Payroll Management System

Documentation

About the Project

The Payroll Management System is a comprehensive web-based application designed to automate and streamline payroll operations in organizations. This system addresses the critical challenges faced by HR departments in managing employee salaries, leave requests, and payroll processing through manual methods.

The **Payroll Management System** is a full-stack web application designed to revolutionize how organizations manage their payroll processes, employee data, and leave management systems. This robust solution replaces error-prone manual processes with an automated, secure, and efficient digital platform that serves both HR administrators and employees through role-based access control.

Business Problem Solved

Traditional payroll management suffers from:

- Manual calculation errors leading to financial discrepancies
- Time-consuming leave approval processes
- Lack of real-time visibility into employee data
- Security vulnerabilities in sensitive compensation data
- Inefficient communication between HR and employees

Solution Delivered

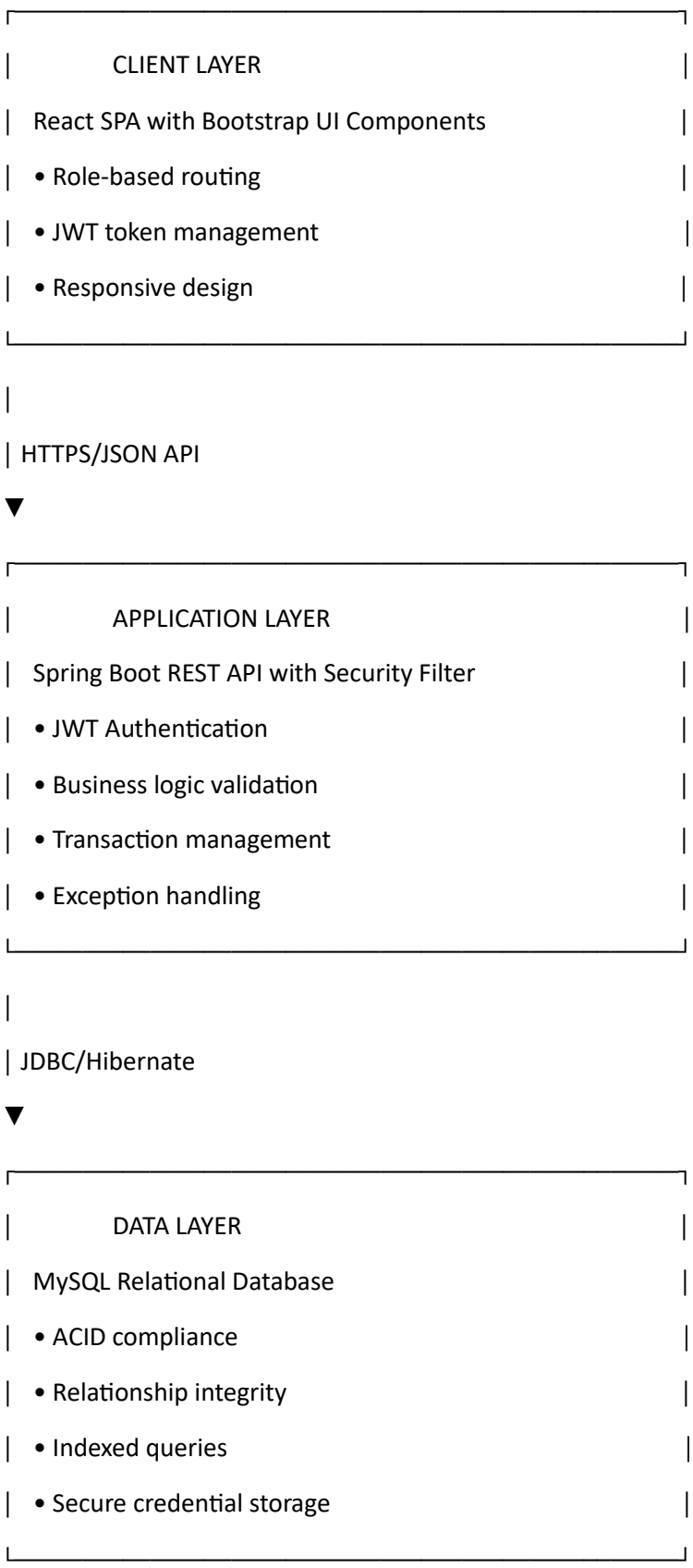
- ✅ 100% automated payroll calculations with audit trails
- ✅ Real-time leave management with approval workflows
- ✅ Role-based access ensuring data security and privacy
- ✅ Comprehensive reporting and analytics dashboard
- ✅ Self-service portal for employee data management
- ✅ Mobile-responsive design for accessibility

