HR Management Portal (hrSphere) - Complete System Documentation

■ Table of Contents

■ Project Overview

The HR Management Portal (hrSphere) is a comprehensive full-stack web application designed to streamline human resource management processes. Built with modern technologies, it provides both administrative and employee interfaces for managing HR operations efficiently.

Key Features

• Role-Based Authentication : Admin and User roles with different access levels

• Employee Management : Complete CRUD operations for employee records

• Leave Management : Request, approve, and track leave applications

• Dashboard Analytics : Real-time statistics and department-wise summaries

• Payroll Management : Generate and manage employee payroll

• Post/Announcements : Company-wide communication system

• Responsive Design : Mobile-friendly interface with Bootstrap

■■ Architecture Overview

System Architecture

```
[illegible]
```

Project Structure

```
[illegible]
```

■ Technology Stack

Backend Technologies

- Framework : Spring Boot 3.x
- Language : Java 17
- Database : MySQL 8.0
- ORM : Spring Data JPA / Hibernate

• Build Tool : Maven

• Email : Spring Mail

• PDF Generation : iText PDF

• Security : Spring Security (JWT ready)

Frontend Technologies

• Framework : React 18

• Build Tool : Vite

• Routing : React Router DOM

• HTTP Client : Axios

• UI Framework : Bootstrap 5

• Icons : Font Awesome

• Styling : CSS3 + Custom Styles

Development Tools

• IDE : Eclipse/IntelliJ IDEA, VS Code

• Database Tools : MySQL Workbench

• API Testing : Postman

• Version Control : Git

■■ Database Schema

Core Tables

1. EMPLOYEE Table

```
CREATE TABLE EMPLOYEE (
id INT PRIMARY KEY AUTO_INCREMENT,
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
password VARCHAR(255),
[illegible]
```

2. COMPOSE Table (Leave Requests)

```
[illegible]
```

3. CREATE_POST Table (Announcements)

```
[illegible]
```

4. PAYROLL Table

```
[illegible]
```

■ Backend Code Flow Analysis

1. Application Startup Flow

Main Application Class

```
[illegible]
```

Configuration Properties

```
spring.application.name=HR-Management-Portal
```

```
[illegible]
```

2. Controller Architecture

AuthController - Authentication Management

```
[illegible]
}
```

```
[illegible]
```

EmployeeController - Employee Management

```
[illegible]
System.err.println("Failed to send welcome email: " +
[illegible]
}
```

```
}
```

LeaveController - Leave Management

```
[illegible]
textBuilder.append("From: ").append(fromDate != null ? fromDate :
[illegible]
textBuilder.append("Days: ").append(leaveDaysObj != null ?
[illegible]
}
}
}
```

■■ Frontend Code Flow Analysis

1. Application Structure

Main App Component

```
[illegible]
import { BrowserRouter as Router, Routes, Route, Navigate } from
[illegible]
useEffect(() => {
```

```
[illegible]
```

API Service Layer

```
[illegible]
throw new Error(error.response.data?.message || error.response.statusText ||
'API request
```

```
[illegible]
```

Axios Configuration

```
[illegible]
},
[illegible]
error.message = 'Cannot connect to backend server. Please ensure the Spring Boot
[illegible]
```

2. Component Examples

Login Component

```
[illegible]
localStorage.setItem('user', JSON.stringify(demoUser));
```

```
[illegible]
```

Dashboard Component

```
[illegible]
```

```
[illegible]
{ id: 1, title: 'Welcome to HR Portal', content: 'Welcome message', author:
[illegible]
```

■ Complete Feature Flows

1. Login Flow Sequence

```
User Input → Frontend Validation → API Call → Backend Authentication →
[illegible]
```

Detailed Steps: 1. User enters credentials in Login.jsx 2. Form validation occurs on frontend 3. API call sent via axios to /api/auth/login 4. Backend controller processes request in AuthController 5. Database query executed via EmployeeRepo 6. Authentication logic processes result 7. Response sent back to frontend with user data and token 8. LocalStorage updated with user info and auth token 9. App state updated with authentication status 10. Navigation triggered to appropriate dashboard

2. Employee Management Flow

```
Component Mount → API Request → Controller → Service → Repository → Database
[illegible]
```

Detailed Steps: 1. AllEmployee component mounts and triggers useEffect 2. fetchEmployees function calls employeeAPI.getAll() 3. API request sent to /api/employees endpoint 4. EmployeeController.getAllEmployees() method executed 5. HrService.getAllEmployee() called for business logic 6. EmployeeRepo.findAll() queries database 7. Employee entities retrieved from EMPLOYEE table 8. Entity-to-DTO mapping via convertEmployeeToMap() 9. JSON response sent back to frontend 10. Component state updated with employee data 11. UI re-renders with employee table

