

**EASTERN INTERNATIONAL UNIVERSITY  
SCHOOL OF COMPUTING AND INFORMATION TECHNOLOGY**

**DEPARTMENT OF SOFTWARE ENGINEERING**



**Đại Học Quốc Tế Miền Đông  
Eastern International University**

**PROJECT 1 REPORT**

**VETERINARY TRACKER**

**Student(s)**

**Đặng Bảo Phúc – 2131209001**

**Đặng Thị Ánh Tuyết - 2131200054**

***Supervisor(s)***

***MSc. Trần Văn Tài***

**Binh Duong, March, 2024**

## **ABSTRACT**

"Veterinary Tracker" is a web application designed to revolutionize pet healthcare by offering a comprehensive platform for both veterinarians and pet owners. By centralizing pet health information, the app streamlines communication and ensures seamless coordination between all stakeholders involved in a pet's care journey. For veterinarians, the platform provides efficient management of patient records, allowing easy tracking of medical histories, appointments, treatments, and prescriptions. Additionally, veterinarians can utilize the app to communicate with pet owners, share vital information, and offer personalized recommendations for pet care. Pet owners benefit from secure access to their pet's medical records, appointment reminders, and educational resources on pet health. By empowering pet owners with knowledge and facilitating proactive pet care, Veterinary Tracker aims to enhance the overall well-being and longevity of pets everywhere. With its user-friendly interface and robust functionality, the app serves as an indispensable tool for promoting pet health and strengthening the bond between pets and their owners.

## **ACKNOWLEDGEMENT**

We would like to express our heartfelt gratitude to our college faculty and project guide, Mr. Trần Văn Tài, for his unwavering support and guidance throughout the duration of this project. His expertise, encouragement, and mentorship have played a crucial role in shaping the project's direction and ensuring its successful completion.

Simultaneously, we would like to acknowledge the many programmers who have generously shared insightful articles online. Their contributions have provided us with a wealth of valuable knowledge, enhancing our understanding and skills in numerous ways.

## TABLE OF CONTENTS

<b>ABSTRACT.....</b>	<b>2</b>
<b>ACKNOWLEDGEMENT.....</b>	<b>2</b>
<b>TABLE OF CONTENTS.....</b>	<b>4</b>
<b>TABLE OF FIGURES.....</b>	<b>6</b>
<b>CHAPTER I: INTRODUCTION.....</b>	<b>7</b>
1.1 Motivation.....	7
1.2 Objectives.....	8
<b>CHAPTER II: TECHNOLOGY.....</b>	<b>9</b>
2.1 Spring Boot.....	9
2.1.1 Spring boot Introduction.....	9
2.1.2 Spring Boot Advantages.....	9
2.2 MySQL.....	10
2.2.1 MySQL Introduction.....	10
2.2.2 MySQL Advantages.....	11
2.3 JavaScript.....	12
2.3.1 JavaScript Introduction.....	12
2.3.2 JavaScript Advantages.....	12
2.4 HTML & CSS.....	13
2.4.1. HTML Introduction.....	13
2.4.2 HTML Advantages.....	13
2.4.3 CSS Introduction.....	15
2.4.4 CSS Advantages.....	15
2.5 MVC.....	16
2.5.1 MVC Introduction.....	16
2.5.2 MVC Advantages .....	16
2.6 JAVA.....	18
2.6.1 JAVA Introduction.....	18
2.6.2 JAVA Advantages.....	18
<b>CHAPTER III: SYSTEM ANALYSIS.....</b>	<b>19</b>
3.1 UseCase.....	20
3.2 Database Diagram.....	21
<b>Chapter IV: IMPLEMENTATION.....</b>	<b>23</b>
4.1 Building REST services with Spring.....	23
4.1.1 Model:.....	23
4.1.2 Service.....	26
4.1.3 Controller.....	28
4.2 Admin Management Page.....	42
4.2.1 Pets Owners:.....	42
4.2.2 Pets:.....	50
4.2.3 Veterinarian:.....	51

4.2.4 Medicine:.....	52
4.2.5 Vaccine:.....	53
4.2.6 Appointment:.....	54
4.2.7 Records:.....	55
<b>CHAPTER V: RESULT AND FUTURE WORK.....</b>	<b>57</b>
5.1 Achievement:.....	57
5.1.1 Functional API Endpoints Development.....	57
5.1.2 Functional Admin Page Management.....	57
5.2 Future Work.....	57
<b>REFERENCE.....</b>	<b>58</b>

## TABLE OF FIGURES

Figure 1: Logo of Spring Boot.....	9
Figure 2: Logo of MySQL.....	10
Figure 3: Logo of JavaScript.....	12
Figure 4: Logo of HTML.....	13
Figure 5: Logo of CSS.....	15
Figure 6: MVC Pattern.....	16
Figure 7: Logo of Java.....	18
Figure 8: Veterinary Tracker UseCase.....	21
Figure 9: Veterinary Tracker Database diagram.....	22
Figure 10: Table of Deleting user.....	41
Figure 11: Table of Adding new user.....	41
Figure 12: Table of Editing user.....	42
Figure 14: Pets Owner Table (Admin site).....	43
Figure 15: Edit Pets Owner (Admin site).....	48
Figure 16: Delete Pets Owner (Admin site).....	49
Figure 18: Add new pet form (Admin site).....	51
Figure 20: Add new Veterinarian form (Admin site).....	52
Figure 21: Table of Medicine (Admin site).....	53
Figure 22: Add new Medicine form (Admin site).....	53
Figure 23: Table of Vaccine (Admin site).....	54
Figure 25: Table of Appointment (Admin site).....	55
Figure 26: Add new Appointment form (Admin site).....	55
Figure 27: Table of Veterinarians (Admin site).....	56
Figure 28: Add new Veterinarian form (Admin site).....	56

# CHAPTER I: INTRODUCTION

## 1.1 Motivation

The motivation behind the creation of "Veterinary Tracker" stems from a deep-seated desire to address the challenges and inefficiencies present in traditional pet healthcare systems. As pet lovers ourselves, we understand the emotional attachment and responsibility that comes with caring for a beloved companion animal. However, we also recognize the obstacles that both veterinarians and pet owners face when it comes to managing and monitoring pet health effectively.

One of the primary motivations for developing "Veterinary Tracker" is to simplify and streamline the process of pet healthcare management for both veterinarians and pet owners. Traditional methods of record-keeping and communication can be cumbersome and prone to errors, leading to gaps in care and missed opportunities for early intervention. By harnessing the power of technology, we aim to eliminate these obstacles and provide a centralized platform that fosters collaboration and communication between all stakeholders involved in a pet's care journey.

Another driving force behind "Veterinary Tracker" is the desire to empower pet owners with the knowledge and resources they need to take an active role in their pet's health and well-being. Many pet owners are eager to provide the best possible care for their furry friends but may lack access to reliable information or guidance. Through educational resources, personalized recommendations, and timely reminders, "Veterinary Tracker" aims to empower pet owners to make informed decisions and take proactive steps to ensure their pet's health.

Furthermore, the motivation behind "Veterinary Tracker" is rooted in a commitment to advancing the field of pet healthcare and promoting the overall well-being of animals everywhere. By providing veterinarians with powerful tools for managing patient records, analyzing trends, and delivering personalized care, we believe that "Veterinary Tracker" has the potential to revolutionize the way veterinary care is delivered.

Ultimately, our motivation for creating "Veterinary Tracker" is driven by a passion for animals and a commitment to improving their quality of life. We believe that every pet deserves access to high-quality healthcare, and we are dedicated to leveraging technology to make this vision a reality. With "Veterinary Tracker," we hope to empower veterinarians, support pet owners, and ultimately, enhance the health and happiness of pets around the world.

## 1.2 Objectives

"Veterinary Tracker" aims to revolutionize the landscape of pet healthcare by addressing various objectives designed to streamline processes, enhance communication, empower pet owners, improve practice efficiency, promote preventive healthcare, ensure data security, and continuously enhance user experience.

Efficiency is prioritized through simplifying pet healthcare management for veterinarians and pet owners. The platform facilitates seamless communication between all stakeholders involved in a pet's care journey. Pet owners are empowered with tools and knowledge for proactive care, while veterinarians benefit from streamlined administrative tasks, optimizing practice efficiency.

Preventive healthcare is encouraged through timely reminders and educational resources, promoting early intervention and better health outcomes for pets. Data security measures are implemented to safeguard sensitive pet health information, ensuring confidentiality and integrity.

Continuous improvement of the user experience is prioritized, driven by feedback and emerging technologies. With its comprehensive approach, "Veterinary Tracker" aims to transform pet healthcare management, ultimately enhancing the well-being of pets and strengthening the bond between pets and their owners.



## CHAPTER II: TECHNOLOGY

### 2.1 Spring Boot

#### 2.1.1 Spring boot Introduction



*Figure 1: Logo of Spring Boot*

Spring Boot is an open-source Java-based framework used to create standalone, production-grade Spring-based applications with minimal setup. It is part of the larger Spring Framework ecosystem, designed to simplify the development of Java applications by providing a range of preconfigured settings and conventions.[1]

#### 2.1.2 Spring Boot Advantages[2]

**Rapid Development:** Spring Boot provides a range of preconfigured settings and conventions, enabling developers to quickly bootstrap and develop applications without spending time on boilerplate configuration.

**Reduced Configuration:** Spring Boot's auto-configuration feature automatically configures the Spring application based on the dependencies present in the class path. This reduces the need for manual configuration, making development more efficient.

**Standalone Applications:** Spring Boot allows developers to create standalone applications with embedded servers (like Tomcat, Jetty, or Undertow). This simplifies deployment as applications can be packaged as executable JAR files without the need for external application servers.

**Starters:** Spring Boot Starters provide pre-configured dependencies to streamline the setup of common application features such as web applications, data access, security, messaging, etc. This accelerates development by eliminating the need to manually configure dependencies.

**Opinionated Defaults:** Spring Boot follows convention over configuration principles, providing sensible defaults for configurations. Developers can start coding immediately and override defaults only when necessary, reducing cognitive load and improving productivity.

**Integration with Spring Ecosystem:** Spring Boot seamlessly integrates with the broader Spring ecosystem, including Spring Framework, Spring Data, Spring Security, Spring Cloud, etc. This allows developers to leverage a wide range of libraries and tools for building robust and scalable applications.

**Actuator:** Spring Boot Actuator provides production-ready features for monitoring and managing applications. It includes built-in endpoints for metrics, health checks, environment details, etc., which are essential for monitoring application health and performance in production environments.

**Community and Ecosystem:** Spring Boot has a large and active community of developers, providing extensive documentation, tutorials, and support. Additionally, Spring Boot has a vibrant ecosystem with third-party libraries, plugins, and integrations, further enhancing its capabilities and flexibility.

## 2.2 MySQL

### 2.2.1 MySQL Introduction



*Figure 2: Logo of MySQL*

MySQL is a widely-used open-source Relational Database Management System (RDBMS) renowned for its reliability, performance, and ease of use. It follows the relational model and utilizes Structured Query Language (SQL) for data manipulation. MySQL is cross-platform compatible, scalable, and optimized for performance. It offers high availability, robust security features, and seamless integration with various programming languages and frameworks. With a vibrant community and extensive support resources, MySQL is a preferred choice for developers and organizations seeking a versatile and efficient database solution.[3]

### 2.2.2 MySQL Advantages

**Cost-Effectiveness:** Being open-source, MySQL eliminates the need for expensive licensing fees, making it an economical choice for businesses, startups, and individual

developers. This cost-effectiveness allows organizations to allocate resources to other critical areas of development or infrastructure.

