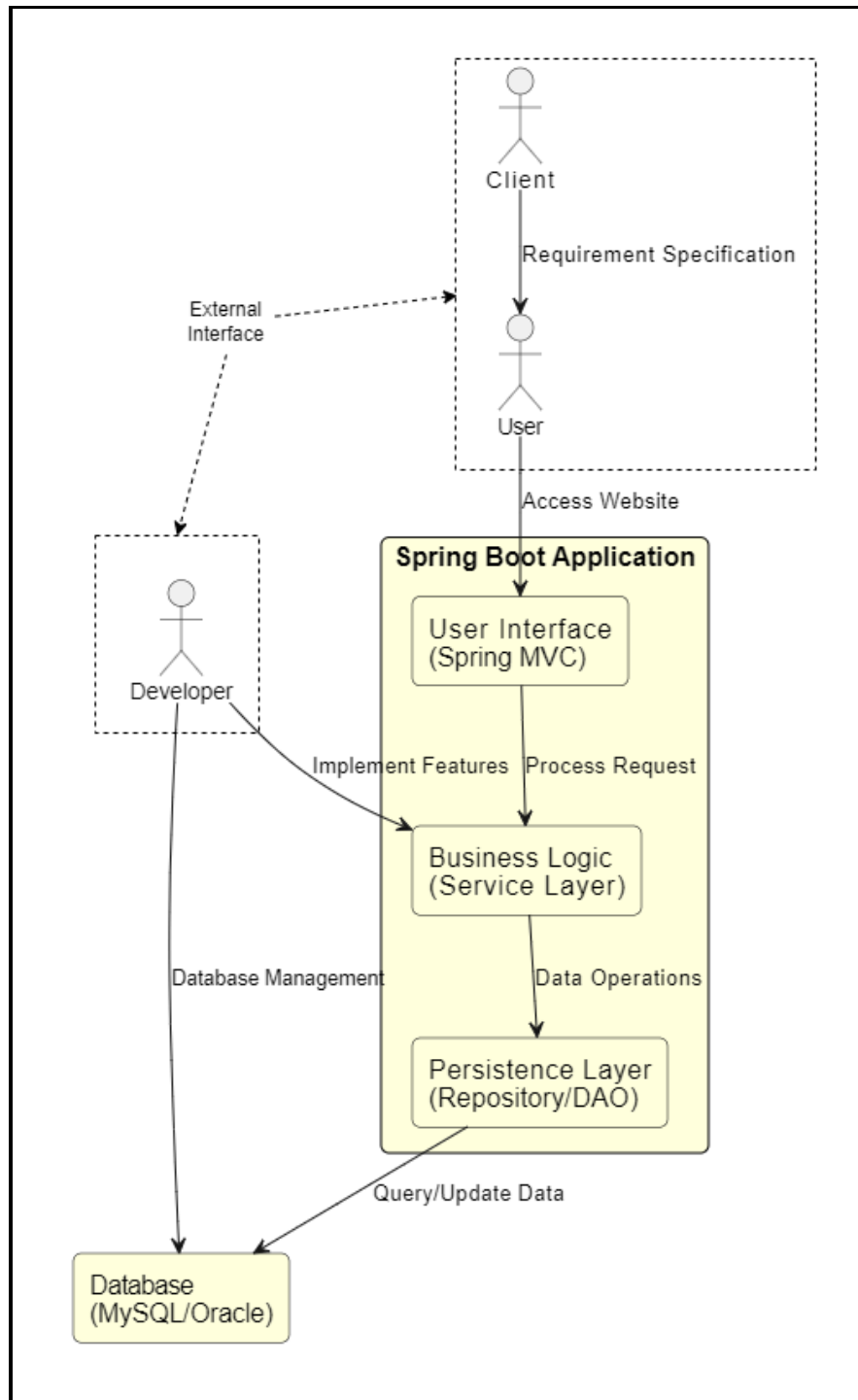
World Intellectual Property Organization (n.d.), Copyright, WIPO accessed August 25, 2024. <https://www.wipo.int/copyright/en/>

WP Crux (2024) Why MySQL is better than other databases in 2024? accessed August 24, 2024, from <https://wpcrux.com/blog/why-mysql-is-better-than-other-databases>

2. Overall Description

2.1 Product Perspective

Figure 1 : Product Perspective Diagram



The VetCare Online Vet Clinic Management System is a newly developed, standalone web application. It is not part of an existing product family nor is it a replacement for any previous system. VetCare was created from the ground up to meet the specific needs of modern veterinary practices and pet owners, offering a comprehensive solution for managing pet healthcare.

VetCare is self-contained, managing all operations internally, without reliance on external systems. However, the system is designed with potential future integrations in mind, such as connecting with e-commerce platforms or external veterinary management systems.

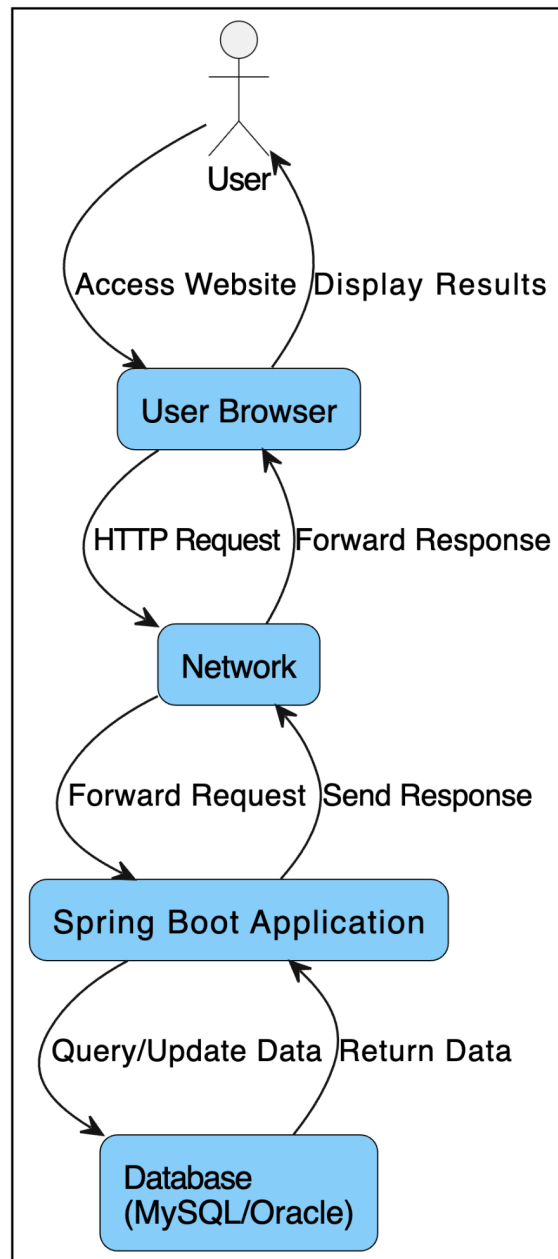
The application is structured into three key layers:

1. **User Interface (UI) Layer:** This layer provides a responsive and intuitive interface for users, including pet owners, veterinarians, and clinic administrators. It facilitates easy interaction with the system's features.
2. **Business Logic Layer:** This is the core of the system, where all the business rules and processes are implemented. It manages functionalities such as appointment scheduling, medical record management, prescription handling, and notification delivery.
3. **Persistence Layer:** This layer handles data storage and retrieval using a MySQL database. It ensures secure and efficient management of all user data, including medical histories, appointment details, and other critical information.

This SRS document covers the requirements for the entire VetCare system as implemented in version 1.0, focusing on the internal functionalities provided by these three layers. Future enhancements and integrations will be considered in subsequent versions of the SRS.

2.2 Product Functions

Figure 2: Top level data flow diagram



The VetCare system is designed to perform a range of functions that facilitate the management of pet healthcare. The major functions include:

Medical Records Management: Securely store and manage comprehensive pet health records, including vaccination histories, treatment plans, and more.

Appointment Scheduling: Allow users to book, reschedule, or cancel veterinary appointments. Automated reminders are sent to ensure that users remember their upcoming appointments.

Prescription Management: Enable pet owners to request additional medication for their pets, with VetCare coordinating the processing and delivery of these prescriptions.

Educational Resources: Provide users with access to a library of articles, videos, and guides on pet care and wellness, helping them stay informed about best practices in veterinary medicine.

Notifications and Reminders: Automatically send notifications to users about upcoming appointments, medication schedules, and other important events related to their pets' healthcare.

These functions are designed to work seamlessly together, ensuring that users can efficiently manage their pets' healthcare needs within a single, integrated system.

2.3 User Classes and Characteristics

In the context of the VetCare Online Vet Clinic Management System, the following user classes have been identified, focusing on the primary users for whom the system is being developed:

1. Pet Owners

- **Usage Frequency:** Pet owners are expected to be the most frequent users of the system, regularly accessing it to manage their pets' healthcare needs.
- **Primary Functions:** Pet owners will use the system to schedule appointments, view and manage their pets' medical records, request prescription refills, and access educational resources. They will also receive notifications and reminders for appointments and medication schedules.
- **Technical Proficiency:** The system is designed to be user-friendly, with an intuitive interface that caters to users with basic to intermediate technical skills.
- **Security and Access Level:** Pet owners have restricted access, limited to their own accounts and their pets' information. They cannot access or modify data related to other users or system settings.
- **Education and Experience:** The system should be accessible to a wide range of users, with varying levels of education and experience, ensuring ease of use for all.

2. Developers (Project Team)

- **Usage Frequency:** As the team responsible for building and maintaining the system, developers interact with the system regularly throughout its development lifecycle.
- **Primary Functions:** Developers are responsible for implementing, testing, and refining the system's features, including the user interface, business logic, and data management.

- **Technical Proficiency:** Developers possess a strong technical background, with skills in programming, database management, and web development.
- **Security and Access Level:** Developers have full access to the system's codebase, databases, and configuration settings, enabling them to make necessary changes and improvements.
- **Education and Experience:** Developers are students gaining practical experience in software development, applying their theoretical knowledge to create a functional system.

Importance of User Classes

Primary Users: Pet Owners are the main focus of the VetCare system, as they will be the most frequent users and are the intended beneficiaries of the system's features.

Support Users: Developers are essential during the project development phase, ensuring the system is built and functions as intended.

2.4 Operating Environment

The VetCare Online Vet Clinic Management System operates in a standard web-based environment, and its deployment involves the following key components:

1. Hardware Platform:

- **Server Requirements:** The application is deployed on a server, which can be cloud-based or on-premise. Minimum specifications include a quad-core CPU, 8 GB RAM, and 100 GB SSD storage.
- **Continuity:** An estimate of how much compute resources are required can be predicted once the app is live and an estimate of the number of users per hour is known according to which changes will be made to the run time environment and available resources compute resources.
- **Client Devices:** The system is accessible on any device with a web browser that supports JavaScript and is able to render HTML, including desktops, laptops, tablets, and smartphones.

2. Operating System:

- **Server Operating System:** Compatible with Linux distributions (e.g., Ubuntu 20.04 LTS).
- **Client Operating Systems:** Users can access the system on any device that allows users to make HTTP requests, execute JavaScript and display HTML.

3. Web Server:

- The system is hosted on an HTTP server, which could be Apache Tomcat, Jetty, or any compatible server that supports Java-based web applications.

4. Software Components:

- **User Interface:** Built using Spring MVC, the UI is rendered as HTML/CSS in the browser.
- **Business Logic:** Handled by the service layer within the Spring Boot framework.

- **Persistence Layer:** Manages data through a MySQL or Oracle database.
 - **Data Exchange:** JSON is used for communication between the front end and back end.
5. **Development and Testing Tools:**
- **Code Repository and CI/CD:** Managed via GitHub and GitHub Actions.
 - **Build Tool:** Maven is used for project management and builds.
 - **Unit Testing:** JUnit5 is used for testing.
6. **Network Environment:**
- Users connect to the VetCare system through a web browser over the internet, with the HTTP server processing requests and interacting with the database.

This operating environment ensures that the VetCare system runs smoothly across various devices and platforms, providing a reliable and consistent user experience.

2.5 Design and Implementation Constraints

1. Technology Stack:

Mandatory Technologies: The project requires the use of Java 17 or later, Spring Boot for the web framework, and MySQL or Oracle as the database management system. These technologies have been chosen based on project requirements and must be adhered to throughout the development process.

Build and Dependency Management: Maven is mandated as the build tool for managing dependencies and project builds, restricting the use of other tools like Gradle.

2. Monolithic Architecture Constraints:

Single Deployment Unit: The system is built as a monolithic application, where all components are packaged and deployed together. This complicates scalability and makes it difficult to update or modify specific parts of the application independently.

Tight Coupling: Components within the monolithic architecture are tightly coupled, meaning changes in one part of the system can affect others, increasing the complexity of updates and testing.

Scalability Challenges: The entire application must be scaled as a single unit, potentially leading to inefficient resource use when only certain parts of the application require scaling.

Deployment Complexities: Any change requires redeployment of the entire application, leading to longer downtime and more complex deployment processes as the system grows.

Maintenance Challenges: Maintaining a monolithic architecture can become increasingly difficult over time. It becomes harder for any single developer to understand the entire program, leading to reliance on a few key individuals with deep system knowledge.

Big Ball of Mud: Over time, the monolithic architecture can lead to a "Big Ball of Mud," where the system becomes entangled with dependencies, making it difficult to manage, test, and extend.

Testing and Deployment Delays: The tightly coupled nature of the monolithic architecture means that testing and deployment can take longer, slowing down release cycles.

3. MVC Structure Constraints:

Complexity in Large Applications: While MVC (Model-View-Controller) architecture is effective for separating concerns, it can lead to increased complexity as the application grows. Maintaining clear boundaries between models, views, and controllers can become challenging.