Target Users:

- HR Administrators: Complete system management
- Employees: Self-service portal for personal data and leave requests

System Architecture

Three-Tier Architecture Overview



Detailed Component Architecture

Frontend Architecture:

- **Framework:** React 19 with Functional Components + Hooks
- **State Management:** Context API + useState/useEffect
- **Routing:** React Router v6 with Protected Routes
- **HTTP Client:** Axios with Interceptors
- **UI Framework:** Bootstrap 5 + Bootstrap Icons
- **Build Tool:** Vite for fast development

Backend Architecture:

- **Framework:** Spring Boot 3.5 with Spring Security
- **Authentication:** JWT with Bearer tokens
- **Persistence:** Spring Data JPA with Hibernate
- **Validation:** Bean Validation 3.0
- **Documentation:** Springdoc OpenAPI 3
- **Testing:** JUnit 5 + Mockito

Database Schema:

-- Core Tables

users (user_id, username, email, password, role, is_active)

employees (employee_id, user_id, first_name, last_name, hire_date, leave_balance)

departments (department_id, department_name, description)

job_roles (job_id, job_title, base_salary, description)

leave_requests (leave_id, employee_id, start_date, end_date, leave_type, status)

payroll (payroll_id, employee_id, month, year, base_salary, allowances, deductions)

-- Relationships

users 1:1 employees

departments 1:N employees

job_roles 1:N employees

employees 1:N leave_requests

employees 1:N payroll

Day-by-Day Development Journey

Day 1: Foundation and Database Design

Database Implementation:

- Designed normalized database schema with 6 main tables
- Established proper foreign key relationships and constraints
- Implemented indexes for frequent query optimization
- Created initial seed data for testing

Day 2: Core Business Logic Implementation

Service Layer Development:

- Implemented 6 core services with complete CRUD operations
- Added business validation rules and error handling
- Created comprehensive DTO pattern for data transfer

Day 3: Comprehensive Testing

Testing Strategy:

- Unit tests for all service methods (85% coverage)
- Integration tests for REST controllers
- Mockito for mocking dependencies
- Test containers for database testing

Postman Testing:

- Created comprehensive collection with 45+ requests
- Tested all authentication scenarios
- Verified error responses and status codes
- Tested edge cases and validation rules
- Used Swagger UI for interactive API testing