**Performance Optimization:** MySQL provides various optimization techniques such as indexing, caching, and query optimization. Indexing speeds up data retrieval by enabling faster lookup of records, while caching frequently accessed data enhances overall performance. Additionally, MySQL's query optimizer analyzes SQL queries to generate efficient execution plans, further boosting performance.

**Storage Engines Flexibility:** MySQL supports multiple storage engines, each optimized for different use cases. For instance, InnoDB is well-suited for transactional applications due to its support for ACID transactions and row-level locking, while MyISAM is suitable for read-heavy workloads. This flexibility allows developers to choose the most appropriate storage engine based on their application requirements.

**High Availability and Reliability:** MySQL offers features such as replication, clustering, and automatic failover to ensure high availability and fault tolerance. Replication enables the creation of redundant copies of data across multiple servers, while clustering allows for load balancing and failover management. These features minimize downtime and ensure data consistency and reliability, critical for mission-critical applications.

**Security Features:** MySQL provides robust security features to protect data integrity and confidentiality. It supports authentication mechanisms for user access control, encryption for data protection, and auditing capabilities for compliance requirements. Additionally, MySQL's role-based access control (RBAC) allows granular control over user privileges, ensuring that only authorized users can access sensitive data.

**Scalability:** MySQL is highly scalable, capable of handling large datasets and accommodating growing application demands. It supports both vertical scaling (increasing resources on a single server) and horizontal scaling (scaling out across multiple servers), enabling developers to scale their applications seamlessly as the workload increases.

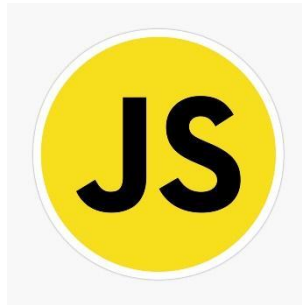
**Community and Support:** MySQL has a large and active community of developers, users, and contributors who provide extensive documentation, tutorials, and support resources. This vibrant community fosters collaboration, knowledge sharing, and problem-solving, making it easier for developers to troubleshoot issues and optimize their MySQL deployments.

**Cross-Platform Compatibility and Integration:** MySQL is compatible with various operating systems and integrates seamlessly with popular programming languages, frameworks, and tools. It provides drivers and connectors for languages like Java, Python,

PHP, and frameworks like Spring and Django, facilitating interoperability and ease of development across different environments.[3]

## 2.3 JavaScript

### 2.3.1 JavaScript Introduction



*Figure 3: Logo of JavaScript*

JavaScript, created by Brendan Eich in 1995, is essential for dynamic web applications. It enables interactive user experiences without constant server interaction, with intuitive syntax and dynamic typing. Evolving through the ECMAScript standard, it's used not only for front-end but also back-end and mobile development. Its asynchronous nature ensures smooth performance, making it indispensable in modern web development.[4]

### 2.3.2 JavaScript Advantages

**Client-Side Interactivity:** JavaScript allows for dynamic and interactive elements on web pages without needing to reload the entire page. This enhances user experience by providing immediate feedback and responsiveness.

**Cross-Platform Compatibility:** JavaScript runs on all major browsers and platforms, making it a versatile choice for web development. It ensures consistent behavior across different devices and operating systems.

**Rich Ecosystem:** JavaScript has a vast ecosystem of libraries, frameworks, and tools that streamline development processes. These resources enable developers to build complex applications efficiently and with minimal effort.

**Asynchronous Programming:** JavaScript's asynchronous nature allows non-blocking operations, such as fetching data from servers or handling user input, without freezing the user interface. This enhances performance and responsiveness.

**Versatility:** JavaScript is not limited to front-end development. With frameworks like Node.js, it can also be used for server-side development, enabling full-stack development with a single programming language.

**Easy to Learn:** JavaScript has a relatively simple syntax compared to other programming languages, making it accessible to beginners. Additionally, its dynamic typing eliminates the need for explicit data type declarations, reducing development time and complexity.

**Community Support:** JavaScript has a large and active community of developers who contribute to open-source projects, share knowledge, and provide support. This vibrant community ensures that developers have access to resources and assistance when needed.

## 2.4 HTML & CSS

### 2.4.1. HTML Introduction



*Figure 4: Logo of HTML*

HTML is the standard markup language used to create the structure and content of web pages. It consists of a series of elements, each enclosed in angle brackets, which define the different parts of a webpage's content. These elements can represent headings, paragraphs, images, links, forms, and more. [4]

### 2.4.2 HTML Advantages

**Universal Compatibility:** HTML is supported by all major web browsers, ensuring that web pages built with HTML can be accessed by a wide range of users regardless of their browser or device.

**Simple and Easy to Learn:** HTML has a straightforward syntax and requires minimal setup, making it easy for beginners to learn and understand. Its simplicity allows for rapid development of basic web pages.

**Semantics:** HTML provides semantic elements that describe the meaning and structure of content, making web pages more accessible to users and search engines. Semantic HTML elements, such as `<header>`, `<nav>`, `<section>`, and `<footer>`, enhance the clarity and organization of web documents.

**SEO (Search Engine Optimization):** Semantic HTML helps improve the search engine ranking of web pages by providing search engines with clear and structured content to

index. Properly structured HTML documents with meaningful tags and attributes make it easier for search engines to understand and rank the content.

**Flexibility:** HTML can be combined with other web technologies, such as CSS for styling and JavaScript for interactivity, to create rich and dynamic web experiences. This flexibility allows developers to build a wide range of web applications, from simple static websites to complex web applications.

**Scalability:** HTML is scalable and can accommodate the needs of various types of websites and web applications. Whether building a small personal blog or a large e-commerce platform, HTML provides the foundation for creating scalable and robust web solutions.

**Cross-Platform Compatibility:** HTML-based web pages can be accessed on various platforms, including desktop computers, laptops, tablets, and smartphones. This cross-platform compatibility ensures that web content can reach a diverse audience across different devices and operating systems.

**Cost-Effectiveness:** HTML development is cost-effective compared to other web development technologies. Since HTML is an open standard and requires no licensing fees, businesses and developers can create and deploy web content without incurring additional costs.

### 2.4.3 CSS Introduction



*Figure 5: Logo of CSS*

CSS (Cascading Style Sheets) is a powerful styling language used in web development to enhance the visual presentation and layout of HTML documents. It allows developers to define the appearance of web pages, including colors, fonts, spacing, and positioning.[4]

#### 2.4.4 CSS Advantages

**Separation of Concerns:** CSS allows for the separation of content (HTML) from presentation (styling). This separation makes code more maintainable, improves readability, and facilitates collaboration among developers, designers, and content creators.

**Consistent Styling:** CSS enables developers to apply consistent styling across multiple web pages by defining styles once and applying them universally. This consistency enhances the user experience and strengthens branding and identity across a website or web application.

**Flexibility and Control:** CSS provides granular control over the styling of HTML elements, allowing developers to customize the appearance of elements precisely according to design requirements. Properties like color, font, size, spacing, and layout can be adjusted with ease.

**Responsive Design:** CSS supports responsive web design techniques, allowing developers to create layouts that adapt to different screen sizes and devices. Media queries, flexible layout options (e.g., flexbox, grid), and viewport settings enable developers to create designs that look and function well across desktops, tablets, and smartphones.

**Fast Loading Times:** Separating styling into external CSS files allows browsers to cache stylesheets, resulting in faster loading times for subsequent page visits. This optimization reduces bandwidth usage and improves overall website performance.

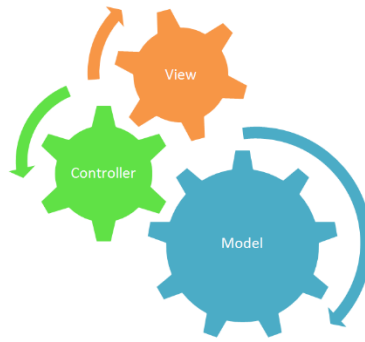
**Accessibility:** CSS supports accessibility by allowing developers to define semantic HTML elements and apply appropriate styling to enhance readability and usability for users with disabilities. Accessible styling practices, such as high contrast, proper text sizing, and keyboard navigation support, ensure that websites are inclusive and accessible to all users.

**Modularity and Reusability:** CSS promotes modularity and reusability through the use of classes, IDs, and reusable style rules. Developers can create a library of reusable styles that can be applied to multiple elements throughout a website, reducing redundancy and improving code maintainability.

**Ease of Maintenance:** Centralizing styling in external CSS files makes it easier to update and maintain styles across an entire website or web application. Changes made to CSS styles are automatically applied to all HTML elements associated with those styles, simplifying the maintenance process and reducing the risk of errors.

## 2.5 MVC

### 2.5.1 MVC Introduction



*Figure 6: MVC Pattern*

MVC provides a structured approach to building web applications, emphasizing modular design, code organization, and separation of concerns. By dividing an application into models, views, and controllers, MVC enables developers to create scalable, maintainable, and testable applications that meet the evolving needs of users and stakeholders.[5]

### **2.5.2 MVC Advantages [6]**

**Separation of Concerns:** MVC separates the application into three distinct components, each responsible for a specific aspect of the application: the Model handles data and business logic, the View handles presentation and user interface, and the Controller manages user input and application flow. This separation enhances code organization, maintainability, and scalability by isolating different concerns and allowing for independent development and testing of each component.

**Modular and Reusable Components:** MVC promotes modular design, allowing developers to create reusable components that can be shared across different parts of the application or even across multiple applications. This modularity increases code reusability, reduces redundancy, and simplifies maintenance and updates.

**Testability:** The separation of concerns in MVC makes it easier to write unit tests for individual components. Models, Views, and Controllers can be tested independently, allowing for more comprehensive and reliable testing of the application's functionality. This leads to improved code quality, fewer bugs, and faster development cycles.

**Scalability:** MVC architecture facilitates scalability by allowing developers to add new features or modify existing ones without affecting other parts of the application. The modular design makes it easier to extend the application's functionality, accommodate changing requirements, and adapt to evolving business needs.

**Enhanced Code Organization:** MVC encourages clean and organized code by providing clear guidelines for structuring an application. With separate folders or directories for Models, Views, and Controllers, developers can easily locate and manage code related to specific functionalities, leading to improved code readability and maintainability.



**Improved Collaboration:** MVC promotes collaboration among developers, designers, and testers by providing a clear separation of concerns and well-defined interfaces between components. This allows team members to work on different parts of the application simultaneously, reducing dependencies and improving productivity.

**Flexibility and Extensibility:** MVC architecture offers flexibility and extensibility, allowing developers to customize and extend the application's functionality easily. New features can be added by creating additional Models, Views, and Controllers, or by modifying existing ones, without disrupting the overall structure of the application.

**Support for Different Platforms:** MVC is platform-independent and can be used to develop web applications for various platforms, including desktops, mobile devices, and IoT devices. The modular design and separation of concerns make it suitable for building applications with diverse requirements and target audiences.

## 2.6 JAVA

### 2.6.1 JAVA Introduction



*Figure 7: Logo of Java*

Java is a high-level, object-oriented programming language originally developed by Sun Microsystems (now owned by Oracle Corporation) in 1995. It was designed with the principle of "Write Once, Run Anywhere" (WORA), meaning that Java code can be compiled into bytecode and executed on any platform that has a Java Virtual Machine (JVM), without the need for recompilation.[7]

### 2.6.2 JAVA Advantages

**Platform Independence:** One of the most significant advantages of Java is its platform independence. Java code is compiled into bytecode, which can run on any device or operating system with a Java Virtual Machine (JVM) installed. This "write once, run anywhere" capability simplifies cross-platform development and deployment.