Dependency Management: Controllers in MVC often handle multiple dependencies and business logic, which can make the code harder to maintain and test.

Testing Challenges: The MVC structure requires thorough testing of each layer independently. Due to interdependencies, achieving high test coverage and isolating tests can be challenging.

4. Testing and Scalability Constraints:

Limited User Base for Testing: As a university project, the system will not undergo mass testing by real users, limiting the ability to accurately gauge the system's performance, scalability, and user experience under load.

Scalability Uncertainty: Due to the lack of mass user testing, the system's scalability is unverified. There is no practical data on how well it scales with a significant increase in users or data transactions.

5. Hardware Limitations:

Server Specifications: The application must run on servers with a minimum specification of 8 GB RAM and a quad-core CPU, which may limit scalability under high user loads.

Client Devices: The application should be optimized for devices with lower-end specifications (e.g., 4 GB RAM), affecting the complexity of the user interface and client-side processing.

6. Interface Constraints:

Database Connectivity: The system is designed to connect only to MySQL or Oracle databases, limiting flexibility to integrate with other database systems.

HTTP Server: The application must be hosted on an HTTP server compatible with Java-based web applications (e.g., Apache Tomcat, Jetty).

7. Security Considerations:

Data Privacy: The system must ensure secure handling of user data, particularly sensitive information related to pets' medical records, requiring strong encryption protocols for data at rest and in transit.

Authentication and Authorization: Role-based access control (RBAC) must be implemented to ensure users only access information and features relevant to their role.

8. Design Conventions and Standards:

Coding Standards: The project adheres to standard Java coding conventions, using JUnit5 for testing and GitHub for version control. All code must be documented, and pull requests reviewed before merging.

Responsive Design: The user interface must be responsive, functioning smoothly on various devices and screen sizes, limiting the complexity of the UI design.

9. Project Scope and Time Constraints:

Limited Development Time: Development time is constrained by the academic calendar, requiring prioritization of core functionalities such as appointment scheduling, medical record management, and prescription handling.

Team Size and Skills: The project team consists of students with varying levels of experience, limiting the use of complex or less familiar technologies.

10. Regulatory Compliance (If Applicable):

Data Protection Regulations: While not strictly enforced at the university level, the project should consider basic data protection principles (e.g., GDPR) for potential real-world applications.

11. Post-Delivery Maintenance Responsibility:

Maintenance Responsibility: As a university project, the development team is responsible for initial development, testing, and deployment. In a real-world scenario, another organization or team would assume long-term maintenance, requiring comprehensive documentation and relevant technical skills for ongoing support.

2.6 User Documentation

The following user documentation components will be delivered along with the VetCare platform:

- **Video Tutorials:** We will provide video tutorials that demonstrate how to use the various features of the application. These tutorials will offer users a visual and step-by-step guide to understanding the platform's functionalities.

- **Interactive User Onboarding Guide:** An interactive User Onboarding Guide will be included to assist new users in familiarizing themselves with the platform. This guide will automatically appear when users first access the site or when new features are introduced. It offers step-by-step instructions and highlights key features of the platform, guiding users with "Next" and "Previous" buttons. This interactive element ensures that users can quickly and effectively learn how to navigate and use the platform, thereby enhancing their overall experience.
- **Frequently Asked Questions (FAQ):** An FAQ section will be included as part of the user documentation. This section will address common questions and concerns that users might have while using the platform. The FAQ will cover a range of topics, from basic usage to more advanced features, and will provide quick, concise answers. The FAQ is intended to help users find solutions to common issues without needing to consult the full user manual or contact support, thereby improving their overall experience with the platform.
- **Contact Web Administrator:** Users will have access to a dedicated "Contact Web Administrator" section within the platform. This feature will provide users with a direct way to reach out to the web admin team for assistance with technical issues, account problems, or other concerns that cannot be resolved through the user manual or FAQ. The contact section will include a form for submitting inquiries, as well as information on how to reach the support team via email or phone. This ensures that users have access to personalized support when needed, further enhancing their experience with the VetCare platform.

2.7 Assumptions and Dependencies

The development of the VetCare Online Vet Clinic Management System is based on several assumptions and dependencies that, if incorrect or altered, could impact the requirements and overall success of the project. These include:

Assumptions:

- **Consistent Development Environment:** It is assumed that all development team members will have access to a consistent development environment, including the necessary software, hardware, and network resources. Any variation in these environments could lead to integration issues or unexpected bugs.
- **Availability of Required Tools and Technologies:** The project assumes that all required tools and technologies (e.g., Java 17, Spring Boot, MySQL, Docker) will be available and functional throughout the development process. Any disruption in access to these tools, such as licensing issues or software deprecations, could delay the project.
- **Stable Network Connectivity:** It is assumed that the development, testing, and deployment environments will have stable and reliable network connectivity. This is critical for accessing cloud-based resources, collaborating via GitHub, and deploying the application.
- **Third-Party Services and APIs:** The project assumes that any third-party services or APIs (e.g., SMTP servers for email notifications) used in the system will remain stable and available throughout the project lifecycle. Changes or discontinuation of these services could require significant rework.

- **User Familiarity with Web Applications:** It is assumed that the end-users (pet owners, veterinarians, and clinic administrators) have basic familiarity with using web applications. Extensive user training is not anticipated as part of this project.

Dependencies:

- **Database Management System:** The project depends on the availability and reliability of MySQL or Oracle as the database management system. Any issues with these databases, such as version incompatibilities or performance problems, could impact the system's functionality.
- **Docker and Containerization:** The project relies on Docker for containerizing the monolithic application to ensure environment consistency across the development, testing, and production stages. Any issues with Docker or related tools could affect deployment and testing processes.
- **External Libraries and Frameworks:** The project depends on external libraries and frameworks (e.g., Spring Boot, JUnit) for critical functionalities. If these components receive significant updates or become deprecated, the project may need to adapt, potentially affecting timelines and stability.
- **GitHub for Version Control and CI/CD:** The project assumes that GitHub will be used consistently for version control and continuous integration/continuous deployment (CI/CD) throughout the development process. Any interruptions in GitHub's service or issues with integrating CI/CD pipelines could impact the project's progress.
- **Project Team Availability:** The project assumes that all team members will be available and able to contribute according to the project plan. Any unforeseen circumstances (e.g., illness, conflicting academic responsibilities) that reduce team availability could delay the project.

3. External Interface Requirements

3.1 User Interfaces

The VetCare platform has been designed for all users that require an all in one application to manage their pet's health. It includes features such as a display of all previous medical records, prescriptions, appointments as well as account details. Our platform has an emphasis on intuitive design, allowing for ease of use amongst our users, through our standardised navigation systems across our pages. The key features of our platform are as follows:

- **Dashboard Interface:** The homepage includes a user friendly dashboard that utilises simplicity for ease of navigation for the user. The use of tiles helps us achieve this, whereby the user is prompted with pages they can click on depending on their needs.
- **Navigation Bar and Footer:** We have implemented a consistent navigation and footer across all our pages serving as a default way to navigate through the entire site. This will reduce the chance of users getting lost on the platform as well as help standardise the application, enhancing user experience. Furthermore, for mobile phone screen size

constraints, we have developed a condensed, drop-down menu for our navigation bar that overcomes the issue of size constraints.

- **Page Interfaces:** The platform will have a variety of different interfaces that the user will encounter across the website, we have standardised the tables across both desktops and mobiles to help ensure consistency for the user. This is in an attempt to simplify any searches for information that the user may require.
- **Calendars:** We have implemented a calendar UI design that has a focus on ease of use for our users. To achieve this, we mimicked calendars that users would already be familiar with in real world scenarios, making booking appointments simple and familiar.
- **Buttons:** Each of our button components follow the same structure and design, this is to aid the user in drawing their attention to focal points to help them navigate through the site.
- **User Onboarding using Tooltips:** The VetCare platform includes an interactive **User Onboarding Guide** that uses tooltips to assist new users in familiarizing themselves with the platform. This guide appears when users first access the site or when new features are introduced. It provides step-by-step instructions and highlights key features of the platform, with "Next" and "Previous" buttons to guide users through each step. This interactive element ensures that users can quickly learn how to navigate and use the platform effectively, enhancing their overall experience.

3.2 Hardware Interfaces

The VetCare application is built on a monolithic architecture, interfacing with various hardware components through standard protocols. This section outlines the logical and physical characteristics of the hardware interfaces that the application interacts with.

Supported Device Types

- **User Devices:**
 - **Desktops and Laptops:** The application supports modern web browsers such as Google Chrome, Mozilla Firefox, Safari, and Microsoft Edge running on desktop and laptop computers. These devices connect to the application through HTTP/HTTPS protocols and render HTML, CSS, and JavaScript content provided by the server.
 - **Mobile Devices:** The system is accessible on mobile devices running Android and iOS. The user interface adapts responsively to different screen sizes and orientations, ensuring a consistent experience across all devices.
- **Servers:**
 - **HTTP Server:** The application leverages an HTTP server like Jetty or Apache to handle incoming requests from user devices. This server processes HTTP/HTTPS requests and serves the required web pages.
 - **Application Server:** The business logic of the VetCare application is managed by an application server running Java 17, utilizing the Spring Boot framework. This server manages core functionalities, including business operations and database interactions.

- **Database Server:** The application's persistence layer connects to a relational database server (e.g., Oracle, MySQL) to store and manage application data. The database server is a critical component, ensuring reliable data storage and retrieval.

Data and Control Interactions

- **User Interface to HTTP Server:** User interactions are handled by the HTTP server, which processes user requests and serves dynamic content. Communication between the client's web browser and the server is managed through HTTP/HTTPS protocols.
- **Business Logic to Persistence Layer:** The business logic interacts with the database via the Repository/DAO pattern, performing CRUD operations on the database. These interactions are managed through JDBC or another ORM tool within the Spring Boot application.
- **Peripheral Devices:** The system may interface with peripherals like printers and scanners through standard operating system interfaces. This enables functionalities such as printing documents or uploading scanned files.

Communication Protocols

- **HTTP/HTTPS:** The primary protocol for communication between client devices and the HTTP server, with HTTPS ensuring secure, encrypted data transmission.
- **TCP/IP:** Used for communication between the application server and the database server, ensuring reliable data transfer.
- **JSON:** Used for data interchange between the front-end and back-end components, particularly for transferring structured data in a lightweight format.

Hardware Requirements

- **Client Devices:**
 - **Desktops/Laptops:** Requires a modern web browser, 4GB RAM, Intel i3 processor or equivalent, 250GB HDD/SSD.
 - **Mobile Devices:** Requires Android or iOS with 2GB RAM, 16GB storage.
- **Servers:**
 - **HTTP Server:** Requires 8GB RAM, Quad-Core processor, 100GB SSD storage.
 - **Application Server:** Requires Java 17, 8GB RAM, Quad-Core processor, 100GB SSD.
 - **Database Server:** Requires 16GB RAM, Quad-Core processor, 500GB SSD for Oracle, MySQL, or equivalent RDBMS.

3.3 Software Interfaces

Overview

The VetCare application is containerized using Docker, which allows all components—including the application server, database, and other services—to run in isolated and consistent environments. This section describes the connections between the VetCare application and other specific software components, detailing the data exchanges, communication protocols, and integration points.

Connections to Specific Software Components

- **Operating System:**
 - The application runs within Docker containers, which are OS-agnostic, but typically run on a Linux-based host in production environments. The containers ensure that the application and all its dependencies are consistent across different environments.
- **Database:**
 - **MySQL/Oracle:** The persistence layer of the VetCare application interacts with a MySQL or Oracle database, which is also containerized using Docker. The application uses JDBC (Java Database Connectivity) through Spring Data JPA to perform CRUD operations. Data items include user profiles, medical records, appointments, and transactional data.
- **Web Framework:**
 - **Spring Boot (Java 17):** The core application is implemented using Spring Boot, which is responsible for handling HTTP requests, executing business logic, and interacting with the database. The application's web layer is managed by Spring MVC, rendering HTML/CSS for the user interface.
- **JSON Communication:**
 - **Front-End to Back-End:** Data interchange between the user interface (front-end) and the application logic (back-end) is facilitated using JSON. This format is used for transmitting data such as form inputs, API responses, and other dynamic content.
- **CI/CD Pipeline:**
 - **GitHub Actions:** Continuous Integration and Continuous Deployment (CI/CD) are managed via GitHub Actions. The pipeline automates testing (using JUnit 5), building (using Maven), and deploying the application into Docker containers.
- **Code Repository:**
 - **GitHub:** The source code is managed using GitHub, with version control and collaboration features enabling smooth development workflows.

Services and Communication

- **HTTP/HTTPS:**
 - Communication between client devices (e.g., web browsers) and the server is handled via HTTP/HTTPS, ensuring secure data transmission.
- **API Documentation:**
 - API endpoints are documented using Swagger or Spring REST Docs, providing clear guidelines for how different services interact within the application.

Data Sharing Mechanism

- **Session Management:**
 - **Spring Security:** User session data is managed through Spring Security, ensuring that user authentication and session states are maintained securely across different components of the application.
- **Containerized Environment:**
 - All components (application server, database, etc.) run in Docker containers, ensuring isolated and consistent environments. Data sharing and communication

between containers (e.g., the application server and the database) occur over Docker's internal networking, typically using standard network protocols like TCP/IP.

External APIs and Integration Points

- **Payment Gateways (if applicable):**
 - The application can integrate with external payment gateways (e.g., Stripe, PayPal) through their APIs, facilitating online transactions securely.

3.4 Communications Interfaces

Overview

The VetCare system utilizes a variety of communication methods to ensure users receive timely updates and alerts. These methods include email, SMS, web browser notifications, and mobile push notifications. The communication functions are designed to keep users informed about appointments, medical records, prescription renewals, and other relevant updates.