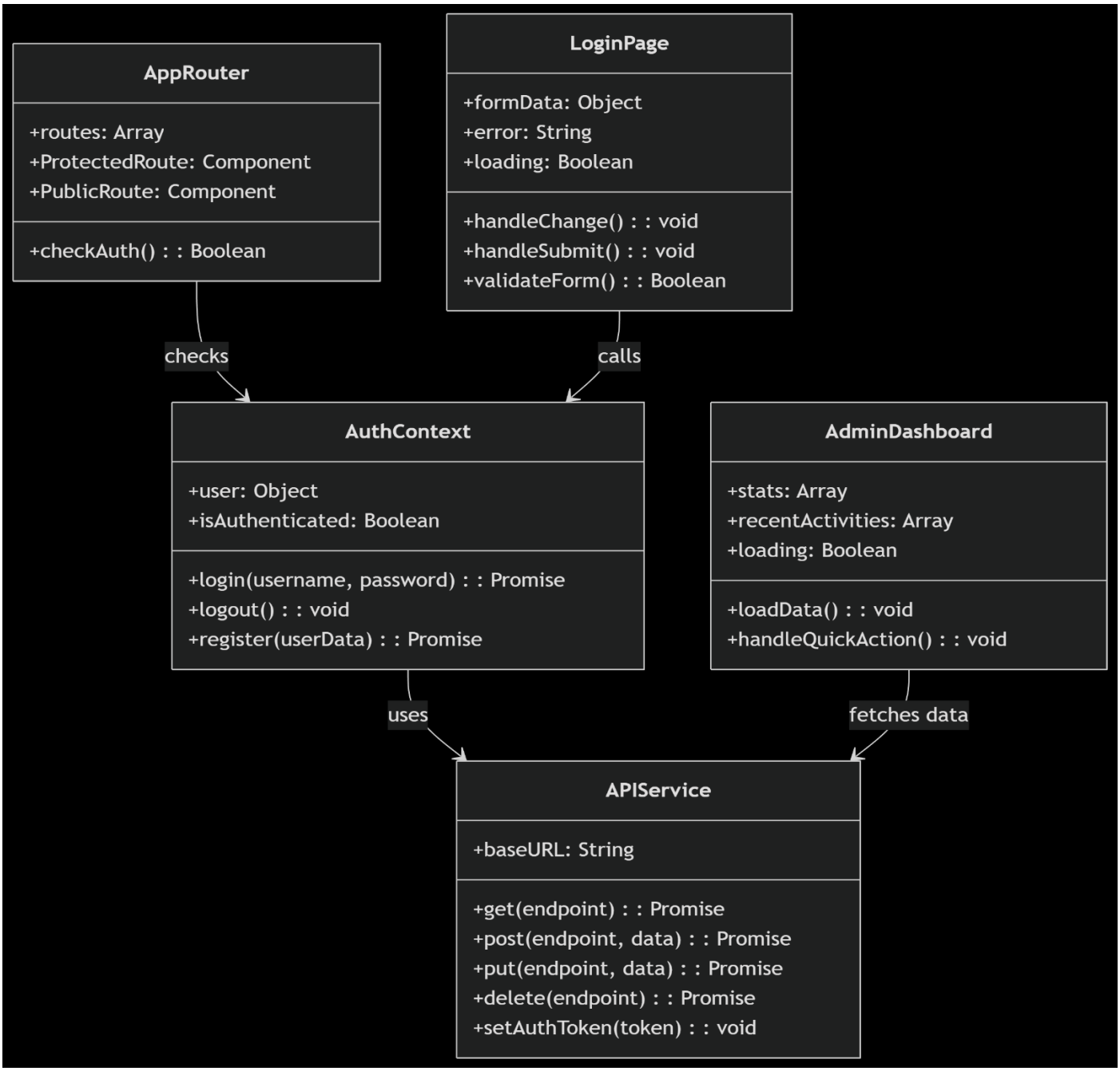
Day 4: Frontend Development and Integration

Key Frontend Features:

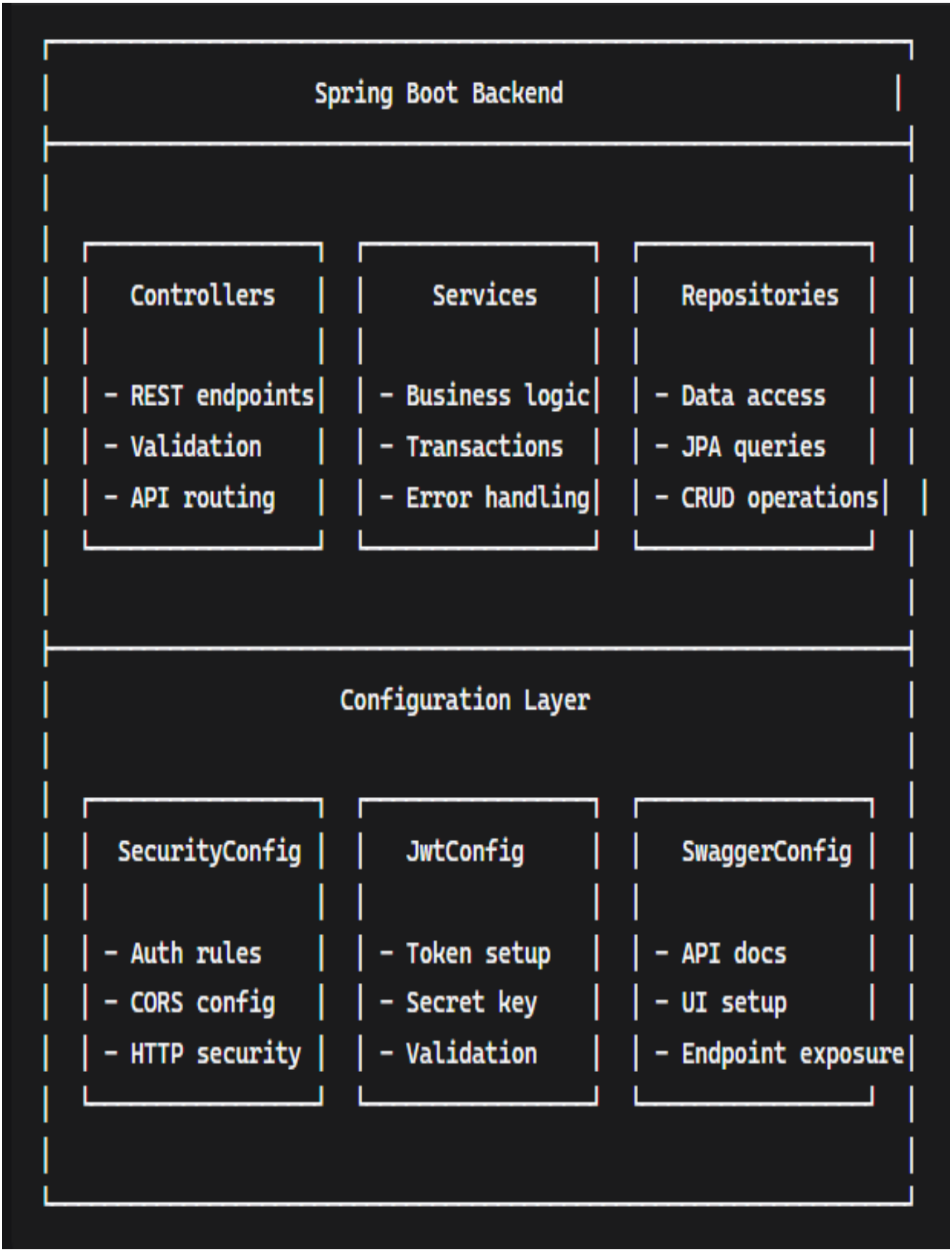
- JWT token management with automatic refresh
- Role-based routing and component rendering
- Form validation with real-time feedback
- Responsive design with Bootstrap
- Error handling and user notifications