**Object-Oriented Programming (OOP):** Java is based on the object-oriented programming paradigm, which promotes modular, reusable, and maintainable code. OOP

concepts such as encapsulation, inheritance, and polymorphism facilitate code organization and enhance code quality.

**Rich Standard Library:** Java comes with a comprehensive standard library (Java API) that provides a vast array of classes and methods for common programming tasks. This rich set of libraries covers areas such as input/output operations, networking, data structures, concurrency, and more, reducing the need for developers to write code from scratch.

**Strongly Typed Language:** Java is a strongly typed language, meaning that all variables must be declared with a specific data type. This helps catch errors at compile-time, enhances code clarity, and improves maintainability. Strong typing also contributes to better code documentation and IDE support.

**Automatic Memory Management:** Java features automatic memory management through garbage collection. The JVM automatically handles memory allocation and deallocation, freeing developers from the burden of manual memory management. This prevents memory leaks and improves application stability and performance.

**High Performance:** Despite being interpreted bytecode, Java applications can achieve high performance through just-in-time (JIT) compilation and optimization techniques. Modern JVM implementations optimize code execution at runtime, resulting in competitive performance compared to natively compiled languages.

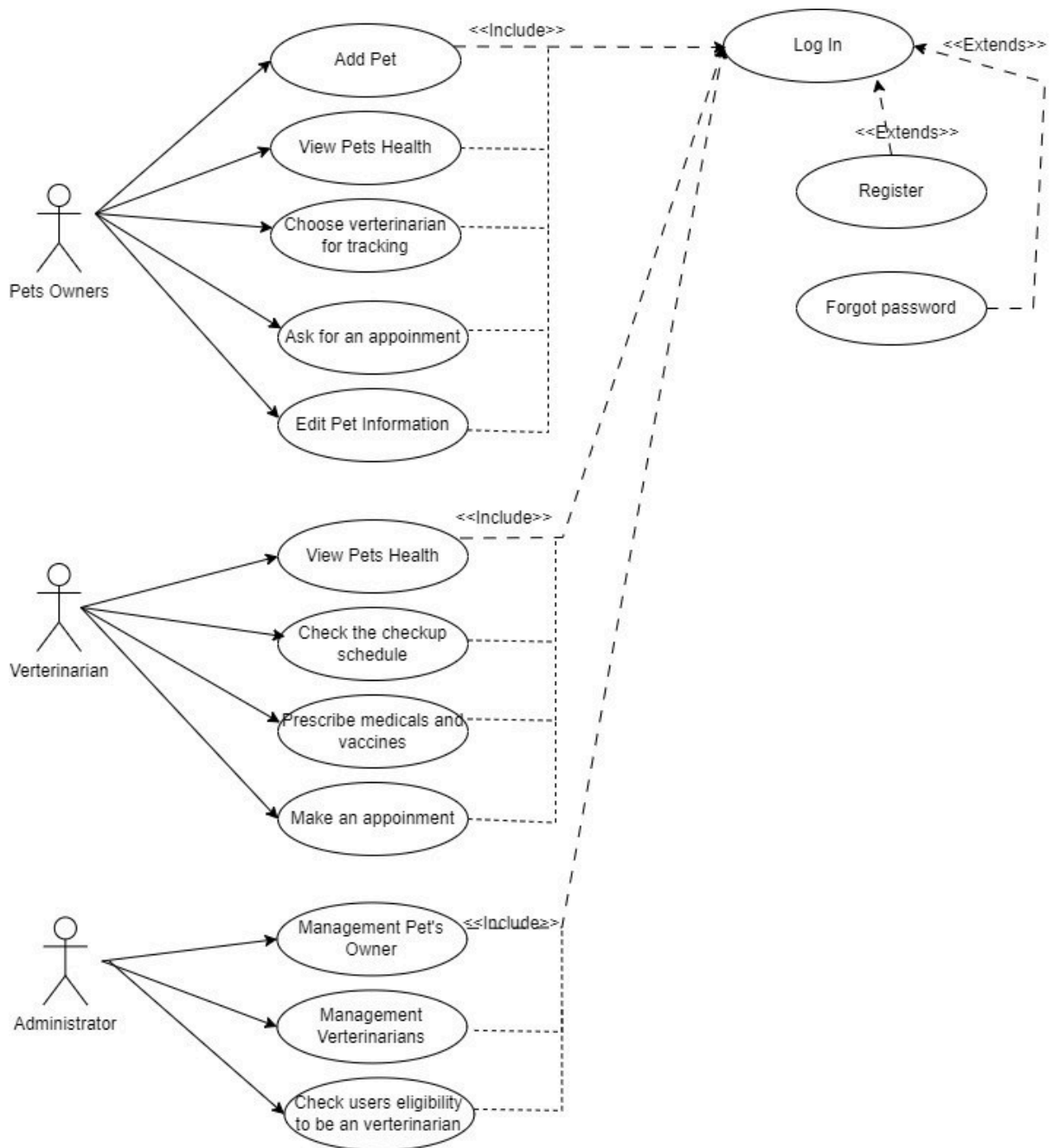
**Security Features:** Java has built-in security features, including a robust security model that protects against various security threats such as unauthorized access, tampering, and malware. The JVM implements strict access controls, sandboxing, and encryption mechanisms to ensure the integrity and confidentiality of Java applications.

**Large Ecosystem and Community:** Java has a vast ecosystem of libraries, frameworks, tools, and resources supported by a large and active developer community. Popular frameworks such as Spring, Hibernate, and Apache Commons provide additional utilities and abstractions for building scalable and maintainable applications.

## CHAPTER III: SYSTEM ANALYSIS

### 3.1 UseCase

Use cases play a crucial role in software development by documenting user requirements, clarifying system behavior, facilitating communication among stakeholders, guiding system design, supporting testing and validation, enabling requirement traceability, and serving as a basis for Agile development practices. They help ensure that the resulting system meets user needs and expectations effectively.



*Figure 8: Veterinary Tracker UseCase[8]*

In our system, Pets Owners, Veterinarians and Administrators are the main interacting subjects. By logging, Pets Owners can manage their own pets (add a pet, view pets health and edit the information of their pets), they can also contact a Veterinarian to track the pets on the system.

Veterinarian users would be authenticated (by providing business license and related certificates and degrees) to improve Service Reliability. By the way, the Veterinarian users can view the medical records, prescribe medicals and vaccines to the registered pets.

The Administrator is the manager of the entire system including viewing all data sets, editing, adding new object data or removing a data object. Additionally, the admin also can check if the user is eligible to register a veterinary business.

### **3.2 Database Diagram**

A database diagram, also known as an entity-relationship diagram (ER diagram), is a visual representation of the structure of a database. It illustrates the relationships between different entities (tables) in the database and the attributes (columns) associated with each entity.

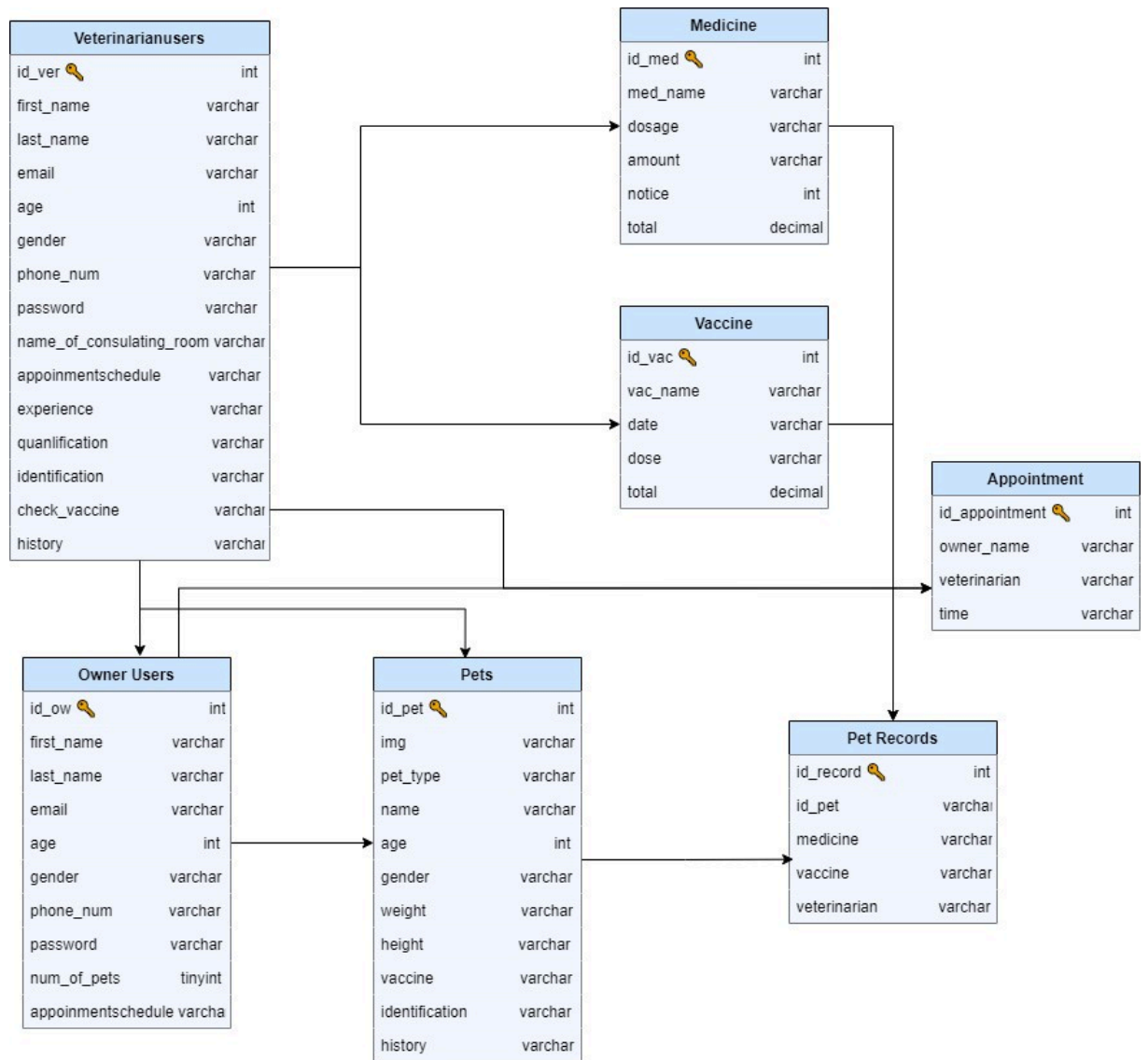


Figure 9: Veterinary Tracker Database diagram[9]

## Chapter IV: IMPLEMENTATION

### 4.1 Building REST services with Spring

The use of RESTful API architecture in building web offers benefits such as modular development, scalability, flexibility, statelessness, separation of concerns, enhanced security, and streamlined documentation and testing processes. These benefits contribute to the development of a robust, maintainable, and extensible system that can adapt to evolving business requirements and user needs[10]. Accordingly, we used the Spring portfolio to build a RESTful service while leveraging the stackless features of REST:

#### 4.1.1 Model:

The Model represents the data and business logic of the application. In the context of a REST API, the Model typically consists of data structures or classes that represent the resources being manipulated by the API. These resources could be objects like users, medicines, appointments, etc. Following that:

`@Id` is the JPA annotation used to designate the field as the primary key of the entity.

`@GeneratedValue(strategy = GenerationType.IDENTITY)` is the other JPA annotation used to specify how the primary key value will be generated. In this case, we just set `GenerationType.IDENTITY`, which typically means that the database will automatically assign a unique identifier to the primary key field when a new row is inserted.