Communication Methods

Email:

- **Purpose:** Will be utilised as the standard form of contact with the user for all functions such as booking confirmations and prescriptions.
- **Formatting:** Emails will use an HTML structure, ensuring consistency amongst users and their emailing platforms.
- **Protocol/Security:** We will use SMTP for sending our emails for reliability, as well as TLS to ensure a secure communication line with all users.

SMS:

- **Purpose:** Allows for a more personal and direct notification system, focusing on critical alerts such as upcoming appointments and deliveries.
- **Formatting:** Will be integrated using HTTPS to ensure a seamless communication between our platform and SMS service.
- **Protocol/Security:** TLS will be used to ensure a secure communication line with all users.

Web Browser Notifications:

- **Purpose:** Delivers in-app alerts to our users to notify them of changes and updates on the platform, including confirmations of purchases and bookings.
- **Formatting:** Notifications will be succinct and direct to the user, using JavaScript to notify them of any reminders/notifications. This enables real time communication with the user.
- **Protocol/Security:** All web browser notifications will be using HTTPS to maintain data security for our users.

Communication Standards and Protocols

- **HTTP/HTTPS:** All web-based communications, including email, SMS, and notifications, use HTTP/HTTPS protocols to ensure secure data transmission between the server and client.
- **SMTP:** Emails are sent using SMTP, with TLS encryption to protect the contents during transit.
- **Web Push Protocol:** Used for sending browser notifications, ensuring compatibility across different web browsers and secure delivery.
- **Firestore Cloud Messaging (FCM)/Apple Push Notification service (APNs):** These protocols are used for sending mobile push notifications securely to Android and iOS devices, respectively.

Security and Encryption

- **TLS Encryption:** Ensures that all emails are encrypted during transmission to prevent unauthorized access.
- **HTTPS:** Used for all web-based communications to protect data integrity and confidentiality.
- **Data Transfer Rates:** While specific rates depend on the network, the system is optimized to ensure timely delivery of all communications without noticeable delays to the end user.
- **Synchronization Mechanisms:** The system uses periodic checks and acknowledgments to ensure that notifications and alerts are synchronized across devices and communication channels.

4. Nonfunctional Requirements

4.1 Performance Requirements

Performance requirements are crucial to ensure that the VetCare platform provides a seamless and efficient experience for both pet owners and veterinary clinic staff. These requirements must be met under various operating conditions to maintain the integrity and usability of the system.

1. **Response Time:**
 - **Normal Load:** The application should respond to user inputs and requests within **2 seconds** under normal load conditions.
 - **Peak Load:** During peak load times, response times should not exceed **5 seconds**. This ensures a fluid user experience and minimizes frustration during busy periods.
2. **System Availability:**
 - The VetCare application should maintain **99.9% uptime**, allowing for scheduled maintenance windows, which should be communicated to users in advance. High availability is critical to ensure users can access the service when needed, especially during emergencies.
3. **Concurrent Users:**

- The system must support up to **1,000 concurrent users** without degradation in performance. This requirement is based on anticipated peak usage scenarios, ensuring that the system remains responsive during high-traffic periods.
- 4. **Data Processing:**
 - The application should process data transactions, such as appointment bookings or medical record updates, within **3 seconds 95%** of the time. This speed is essential to ensure that records are up-to-date and accurate, reflecting changes in real-time, which is crucial for proper healthcare management.
- 5. **Scalability:**
 - VetCare must be designed to scale seamlessly with increases in user base and data volume. The system should support a **50% increase in concurrent users** with proportional increases in infrastructure, without significant loss in performance. This ensures that the platform can accommodate future growth.

4.2 Safety Requirements

Safety is paramount due to the sensitive nature of the data and the critical services provided by the VetCare application. These safety requirements are designed to minimize the risks associated with both human error and system failures, ensuring a secure and reliable user experience.

- **Data Backup and Recovery:**
 - The application must implement robust data backup procedures to prevent data loss in the event of a system failure. Regular backups should be scheduled, with backups stored in a secure, off-site location. Data recovery processes must be in place, tested regularly, and capable of restoring data to its most recent state without data corruption or loss.
- **Error Handling and Logging:**
 - The system must include comprehensive error handling mechanisms to prevent system crashes, data corruption, or unintended operations. All critical errors should be logged with detailed information to facilitate rapid troubleshooting. User activities and system interactions must also be logged to monitor for potential security breaches or operational failures.
- **Regulatory Compliance:**
 - The application must comply with all relevant veterinary medical privacy laws and regulations applicable in the jurisdictions where the system is deployed. For example, in the U.S., the system should comply with the Health Insurance Portability and Accountability Act (HIPAA), which mandates the protection and confidential handling of medical information.
- **Safety Certifications:**
 - The system should aim to obtain and maintain certifications such as [ISO 27001](#), which provides a framework for establishing, implementing, maintaining, and continually improving an information security management system (ISMS). This certification ensures that the system adheres to industry-recognized best practices for security management and comprehensive security controls as outlined in ISO 27001.
- **Preventive Measures:**

- The application must implement preventive measures to protect against data breaches, unauthorized access, and other forms of security threats. This includes regular security audits, vulnerability assessments, and adherence to secure coding practices.

4.3 Security Requirements

Security in the VetCare application is critical to protect both the privacy of users and the integrity of the medical data it handles. The following security requirements are established to ensure comprehensive protection in compliance with Australian regulations.

1. **Authentication and Authorization:**

- **Multi-Factor Authentication (MFA):** Implement MFA to provide an additional layer of security, ensuring that only authorized users can access the system. This may include a combination of passwords, security tokens, or biometric verification.
- **Role-Based Access Control (RBAC):** Define user roles and permissions carefully to ensure that users can only access the information and functionalities relevant to their role (e.g., administrators, veterinarians, pet owners).

2. **Data Encryption:**

- **Data at Rest:** Encrypt all sensitive data stored within the system, including the database, backup files, and any other data storage systems, using strong encryption standards such as AES-256.
- **Data in Transit:** Protect data during transmission using TLS (Transport Layer Security) to prevent unauthorized access and ensure data integrity as it moves between the user's device and the server.

3. **Compliance and Privacy Policies:**

- **[Australian Privacy Act 1988](#):** Ensure compliance with the Australian Privacy Act 1988, which governs the collection, use, storage, and disclosure of personal information. This includes adhering to the Australian Privacy Principles (APPs) that outline how personal information must be managed.
- **Data Breach Notification:** Implement processes to comply with the [Notifiable Data Breaches](#) (NDB) scheme under the Privacy Act, which requires organizations to notify affected individuals and the Office of the Australian Information Commissioner (OAIC) of eligible data breaches.
- **User Data Transparency:** Provide mechanisms for users to view the data stored about them, including the ability to request corrections or deletion in accordance with the Privacy Act.

4. **Security Certifications:**

- **[ISO/IEC 27001 Certification](#):** Pursue ISO/IEC 27001 certification to demonstrate the application's commitment to comprehensive information security management, which is recognized internationally and can bolster trust with Australian users.
- **[Australian Signals Directorate \(ASD\) Essential Eight](#):** Implement the Essential Eight mitigation strategies as recommended by the ASD to improve security posture and protect against cybersecurity threats.

5. **Security Monitoring and Incident Response:**

- **Continuous Monitoring:** Implement continuous security monitoring to detect and respond to potential security threats in real-time, including unauthorized access attempts and suspicious activities.
- **Incident Response Plan:** Develop and maintain a robust incident response plan that aligns with Australian regulatory requirements, outlining procedures for responding to security breaches, including containment, mitigation, and notification protocols.

4.4 Software Quality Attributes

The VetCare application's development will focus on several key quality attributes to ensure that it meets the expectations of both users and developers. These attributes must be specific, quantifiable, and verifiable to contribute effectively to the overall quality and reliability of the application.

1. Reliability:

- **Uptime:** The application must maintain *99.9% uptime*, with downtime limited to scheduled maintenance windows. Any unexpected downtime must be resolved within *1 hour* to minimize disruption to users, ensuring continuous availability of critical services.

2. Usability:

- **Ease of Use:** The user interface should be designed to be intuitive and user-friendly, requiring no more than *30 minutes of training* for new users. Usability should be assessed through regular user satisfaction surveys, with a target satisfaction score of *90%* or higher. User feedback will be actively used to make iterative improvements.

3. Performance:

- **Responsiveness:** The application must handle user interactions efficiently, ensuring that response times do not exceed *2 seconds* under normal load conditions. During peak load times, response times should not exceed *5 seconds* to maintain a seamless user experience.

4. Maintainability:

- **Modularity:** The system should be built using a modular architecture, allowing for straightforward updates and maintenance. The code should be well-documented and adhere to clean coding practices, making it easy for developers to manage dependencies and implement updates with minimal impact on the overall system.

5. Portability:

- **Cross-Platform Compatibility:** The application must be easily deployable across multiple environments, including major operating systems (Windows, macOS, Linux) and web browsers (Chrome, Firefox, Safari, Edge). The platform should also support mobile and desktop use with minimal configuration changes required for each environment.

6. Testability:

- **Automated Testing:** The application must support comprehensive automated testing, including unit tests, integration tests, and regression tests. These tests should cover *100% of critical functionalities* and aim for at least *80% overall code coverage*. Detailed logging must be implemented to facilitate debugging and validation of test results.

4.5 Business Rules

Business rules dictate the operational guidelines and principles that govern the functions of the VetCare system. These rules are not only essential for maintaining the order and integrity of the system but also imply certain functional requirements necessary for enforcing these rules.

- **Role-Based Access Control (RBAC):** Access to different parts of the system should be governed by user roles. For example:
 - Veterinary Staff can access and update medical records, manage appointments, and process prescriptions.
 - Pet Owners are restricted to viewing and managing their pets' appointments, medical records, and prescriptions.
 - Administrators have overarching access to the entire system for management and maintenance purposes.
- **Data Privacy:** Any access to pet medical records must comply with relevant privacy laws and regulations. Only authorized users (vet staff and pet owners) can view specific medical records, and the sharing of information must be explicitly authorized by the pet owner.
- **Appointment Booking Rules:** Appointments can only be booked if available slots are confirmed. Automatic checks should prevent double bookings and provide alternatives when slots are not available.
- **Prescription Access:** Only licensed veterinary professionals can create or modify prescription details. Pet owners can view and request refills but cannot alter the prescription data.
- **Audit Trails:** All significant actions within the system, such as access to sensitive information or changes to medical records, must be logged to ensure traceability and accountability.

5. Other Requirements

Other requirements include:-

Animal Welfare Regulations

- **State and Territory Legislation:**
 - **Overview:** Animal welfare in Australia is governed by laws specific to each state and territory. These laws ensure that animals are treated humanely and that veterinary practices comply with welfare standards.
 - **Requirements:**
 - **Veterinary Practices:** Ensure that all veterinary services offered through your platform comply with state and territory regulations. This includes ensuring that the veterinarians are licensed and adhere to professional standards.

- **Advertising and Information:** Ensure that any information related to pet care, treatment options, or breed-specific issues provided through the platform is accurate and not misleading. Misrepresentation can lead to legal liabilities.

Professional Regulations

- **Veterinary Surgeons Acts (State-Specific):**
 - **Overview:** Each state in Australia has its own Veterinary Surgeons Act that regulates the practice of veterinary medicine. These acts establish requirements for the licensing of veterinarians and the standards they must adhere to.
 - **Requirements:**
 - **Verification of Credentials:** Your platform must verify the credentials of veterinarians listed on the platform to ensure they are properly licensed to practice in their respective states.
 - **Professional Conduct:** Ensure that the services provided by veterinarians through your platform comply with the professional conduct standards as defined by state legislation.

Animal Registration and Microchipping

- **State and Local Government Regulations:**
 - **Overview:** Regulations around animal registration and microchipping vary by state and local government. In many areas, it is mandatory for pets to be registered and microchipped.
 - **Requirements:**
 - **Integration with Local Regulations:** Your application should provide users with information about the registration and microchipping requirements in their area. It may also be beneficial to offer features that help users manage these aspects (e.g., reminders for registration renewals).

Advertising and Marketing Regulations

- **Spam Act 2003:**
 - **Overview:** The Spam Act regulates the sending of commercial electronic messages, including emails and SMS.
 - **Requirements:**
 - **Consent:** Obtain explicit consent from users before sending marketing communications.
 - **Unsubscribe Mechanism:** Include a clear and functional unsubscribe mechanism in all marketing communications.
 - **Identification:** Ensure that all communications clearly identify your business as the sender.

Intellectual Property

- **Trademark and Copyright Laws:**

- **Overview:** If your platform uses any proprietary content, brand names, or logos, ensure that these are properly protected under trademark or copyright laws.
- **Requirements:**
 - **Trademark Registration:** Consider registering your platform's name, logo, and other branding elements as trademarks to protect your intellectual property.
 - **Copyright Compliance:** Ensure that any content provided on your platform (e.g., educational materials, images) does not infringe on third-party copyrights.
- Any publicly available data on the application is the property of the application owner and is not meant to be scraped/used in any way/form on any public or private platform without the permission of the application owner.