3. Leave Request Flow

```
Form Submission → Data Processing → API Call → Controller → Entity Creation →
[illegible]
[illegible]
```

Detailed Steps: 1. User fills leave request form in UserCompose.jsx 2. Date calculation and validation performed 3. Leave request object constructed with user data 4. API call made to /api/leave/requests 5. LeaveController.createLeaveRequest() processes request 6. Compose entity created and populated 7. Structured text built with leave details 8. Database save via ComposeRepo.save() 9. Success response returned with saved entity 10. UI feedback shown to user 11. Form reset for next request

4. Dashboard Analytics Flow

```
Component Load → Multiple API Calls → Parallel Processing → Data Aggregation →
[illegible]
[illegible]
```

Detailed Steps: 1. Dashboard component mounts triggering multiple useEffect calls 2. Parallel API requests for: - Dashboard statistics (/api/dashboard/stats) - Department summary (/api/dashboard/department-summary) - Leave summary (/api/leave/summary/{userId}) - Posts data (/api/posts) 3. Backend controllers process each request independently 4. Database queries executed for different data sources 5. Data aggregation performed in respective controllers 6. Multiple responses returned to frontend 7. State updates trigger component re-renders 8. Real-time dashboard displays current statistics

■ API Documentation

Authentication Endpoints

POST /api/auth/login

Description : Authenticate user and return JWT token Request Body :

```
[illegible]
[illegible]
[illegible]
[illegible]
```

Response :

```
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
```

POST /api/auth/change-password

Description : Change user password Request Body :

```
[illegible]
"userId": "1",
[illegible]
```

Employee Management Endpoints

GET /api/employees

Description : Get all employees Response :

```
[illegible]
```

POST /api/employees

Description : Create new employee Request Body :

```
[illegible]
```

PUT /api/employees/{id}

Description : Update employee by ID

DELETE /api/employees/{id}

Description : Delete employee by ID

Leave Management Endpoints

GET /api/leave/requests

Description : Get all leave requests (Admin only)

GET /api/leave/requests/user/{userId}

Description : Get leave requests for specific user

POST /api/leave/requests

Description : Create new leave request Request Body :

```
[illegible]
```

PUT /api/leave/requests/{id}/status

Description : Update leave request status (Admin only) Request Body :

```
[illegible]
```

Dashboard Endpoints

GET /api/dashboard/stats

Description : Get dashboard statistics Response :

```
[illegible]
```

GET /api/dashboard/department-summary

Description : Get department-wise employee count Response :

```
[illegible]
```

■ Security Implementation

Authentication & Authorization

JWT Token Implementation (Ready)

```
[illegible]
```

Role-Based Access Control

```
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
```

CORS Configuration

```
[illegible]
```

Input Validation

```
[illegible]
[illegible]
@Size(min = 3, max = 100, message = "Employee Name must be between 3 to 100
[illegible]
[illegible]
[illegible]
[illegible]
@Pattern(regexp = "^[6-9]\\d{9}$", message = "Mobile number must start with 6 to
[illegible]
[illegible]
```

Data Protection

Password Security

```
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
```

Error Handling

```
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
```

■ Deployment Guide

Development Environment Setup

Prerequisites

2. Java 17 or higher

4. Node.js 16 or higher

6. MySQL 8.0 or higher

8. Maven 3.6 or higher

Backend Setup

2. Clone the repository : bash git clone cd HR-Management-Portal-hrSphere/HR-Management-Portal

4. Configure database in application.properties : properties spring.datasource.url=jdbc:mysql://localhost:3306/hr spring.datasource.username=root spring.datasource.password=your_password

6. Create database : sql CREATE DATABASE hr;

8. Run the application : bash mvn spring-boot:run

10. Verify backend at http://localhost:8080

Clone the repository : bash git clone cd HR-Management-Portal-hrSphere/HR-Management-Portal

Configure database in application.properties : properties spring.datasource.url=jdbc:mysql://localhost:3306/hr spring.datasource.username=root spring.datasource.password=your_password

Create database : sql CREATE DATABASE hr;

Run the application : bash mvn spring-boot:run

Verify backend at http://localhost:8080

Frontend Setup

2. Navigate to frontend directory : bash cd ../my-react-app

4. Install dependencies : bash npm install

6. Configure Vite proxy in vite.config.js : javascript export default defineConfig({ plugins: [react()], server: { proxy: { '/api': { target: 'http://localhost:8080', changeOrigin: true } } } })