Below would be the structure of one of the main models:

- **Model of Owner**

`@Entity`

`@Table(name = "owneruser")`

`@JsonIgnoreProperties({ "hibernateLazyInitializer", "handler" })`

```
public class Owner {  
    private String firstName;  
    private String lastName;  
    private String email;  
    private String gender;  
    private String phoneNum;  
    private String password;  
    private String appointmentschedule;  
    private Integer age;  
    private Integer numOfPets;  
}
```

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Integer idOw;
```

#### 4.1.2 Service

Define methods for CRUD operations (Create, Read, Update, Delete) and any additional business logic required for processing requests. The JpaRepository interface defines methods for CRUD operations on entities.

We created a service that uses JpaRepository to access data from the database. Below would be the structure of one of the main repositories:

- **OwnerRepository**

```
package com.example.project.repository;
import org.springframework.data.jpa.repository.JpaRepository;
import com.example.project.model.Owner;
public interface OwnerRepository extends JpaRepository<Owner, Integer> {
}
```

#### 4.1.3 Controller

The Controller is responsible for handling incoming requests, interpreting them, invoking the appropriate operations on the Service layer, and returning the corresponding responses to the client. In a REST API, Controllers typically correspond to different endpoints or routes exposed by the API. Each Controller is responsible for a specific set of related functionalities.

Controllers extract data from incoming requests (request parameters, request body) and pass it to the appropriate Service methods for processing. After receiving the processed data from the Service layer, Controllers format it into an appropriate response format (JSON) and send it back to the client.

@RestController annotation is a combination of @Controller and @ResponseBody annotations. @RequestMapping("") to declare the urls of the apis in the controller.

#### REST API GET method

We used a Java method annotated with @GetMapping in Spring Framework for defining RESTful endpoints. This specific method defines a GET endpoint that returns all subject data from a repository. The findAll() or findById(id) method fetches all data entities from the repository and returns them as a list, here is the structure of the class:

- **OwnerController**

***Get all Owner:***

```
@GetMapping("/get-all-owner")

public List<Owner> getAllOwner() {
    return ownerRepository.findAll();
}
```

***Get Owner by ID:***

```
@GetMapping("/get-all-owner/{identity}")
public Owner getSingleOwner(@PathVariable("identity") Integer idOw) {
    return ownerRepository.findById(idOw).get();
}
```

**REST API DELETE method**

When a DELETE request is made to the `"/remove.../{id}"` endpoint with a specific ID, this method will attempt to find and delete the object with that ID from the repository. If the object is successfully deleted, the method returns true; otherwise, it returns false.

- **OwnerController**

```
@DeleteMapping("/removeOwner/{id}")
public boolean deleteOwnerRow(@PathVariable("id") Integer idOw) {
    if (!ownerRepository.findById(idOw).equals(Optional.empty())) {
        ownerRepository.deleteById(idOw);
        return true;
    }
    return false;
}
```

**REST API PUT method**

This specific method defines a PUT endpoint that updates an object's information based on their ID. The `@PathVariable("id")` annotation indicates that the value of the id path variable from the URI should be bound to the id parameter. The `@RequestBody` annotation indicates that the method parameter body should be bound to the body of the HTTP request.

So, when a PUT request is made to the `"/update.../{id}"` endpoint with a specific ID and a request body containing updated object information, this method will update the object's information in the repository and return the updated object entity.

- **OwnerController**

```
@PutMapping("/updateOwner/{id}")
```



```

public Owner updateOwner(@PathVariable("id") Integer idOw, @RequestBody
Map<String, String> body) {
    Owner current = ownerRepository.findById(idOw).get();
    current.setAge(Integer.parseInt(body.get("age")));
    current.setAppointmentschedule(body.get("appointmentschedule"));
    current.setEmail(body.get("email"));
    current.setFirstName(body.get("firstName"));
    current.setGender(body.get("gender"));
    current.setLastName(body.get("lastName"));
    current.setNumOfPets(Integer.parseInt(body.get("numOfPets")));
    current.setPassword(body.get("password"));
    current.setPhoneNum(body.get("phoneNum"));
    ownerRepository.save(current);
    return current;
}

```

## REST API POST method

This method handles HTTP POST requests to the "/addOwner" endpoint. It expects a JSON request body containing key-value pairs representing the attributes of an owner. It extracts these attributes from the request body, creates a new Owner object with the extracted data, and then saves the new owner to the repository using the save() method of ownerRepository. Finally, it returns the saved owner.

- **OwnerController**

```

@PostMapping("/addOwner")
public Owner addOwner(@RequestBody Map<String, String> body) {
    String firstName = body.get("firstName");
    String lastName = body.get("lastName");
    String email = body.get("email");
    String gender = body.get("gender");
    String phoneNum = body.get("phoneNum");
    String password = body.get("password");
    String appointmentschedule = body.get("appointmentschedule");
    Integer age = Integer.parseInt(body.get("age"));
    Integer numOfPets = Integer.parseInt(body.get("numOfPets"));
    Owner newOwner = new Owner(firstName, lastName, email, gender, phoneNum,
password,
    appointmentschedule, age, numOfPets);
}

```

```
    return ownerRepository.save(newOwner);
}
```

**We configure the connection to the database in the application.properties file**

```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/client
spring.datasource.username=root
spring.datasource.password=123123123
```

**For the testing of REST API (In case of Pet Owners Testing):**

- *Get All users:*

```
{
  "firstName": "John",
  "lastName": "Doe",
  "email": "johndoe@example.com",
  "gender": "Male",
  "phoneNum": "1234567890",
  "password": "hashed_password_1",
  "appointmentschedule": "Monday 10:00 AM",
  "age": 30,
  "numOfPets": 2,
  "idOw": 1
},
{
  "firstName": "Jane",
  "lastName": "Smith",
  "email": "janesmith@example.com",
  "gender": "Female",
  "phoneNum": "9876543210",
  "password": "hashed_password_2",
  "appointmentschedule": "Wednesday 2:00 PM",
  "age": 25,
  "numOfPets": 1,
  "idOw": 2
},
```

```

{
  "firstName": "Michael",
  "lastName": "Johnson",
  "email": "michaeljohnson@example.com",
  "gender": "Male",
  "phoneNum": "1112223333",
  "password": "hashed_password_3",
  "appointmentschedule": "Friday 3:00 PM",
  "age": 40,
  "numOfPets": 3,
  "idOw": 3
},
{
  "firstName": "Emily",
  "lastName": "Williams",
  "email": "emilywilliams@example.com",
  "gender": "Female",
  "phoneNum": "4445556666",
  "password": "hashed_password_4",
  "appointmentschedule": null,
  "age": 35,
  "numOfPets": 0,
  "idOw": 4
}

```

- *Get user by ID:*

```

{
  "firstName": "John",
  "lastName": "Doe",
  "email": "johndoe@example.com",
  "gender": "Male",
  "phoneNum": "1234567890",
  "password": "hashed_password_1",
  "appointmentschedule": "Monday 10:00 AM",
  "age": 30,

```

```

    "numOfPets": 2,

    "idOw": 1

}

```

- *Delete user by ID*: return true and update to database

true

id_ow	first_name	last_name	email	age	gender	phone_num	password	num_of_pets	appointmentschedule
1	John	Doe	johndoe@example.com	30	Male	1234567890	hashed_password_1	2	Monday 10:00 AM
2	Jane	Smith	janesmith@example.com	25	Female	9876543210	hashed_password_2	1	Wednesday 2:00 PM
3	Michael	Johnson	michaeljohnson@example.com	40	Male	1112223333	hashed_password_3	3	Friday 3:00 PM
4	Emily	Williams	emilywilliams@example.com	35	Female	4445556666	hashed_password_4	0	NULL
5	David	Brown	davidbrown@example.com	28	Male	7778889999	hashed_password_5	2	Tuesday 4:00 PM
6	Sarah	Jones	sarahjones@example.com	45	Female	5554443333	hashed_password_6	1	Thursday 1:00 PM
7	Chris	Lee	chrisee@example.com	33	Male	2223334444	hashed_password_7	4	Saturday 11:00 AM
8	Amanda	Davis	amandadavis@example.com	29	Female	6667778888	hashed_password_8	2	Wednesday 10:00 AM
9	Matthew	Miller	matthewmiller@example.com	38	Male	9998887777	hashed_password_9	1	Monday 2:00 PM
10	Laura	Wilson	laurawilson@example.com	32	Female	3332221111	hashed_password_10	3	Thursday 3:00 PM
11	Kevin	Taylor	kevintaylor@example.com	27	Male	8889990000	hashed_password_11	0	NULL
12	Hannah	Martinez	hannahmartinez@example.com	36	Female	6661112222	hashed_password_12	2	Friday 10:00 AM
13	Andrew	Anderson	andrewanderson@example.com	31	Male	5556667777	hashed_password_13	1	Tuesday 3:00 PM
14	Megan	Thomas	meganthomas@example.com	39	Female	2221113333	hashed_password_14	2	Saturday 2:00 PM
15	Jason	Hernandez	jasonhernandez@example.com	26	Male	7776665555	hashed_password_15	3	Wednesday 11:00 AM

Figure 10: Table of Deleting user

- *Add new User*: test by JSON Body in Postman