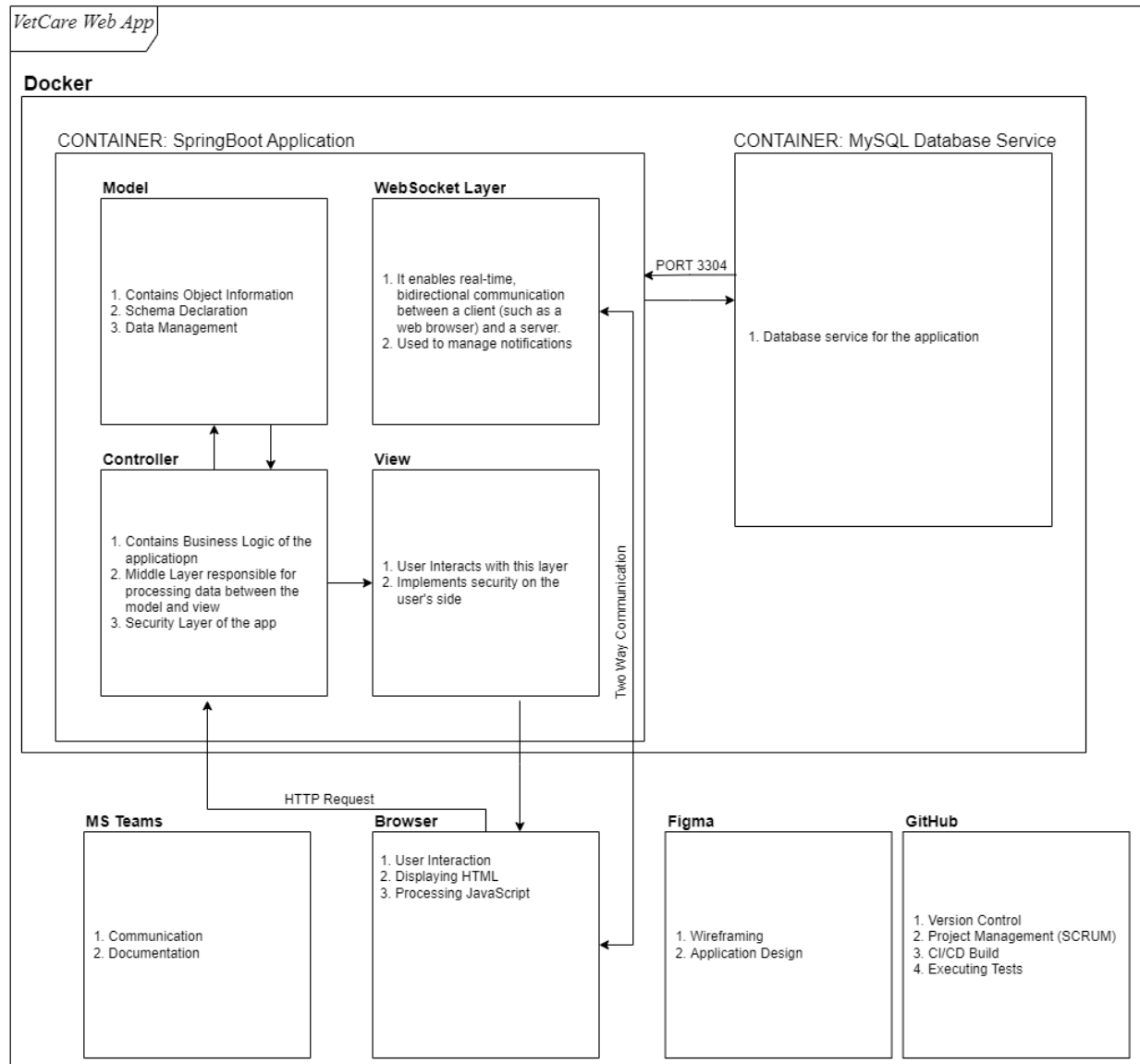
Contractual Obligations

- **Terms of Service and User Agreements:**
 - **Overview:** Clearly define the terms of service and user agreements that govern the use of your platform.
 - **Requirements:**
 - **Liability Limitations:** Include clauses that limit your platform's liability in cases where services provided by third-party veterinarians do not meet user expectations.
 - **Dispute Resolution:** Establish a dispute resolution process to handle conflicts between users and service providers.

6. System Architecture

Figure 3: Architecture Diagram

Layered Monolithic Application Architecture Diagram



The VetCare application follows a monolithic architecture, encapsulating all components within a single, cohesive Docker container environment. This approach simplifies deployment and scaling while ensuring that all parts of the application can efficiently interact within the same runtime environment.

Fundamental Decisions and Solution Strategies:

Technology Stack:

- **Spring Boot (Spring MVC Framework):** The application is built using Spring Boot, leveraging the Spring MVC framework for handling web requests and responses. This framework was chosen for its robustness, ease of use, and extensive support for web-based applications.
- **Docker:** The entire application is containerized using Docker, enabling consistent deployment across various environments and simplifying the management of dependencies.
- **MySQL Database:** MySQL serves as the relational database, selected for its reliability, scalability, and wide adoption, making it a solid choice for handling structured data.

Top-Level Decomposition:

The system is decomposed into several key components:

- **User Interface (UI):** Delivered through a web browser, it interacts with the application via HTTP requests.
- **Spring MVC Application:** Comprising the HTTP server, controller, model, view, service layer (business logic), and persistence layer (repository/DAO), this component handles all business operations and user interactions.
- **Notification Service:** Implements real-time notifications using WebSocket connections, ensuring that users receive updates promptly within the browser.
- **Database (MySQL):** Manages all data storage and retrieval operations, interfaced by the persistence layer of the application.

Approaches to Achieve Top-Quality Goals:

- **Performance and Scalability:** The monolithic architecture, when combined with Docker, allows the application to be easily scaled by replicating the container. The use of MySQL ensures that data operations are efficient, even under high load.
- **Reliability and Availability:** The architecture is designed to achieve high reliability, with a robust error-handling mechanism within the service layer. Docker's containerization provides consistent environments, reducing the risk of deployment issues.
- **Real-Time Communication:** Real-time notifications are handled via WebSocket connections, allowing the application to push updates to users instantly without requiring them to refresh their browsers.

Relevant Organizational Decisions:

- **Centralized Development:** The monolithic approach was chosen to simplify development and maintenance, allowing a centralized team to work on the entire application without the complexities of microservices.
- **Deployment Strategy:** By using Docker, the application can be deployed consistently across different environments (development, testing, production) with minimal configuration changes, ensuring that the application behaves the same way regardless of the deployment environment.

6.1 Architecture Overview

The VetCare application is structured as a monolithic system encapsulated within a Docker container. The system is built using a hierarchical approach where the main components (white boxes) contain subcomponents (black boxes), providing a clear abstraction of the source code and its organization.

1. VetCare Application (White Box)

- The overall VetCare application is contained within a Docker environment, ensuring consistency across development, testing, and production environments.
- 1.1 Docker Container (White Box)
 - The Docker container serves as the environment for the entire application, encapsulating all components and ensuring that dependencies and configurations are managed consistently.
 - 1.1.1 Spring Boot Application (White Box)
 - The Spring Boot Application is the core of the VetCare system, implementing the MVC (Model-View-Controller) pattern to manage user interactions and business logic.
 - 1.1.1.1 HTTP Server (Black Box)
 - Manages incoming HTTP requests from the user's browser and routes them to the appropriate controllers.
 - 1.1.1.2 Controller Layer (Black Box)
 - Handles the incoming requests by interacting with the service layer (business logic) and preparing the data needed for the view layer.
 - 1.1.1.3 Model Layer (Black Box)
 - Represents the data structures used within the application. It interacts with the persistence layer to fetch and update data stored in the database.
 - 1.1.1.4 View Layer (Black Box)
 - Responsible for rendering the user interface by using HTML and CSS. The view layer presents the data processed by the model layer to the user.
 - 1.1.1.5 Business Logic (Service Layer) (Black Box)
 - Implements the core business rules and logic of the application. It processes data received from the controller layer and interacts with the persistence layer for data operations.
 - 1.1.1.6 Persistence Layer (Repository/DAO) (Black Box)
 - Manages database interactions by interfacing with the MySQL database. It performs CRUD (Create, Read, Update, Delete) operations on the data.
 - 1.1.1.7 Notification Service (Black Box)
 - Responsible for managing real-time notifications. It pushes updates to the user interface via WebSocket connections.
 - 1.1.1.8 WebSocket Connection (Black Box)

- Manages real-time, bi-directional communication between the server and the user's browser, ensuring that users receive instant updates.
- 1.1.2 Database (MySQL) (White Box)
 - The MySQL database serves as the primary data store for the application, handling all persistent data.
 - 1.1.2.1 Database Tables (Black Box)
 - Represents the structured data stored in various tables, such as Users, Pets, Appointments, and Medical Records. Each table corresponds to a specific model in the application.
 - 1.1.2.2 SQL Queries and Procedures (Black Box)
 - Encapsulates the SQL queries and stored procedures that the persistence layer uses to interact with the database, ensuring efficient data retrieval and manipulation.

2. User Interface (White Box)

- The user interface is the front-end component that users interact with, accessed through a web browser.
- 2.1 Browser (White Box)
 - The browser acts as the client interface for accessing the VetCare application over the network.
 - 2.1.1 HTML/CSS (Black Box)
 - Defines the structure and styling of the web pages that are rendered in the user's browser, ensuring a consistent and user-friendly interface.
 - 2.1.2 JavaScript (Black Box)
 - Manages client-side logic, enabling dynamic interactions within the web pages, such as form validations, AJAX requests, and real-time updates.

3. Network Connection (White Box)

- The network connection facilitates communication between the user's browser and the Docker container hosting the VetCare application.
- 3.1 HTTP Requests and Responses (Black Box)
 - Represents the communication protocol for transmitting data between the client and server, handling standard web requests and responses.
- 3.2 WebSocket Connection (Black Box)
 - Manages real-time updates from the server to the client, allowing for instant notifications and updates in the browser without needing to refresh the page.

6.2 Architectural Decisions

In developing the VetCare application, several key architectural decisions were made that are important due to their impact on the system's functionality, cost, scalability, and risk management. Below are some of these decisions, along with the rationales behind them:

1. Monolithic Architecture Decision

Decision: The decision was made to use a [monolithic architecture](#) instead of a microservices architecture.

Rationale:

Simplicity: A monolithic architecture is simpler to design, develop, and manage for a small to medium-sized application like VetCare. It allows all components to be tightly integrated within a single codebase, reducing the overhead of managing multiple services.

Cost-Effectiveness: Developing and deploying a monolithic application is typically less expensive than a microservices architecture, which requires more complex infrastructure, orchestration, and monitoring tools.

Performance: A monolithic architecture can offer better performance in this context, as it avoids the latency and overhead associated with inter-service communication in microservices.

2. Use of Docker for Containerization

Decision: The entire VetCare application is containerized using [Docker](#).

Rationale:

Consistency Across Environments: Docker ensures that the application behaves the same across development, testing, and production environments by encapsulating all dependencies and configurations within the container.

Scalability: While VetCare uses a monolithic architecture, Docker allows the application to be easily scaled horizontally by deploying additional container instances as needed.

Simplified Deployment: Docker simplifies the deployment process, allowing for continuous integration and continuous deployment (CI/CD) pipelines to be easily implemented, reducing the risk of deployment failures.

3. Spring Boot and Spring MVC Framework

Decision: The application is built using [Spring Boot](#) with Spring MVC for web framework functionality.

Rationale:

Mature Ecosystem: Spring Boot is a mature and widely-used framework that provides comprehensive support for building enterprise-grade applications. It comes with built-in tools and libraries for handling web requests, business logic, security, and data persistence.

Rapid Development: Spring Boot's convention-over-configuration approach accelerates development by minimizing boilerplate code and simplifying configuration.

Community Support: The large Spring community ensures that there is extensive documentation, libraries, and third-party integrations available, reducing development time and risks associated with less established frameworks.

4. MySQL as the Primary Database

Decision: [MySQL](#) was selected as the primary relational database for VetCare.

Rationale:

Reliability and Performance: MySQL is a proven, reliable, and performant relational database system that can handle the transactional workload of VetCare, ensuring data integrity and fast query performance.

Cost-Effectiveness: MySQL is open-source and free to use, making it a cost-effective choice for the project while still offering enterprise-level features.

Scalability: MySQL can scale both vertically (on a single server) and horizontally (using replication), providing flexibility as the application's data and user base grow.

5. Real-Time Notifications via WebSockets

Decision: Real-time notifications are implemented using [WebSockets](#).

Rationale:

User Experience: WebSockets allow for instant communication between the server and the client, providing users with real-time updates without the need for page refreshes. This enhances the user experience by ensuring timely updates, especially for critical features like appointment reminders and prescription alerts.

Performance: WebSockets are more efficient than HTTP polling or long-polling, as they maintain a single open connection between the client and server, reducing overhead and improving responsiveness.

6. CI/CD Pipeline with GitHub Actions

Decision: The CI/CD pipeline is managed using GitHub Actions.

Rationale:

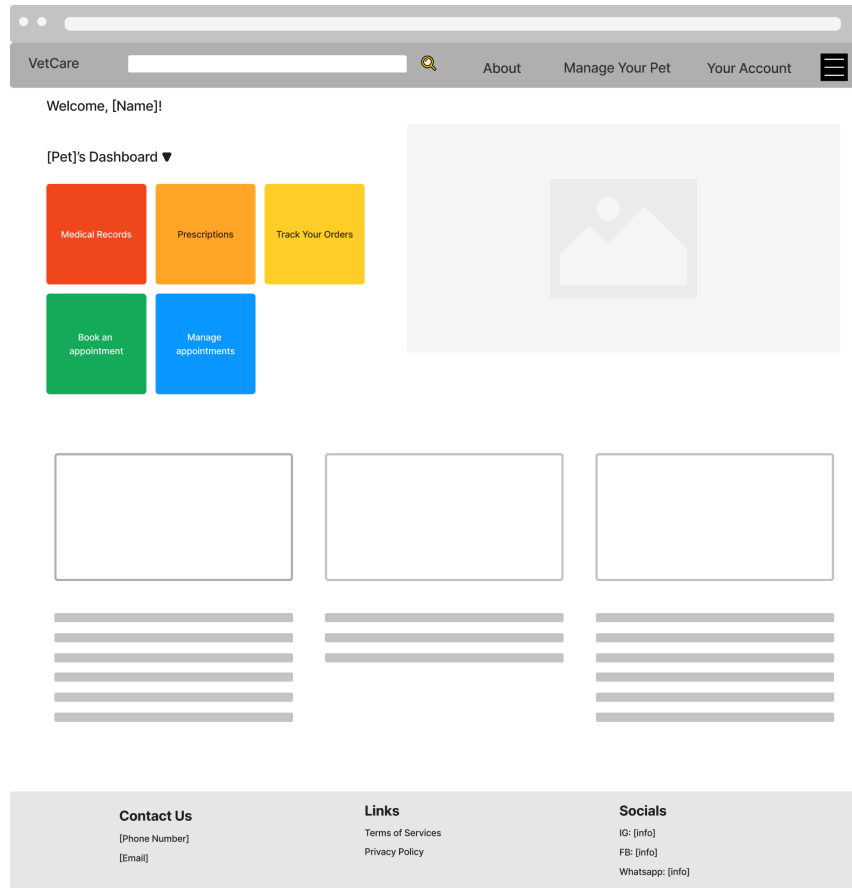
Integration with Codebase: GitHub Actions is tightly integrated with the GitHub repository, allowing seamless automation of build, test, and deployment processes whenever changes are pushed to the codebase.

