PETCLINIC MANAGEMENT SYSTEM – PROJECT OVERVIEW

PROJECT GOAL:

To build a role-based full-stack application for managing a veterinary clinic using:

- Backend: Spring Boot + MySQL
- Frontend: React.js + TailwindCSS + Axios
- Security: JWT-based Authentication & Role-based Authorization

USER ROLES:

1. Admin – Full access: manage pets, owners, vets, appointments
2. Vet – View assigned appointments, update visit reports
3. Owner – Book appointments, manage own pets, view visit history

SYSTEM ARCHITECTURE:

React.js (Frontend)
↓ (HTTP via Axios)
Spring Boot REST API
↓
Service Layer (Business Logic)
↓
Repository Layer (Spring Data JPA)
↓
MySQL Database

BACKEND MODULES (SPRING BOOT):

- model: JPA Entity classes (Pet, Owner, Vet, Appointment, Visit)
- repository: Interfaces extending JpaRepository for DB operations
- service: Business logic and transaction handling
- controller: REST endpoints for frontend integration
- security: JWT auth, filters, password encoding
- dto: Request and response data transfer
- exception: Global error handling using @ControllerAdvice

FRONTEND MODULES (REACT.JS):

- Login/Register: Auth forms with JWT login

- Dashboard: Role-based routing (Admin/Vet/Owner views)
- Pet Management: Add/edit/view pets
- Appointment Booking: Schedule/manage appointments
- Visit Report: Vets log visit outcomes
- User Profile: Owners update their profile

AUTHENTICATION & AUTHORIZATION:

- Authentication: Users login via /api/auth/login and receive JWT token
- Authorization: JWT sent in Authorization header for protected routes
- Role-based Routing:
    - /admin/dashboard → Admin
    - /vet/dashboard → Vet
    - /owner/dashboard → Owner

DATABASE TABLES (MYSQL):

- users: Stores login credentials + role info
- roles: Role names (ADMIN, VET, OWNER)
- owners: Owner details
- vets: Vet details
- pets: Pet info with owner relationship
- appointments: Booking details with pet & vet link
- visits: Visit reports with appointment link

APP FLOW:

1. User registers and logs in → receives JWT
2. Redirected to dashboard based on role
3. Owners manage pets and book appointments
4. Vets view appointments and submit visit reports
5. Admin manages all data: pets, users, vets, appointments

EXAMPLE API ENDPOINTS:

- POST /api/auth/login → Login (All roles)
- GET /api/pets → List pets (Admin/Owner)
- POST /api/appointments → Book appointment (Owner)
- PUT /api/appointments/{id} → Update visit (Vet)

- GET /api/users/role/{id} → Get role (Admin)

TECHNOLOGY STACK:

BACKEND:

- Java
- Spring Boot
- Spring Data JPA + Hibernate
- MySQL
- Spring Security + JWT
- Hibernate Validator (JSR 380)
- Maven
- JUnit + Mockito

FRONTEND:

- React.js
- React Router DOM
- TailwindCSS + ShadCN
- Axios / Fetch API
- React Hook Form
- React Context / Redux (optional)

DEPLOYMENT OPTIONS:

- Backend: Railway, Render, AWS EC2
- Frontend: Vercel, Netlify
- Database: PlanetScale, Railway, AWS RDS

PROJECT MILESTONES:

1. Setup Spring Boot project with MySQL and entity relationships
2. Build Authentication System (JWT)
3. Implement REST APIs (CRUD)
4. Create React.js UI with Axios API calls
5. Add Role-based routing and protected pages
6. Write unit and integration tests
7. Deploy backend and frontend online