```

{

    "firstName": "Matthew",

    "lastName": "Will",

    "email": "math123@example.com",

    "gender": "male",

    "phoneNum": "8812340000",

    "password": "123123123",

    "appointmentschedule": "Wednesday 11:00 AM",

    "age": 20,

    "numOfPets": 2,

    "idOw": 17

}

```

id_ow	first_name	last_name	email	age	gender	phone_num	password	num_of_pets	appointmentschedule
1	John	Doe	johndoe@example.com	30	Male	1234567890	hashed_password_1	2	Monday 10:00 AM
2	Jane	Smith	janesmith@example.com	25	Female	9876543210	hashed_password_2	1	Wednesday 2:00 PM
3	Michael	Johnson	michaeljohnson@example.com	40	Male	1112223333	hashed_password_3	3	Friday 3:00 PM
4	Emily	Williams	emilywilliams@example.com	35	Female	4445556666	hashed_password_4	0	NULL
5	David	Brown	davidbrown@example.com	28	Male	7778889999	hashed_password_5	2	Tuesday 4:00 PM
6	Sarah	Jones	sarahjones@example.com	45	Female	5554443333	hashed_password_6	1	Thursday 1:00 PM
7	Chris	Lee	chrislee@example.com	33	Male	2223334444	hashed_password_7	4	Saturday 11:00 AM
8	Amanda	Davis	amandadavis@example.com	29	Female	6667778888	hashed_password_8	2	Wednesday 10:00 AM
9	Matthew	Miller	matthewmiller@example.com	38	Male	9998887777	hashed_password_9	1	Monday 2:00 PM
10	Laura	Wilson	laurawilson@example.com	32	Female	3332221111	hashed_password_10	3	Thursday 3:00 PM
11	Kevin	Taylor	kevintaylor@example.com	27	Male	8889990000	hashed_password_11	0	NULL
12	Hannah	Martinez	hannahmartinez@example.com	36	Female	6661112222	hashed_password_12	2	Friday 10:00 AM
13	Andrew	Anderson	andrewanderson@example.com	31	Male	5556667777	hashed_password_13	1	Tuesday 3:00 PM
14	Megan	Thomas	meganthomas@example.com	39	Female	2221113333	hashed_password_14	2	Saturday 2:00 PM
15	Jason	Hernandez	jasonhernandez@example.com	26	Male	7776665555	hashed_password_15	3	Wednesday 11:00 AM
17	Matthew	Will	math123@example.com	20	male	8812340000	123123123	2	Wednesday 11:00 AM

Figure 11: Table of Adding new user

- *Edit User by ID:* test by JSON Body in Postman

17	Matthew	Will	math123@example.com	20	female	8812340000	123123123	2	Wednesday 11:00 AM
----	---------	------	---------------------	----	--------	------------	-----------	---	--------------------

Figure 12: Table of Editing user

## 4.2 Admin Management Page

The Admin user interface is designed to support a maximum for system user management, including the functions of viewing all objects, adding, deleting, and editing objects for different storage purposes.

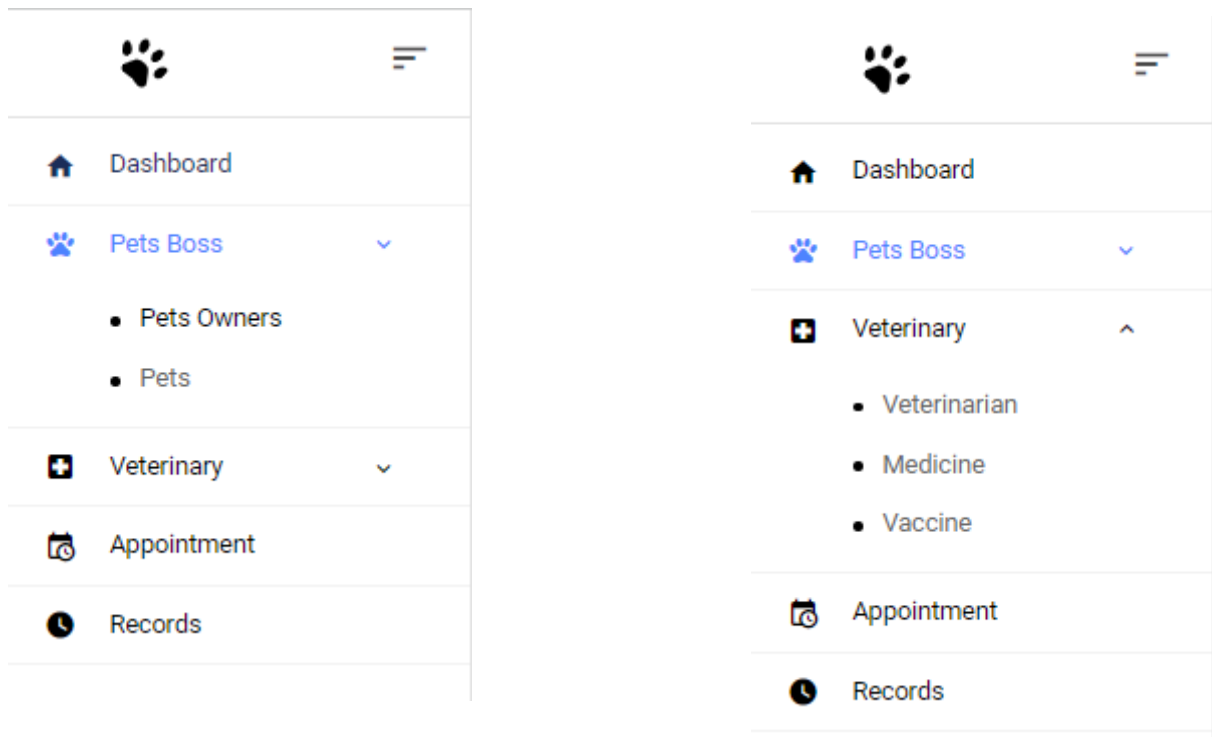


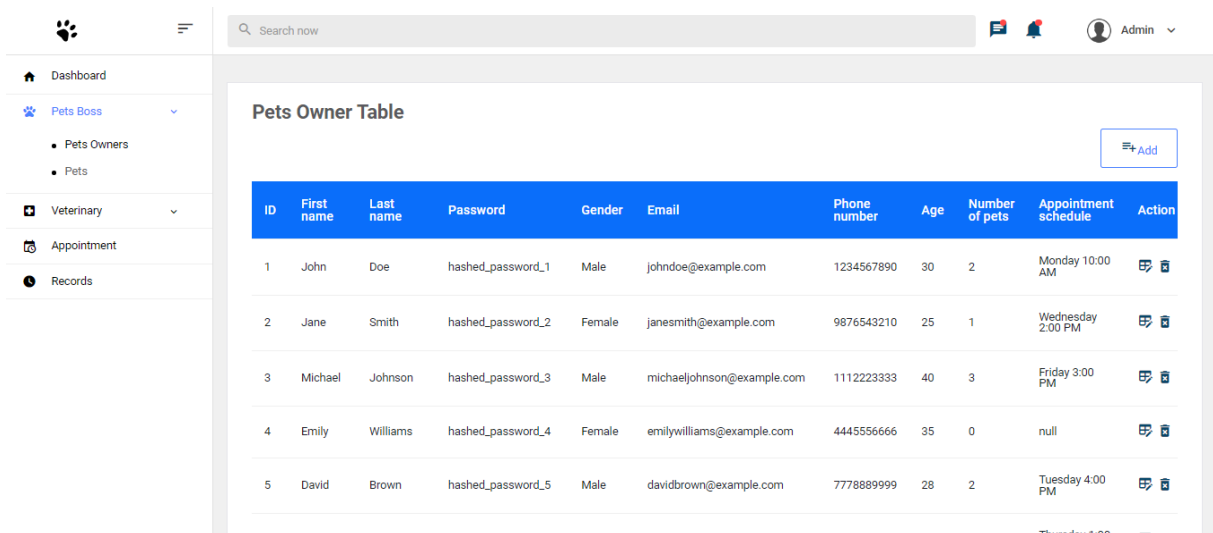
Figure 13: Menu (Admin site)

The sidebar menu contains a total of eight items. The dashboard provides an overview of key metrics and insights such as the number of registered users and recent activities. Beside

that, in indexing Pets Boss, it has two submenus including Pets Owners and Pets. The indexing Veterinary has three submenus Veterinarian, Vaccine and Medicine, and the rest are Appointment and Records.

#### 4.2.1 Pets Owners:

- Get all Pets Owners user













ID	First name	Last name	Password	Gender	Email	Phone number	Age	Number of pets	Appointment schedule	Action
1	John	Doe	hashed_password_1	Male	john.doe@example.com	1234567890	30	2	Monday 10:00 AM	 
2	Jane	Smith	hashed_password_2	Female	jane.smith@example.com	9876543210	25	1	Wednesday 2:00 PM	 
3	Michael	Johnson	hashed_password_3	Male	michael.johnson@example.com	1112223333	40	3	Friday 3:00 PM	 
4	Emily	Williams	hashed_password_4	Female	emily.williams@example.com	4445556666	35	0	null	 
5	David	Brown	hashed_password_5	Male	david.brown@example.com	7778889999	28	2	Tuesday 4:00 PM	 

Figure 14: Pets Owner Table (Admin site)

The table has eleven columns following the data: ID, First name, Last name, Password, Gender, Email, Phone number, Age and Appointment schedule. The column Action just created to help admin to be able to edit and delete the data with ease.

We used the fetch API to get the data from the RESTful API through the url '<http://localhost:8080/get-all-owner>'. The key async and await allows us to write asynchronous code in a more synchronous and readable way. In the function RenderItem(data), DOM was used to create the content for the table data.