Flexibility and Customization: GitHub Actions provides a flexible framework that can be easily customized to fit the specific needs of the VetCare application's development and deployment workflow.

Cost Considerations: As part of the GitHub ecosystem, GitHub Actions is cost-effective, especially for smaller teams, as it avoids the need for additional CI/CD tools or platforms.

7. User Interface Design

Figure 4: Dashboard Wireframe



The homepage dashboard for the VetCare website has a comprehensive and user friendly interface that is simple and minimalistic, adhering to the 8th Nielsen's heuristic. At the top, there is a header with relevant navigation links, a search bar and a hamburger dropdown, which helps to maintain the visibility of system status, and also allows easy navigation across the application. This header will also be consistent throughout the rest of the application.

Panning down, there is a welcome text that will display a welcome message to the logged in user, and below that, and dropdown bar that allows the user to switch between their pets if they have multiple pets, to allow for flexibility and efficiency when managing pets.

The dashboard itself is structured in blocks, with each block representing a key function of the application, such as accessing medical records, managing prescriptions, booking and managing appointments, and tracking your order. These blocks utilise vibrant and contrasting colours, which enhance the aesthetic appeal, but also allows the user to clearly identify the difference between each feature and choose the one they would like to use with ease.

To the right of the dashboard, an image placeholder is used to allow users to manually upload pictures of their pets, to add customisation to the page to make the page more engaging to the user.

Scrolling down, a section of the homepage is reserved for personalised education articles. This area is reserved for educational resources so users can easily discover and access information relevant to them, which enhances the website's overall utility. A footer is also included at the bottom to provide users with relevant links and contact details should the user want extra information.

Figure 5: Medical Records Wireframe

[Pet]'s Medical Records

Overview

Age: [age]
 Gender: [gender]
 Species: [species]
 Breed: [breed]
 Weight: [weight]
 Microchipped: [yes/no]
 Notes: [info]

General Health ▼

- [Past physical exams](#)
- [Weight tracker](#)
- [Vaccinations](#)

Treatment History ▼

• [Medical History ▼](#) Sort by ▼

Practitioner	Treatment	Veterinarian	Date	Start time	End time	Notes

[Download](#)

Contact Us
 [Phone Number]
 [Email]

Links
[Terms of Services](#)
[Privacy Policy](#)

Socials
 IG: [info]
 FB: [info]
 Whatsapp: [info]

The medical records page is designed to be clear and concise for the comfort of the user. It features a general overview that summarises the basic details in relation to the pet's health, such as age, gender, weight etc. This is to ensure the basic information about the pet is correct if a practitioner requires this information, and also allows the user to identify errors or incorrect information regarding their pet in order to resolve it in a timely manner. An image placeholder is present on the right for the same purpose as the one on the dashboard, and will also utilise the same image as the dashboard.

The first section of the page, the “General Health” section, enables the user easy access to general pet records, such as past physical exam results, weight tracking access and vaccination history, if the user requires access to this information. The second section of this page features a sortable table detailing any past treatments they may have done, which will primarily consist of appointments and check-ups, but may also include any information about previous scans and tests. The table can be filtered as needed, and the results can be exported into a pdf or excel document using the “Download” button at the bottom right hand corner underneath the table. This structure is highly intuitive and is an efficient and accurate way of allowing the user to obtain relevant information.

Figure 6: Prescription Management Wireframe

[Pet]'s Prescription Management

Overview

Age: [age]
 Gender: [gender]
 Species: [species]
 Breed: [breed]
 Weight: [weight]
 Microchipped: [yes/no]
 Notes: [info]

Current Prescriptions ▼

Add prescriptions Edit prescriptions Delete prescriptions

Practitioner	Prescription	Veterinarian	Date filled	Dosage	Recommended date of refill	Product description	Track Order

Refill

Prescription History ▼

Sort by ▼

Practitioner	Prescription	Veterinarian	Date filled	Dosage	Date of Refill	More Information

Download

Contact Us
 [Phone Number]
 [Email]

Links
 Terms of Services
 Privacy Policy

Socials
 IG: [info]
 FB: [info]
 Whatsapp: [info]

The prescription management page is designed to be an organised and succinct way of displaying past and present prescription details. It is structured similarly to the medical records page for the benefit of the user, and this follows Nielsen’s 4th heuristic of consistency and standard.

The first section has an overview of key information about the pet, for both the convenience of the user and any practitioners requiring access to this information. An image placeholder is present on the right for the same purpose as the one on the dashboard, and will also utilise the same image as the dashboard.

The middle section has a list of the pet's current prescriptions in a table format, with important information pertaining to the dosage and description of each prescription, alongside information such as the name of the practice that filled the prescription, the date of this filling, and the recommended date of refill. If the user has yet to receive a prescription, relevant details regarding order tracking can also be seen via a full page popup that can be clicked in the last column of the table. There is also a button that redirects directly to the refill page for the user's convenience, as well as buttons to add, edit or delete prescriptions. This allows more control to the user.

The last section is akin to the medical history table of the medical record access page. It lists every past prescription into a sortable table, which has all the applicable information required, such as dosage numbers and a comprehensive description. The table can be filtered as needed, and the results can be exported into a pdf or excel document using the "Download" button found in the bottom right hand corner underneath the table. This structure is consistent with different parts of this application, and therefore is an intuitive way of displaying all the relevant information for use.

Figure 7: Profile Management Wireframe

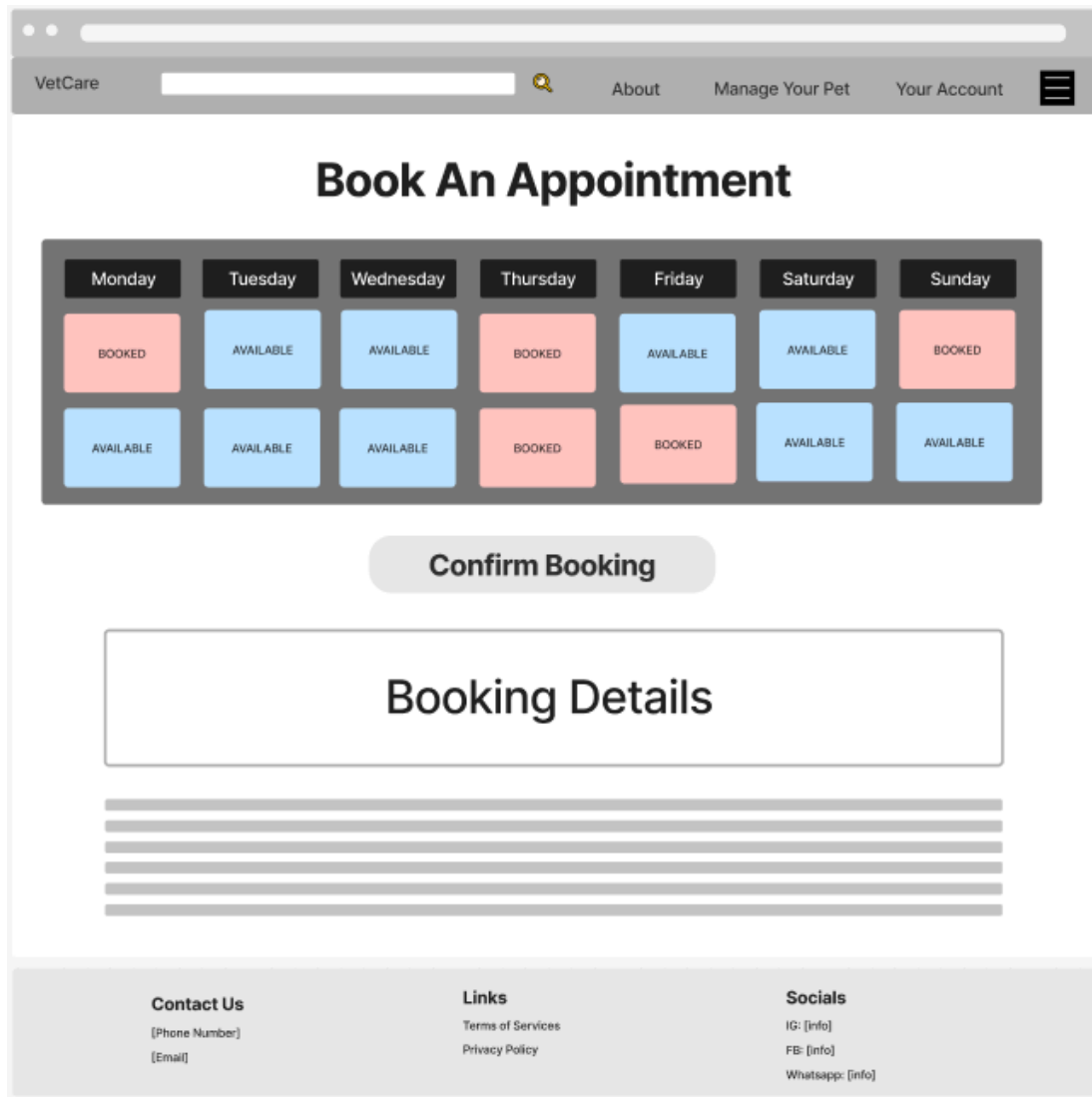
The wireframe shows a user account settings page for 'VetCare'. The page is divided into a main content area on the left and a settings area on the right. The main content area has a dark sidebar with a menu: 'Account Details' (selected), 'Security', 'Payment Settings', and 'Notification Preferences'. The 'Account Details' section includes fields for First Name, Last Name, Email, and Mobile, with an 'UPDATE' button. The right sidebar contains three sections: 'Security' with password fields and an 'UPDATE' button; 'Payment Settings' with fields for Card #, Expiry, and CVC, with an 'UPDATE' button; and 'Notification Preferences' with checkboxes for Appointment Reminders, Prescription Confirmations, Delivery Notifications, and Advertisements, with a 'CONFIRM' button. A footer section contains 'Contact Us', 'Links', and 'Socials' information.

The design of the Account Details page is intuitive and user friendly. It has a focus on Nielsen's heuristic of visibility, in that there is a clear distinction between the settings tab in black on the left, as well as the profile details on the right.

The layout has several menu options on the left that are hyperlinked to indicate that it is a menu option. We then have all the edit settings sections on the left (gray boxes) that will interchange between one another depending on the setting option selected. The forms/check boxes follow a simple structure that is used consistently between our site as well as other sites in the industry. This is in an attempt to ensure consistency and standards heuristic is being adhered to.

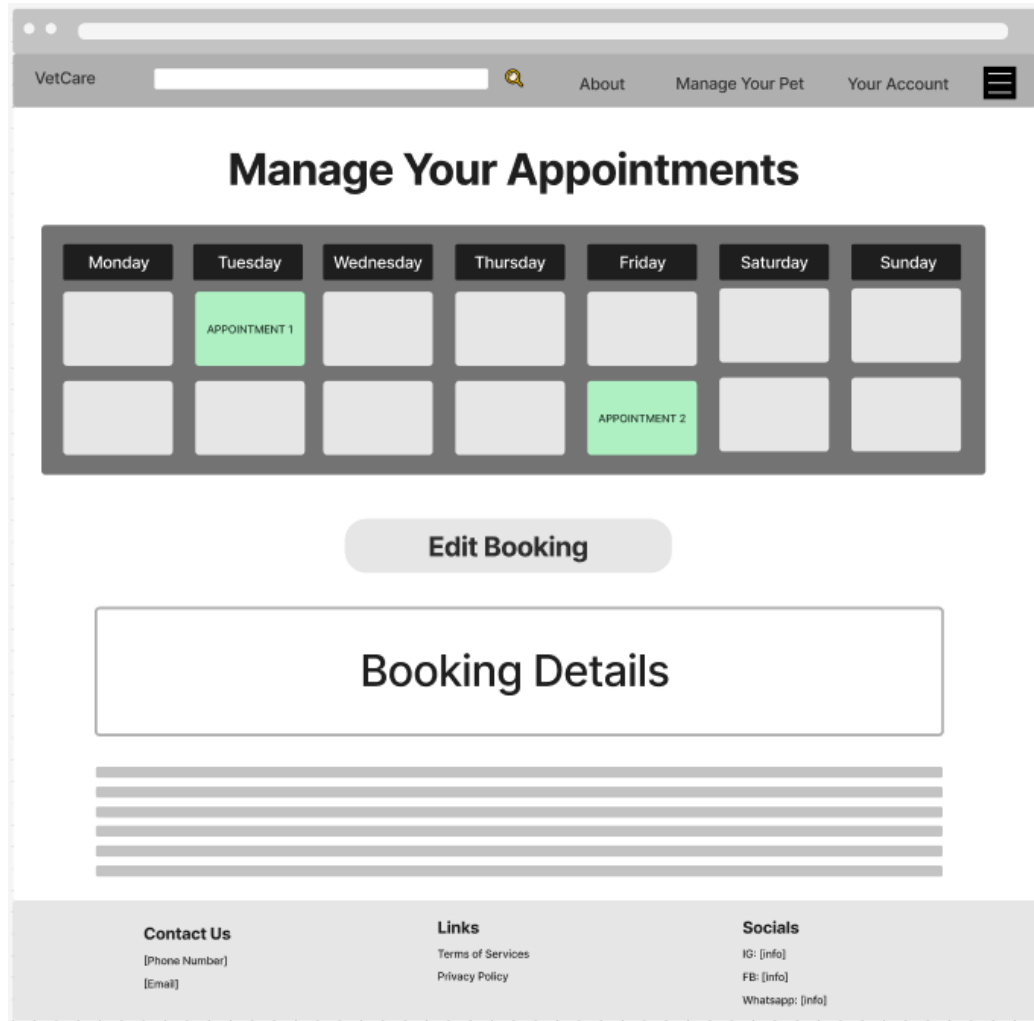
Moreover, the large buttons used are clearly labeled and colour, not only drawing the attention to the user, but also adhering to the recall principle, making the process simple for the user to follow. Moreover, this minimalist design, helps reduce clutter on the site, tying into the aesthetic and design heuristic.