8. Start development server : bash npm run dev

10. Access application at http://localhost:3000

Navigate to frontend directory : bash cd ../my-react-app

Install dependencies : bash npm install

Configure Vite proxy in vite.config.js : javascript export default defineConfig({ plugins: [react()], server: { proxy: { '/api': { target: 'http://localhost:8080', changeOrigin: true } } } })

Start development server : bash npm run dev

Access application at http://localhost:3000

Production Deployment

Backend (Spring Boot)

2. Build the application : bash mvn clean package -DskipTests

4. Run the JAR file : bash java -jar target/HR-Management-Portal-0.0.1-SNAPSHOT.jar

Build the application : bash mvn clean package -DskipTests

Run the JAR file : bash java -jar target/HR-Management-Portal-0.0.1-SNAPSHOT.jar

Frontend (React)

2. Build for production : bash npm run build

4. Serve static files using nginx or Apache

Build for production : bash npm run build

Serve static files using nginx or Apache

Docker Deployment (Optional)

```
[illegible]
```

Database Migration

```
[illegible]
```

■ Performance Considerations

Backend Optimizations

• Connection Pooling : HikariCP (default in Spring Boot)

• JPA Optimization : Lazy loading, query optimization

• Caching : Consider Redis for session management

• Database Indexing : Index on frequently queried columns

Frontend Optimizations

• Code Splitting : Lazy load components

• Bundle Optimization : Vite's built-in optimizations

• API Caching : Consider React Query for data fetching

• Image Optimization : Compress and optimize images

Monitoring & Logging

• Backend Logging : Logback configuration

• API Monitoring : Spring Boot Actuator

• Frontend Error Tracking : Error boundaries

• Performance Metrics : Consider APM tools

■ Testing Strategy

Backend Testing

```
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
ResponseEntity response = restTemplate.getForEntity("/api/employees",
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
ResponseEntity response = restTemplate.postForEntity("/api/employees", employee,
[illegible]
[illegible]
[illegible]
[illegible]
```

Frontend Testing

```
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
expect(screen.getByPlaceholderText('Enter your employee
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
```

■ Future Enhancements

Planned Features

2. Advanced Reporting : PDF/Excel exports for various reports

4. Notification System : Email/SMS notifications for important events

6. File Upload : Document management for employees

8. Advanced Analytics : Charts and graphs for HR metrics

10. Mobile App : React Native version for mobile access

12. Integration APIs : Third-party HR tool integrations

14. Audit Logging : Comprehensive activity tracking

16. Multi-tenancy : Support for multiple organizations

Technical Improvements

2. Microservices Architecture : Break into smaller services

4. Event-Driven Architecture : Using message queues

6. GraphQL API : More flexible data fetching

8. WebSocket Support : Real-time notifications

10. Advanced Security : OAuth2, SAML integration

12. Performance Optimization : Caching, CDN implementation

14. CI/CD Pipeline : Automated testing and deployment

16. Monitoring : Comprehensive application monitoring

■ Support & Contact

Development Team

• Project Lead : Shubham Singh

• Backend Developer : Spring Boot Expert

• Frontend Developer : React Specialist

• Database Administrator : MySQL Expert

Documentation

• Project Repository : [GitHub Link]

• API Documentation : Available at /swagger-ui.html (if Swagger is configured)

• User Manual : Separate document for end users

• Technical Specifications : This document

Issue Reporting

• Bug Reports : Create GitHub issues with detailed descriptions

• Feature Requests : Submit enhancement proposals

• Security Issues : Report privately to security team

■ Conclusion

The HR Management Portal (hrSphere) represents a comprehensive, enterprise-grade human resource management solution built with modern web technologies. The application demonstrates:

Technical Excellence

• Clean Architecture : Proper separation of concerns across all layers

• Modern Technologies : Latest versions of Spring Boot and React

• Professional Practices : Proper error handling, validation, and security measures

• Scalable Design : Modular structure ready for future enhancements

Business Value

• Complete HR Solution : Covers all major HR functions

• User Experience : Intuitive interfaces for both admins and employees

• Real-time Data : Live dashboards and analytics

• Efficiency : Streamlined processes for HR operations

Development Standards

• Code Quality : Well-structured, documented, and maintainable code

• Best Practices : Following industry standards and conventions

• Security : Proper authentication, authorization, and data protection

• Testing Ready : Structure supports comprehensive testing strategies

This documentation serves as a complete reference for understanding, maintaining, and extending the HR Management Portal system. The application is production-ready and can be deployed in various environments to serve real-world HR management needs.

Document Version: 1.0 Last Updated: January 2025 Author: System Documentation Team