- Add new Pets Owners user

When pressing the button Add above the table, the site would redirect to the form containing the details of Pet owner user, and when pressing add new button, the new data would be saved and updated to the database also.

## Add Pets Owner

Jack

Hollien

jackpassword\_1

2736613332

jackhollien@example.com

Gender

Male



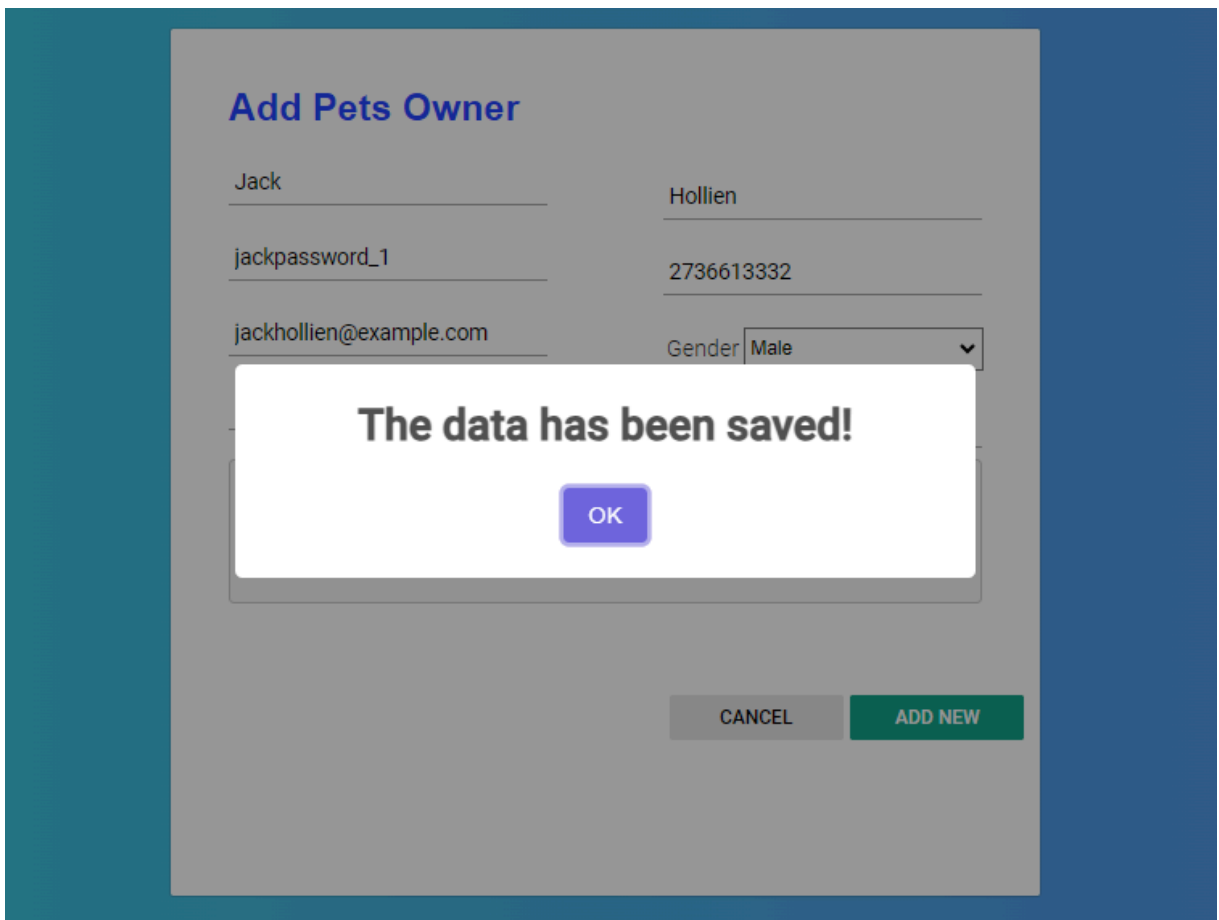
21

1

Appointment Schedule

CANCEL

ADD NEW



*Figure 14: Add new Pets Owner (Admin site)*

The Add new data function, the var payload stored the data of variables (following the structure of Owner class in API site, then we just saved them as the body data type JSON, with the POST method.

- Edit Pets Owners user by ID

The editing form would pop up, the data of the selected row was filled in the form, the admin user just needed to edit the data in the form, and by clicking the edit button the data would be saved.



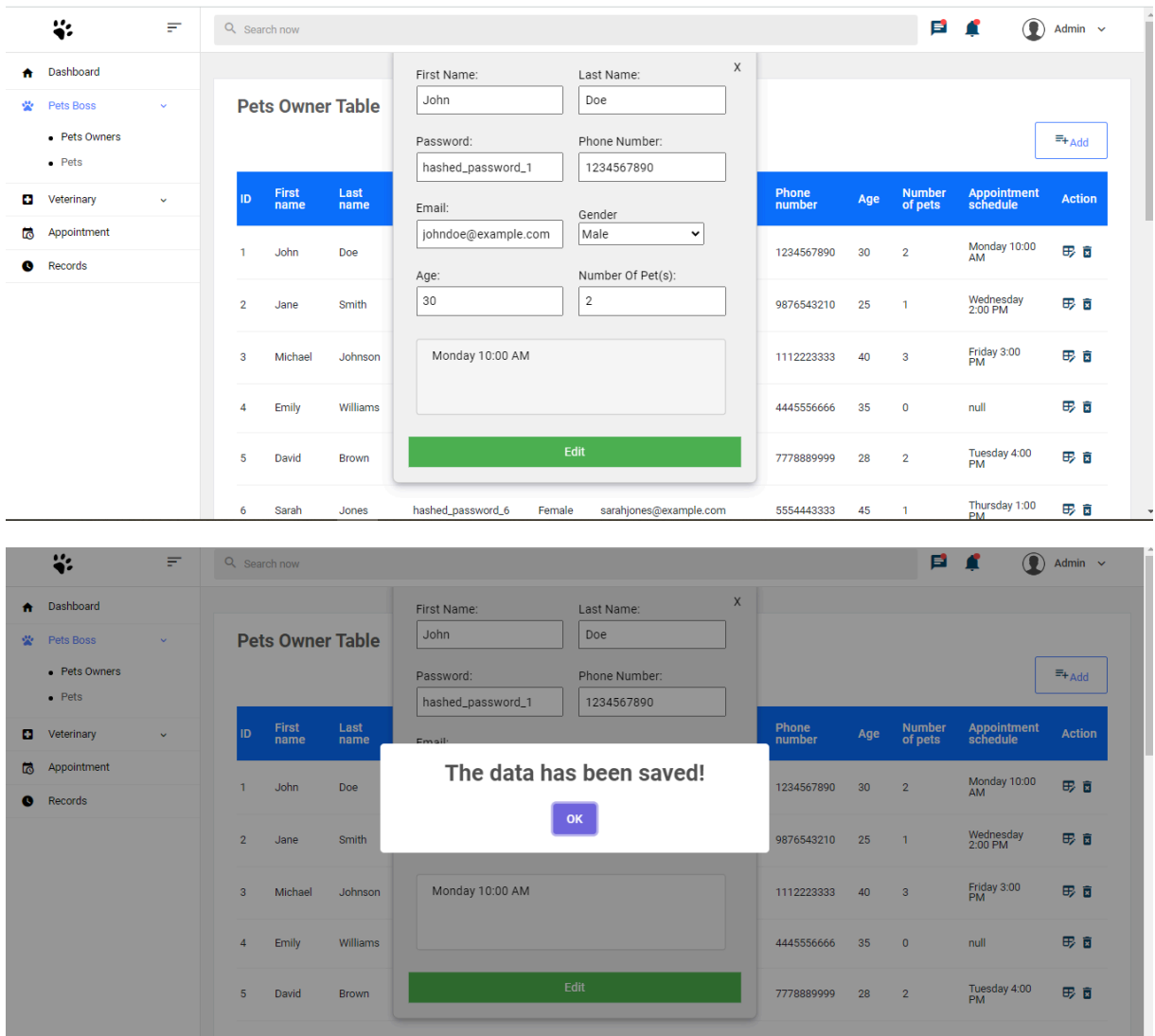


Figure 15: Edit Pets Owner (Admin site)

Be like the POST method, the function Edit data has been written (with the PUT method).

- Delete Pets Owners user by ID

The button delete is set in each row of the data object, before delete it, the system would confirm the user that has been selected.

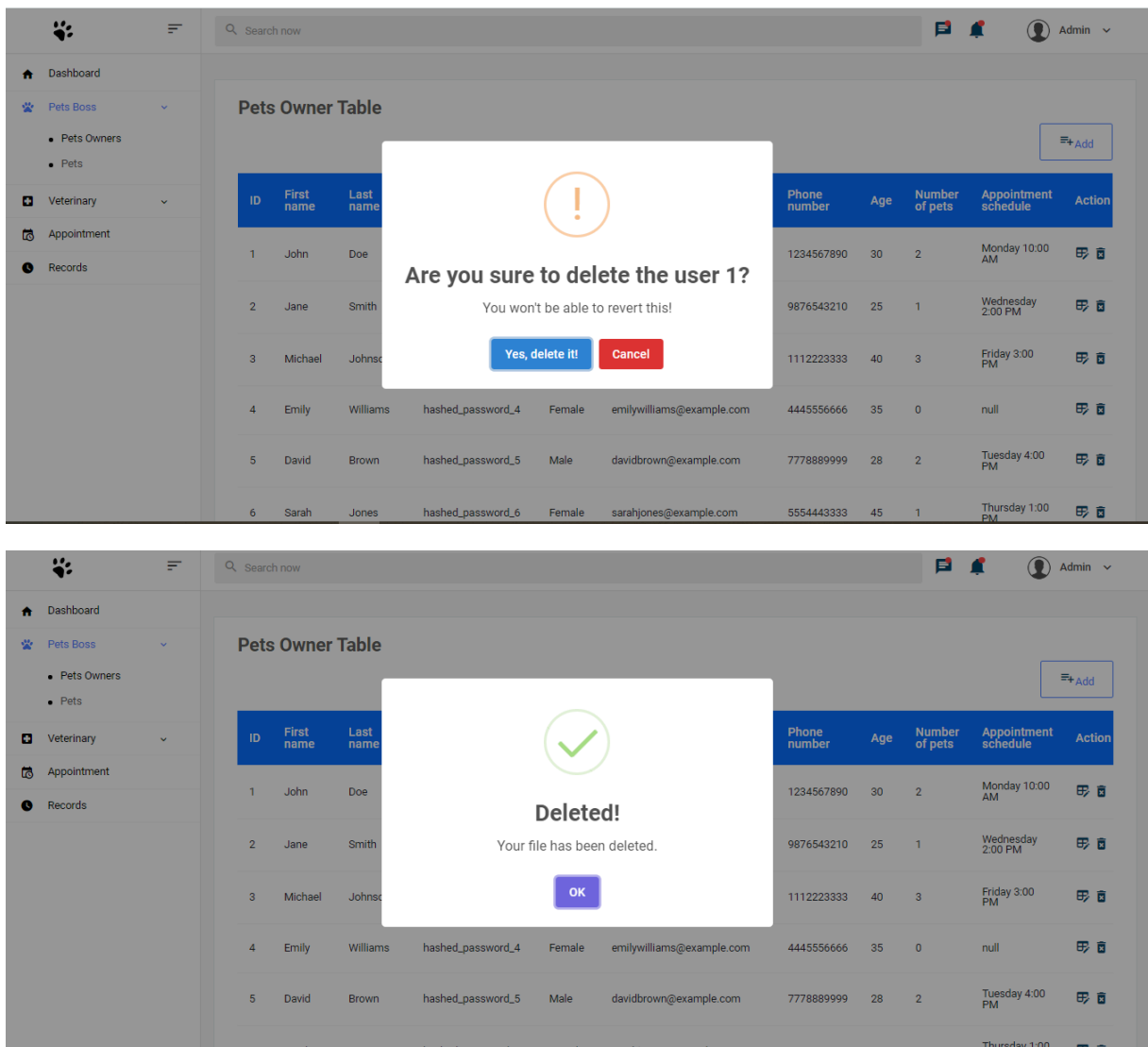


Figure 16: Delete Pets Owner (Admin site)

The function of delete post by key, the key (ID) would be passed by clicking the delete button in the row, the url was called to the address of remove Owner API and following that, the method 'DELETE' was called in the fetch body to remove the item and update it in database set.

Likewise, the data tables of the remaining subjects are also organized according to a similar structure

#### 4.2.2 Pets:

The Pets object has been stored by the data fields ID, name, pet type, gender, weight, height, identification (identifying characteristics of pets), vaccine (which would be assigned by veterinarian) and the history (which would be stored when having a consultation) and for the column pet, the image could be uploaded by user and they would be fixed with the table format.

**Pets Table**

ID	Pet	Name	Pet type	Gender	Age	Weight	Height	Identification	Vaccine	History	Action
1		Buddy	Dog	Male	3	15 kg	50 cm	Golden retriever with a white spot on the chest	Rabies, DHPP	Regular check-ups indicate Buddy is healthy and active.	
2		Whiskers	Cat	Female	2	5 kg	30 cm	Black and white with green eyes	FVRCP	Whiskers has been vaccinated and has no health issues.	
3		Max	Dog	Male	5	20 kg	55 cm	German shepherd with brown patches	Rabies, DHPP, Bordetella	Max has had occasional ear infections, otherwise healthy.	
4		Snowball	Rabbit	Female	1	1.5 kg	20 cm	White with floppy ears	null	Snowball's dental health is good, no other concerns.	
5		Charlie	Bird	Male	2	null	null	Green and yellow feathers	Polyomavirus, Psittacosis	Charlie's wings have been clipped, otherwise in good health.	

Figure 17: Table of Pets (Admin site)

### Add New Pet

Pet Name

Age

Weight

Vaccine

Pet Type

Gender ☐ Male ☐ Female

Height

Pets Identification

Figure 18: Add new pet form (Admin site)

#### 4.2.3 Veterinarian:

The table has fifteen columns: the Veterinarian object has the basic data as the Pets Owners object, Some of the added data fields to serve the clinic business are Consulting Room Name, Qualification, Experience, Identification (to verify eligibility for veterinary business), Check Vaccine (for the pets that has registered) , and medical history.

Pets Boss

Veterinary

- Veterinarian
- Medicine
- Vaccine

Appointment

Records

Search now

Admin

Veterinarian Table

Add

ID	First name	Last name	Password	Gender	Email	Phone number	Age	Consulting Room Name	Quanlification	Experience	
1	John	Doe	pass123	Male	john.doe@email.com	123-456-7890	35	Room 101	DVM	10 years	
2	Alice	Johnson	secret321	Female	alice.johnson@email.com	987-654-3210	28	Room 202	BVSc	5 years	
3	David	Smith	vetpass	Male	david.smith@email.com	555-123-4567	42	Room 303	DVM	15 years	
4	Emily	White	petcare22	Female	emily.white@email.com	111-222-3333	30	Room 404	BVetMed	8 years	
5	Michael	Brown	animaldoc	Male	michael.brown@email.com	777-888-9999	38	Room 505	DVM	12 years	