Figure 8: Appointment Booking Wireframe



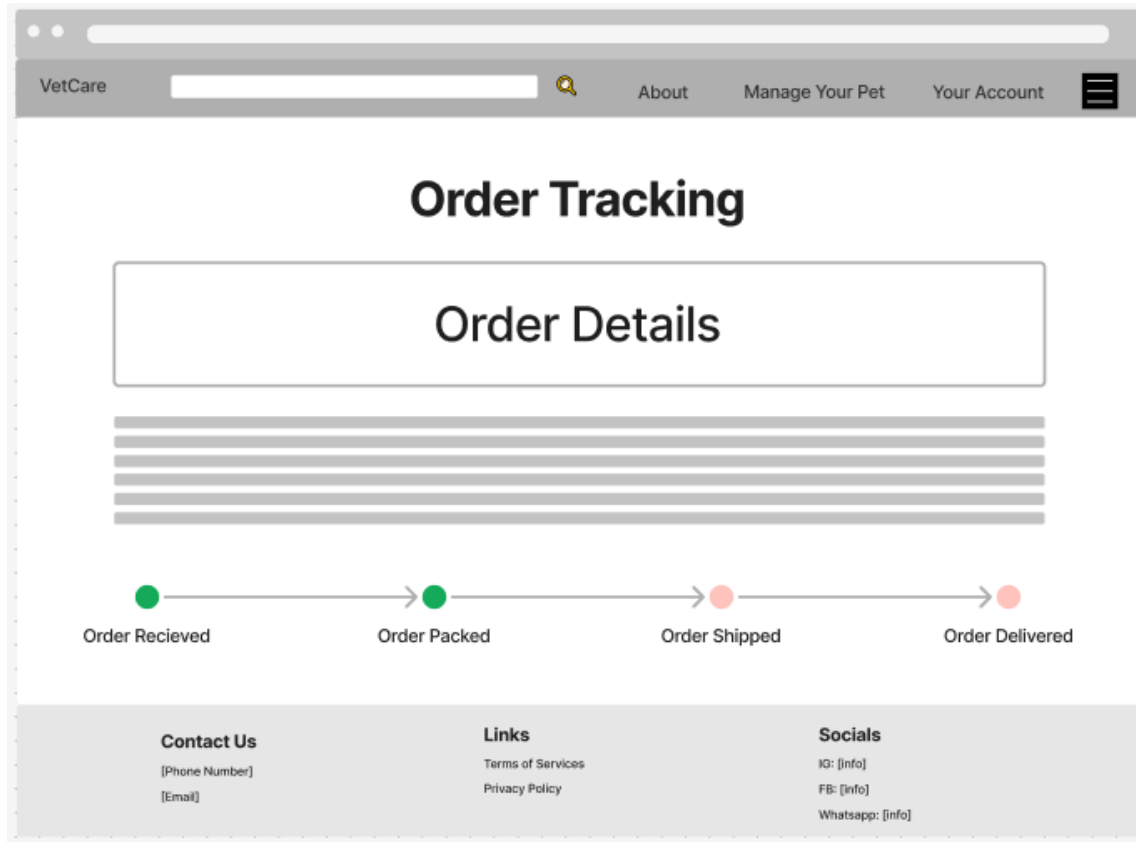
The Book an Appointment page is clear and adheres to the heuristic of matching system to real world objects, as seen through the calendar format. Each day is either 'AVAILABLE' or 'BOOKED', making identification a simple process, reducing the efforts required from the user. This further ties into the recognition rather than recall principle.

Moreover, our simplistic design, combined with the ability to confirm/edit appointments, allows users to easily navigate through the site. This adheres to the error prevention heuristic, as it minimises the chance that the user makes a mistake or gets lost.

Figure 9: Manage Appointments Wireframe

This UI is designed to allow users to easily manage their appointments, clearly highlighting their existing bookings in a green box. The calendar format is consistent with the booking interface, further tying into the recognition rather than recall principle, making the booking system a familiar process for the user.

The clear labels of booked appointments such as “APPOINTMENT 1”, enable quick recognition of when a user has an upcoming appointment with a veterinarian. The option to ‘Edit Booking’, also provides the user with freedom and flexibility to modify existing appointments. The clean, uncluttered design reduces user confusion and supports the aesthetic and minimalist design heuristic.

Figure 10: Order Tracking Wireframe

The Order Tracking UI is designed to be both intuitive and informative, adhering to *Nielsen's heuristic of visibility* of system status by clearly showing the order's progress through different stages: Order Received, Order Packed, Order Shipped, and Order Delivered.

The order tracking UI is designed to be as intuitive, informative and simplistic as possible, to minimise any potential errors that the user may encounter. This is achieved through providing different stages ranging from 'Order Received' through to 'Order Delivered', meaning the user can identify exactly where their product is.

The step by step process also adheres to the match the system and real world heuristic, as it mimics a real world order process. The interface was created to be as straightforward as possible, communicating essential information whilst also reducing clutter. This was in an attempt to adhere to the aesthetic and minimalist design heuristics.

Figure 11: Mobile View Dashboard Wireframe

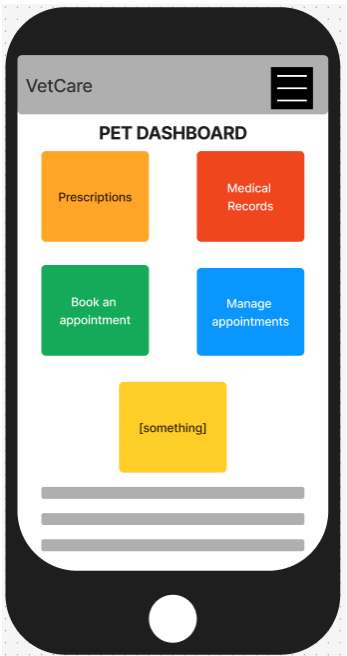
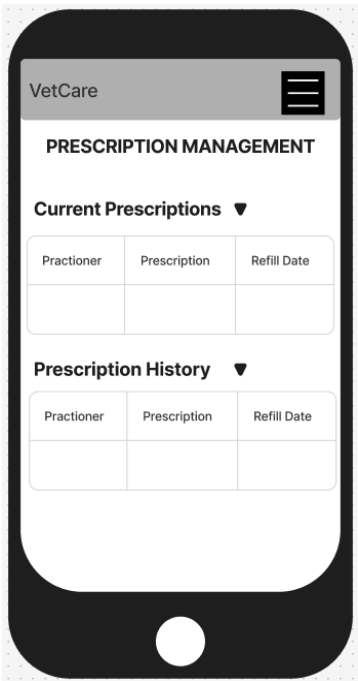


Figure 12: Mobile View Prescription Management Wireframe



Our mobile design for the VetCare web application closely follows *Nielsen's heuristics* to create an intuitive and user-friendly interface. Due to the size constraints of a mobile view, we had to prioritise visibility and clarity, due to the lack of space to provide information. We achieved this through the use of expandable menus, such as the burger menu in the top right corner, which allows the user to view different pages if necessary. However, when it is not in use, it is reduced in size and out of the way, which reduces clutter as well as cognitive load of our user.

Moreover, our design incorporates collapsible tables for our Prescription Management screen. This allows us to further reduce clutter, but also control the information that they wish to view. This prevents information overload and allows for simplistic navigation.

Lastly, we emphasized consistency and standards throughout the design. We utilised consistent colour patterns across the site, as well as simple iconography, which adheres to the recognition rather than recall heuristic, as the user will be familiar with the environment.

User Interface Implemented In Sprint 1:

Figure 13: Navigation Bar Logged Out View

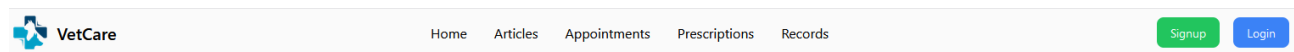
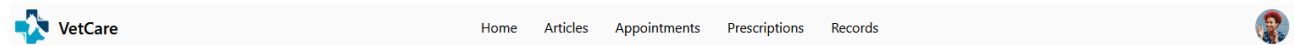


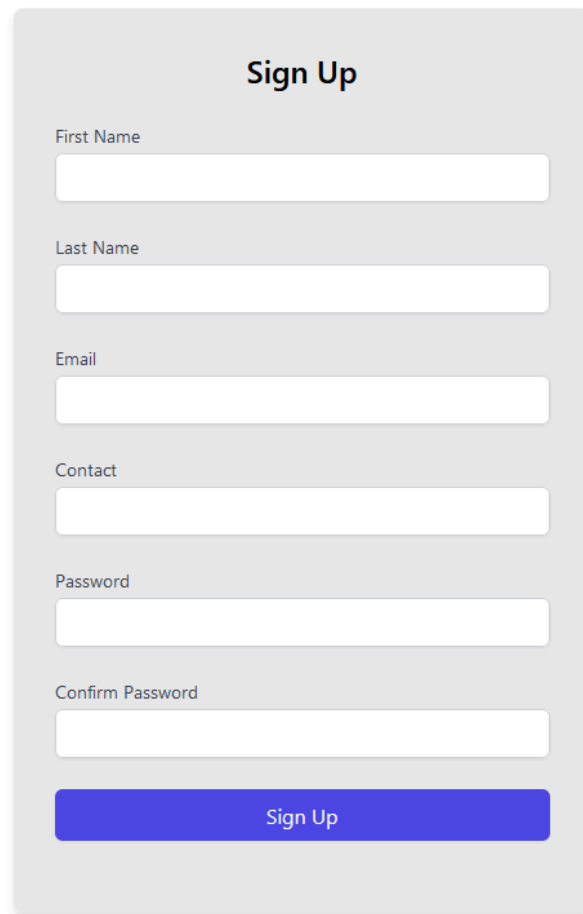
Figure 14: Navigation Bar Logged In View



The navigation layout that we decided to implement was designed to try and provide the user with an intuitive and simplistic style that avoids confusion. In figure 13, we can see the view that a logged out user would see, which has 2 clear 'signup' and 'login' buttons that the user can select depending on their requirements. Furthermore, having each of the pages clearly indicated in the middle of the navigation bar, helps the user navigate through the site to their desired location. This adds to the user experience due to its clear and simple nature.

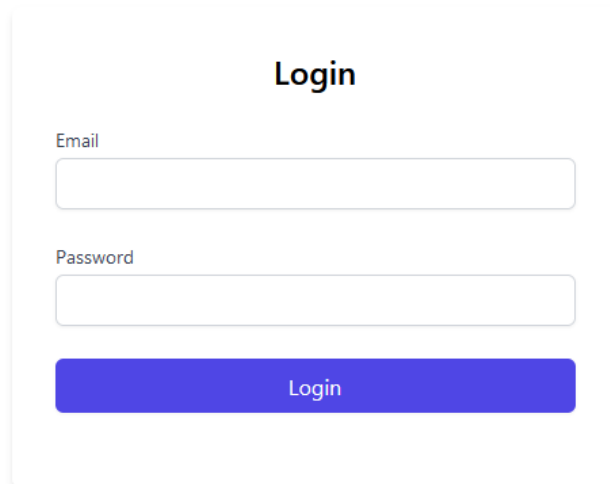
Upon the user logging into their account, they are prompted with a different navigation bar as seen in figure 14, this removes the 'login' and 'signup' button and replaces it with a profile icon that clearly displays the users profile. This also has a drop down menu that allows the user to either logout or view their profile icons. We have a focus on trying to reduce the number of items within the navbar to reduce clutter. This was in an attempt to really stress the key parts of our site, as well as key functionalities such as logging out, such that the user doesn't get disoriented or confused with any processes.

Figure 15: Signup Form



The Signup Form is a vertical rectangle with a light gray background. At the top, the title "Sign Up" is centered in a bold, black font. Below the title, there are six input fields, each with a label to its left: "First Name", "Last Name", "Email", "Contact", "Password", and "Confirm Password". Each input field is a white rectangle with rounded corners and a thin gray border. At the bottom of the form is a blue button with rounded corners and the text "Sign Up" in white, centered on the button.

Figure 16: Login Form



The Login Form is a vertical rectangle with a white background. At the top, the title "Login" is centered in a bold, black font. Below the title, there are two input fields, each with a label to its left: "Email" and "Password". Each input field is a white rectangle with rounded corners and a thin gray border. At the bottom of the form is a blue button with rounded corners and the text "Login" in white, centered on the button.

