SOFTWARE REQUIREMENTS SPECIFICATION (SRS)

Project Name: PetClinic Management System
Developed By: Praveen N
Tech Stack: Spring Boot (Java) + MySQL + React.js

1. INTRODUCTION

1.1 Purpose

The purpose of this project is to develop a full-stack **PetClinic Management System** to manage pets, their owners, veterinarians, and appointments efficiently. This document defines all functional and non-functional requirements to be fulfilled by the system.

1.2 Scope

This system allows:

- Owners to manage their pets and book appointments.
- Vets to manage appointments and record visit reports.
- Admins to manage users, pets, appointments, and vet details.

The system includes a RESTful backend API built with **Spring Boot**, **MySQL** as the relational database, and a modern **React.js frontend**. It will implement **JWT-based authentication and role-based access control**.

1.3 Definitions, Acronyms, and Abbreviations

- **SRS**: Software Requirements Specification
- **JWT**: JSON Web Token
- **API**: Application Programming Interface
- **CRUD**: Create, Read, Update, Delete

- **UI**: User Interface
- **Vet**: Veterinarian

1.4 References

- Spring Boot Documentation
- React.js Documentation
- IEEE SRS Standard

2. OVERALL DESCRIPTION

2.1 Product Perspective

The system will be a self-contained web-based application, composed of:

- **Frontend**: React.js with TailwindCSS for UI.
- **Backend**: Spring Boot with REST APIs.
- **Database**: MySQL to persist application data.

2.2 Product Functions

- User registration and login with JWT token issuance
- Role-based dashboards (Admin, Vet, Owner)
- Pet and owner management
- Appointment scheduling and management
- Visit report submission by vets
- Admin control over users, vets, pets, and appointments

2.3 User Characteristics

- **Admin**: Tech-savvy, full access to all modules
- **Vet**: Medical staff, limited access to appointments and visit logs
- **Owner**: General user, non-technical

2.4 Constraints

- Must run on modern web browsers
- Backend must support REST over HTTP
- Database: MySQL (v8 or higher)
- Use of secure password storage (BCrypt)
- JWT expiry and refresh tokens must be implemented

3. SYSTEM FEATURES

3.1 User Authentication

- Login/Register functionality
- Password encryption using BCrypt
- JWT token generation and validation

3.2 Role-Based Access Control

- Admin: Access to all routes and CRUD operations
- Vet: Access to appointments and visit logs only
- Owner: Access to their own pets and appointments

3.3 Pet Management

- Add, view, update, delete pets
- Link pets to specific owners
- View pet medical history

3.4 Appointment Scheduling

- Book appointment with vet
- View upcoming and past appointments
- Appointment status updates

3.5 Visit Reports

- Vets record details of pet visits
- Owners can view visit summaries

3.6 Admin Panel

- View all users and roles
- Promote users to VET/ADMIN
- Add/remove Vets
- View system-wide analytics (optional)

4. EXTERNAL INTERFACES

4.1 User Interface (Frontend)

- Built using React.js
- TailwindCSS and ShadCN for styling
- React Router DOM for navigation
- React Hook Form for form handling

4.2 Backend REST APIs

- Built using Spring Boot
- Exposed via `/api/...` endpoints
- JSON request/response format

4.3 Database (MySQL)

- Tables: Users, Roles, Pets, Owners, Vets, Appointments, Visits
- Managed via Spring Data JPA and Hibernate

5. NON-FUNCTIONAL REQUIREMENTS

5.1 Performance Requirements

- API should respond within 500ms on average
- System should support 100+ concurrent users

5.2 Security Requirements

- JWT authentication with role claims
- BCrypt password hashing
- CORS enabled only for frontend origin
- Prevent unauthorized access to APIs

5.3 Reliability and Availability

- System should be available 99.5% of the time
- Database connection should be persistent and fault-tolerant

5.4 Maintainability

- Code will follow MVC and modular structure
- Commented code and clear naming conventions
- Separation of concerns between layers

5.5 Scalability

- App designed for easy deployment on cloud platforms like AWS, Railway, or Render
- Stateless APIs to support horizontal scaling

6. DATABASE OVERVIEW

Entities & Relationships:

- **User** (id, name, email, password, role_id)

- **Role** (id, name)
- **Owner** (id, user_id, contact info)
- **Vet** (id, user_id, specialization)
- **Pet** (id, name, birth_date, type, owner_id)
- **Appointment** (id, date, pet_id, vet_id, status)
- **Visit** (id, appointment_id, summary, prescription)

7. USE CASES

1. **Register a new user**
   Actor: Public
   Flow: User submits registration form → account created with 'OWNER' role
2. **Login with credentials**
   Actor: All
   Flow: Submits email/password → receives JWT token
3. **Book a new appointment**
   Actor: Owner
   Flow: Selects pet and vet → appointment created
4. **Vet updates visit report**
   Actor: Vet
   Flow: Opens appointment → fills report → submits
5. **Admin adds a new vet**
   Actor: Admin
   Flow: Fills vet registration form → user created with VET role

8. FUTURE ENHANCEMENTS

- Email/SMS reminders for appointments
- Prescription PDF download
- Payment gateway integration
- Admin dashboard charts and analytics
- Export visit reports to CSV/PDF

9. APPENDIX

- Tools: IntelliJ, MySQL Workbench, Postman, VS Code
- Libraries: Spring Boot, React, Axios, TailwindCSS
- Dependencies: spring-boot-starter-security, spring-boot-starter-data-jpa, jwt, etc.