Software Requirements Specification (SRS)

Enterprise Resource Planning (ERP) System

---

1. Introduction

1.1 Purpose

This document describes the Software Requirements Specification (SRS) for the Enterprise Resource Planning (ERP) System. The system is designed using the **Model–View–Controller (MVC) architectural pattern** to ensure scalability, maintainability, and separation of concerns. This SRS is intended for developers, instructors, and stakeholders involved in the design, implementation, and evaluation of the system.

1.2 Scope

The ERP system integrates multiple business modules including User & Role Management, Finance & Accounting, Inventory Management, Procurement, Human Resources, Sales & CRM, Project Management, Document Management, Workflow Engine, and Reporting Dashboards. The system supports multi-tenant organizations and integrates AI-based assistance for analytics and automation.

1.3 Definitions, Acronyms, and Abbreviations

- ERP: Enterprise Resource Planning
- MVC: Model–View–Controller
- RBAC: Role-Based Access Control
- JWT: JSON Web Token
- API: Application Programming Interface
- UI: User Interface

---

2. Overall Description

2.1 Product Perspective

The ERP system follows a **distributed MVC architecture**:

- **Model**: Represents business data and database schemas (MongoDB / PostgreSQL)
- **View**: User interface implemented using React + TypeScript
- **Controller**: Backend logic implemented using Go microservices and Java Spring Boot

Each ERP module follows MVC principles independently while communicating via REST or GraphQL APIs.

2.2 Product Functions

- User authentication and role management
- Financial transactions and reporting
- Inventory tracking and vendor management
- Employee and payroll management
- Sales and customer relationship management
- Project and task tracking
- Workflow automation and approvals
- AI-assisted reporting and insights

2.3 User Classes and Characteristics

- **Admin**: Full access to all modules and configurations
- **Manager**: Access to department-level data and reports
- **Employee**: Limited access based on assigned role

2.4 Operating Environment

- OS: Windows / Linux / macOS
- Backend: Go, Java (Spring Boot)
- Frontend: React + TypeScript
- Database: MongoDB, PostgreSQL
- Deployment: Docker / Docker Compose

---

3. System Architecture (MVC Implementation)

3.1 Model Layer

The Model layer manages application data, business rules, and database interaction.

Examples: - User, Role, Permission - Inventory Item, Stock, Warehouse - Financial entities (GLAccount, Transaction)

Responsibilities: - Data validation - Database CRUD operations - Multi-tenant data isolation using tenant_id

3.2 View Layer

The View layer is responsible for presenting data to the user and capturing user input.

Technologies: - React + TypeScript - shadcn UI components

Responsibilities: - Display dashboards, forms, and tables - Send user input to controllers via APIs - Display responses and errors

3.3 Controller Layer

The Controller layer acts as an intermediary between Model and View.

Technologies: - Go (REST / GraphQL microservices) - Java Spring Boot (Finance, HR modules)

Responsibilities: - Process client requests - Enforce authentication and authorization - Call Model layer for data operations - Return responses to View

---

4. Functional Requirements

4.1 User & Role Management

- The system shall allow admins to create, update, and delete users
- The system shall assign roles and permissions using RBAC
- The controller shall validate user actions before updating the model

4.2 Inventory Management

- The system shall allow users to add and update inventory items
- The system shall notify users of low stock
- The model shall store inventory data in MongoDB

4.3 Finance & Accounting

- The system shall record financial transactions
- The system shall generate financial reports
- The controller shall ensure data consistency

4.4 Human Resource Management

- The system shall manage employee records
- The system shall calculate payroll
- The view shall display leave calendars and directories

---

5. Non-Functional Requirements

5.1 Performance

- API response time shall be under 2 seconds

5.2 Security

- OAuth2 and JWT-based authentication
- Role-based access control

5.3 Scalability

- Microservice-based architecture
- Support for multiple tenants

5.4 Maintainability

- MVC architecture ensures separation of concerns
- Modular code structure

---

6. Database Requirements

- MongoDB for operational data
- PostgreSQL for reporting and analytics
- Each table/collection includes tenant_id

---

7. Future Enhancements

- Advanced AI-driven analytics
- Mobile application support
- Workflow auto-optimization

---

8. Conclusion

The ERP system is designed using the MVC architectural pattern to ensure modularity, scalability, and ease of maintenance. The separation of Model, View, and Controller enables efficient development and future expansion while supporting modern enterprise requirements.