The signup and login forms were created to be very similar, in an attempt to appeal to Nielsens design heuristics, ensuring that systems are replicated from what the users are used to. Moreover, through repetition, we created very similar forms, with similar colours, which helps create a unified process that users can follow through the site. This can be seen within both figure 15 and 16. Furthermore, each of the form inputs are clearly labeled to help the user identify what to enter in the fields. We also embedded various error codes in case of duplicate emails, non-matching passwords in attempt to help guide the user in case they were doing something incorrectly.

Figure 17: User Profile Page

The screenshot displays the User Profile Page. On the left is a black sidebar with the text 'Welcome!' and 'preeti test' at the top. Below this are four buttons: 'Account Details', 'Security', 'Payment Settings', and 'Notification Preferences'. The 'Account Details' button is highlighted. The main content area on the right is light blue and titled 'Account Details'. It contains four input fields: 'First Name' (preeti), 'Last Name' (test), 'Email' (preeti@gmail.com), and 'Mobile' (04111111). A blue 'Update' button is located at the bottom left of the form.

The user profile page follows a very clear structure, dividing the page into two clear parts. The left side, within figure 17, is black and contains the user details, as well as a series of buttons that the user can click on to edit details. The right side of the page contains the selected buttons display, whether it is regarding account details, security, notifications, payment settings, it will display that relevant section. We implemented this well defined structure such that the user has freedom and control to view and edit what they please.

We also implemented the input fields to contain the preexisting content as seen in figure 17, we can see the existing user name, last name etc... which helps the user to identify what they are editing. Furthermore, we used buttons that are not only distinct, but also the same as what was used in the login and signup forms, to further help guide the user around the site.

Appointment Booking Page has 3 states, which are as follows:

1. Initial stage: Juist displays options for filters and an option to select or change dates and user's preferred veterinarian

CLINIC: VetCare LaTrobe(city)

FILTER BY SERVICES

- ☐ Consultation
- ☐ Senior Pet Care
- ☐ Behavioral Consultation
- ☐ Nutrition
- ☐ Dental Care
- ☐ Puppy & Kitten Care
- ☐ Surgery

FILTER BY DOCTORS

- ☐ Dr. VetFirst1 VetLast1
vet1@vetcare.com
- ☐ Dr. VetFirst2 VetLast2
vet2@vetcare.com
- ☐ Dr. VetFirst3 VetLast3
vet3@vetcare.com

Today: 22 September 2024 Sun Sep 22 2024

Calendar View: 22 Sep 24, 23 Sep 24, 24 Sep 24, 25 Sep 24, 26 Sep 24, 27 Sep 24, 28 Sep 24

Time Slots: 7:00 AM, 8:00 AM, 9:00 AM, 10:00 AM

Select a doctor to view available appointments.

- Veterinarian selected stage: Once the veterinarian has been selected by the user, available slots are shown.

☐ Surgery

FILTER BY DOCTORS

- ☐ Dr. VetFirst1 VetLast1
vet1@vetcare.com
- ☒ Dr. VetFirst2 VetLast2
vet2@vetcare.com
- ☐ Dr. VetFirst3 VetLast3
vet3@vetcare.com

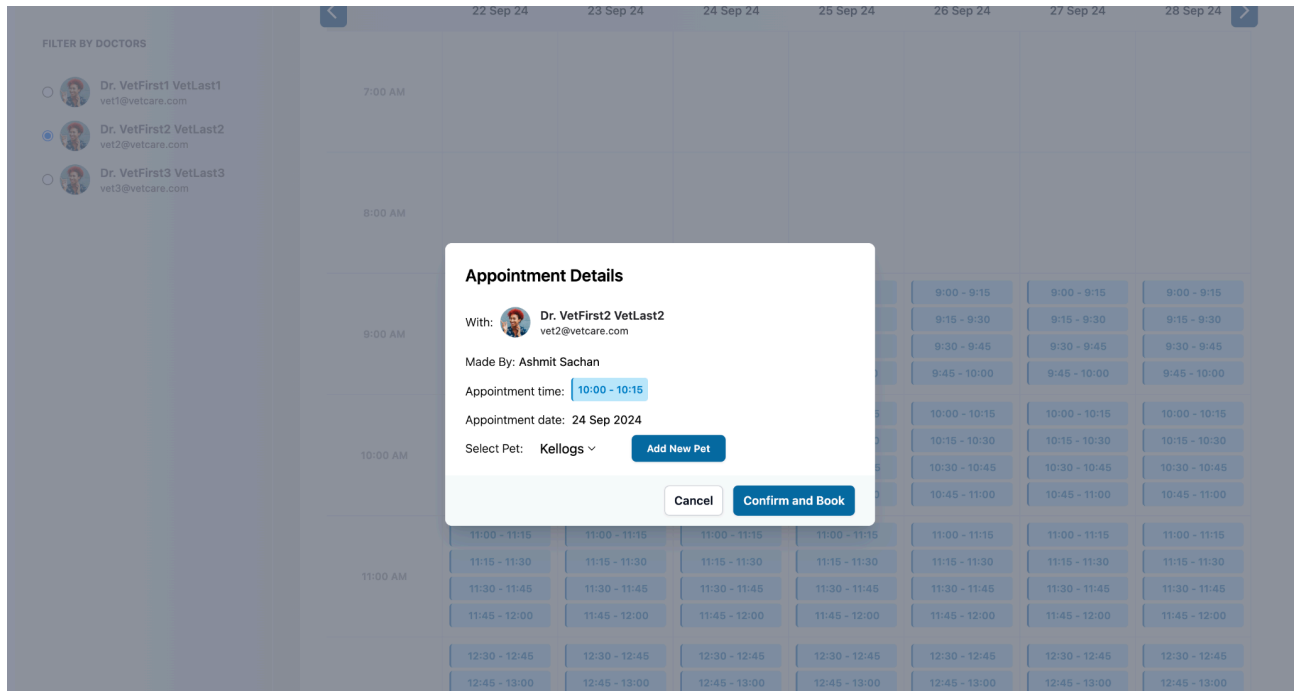
Calendar View: 22 Sep 24, 23 Sep 24, 24 Sep 24, 25 Sep 24, 26 Sep 24, 27 Sep 24, 28 Sep 24

Time Slots: 7:00 AM, 8:00 AM, 9:00 AM, 10:00 AM, 11:00 AM, 12:00 PM

Available Slots (for Dr. VetFirst2 VetLast2):

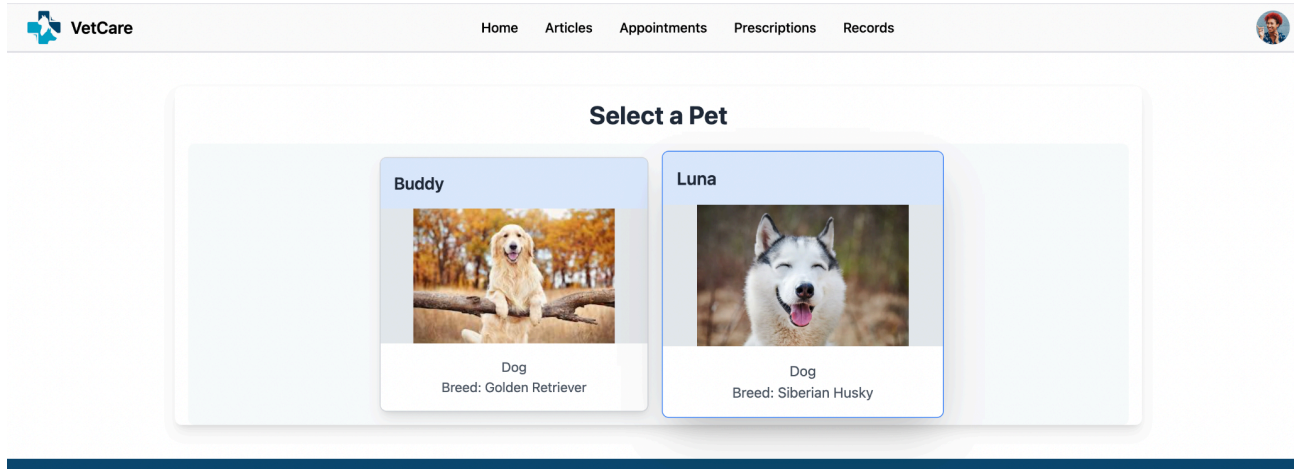
Day	7:00 AM	8:00 AM	9:00 AM	10:00 AM	11:00 AM	12:00 PM
22 Sep 24			9:00 - 9:15, 9:15 - 9:30, 9:30 - 9:45, 9:45 - 10:00	10:00 - 10:15, 10:15 - 10:30, 10:30 - 10:45, 10:45 - 11:00	11:00 - 11:15, 11:15 - 11:30, 11:30 - 11:45, 11:45 - 12:00	12:30 - 12:45, 12:45 - 13:00
23 Sep 24			9:00 - 9:15, 9:15 - 9:30, 9:30 - 9:45, 9:45 - 10:00	10:00 - 10:15, 10:15 - 10:30, 10:30 - 10:45, 10:45 - 11:00	11:00 - 11:15, 11:15 - 11:30, 11:30 - 11:45, 11:45 - 12:00	12:30 - 12:45, 12:45 - 13:00
24 Sep 24			9:00 - 9:15, 9:15 - 9:30, 9:30 - 9:45, 9:45 - 10:00	10:00 - 10:15, 10:15 - 10:30, 10:30 - 10:45, 10:45 - 11:00	11:00 - 11:15, 11:15 - 11:30, 11:30 - 11:45, 11:45 - 12:00	12:30 - 12:45, 12:45 - 13:00
25 Sep 24			9:00 - 9:15, 9:15 - 9:30, 9:30 - 9:45, 9:45 - 10:00	10:00 - 10:15, 10:15 - 10:30, 10:30 - 10:45, 10:45 - 11:00	11:00 - 11:15, 11:15 - 11:30, 11:30 - 11:45, 11:45 - 12:00	12:30 - 12:45, 12:45 - 13:00
26 Sep 24			9:00 - 9:15, 9:15 - 9:30, 9:30 - 9:45, 9:45 - 10:00	10:00 - 10:15, 10:15 - 10:30, 10:30 - 10:45, 10:45 - 11:00	11:00 - 11:15, 11:15 - 11:30, 11:30 - 11:45, 11:45 - 12:00	12:30 - 12:45, 12:45 - 13:00
27 Sep 24			9:00 - 9:15, 9:15 - 9:30, 9:30 - 9:45, 9:45 - 10:00	10:00 - 10:15, 10:15 - 10:30, 10:30 - 10:45, 10:45 - 11:00	11:00 - 11:15, 11:15 - 11:30, 11:30 - 11:45, 11:45 - 12:00	12:30 - 12:45, 12:45 - 13:00
28 Sep 24			9:00 - 9:15, 9:15 - 9:30, 9:30 - 9:45, 9:45 - 10:00	10:00 - 10:15, 10:15 - 10:30, 10:30 - 10:45, 10:45 - 11:00	11:00 - 11:15, 11:15 - 11:30, 11:30 - 11:45, 11:45 - 12:00	12:30 - 12:45, 12:45 - 13:00

- Booking confirmation stage: Upon selecting the slot the user can select the pet and confirm booking.



Medical record Access:

1. First page: the first page shows all pets the owner has added and allows user to select pet



2. Records Page: History of the pet is shown in this page with an option to download the records and share them



8. Testing

User Service Tests

Test 1:

Test Name: testSaveUser_WithEncryption

Purpose: The purpose of this test was to check that the passwords were correctly being encrypted when being stored into the database.

Approach: The test works by having a normal password being created and passed to saveUser, of which the test then checks that the password is no longer saved in its original form, meaning it has been successfully encrypted.

Expected Outcome: The test is successful if the outcome is encrypted, meaning it's different to the input password.

Test 2:

Test Name: testEmailExists

Purpose: This test is a check to see if a given email already exists within the database.

Approach: An email is saved into the database, the test then calls emailExists() to check whether the email does or does not exist within the database.

Expected Outcome: The test should return true when the email already exists in the database and false if the email is unique

Test 3:

Test Name: testValidateUserCredentials_Success

Purpose: This test is used to authenticate user information and whether or not the format is correct

Approach: The test works by creating, then saving a new user into the database, then returning the user object to see if the new user matches the expected results.

Expected Outcome: The test expects the credentials to be validated successfully when the user object is successfully returned.

Test 4:

Test Name: testValidateUserCredentials_IncorrectPassword

Purpose: This test checks that the validateUserCredentials will fail when an incorrect password is used.

Approach: A user is created and saved with a known password. The test then attempts to validate this user with an incorrect password, which will then produce an error due to it being incorrect.

Expected Outcome: The test expects an Exception with the message "Invalid credentials" when the password does not match.

Test 5:

Test Name: testValidateUserCredentials_EmailNotFound

Purpose: This test is to validate that unique email addresses are not found within the database.

Approach: The test uses a unique email address that is tested within the database to prove that it is unique and not found

Expected Outcome: The test expects an Exception with the message "Email not found," which confirms that the email is in fact unique.

User Controller Tests

Test 6:

Test Name: signup_Success

Purpose: This test verifies that when a user signs up, that it successfully populates the database

Approach: A JSON payload containing valid user details is sent to the signup controller, that then checks that a 200 status is returned, which confirms a successful user signup attempt.

Expected Outcome: The test expects a success message 200 status to be returned, confirmed a successful user signup

Test 7:

Test Name: signup_EmailAlreadyExists

Purpose: This test is to check that the signup process successfully checks if an email is already registered.

Approach: The test pre-saves an email, which is then used to test against, if the status 400 is returned, the test is a success as it highlights that the email does in fact already exist in the database and thereby is not unique.

