Project Title :

Spring PetClinic Application

A Full Stack Web Application with React and Springboot

Domain

- Full Stack Web Development
- React and Springboot

Student Details :

S.No	Roll no	Name	E-mail
1	22781A3302	Abhishek Kumar	amazingabhi69@gmail.com
2	22781A3318	B Girinath Reddy	girinathreddy767@gmail.com
3	22781A3326	Chapala Praveen	praveenpraveensimhadri@gmail.com
4	22781A3348	Gudipati Jayasimha Vardhan	jayasimhavardhanj@gmail.com
5	22781A3381	M Subham	msubham246@gmail.com

Batch details : Batch A (CSM-A)

Problem Statement :

Traditional Spring Pet Clinic (Before)

- Built using basic Spring MVC structure with manual configuration.
- Limited architecture — not fully aligned with real enterprise applications.
- No strong security implementation, making access control minimal.
- Relied heavily on old methods for data handling with less automation.
- Clients often did not have complete or accessible data on their pets; records were basic and scattered.
- The system mainly focused on backend operations, not full integration or real business workflows.
- Good for learning fundamentals, but not reflective of today's production-grade systems.

Modern Spring Pet Clinic (Now)

- Uses advanced enterprise technologies: Spring Boot, Spring MVC, Spring Data JPA, Spring Security.
- Follows layered architecture with automated configuration and reduced boilerplate.
- Includes strong security features such as authentication and authorization.
- Offers better data management, allowing clients and staff to view complete pet information, history, appointments, and records in one place.
- Represents real-world, full-stack development with smooth frontend–backend integration.
- Scalable, faster, secure, and easier to maintain.
- Helps students and developers understand how complex enterprise systems operate in real scenarios.

Objectives

The objectives of this project are:

- To design and develop a full stack web application using React and Spring Boot
- To demonstrate CRUD operations on real-world entities such as owners, pets, visits, and veterinarians
- To implement layered architecture using controller, service, and repository layers
- To build RESTful APIs and ensure proper database persistence using JPA
- To provide a clean, maintainable, and easy-to-understand reference application

Scope of the Project

The project focuses on managing veterinary clinic operations including owner management, pet registration, visit tracking, and veterinarian management. The system is designed for educational and learning purposes and supports a single-clinic environment.

Technology Stack

Frontend: React.js v18.2.0, HTML5, CSS3, JavaScript ES6+
Backend: Java v17, Spring Boot v3.9.5, Spring MVC v6.1.x, Spring Data JPA v3.2.x

Database: MySQL v8.0.x
Tools: VS Code, Maven v3.9.5, Git v2.40+, GitHub, Postman v10+

Modules

The system consists of the following modules:

- Owner Module
- Pet Module
- Visit Module
- Veterinarian Module
- Authentication Module

Functional Requirements

- User registration and authentication
- Add, update, view, and delete owner details
- Register and manage pets
- Record and view pet visit history
- View veterinarian details
- Role-based access control

System Architecture (Client–Server Architecture)

The **Spring PetClinic Application** follows a **client–server architecture**, ensuring clear separation between the user interface, business logic, and data layer.

1. Client Layer (Frontend – React)

- The frontend is developed using **React.js (v18.2.0)** along with **HTML5, CSS3, and JavaScript (ES6+)**.
- React provides a **component-based UI**, improving reusability and maintainability.
- Users interact with the system through web pages to manage:
  - Owners
  - Pets
  - Visits
  - Veterinarians
- The frontend does **not directly access the database**.
- All user actions (add, update, view, delete) are sent as **HTTP requests** to the backend using **REST APIs**.

2. Communication Layer (RESTful APIs)

- Communication between frontend and backend is done using **RESTful web services**.
- Data is exchanged in **JSON format**, making it lightweight and platform-independent.
- Common HTTP methods used:
    - **GET** – Fetch data
    - **POST** – Create new records
    - **PUT** – Update existing records
    - **DELETE** – Remove records
- This ensures **loose coupling** between frontend and backend.

3. Backend Layer (Spring Boot Application)

The backend is built using **Java 17** and **Spring Boot**, following a **layered architecture**.

a) Controller Layer

- Implemented using **Spring MVC**.
- Handles incoming HTTP requests from the React frontend.
- Maps API endpoints to appropriate service methods.
- Performs request validation and sends responses back to the client.

b) Service Layer

- Contains the **business logic** of the application.
- Acts as a bridge between controllers and repositories.
- Ensures separation of concerns and better maintainability.
- Handles operations related to:
    - Owner management
    - Pet registration
    - Visit tracking
    - Veterinarian details

c) Repository Layer

- Implemented using **Spring Data JPA**.
- Directly interacts with the **MySQL database**.
- Uses JPA repositories to perform CRUD operations.
- Eliminates boilerplate SQL code through ORM support.

4. Database Layer (MySQL)

- The application uses **MySQL 8.0.x** as the relational database.
- Stores all application data such as:
    - ❖ Owners
    - ❖ Pets
    - ❖ Visits
    - ❖ Veterinarians
    - ❖ Specialties

- Database tables are **normalized** to reduce redundancy.
- Relationships are maintained using **primary keys and foreign keys**.
- Spring Data JPA manages object–relational mapping between Java entities and database tables.

5. Security Layer

- Basic authentication and authorization are implemented using **Spring Security**.
- Supports **role-based access control** to restrict system functionalities.
- Ensures only authorized users can access protected resources.

UML Diagrams :

Advantages

- Implements a **real-world full stack application** using React and Spring Boot
- Follows a **clean, layered, and modular architecture**
- Easy to **understand, maintain, and extend**
- Serves as a **strong learning reference** for modern full stack development

Limitations

- Designed for **single-clinic usage only**
- Does not support **real-time notifications**
- Uses **basic security**, suitable mainly for academic purposes
- Not optimized for **large-scale data or high traffic**

Future Enhancements

- Implement **JWT-based authentication and authorization**
- Deploy the application on **cloud platforms such as AWS or GCP**
- Develop a **mobile application** for better accessibility
- Migrate to a **microservices-based architecture**
- Integrate **AI-based pet health analysis and recommendations**

Conclusion

The Spring PetClinic Application successfully demonstrates the development of a real-world full stack web application using React and Spring Boot. It helps bridge the gap between academic learning and industry-level application development by showcasing best practices in architecture, database design, and frontend-backend integration.