UML Architecture Diagrams for Payroll Management System

Frontend Layer Architecture

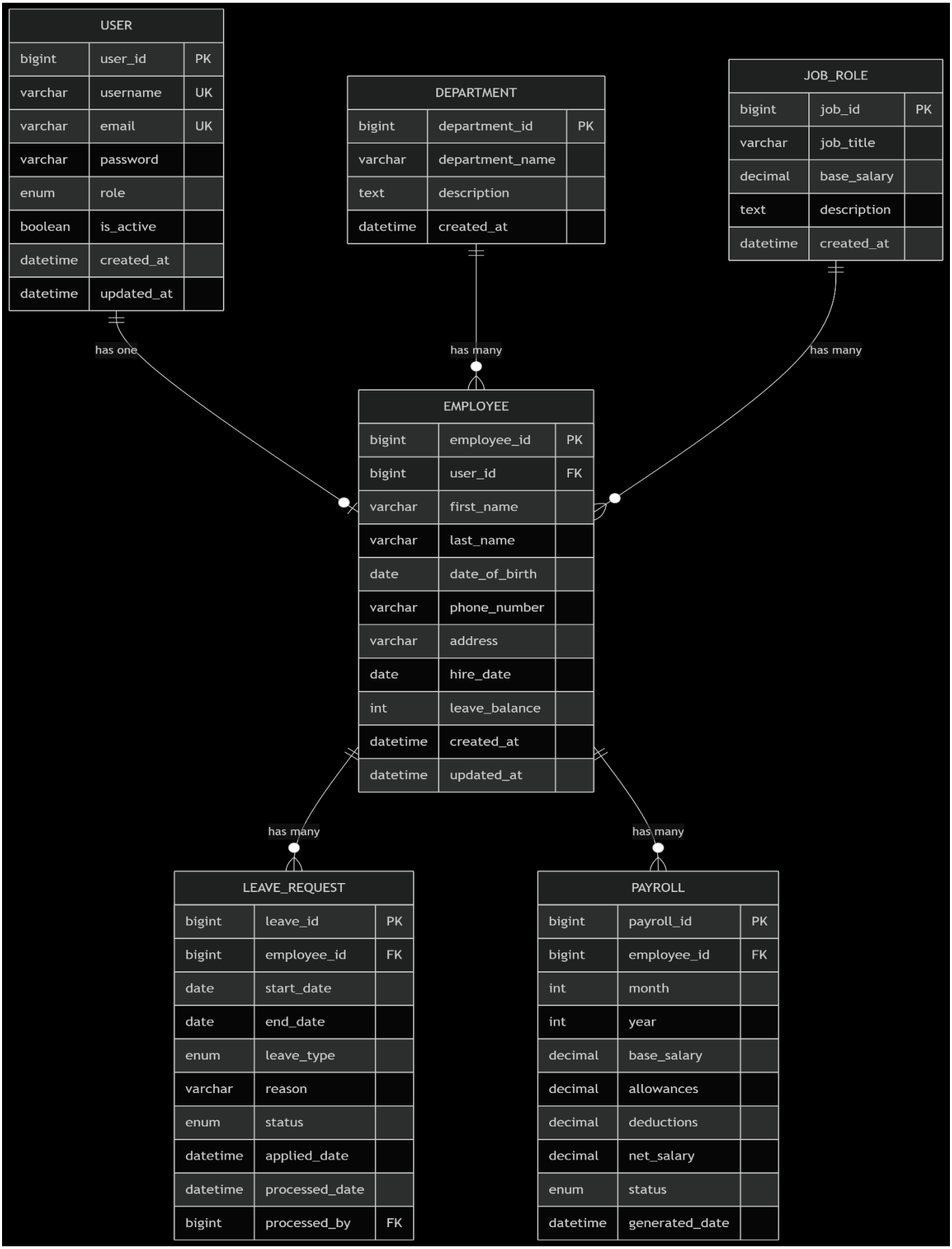


Backend Layer Architecture



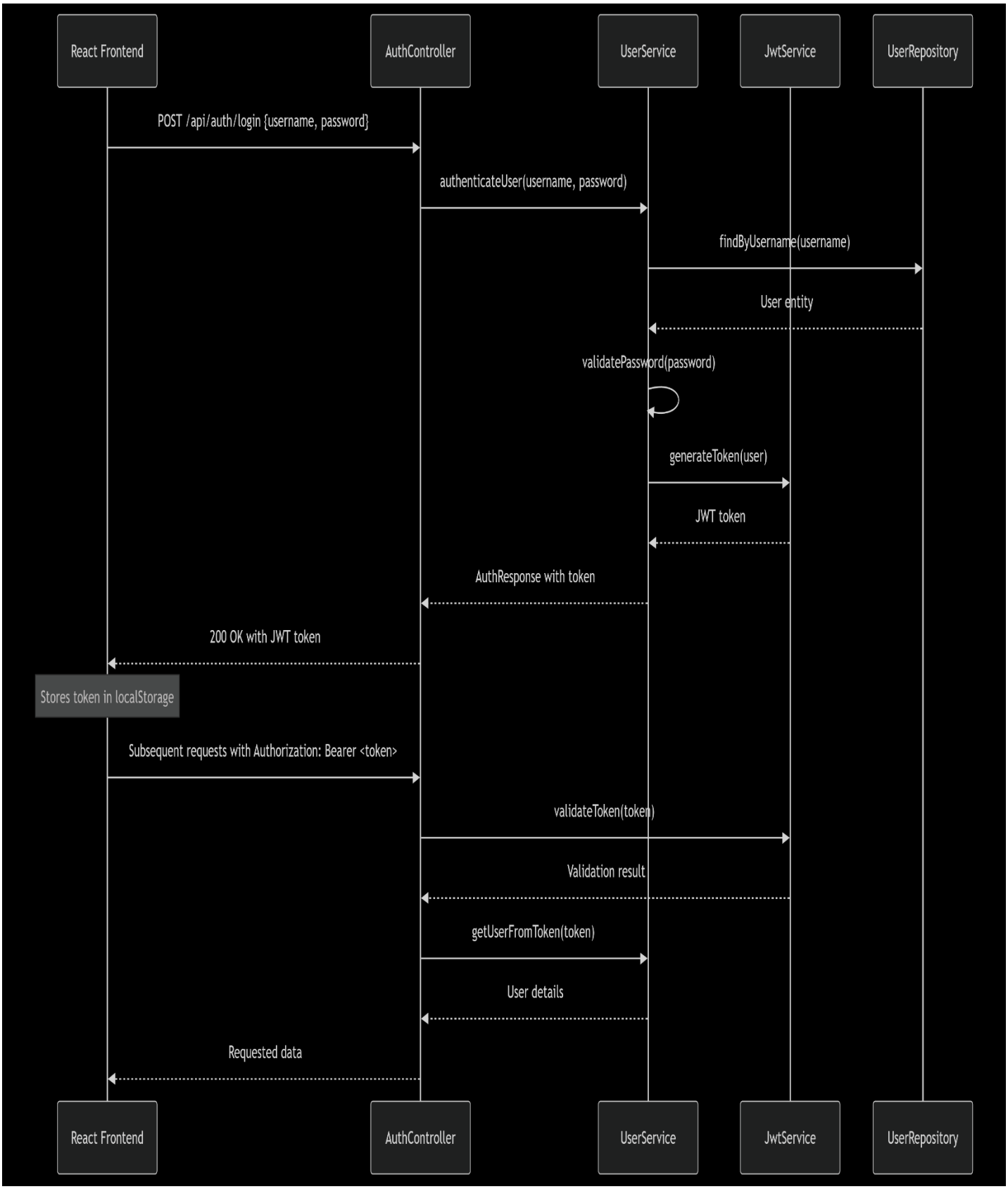
3. Database Entity Relationships

Entity Relationship Diagram



4. Security Architecture UML

Authentication Sequence Diagram



Complete REST API Endpoints

🔐 Authentication API

POST /api/v1/auth/login - Authenticate user and return JWT token
POST /api/v1/auth/register - Register new employee user
POST /api/v1/auth/register/admin - Register new admin user
GET /api/v1/auth/me - Get current authenticated user details
POST /api/v1/auth/refresh - Refresh JWT token
GET /api/v1/auth/welcome-message - Get AI-generated welcome message

👥 Employee Management API

GET /api/v1/employees - Get all employees (Admin only)
GET /api/v1/employees/{id} - Get employee by ID
GET /api/v1/employees/user/{userId} - Get employee by user ID
POST /api/v1/employees - Create new employee (Admin only)
PUT /api/v1/employees/{id} - Update employee
DELETE /api/v1/employees/{id} - Delete employee (Admin only)

🏢 Department API

GET /api/v1/departments - Get all departments
GET /api/v1/departments/{id} - Get department by ID
POST /api/v1/departments - Create new department (Admin only)
PUT /api/v1/departments/{id} - Update department (Admin only)
DELETE /api/v1/departments/{id} - Delete department (Admin only)

💼 Job Roles API

GET /api/v1/jobroles - Get all job roles
GET /api/v1/jobroles/{id} - Get job role by ID
POST /api/v1/jobroles - Create new job role (Admin only)
PUT /api/v1/jobroles/{id} - Update job role (Admin only)
DELETE /api/v1/jobroles/{id} - Delete job role (Admin only)