Expected Outcome: The test expects the response to include the message "Email already exists," as well as a status 400 response, confirming the email exists.

Test 8:

Test Name: login_Success

Purpose: This test validates that the loginUser endpoint in the UserController correctly authenticates a user with valid details.

Approach: A user is created, which is then used to test to see if it can successfully login, if status is 200, it confirms that the details were successfully signed in with.

Expected Outcome: The test expects successful authentication, with the correct user details being returned in the response.

Test 9:

Test Name: login_IncorrectPassword

Purpose: This test checks that the loginUser endpoint fails when an incorrect password is provided when attempting to login.

Approach: A user is saved with a known password, and then a using these details, a login is made with a different password. If a status 400 is returned, it indicates that the password does not match and therefore the user is not logged in.

Expected Outcome: The test expects a failure message and a 400 status, confirming that it was an incorrect password.

Test 10:

Test Name: login_EmailNotFound

Purpose: This test ensures that the loginUser endpoint correctly handles login attempts with an email that hasn't been attached to any account, making it non-existent in the database.

Approach: The test sends a login request with a non-existent email and checks that the response indicates failure due to the email not being found.

Expected Outcome: The test expects a failure response when trying to login with the incorrect email, confirming that it wasn't found in the database.

Article Repository Tests

Setup: Two custom test articles with unique titles, descriptions, and authors are added to an already existing articles database.

Test 11:

Test Name: testSearchArticlesByKeyword_TitleMatch

Purpose: This test ensures that the search functionality for articles by keywords works correctly by matching the keyword with the article's title

Approach: Two separate searches are performed with the input "Curabitur" and "abitur".

Expected Outcome: The test expects both search results to return exactly one article with the title "Curabitur non justo".

Test 12:

Test Name: testSearchArticlesByKeyword_DescriptionMatch

Purpose: This test ensures that the search functionality for articles by keywords works correctly by matching the keyword with the article's description.

Approach: Two separate searches are performed with the input "dolor amet" and "doLoR a".

Expected Outcome: The test expects both search results to return exactly one article with the description "Dolor amet".

Test 13:

Test Name: testSearchArticlesByKeyword_AuthorMatch

Purpose: This test ensures that the search functionality for articles by keywords works correctly by matching the keyword with the article's author.

Approach: Two separate searches are performed with the input "Nullam" and "am volut".

Expected Outcome: The test expects both search results to return exactly one article with the author "Nullam Volutpat".

Test 14:

Test Name: testSearchArticlesByKeyword_NoMatch

Purpose: This test ensures that the search functionality correctly handles cases where no articles match the provided keyword.

Approach: A search is performed with an input "asdfjkl".

Expected Outcome: The test expects that no articles are found and therefore the number of search results is zero.

Test 15:

Test Name: testSearchArticlesByKeyword_LargeDataset

Purpose: This test ensures that the search functionality remains accurate and efficient when dealing with large datasets.

Approach: The test generates 1000 random articles and saves them to the database. A timer is started before the search query is executed to measure how long the search takes. A search is performed using the keyword "Lorem Ipsum". The timer is then stopped when the search query is completed.

Expected Outcome: The search query should complete within one second and return exactly one article.

Prescription Tests

Located in branch "broken-prescription"

Test 16:

Test Name: getCurrentPrescriptions_Success()

Purpose: This test ensures that the system can retrieve the current prescriptions for a specific pet.

Approach: The pet information is fetched from the database using petId, and then checks if the system returns the correct prescription details.

Expected Outcome: The test should return a list of current prescriptions for the test with the correct details, such as medication name and dosage details.

Test 17:

Test Name: getPrescriptionHistory_Success()

Purpose: This test ensures that the system can retrieve the prescription history for a specific pet.

Approach: The pet information is fetched from the database using petId, and then checks if the system returns the correct details of the selected pet's prescription history.

Expected Outcome: The test should return a list of past prescriptions for the test with the correct details, such as medication name, duration and dosage details.

Test 18:

Test Name: addPrescription_Success()

Purpose: This test ensures that new prescriptions can be added successfully to the system via the user's manual input.

Approach: Sends a request to the system with the details of the prescription, with those details being: pet ID, medication, dosage, frequency, duration, issue date and refill date.

Expected Outcome: The test should add these details successfully to the system and return a "Prescription added successfully" message.

Test 19:

Test Name: editPrescription_Success()

Purpose: This test ensures that existing prescriptions can be modified successfully in the system via the user's manual input.

Approach: Sends a request to the system with the details of the prescription, with those details being: pet ID, medication, dosage, frequency, duration, issue date and refill date.

Expected Outcome: The test should update these details successfully to the system and return a "Prescription updated successfully" message.

Test 20:

Test Name: deletePrescription_Success()

Purpose: This test ensures that existing prescriptions can be deleted successfully in the system via the user's request.

Approach: Sends a deletion request to the system containing the pet ID and the prescription information.

Expected Outcome: The test should delete the prescription from the system successfully and return a "Prescription deleted successfully" message.

Test 21:

Test Name: exportToPDF_TriggersPrint()

Purpose: This test ensures that the print button will trigger the browser's native print window in order to turn the prescription information into a PDF.

Approach: When the button is pressed, the system will send an instruction to the browser to open its native print window.

Expected Outcome: The test should trigger the browser's print function and return a "Print function triggered" message.

ServiceControllerTests

This test file checks the behavior of the Service Controller for fetching services using the Spring `MockMvc` framework.

Test 22: `getAllServices_Success`

- **Purpose:** This test ensures that the API endpoint for fetching all services is working as expected.
- **Approach:** It sends a POST request to `/api/service/all` and checks that the response is OK (HTTP status 200) and that the response contains a non-empty array of services.
- **Expected Outcome:** The test expects the API to return a status of 200 (OK) and a non-empty array of services, indicating that services were fetched successfully.

This test validates the system's ability to retrieve a list of all available services, ensuring that the backend service and the endpoint are functioning as intended.

Test 23: `getServiceById_Success`

- **Purpose:** This test checks if a specific service can be fetched by its ID.
- **Approach:** It sends a POST request to `/api/service/1` to fetch the service with ID 1 and expects a successful response.

- **Expected Outcome:** The test expects a 200 (OK) status, confirming that the service with ID 1 is successfully retrieved.

This test ensures that individual services can be queried by ID, which is important for viewing service details in various scenarios, such as a customer checking service offerings.

VeterinarianControllerTests

This test file verifies the Veterinarian Controller's endpoints, particularly fetching veterinarians based on different criteria.

Test 24: getAllVeterinarians_Success

- **Purpose:** This test ensures that the endpoint for retrieving all veterinarians is functioning correctly.
- **Approach:** It sends a POST request to `/api/veterinarian/all` and verifies that the response status is 200 and contains a non-empty list of veterinarians.
- **Expected Outcome:** The test expects an OK status and a non-empty list of veterinarians.

This test guarantees that the application can return a list of all veterinarians, which is essential for functionalities like displaying available veterinarians to users.

Test 25: getVeterinariansByClinic_Success

- **Purpose:** This test verifies that the system can retrieve veterinarians associated with a particular clinic ID.
- **Approach:** A POST request is sent to `/api/veterinarian/clinic/1` to retrieve veterinarians from clinic ID 1. The test expects a successful response.
- **Expected Outcome:** A status of 200 (OK) confirms that the veterinarians from the specified clinic are successfully fetched.

This test ensures that users can filter veterinarians based on clinic, which is important for users who are looking for clinic-specific veterinarians.

Test 26: getVeterinariansByService_Success

- **Purpose:** This test checks if veterinarians can be filtered by a specific service.
- **Approach:** The test sends a POST request to `/api/veterinarian/service/1` to fetch veterinarians who provide a specific service with ID 1.
- **Expected Outcome:** The test expects an OK status, confirming that veterinarians offering the specific service are successfully retrieved.

This test helps in identifying veterinarians based on the services they provide, which is crucial for enabling service-specific searches for customers.

VeterinarianAvailabilityControllerTests

This test file is focused on the availability of veterinarians and ensuring that the system correctly retrieves their availability data.

Test 27: `getAvailabilityByVeterinarianId_Success`

- **Purpose:** The purpose of this test is to verify that the system can retrieve the availability of a specific veterinarian based on their ID.
- **Approach:** A POST request is sent to `/api/veterinarian-availability/1` to fetch the availability of the veterinarian with ID 1. The response is expected to contain availability information.
- **Expected Outcome:** The test expects a 200 status and a non-empty availability array for the veterinarian.

This test ensures that the system provides users with the availability of veterinarians, which is critical for appointment scheduling and resource planning.

MedicalRecordsControllerTests

This test class validates the behavior of the Medical Records Controller by testing various endpoints related to retrieving and downloading medical records for pets.

Test 28: `getUserPets_Success`

- **Purpose:** To ensure the API can successfully fetch all pets associated with a specific user.
- **Approach:** It sends a `GET` request to the `/api/medical-records/user-pets` endpoint with a `userId` parameter and checks that the response contains a non-empty list of pets.
- **Expected Outcome:** The test expects an HTTP status of 200 (OK) and that the response JSON contains a non-empty array, confirming pets are associated with the user.

This test validates that the system can correctly retrieve the pets for a user, which is essential for displaying the user's pets in a dashboard or similar feature.

Test 29: `downloadMedicalRecords_PDF_Success`

- **Purpose:** To verify that medical records for a pet can be downloaded in PDF format.
- **Approach:** It sends a `GET` request to `/api/medical-records/download` with the `selectedPetId`, `format` (set to "pdf"), and specific medical record sections. The test checks that the response is a PDF file.
- **Expected Outcome:** The test expects a 200 (OK) status and verifies that the response header indicates a PDF content type (`application/pdf`).

This test ensures the system can generate and serve medical records as a PDF file, an important feature for users who need downloadable medical documents.

Test 30: downloadMedicalRecords_XML_Success

- **Purpose:** To ensure medical records can be downloaded in XML format.
- **Approach:** It sends a GET request to `/api/medical-records/download` with parameters for the `selectedPetId`, `format` (set to "xml"), and specific sections of the medical records. The test checks that the response is an XML file.
- **Expected Outcome:** The test expects a 200 (OK) status and verifies that the response header indicates an XML content type (`application/xml`).

This test ensures that medical records can be generated in XML format, allowing users or systems to access structured data for further processing or storage.

Test 31: getUserPets_NoPets_SeedsData

- **Purpose:** To verify that the system seeds pet data for a user if no pets exist.
- **Approach:** It sends a GET request to `/api/medical-records/user-pets` with a `userId` that does not have any pets. The test checks if the system returns a non-empty list, implying that pets have been seeded.
- **Expected Outcome:** The test expects a 200 (OK) status and a non-empty JSON array, showing that pets have been seeded when none existed initially.

This test ensures that the system can handle cases where users have no pets and automatically seed data, providing a complete user experience even for new users.

FileGenerationServiceTests

This test class focuses on verifying the functionality of the `FileGenerationService`, particularly for generating XML files with pet medical records.

Test 31: testGenerateXML

- **Purpose:** To test that XML can be successfully generated with all relevant sections of a pet's medical records.
- **Approach:** The test calls the `generateXML` method, passing a pet object and lists of medical history, physical exams, vaccinations, treatment plans, and weight records. It then checks if the generated XML contains the expected sections.
- **Expected Outcome:** The test expects the generated XML stream to be non-null, contain data, and include tags for each section (e.g., `<MedicalHistory>`, `<PhysicalExams>`).

This test verifies that the system can properly serialize a pet's medical records into an XML format, ensuring that all required sections are included.

Test 32: testGenerateXML_EmptySections

- **Purpose:** To test XML generation when no sections are requested.
- **Approach:** The test calls `generateXML` with an empty list of sections. It then verifies that the generated XML does not contain any medical record sections.
- **Expected Outcome:** The test expects the XML to contain the pet information but exclude sections such as `<MedicalHistory>`, `<Vaccinations>`, and others.

This test ensures that the XML generation behaves correctly when specific sections are omitted, providing flexibility in the types of data included in the output.

Appendix A: Glossary

AES-256	Advanced Encryption Standard 256-bit
AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
APNs	Apple Push Notification Service
APPS	Applications (commonly used to refer to software applications)
ASD	Autism Spectrum Disorder
CI/CD	Continuous Integration/Continuous Delivery
CPU	Central Processing Unit
CRUD	Create, Read, Update, Delete
CSS	Cascading Style Sheets
DAO	Data Access Object
FAQ	Frequently Asked Questions
GB	Gigabyte
GDPR	General Data Protection Regulation
GUI	Graphical User Interface
HDD	Hard Disk
HIPAA	Health Insurance Portability and Accountability Act
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
IP	Internet Protocol
IEC 27001	International Electrotechnical Commission 27001
ISMS	Information Security Management System
ISO 27001	International Standard on requirements for information security management
ISO 27002	Information security, Cybersecurity and privacy protection – Information security controls
JDBC	Java Database Connectivity
JSON	JavaScript Object Notation

LTS	Long-Term Support
MFA	Multi-factor Authentication
MVC	Model View Controller
NDB	Notifiable Data Breaches
OAIC	Office of the Australian Information Commissioner
ORM	Object-Relational Mapping
OS	Operating System
RAM	Random Access Memory
RBAC	Role-based Access Control
RDBMS	Relational Database Management System
REST	Representational State Transfer
SMTP	Simple Mail Transfer Protocol
SOC 2 Type II	Service Organization Control Type 2
SMS	Short Message Service
SQL	Structured Query Language
SRS	Software Requirements Specification
SSD	Solid State Drive
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UI	User Interface

Appendix B: Analysis Models

Will be added upon in Milestone 2

Appendix C: To Be Determined List

Don't have any to be determined references for tracking to closure yet