Dashboard

Pets Boss

Veterinary

- Veterinarian
- Medicine
- Vaccine

Appointment

Records

Search now

Admin

Veterinarian Table

Add

	Phone number	Age	Consulting Room Name	Quanlification	Experience	Identification	Check Vaccine	Appointment schedule	History	Action
xe@email.com	123-456-7890	35	Room 101	DVM	10 years	YES	Null	2024-02-01 10:00 AM	Null	
hnson@email.com	987-654-3210	28	Room 202	BVSc	5 years	YES	Null	2024-02-03 02:30 PM	Null	
mith@email.com	555-123-4567	42	Room 303	DVM	15 years	YES	Null	2024-02-05 11:15 AM	Null	
hite@email.com	111-222-3333	30	Room 404	BVetMed	8 years	YES	Null	2024-02-08 03:45 PM	Null	
l.brown@email.com	777-888-9999	38	Room 505	DVM	12 years	YES	Null	2024-02-10 09:30 AM	Null	

Figure 19: Table of Veterinarians (Admin site)

## Add Verterinarian

Gender

Male

▼

Quanlification:

☐ DVM
☐ BVSc
☐ BVetMed

Appointment Schedule

Check Vaccine

**Authentication**  

No

CANCEL

ADD NEW

Figure 20: Add new Veterinarian form (Admin site)

### 4.2.4 Medicine:

The Medicine object is managed by the Veterinarian user, the Veterinarian is responsible for specifying the medicine name, the dosage, the amount as well as the notice, and the total payment for each medicine prescribed in the prescription.

☰

Admin ▼

Dashboard

Pets Boss ▼

Veterinary ▼

- Veterinarian
- Medicine
- Vaccine

Appointment

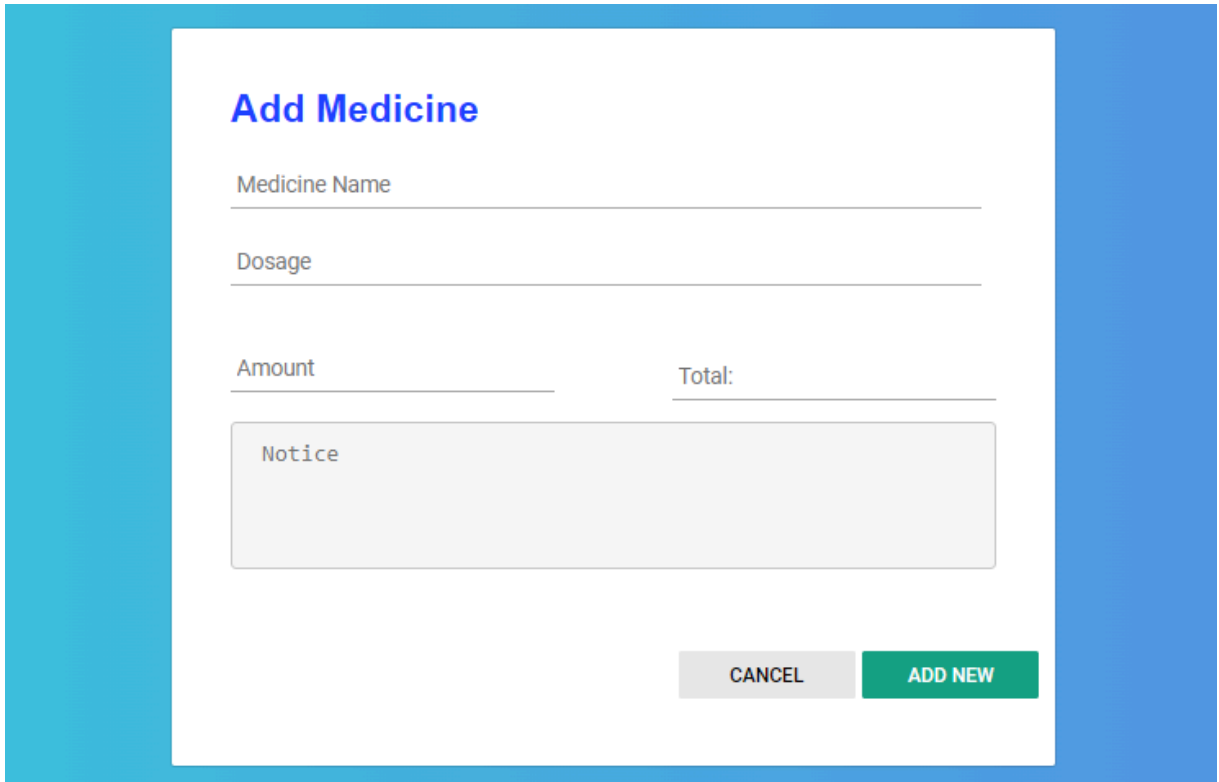
Records

### Medicine Table

➕Add

#	Medicine	Dosage	Amount	Notice	Total	Action
1	APA DICHLO I	2 times a day	100 tablets	null	\$128	
2	Vetoquinol	1 tablet per day	30 tablets	Take with food	\$25.5	
3	Petcam	1 drop in each eye twice a day	10 ml	Shake well before use	\$15.75	
4	Flexadin Advanced	1 chewable tablet daily	60 tablets	For joint health	\$40	
5	NexGard	1 chewable tablet once a month	3 tablets	Effective against fleas and ticks	\$35	
6	Bravecto	1 chewable tablet every 3 months	1 tablet	Protects against fleas and ticks	\$20	

Figure 21: Table of Medicine (Admin site)

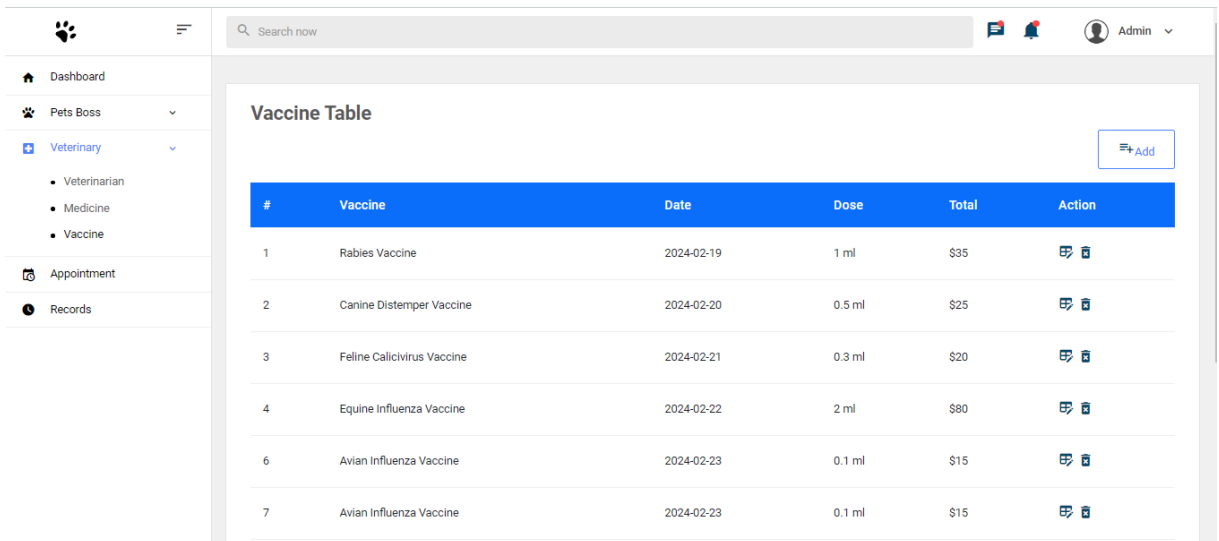


The 'Add Medicine' form is displayed within a blue-bordered container. It features a title 'Add Medicine' in bold blue text. Below the title are three input fields: 'Medicine Name', 'Dosage', and 'Amount'. To the right of the 'Amount' field is a 'Total:' label followed by another input field. A large, light gray rectangular box labeled 'Notice' is positioned below these fields. At the bottom right, there are two buttons: a gray 'CANCEL' button and a green 'ADD NEW' button.













Figure 22: Add new Medicine form (Admin site)

#### 4.2.5 Vaccine:

Similar to the Medicine object, the vaccine is also in the Veterinarian managed category, including the data fields Vaccine name, Date, Dose and Total payment.

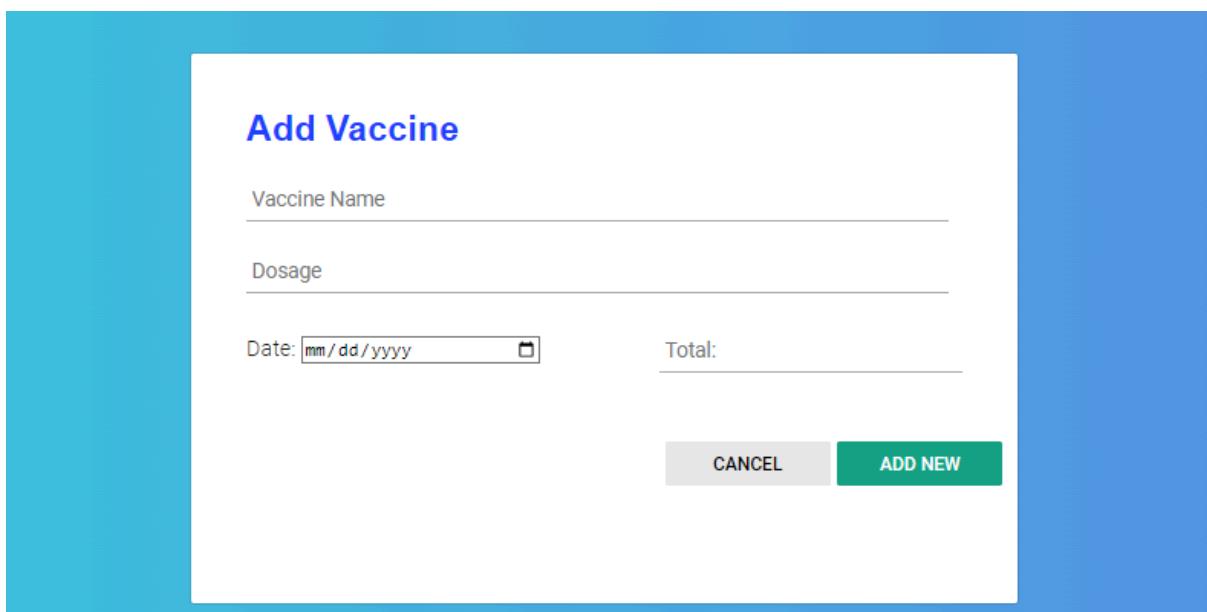


The screenshot shows the 'Vaccine Table' interface. On the left is a sidebar with navigation links: Dashboard, Pets Boss, Veterinary (expanded), Appointment, and Records. The 'Veterinary' section includes sub-links for Veterinarian, Medicine, and Vaccine. The main content area displays a table with the following data:

#	Vaccine	Date	Dose	Total	Action
1	Rabies Vaccine	2024-02-19	1 ml	\$35	 
2	Canine Distemper Vaccine	2024-02-20	0.5 ml	\$25	 
3	Feline Calicivirus Vaccine	2024-02-21	0.3 ml	\$20	 
4	Equine Influenza Vaccine	2024-02-22	2 ml	\$80	 
6	Avian Influenza Vaccine	2024-02-23	0.1 ml	\$15	 
7	Avian Influenza Vaccine	2024-02-23	0.1 ml	\$15	 

An 'Add' button with a plus icon is located in the top right corner of the table area.


Figure 23: Table of Vaccine (Admin site)



**Add Vaccine**

Vaccine Name \_\_\_\_\_

Dosage \_\_\_\_\_

Date:  

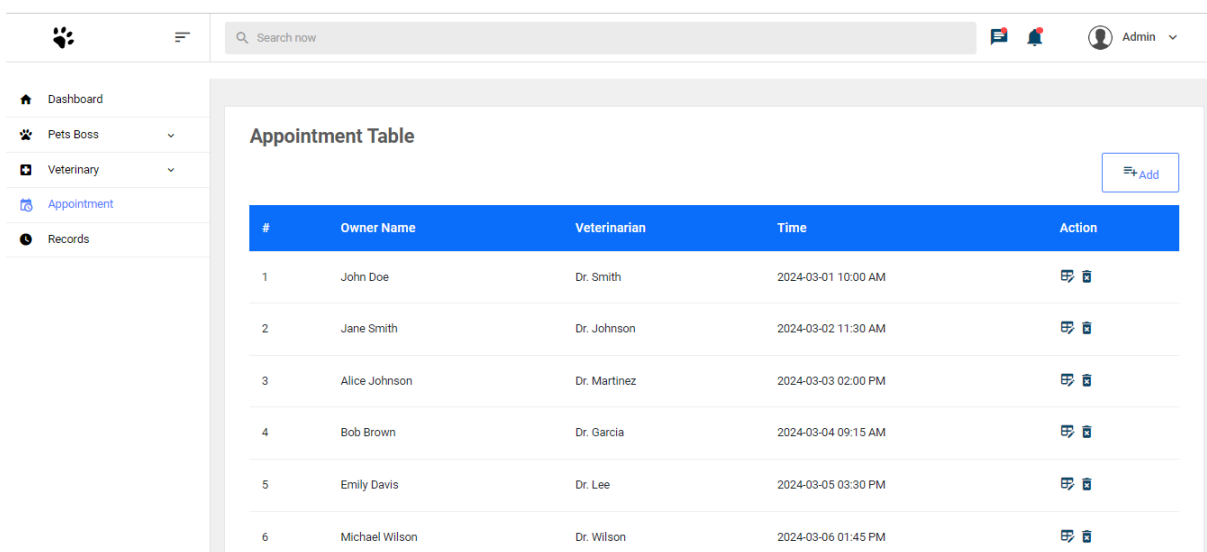
Total: \_\_\_\_\_

**CANCEL** **ADD NEW**

Figure 24: Add new Vaccine form (Admin site)

#### 4.2.6 Appointment:

The Appointment table is used to manage all the consultation between Pets owner users and Veterinarian users.















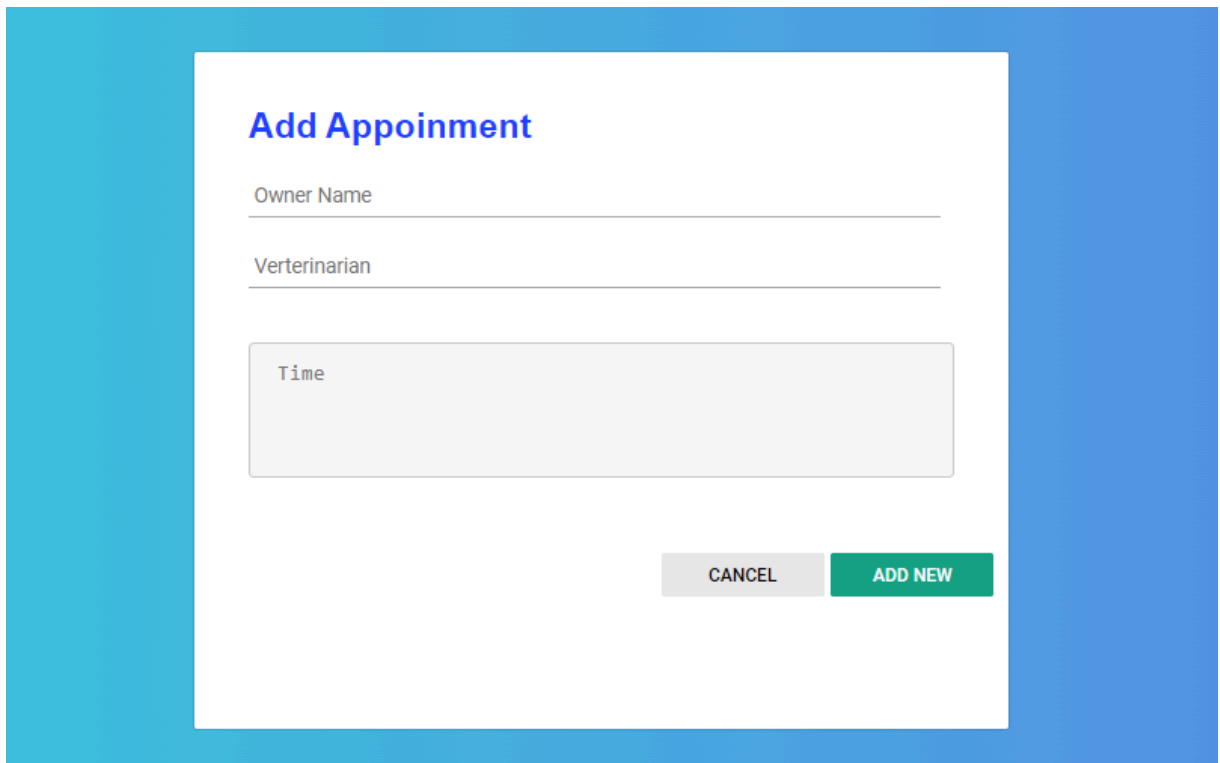
#	Owner Name	Veterinarian	Time	Action
1	John Doe	Dr. Smith	2024-03-01 10:00 AM	 
2	Jane Smith	Dr. Johnson	2024-03-02 11:30 AM	 
3	Alice Johnson	Dr. Martinez	2024-03-03 02:00 PM	 
4	Bob Brown	Dr. Garcia	2024-03-04 09:15 AM	 
5	Emily Davis	Dr. Lee	2024-03-05 03:30 PM	 