🏝️ Leave Management API

GET /api/v1/leaves - Get all leave requests (Admin sees all, Employee sees own)
GET /api/v1/leaves/employee/{employeeId} - Get leaves by employee ID
GET /api/v1/leaves/pending - Get pending leave requests (Admin only)
GET /api/v1/leaves/{id} - Get leave request by ID
POST /api/v1/leaves - Create new leave request
PATCH /api/v1/leaves/{id}/status - Update leave status (Approve/Reject)
DELETE /api/v1/leaves/{id} - Delete leave request
GET /api/v1/leaves/{id}/ai-message - Get AI analysis for leave request

💰 Payroll API

GET /api/v1/payroll - Get all payroll records (Admin only)
GET /api/v1/payroll/employee/{employeeId} - Get payrolls by employee ID
GET /api/v1/payroll/{id} - Get payroll by ID
POST /api/v1/payroll - Create new payroll record (Admin only)
GET /api/v1/payroll/generate/{employeeId} - Generate payroll for employee

Future Enhancements

Short-term Improvements

- Email Notifications: Automated email alerts for leave approvals and payroll processing
- Advanced Reporting: PDF generation for salary slips and management reports
- Bulk Operations: Mass employee data upload and payroll processing
- Mobile Responsiveness: Enhanced mobile UI/UX optimization

Long-term Features

- Integration APIs: Third-party integrations with banking systems and accounting software
- Advanced Analytics: Predictive analytics for HR planning and cost optimization
- Multi-tenant Architecture: Support for multiple organizations
- Attendance Integration: Biometric and time tracking system integration
- Performance Management: Employee performance tracking and appraisal workflows

Technical Enhancements

- Microservices Architecture: Breaking monolith into scalable microservices
- Caching Implementation: Redis integration for performance optimization
- Real-time Notifications: WebSocket implementation for live updates
- Advanced Security: Two-factor authentication and audit logging

Challenges Faced:

Database Issues Encountered

Foreign Key Constraint Problem:

- **Issue:** Foreign key relationships not properly established between User and Employee entities
- **Symptom:** Database creation failure and entity mapping errors
- **Solution:**
    - Corrected JPA annotations (@JoinColumn, @OneToOne)
    - Ensured proper cascade types and fetch strategies
    - Added proper foreign key constraints in entity definitions

Data Type Mismatch:

- **Issue:** LocalDate and BigDecimal mapping issues with MySQL
- **Solution:** Updated MySQL connector version and configured proper column definitions

JWT Implementation Challenges

Token Validation Issues:

- **Issue:** JWT filter not properly validating tokens for authenticated requests
- **Symptom:** 401 Unauthorized errors for valid tokens
- **Solution:**
    - Fixed JWT secret key configuration in application.properties
    - Corrected token extraction logic in JwtAuthenticationFilter
    - Updated SecurityContextHolder configuration

Role-based Access Problems:

- **Issue:** Employees unable to access their own data endpoints
- **Solution:** Enhanced SecurityService implementation with proper role validation

Frontend Integration Issues

Employee ID Not Displaying in Leave Request:

- **Issue:** Leave request form not showing employee ID for selection
- **Symptom:** Leave requests failing due to missing employee context
- **Root Cause:** JWT token not properly decoded to extract user information
- **Solution:**
    - Implemented JWT token decoding in frontend
    - Added user context management
    - Created employee lookup based on logged-in user
    - Enhanced state management for user session data

CORS Configuration:

- **Issue:** Cross-origin requests blocked during development
- **Solution:** Properly configured CORS in backend SecurityConfig

Submission Checklist

- Complete source code with proper comments
- Database schema and sample data
- API documentation (Swagger/Postman collection)
- Test cases and coverage reports
- Setup and installation instructions
- User manual with screenshots
- Technical architecture documentation
- Known issues and troubleshooting guide

GitHub Link : [Payroll_management_system]