6	Michael Wilson	Dr. Wilson	2024-03-06 01:45 PM	 

Figure 25: Table of Appointment (Admin site)

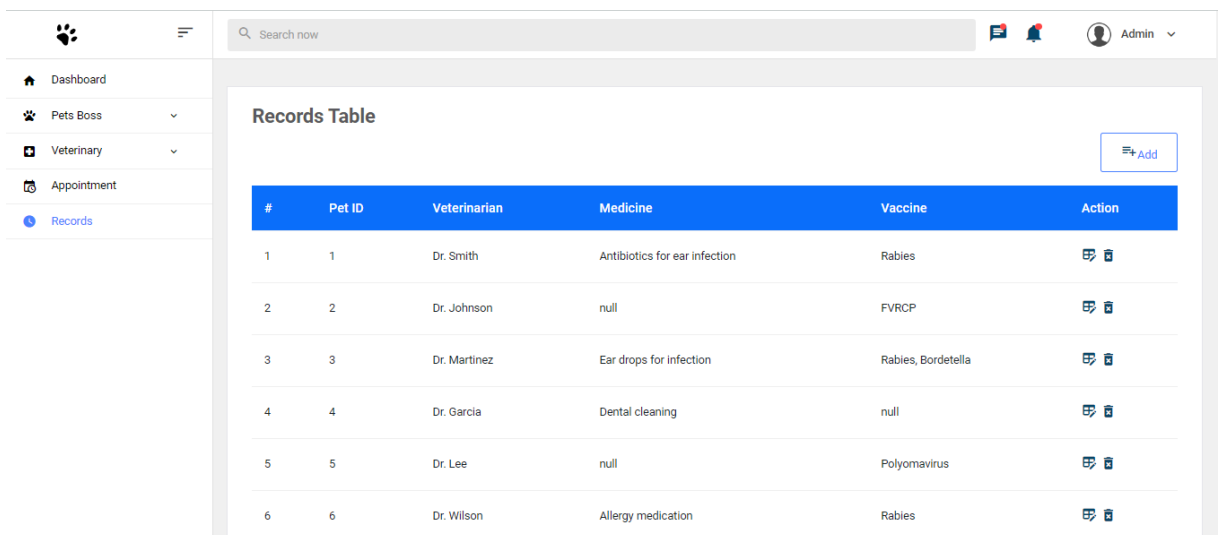


The image shows a web form titled "Add Appointment" in blue text. Below the title are three input fields: "Owner Name", "Verterinarian" (note the spelling), and "Time". The "Time" field is a larger, light gray rectangular box. At the bottom right of the form are two buttons: a gray "CANCEL" button and a green "ADD NEW" button.

Figure 26: Add new Appointment form (Admin site)

#### 4.2.7 Records:

The Records table is used to store the history of all the pets objects. Following the data fields:



The image shows a web application interface with a sidebar on the left containing navigation links: Dashboard, Pets Boss, Veterinary, Appointment, and Records. The main content area displays a "Records Table" with a search bar at the top and an "Add" button. The table has six columns: #, Pet ID, Veterinarian, Medicine, Vaccine, and Action. It contains six rows of data.













#	Pet ID	Veterinarian	Medicine	Vaccine	Action
1	1	Dr. Smith	Antibiotics for ear infection	Rabies	 
2	2	Dr. Johnson	null	FVRCP	 
3	3	Dr. Martinez	Ear drops for infection	Rabies, Bordetella	 
4	4	Dr. Garcia	Dental cleaning	null	 
5	5	Dr. Lee	null	Polyomavirus	 
6	6	Dr. Wilson	Allergy medication	Rabies	 

Figure 27: Table of Veterinarians (Admin site)



**Add Pet Record**

ID Pet \_\_\_\_\_

Verterinarian Name \_\_\_\_\_

Medicine

Vaccine

CANCEL ADD NEW

*Figure 28: Add new Veterinarian form (Admin site)*

## **CHAPTER V: RESULT AND FUTURE WORK**

### **5.1 Achievement:**

#### **5.1.1 Functional API Endpoints Development**

In conclusion, we have successfully created CRUD features that allow clients to interact with the system's resources effectively. Define clear and intuitive URIs for each endpoint to identify the resources within the system. This helps clients understand how to interact with the API and access the desired data or functionality.

Utilize appropriate HTTP methods (such as GET, POST, PUT, DELETE) for CRUD operations and other actions on resources. Each method should be mapped to the corresponding endpoint to ensure a RESTful API design. The development of functional API endpoints results in a reliable, well-designed, and user-friendly interface for interacting with the system's resources.

#### **5.1.2 Functional Admin Page Management**

In terms of User Interface, we have created an intuitive and efficient interface that allows administrators to manage various aspects (providing tool sets such as viewing, deleting, adding, editing and objects) of the system easily.

### **5.2 Future Work**

While significant progress has been made on the 'Veterinary Tracker' project comprehensive validation for veterinary users is pending, sending appointment email and not all functions and interfaces are operating as intended. Many ambitious design elements and functionalities have yet to be implemented. We are committed to addressing these issues and ensuring a smoother and more polished user experience.

For our future, we are going to do our first priority, the Login/ Logout functionality , next is the validation followed by full dreaming design and sending appointment email .Not only that we are going to enhance the user experience by inventing the application that makes the users can access easily and our security of Doctor Approval as well as all functionalities working properly by reporting and advice form the experience of the users.

## REFERENCE

- [1], [2] Craig Walls, Spring Boot in Action, 2015
- [3] Baron Schwartz, High Performance MySQL: Optimization, Backups, and Replication, 2008
- [4] D. R. Brooks, An Introduction to HTML and JavaScript: for Scientists and Engineers, 2007
- [5], [6] Adam Freeman, Pro ASP.NET MVC 5, 2013
- [7] P. Niemeyer, J. Knudsen, Learning Java, 2005
- [8] UML Use Case Diagram Tutorial, from <https://www.lucidchart.com/pages/uml-use-case-diagram>
- [9] Database Structure and Design Tutorial, from <https://www.lucidchart.com/pages/database-diagram/database-design>
- [10] Building REST services with Spring, from <https://spring.io/guides/tutorials/rest>