# Personal Health Management System (PHMS) - Software Design Document (SDD)

**Version**: 1.2  
**Date**: January 2025  
**Project**: CS673 Team 2

## Table of Contents

1. [Introduction](#Xe3d0fc0bea9a42ce7605565d0964033d7f6ee47)
2. [System Overview](#X5508b397684bf4192a0d8d0a48692b0135f807f)
3. [System Architecture](#X0833a92bf396f899e20be29b4b6cbf30c8d8a95)
4. [Component Design](#Xfdf0980badc30f5185b30879f9d513043c0277b)
5. [Database Design](#X6b97c217ebc75afcba20d43deee6c8307895657)
6. [API Design](#X718b2fd4009c28967786be1bce372e50f16bda3)
7. [User Interface Design](#Xb42248d173b6c1d8cf7cfcb7596fe13df6de444)
8. [Security Design](#X66f08edac5b0aa578e1635ff0ec4335c9892d6d)
9. [Technology Stack](#X8fc486e7d4220b3124543e251260ac17df34077)
10. [Deployment Architecture](#X6332cc5cf84dfa4849fa5f85b02a678e218890a)

## 1. Introduction

### 1.1 Purpose

This SDD outlines the architectural design & implementation details for the Personal Health Management System (PHMS). The document serves as a technical blueprint for developers & stakeholders.

### 1.2 Scope

The PHMS is a web-based B/S architecture application built with Spring Boot & Vue.js, providing comprehensive health data management, user authentication, role-based access control (RBAC), & AI-powered health assistance.

### 1.3 Definitions & Abbreviations

* **PHMS**: Personal Health Management System
* **SDD**: Software Design Document
* **RBAC**: Role-Based Access Control
* **API**: Application Programming Interface
* **UI**: User Interface
* **DB**: Database
* **CRUD**: Create, Read, Update, Delete
* **JWT**: JSON Web Token
* **REST**: Representational State Transfer
* **MVC**: Model-View-Controller
* **ORM**: Object-Relational Mapping

## 2. System Overview

### 2.1 Architecture Pattern

The system follows a **3-tier B/S architecture**: - **Presentation Layer**: Vue.js frontend with Element UI - **Business Logic Layer**: Spring Boot REST API services  
- **Data Access Layer**: SQLite DB with MyBatis ORM

### 2.2 Key Features

* Multi-role user authentication & authorization
* Comprehensive health data tracking & analytics
* Exercise logging & management
* AI-powered health consultation via WebSocket
* Responsive web interface
* RESTful API design

## 3. System Architecture

### 3.1 Overall Architecture

The system follows a **three-tier architecture** pattern:

┌─────────────────────────────────────────────────────────────┐  
│ Presentation Layer │  
│ ┌─────────────────┐ ┌─────────────────┐ ┌──────────────┐ │  
│ │ Vue.js SPA │ │ Element UI │ │ WebSocket │ │  
│ │ Frontend │ │ Components │ │ Client │ │  
│ └─────────────────┘ └─────────────────┘ └──────────────┘ │  
└─────────────────────────────────────────────────────────────┘  
 │  
 ▼  
┌─────────────────────────────────────────────────────────────┐  
│ Business Logic Layer │  
│ ┌─────────────────┐ ┌─────────────────┐ ┌──────────────┐ │  
│ │ Spring Boot │ │ Controllers │ │ Services │ │  
│ │ Application │ │ (REST API) │ │ (Business │ │  
│ │ │ │ │ │ Logic) │ │  
│ └─────────────────┘ └─────────────────┘ └──────────────┘ │  
└─────────────────────────────────────────────────────────────┘  
 │  
 ▼  
┌─────────────────────────────────────────────────────────────┐  
│ Data Access Layer │  
│ ┌─────────────────┐ ┌─────────────────┐ ┌──────────────┐ │  
│ │ MyBatis │ │ SQLite │ │ Schema │ │  
│ │ Mappers │ │ Database │ │ Management │ │  
│ └─────────────────┘ └─────────────────┘ └──────────────┘ │  
└─────────────────────────────────────────────────────────────┘

### 3.2 Architectural Patterns

* **MVC (Model-View-Controller)**: Separates concerns between data, presentation, and business logic
* **Repository Pattern**: Abstracts data access through MyBatis mappers
* **Service Layer Pattern**: Encapsulates business logic in service classes
* **Component-Based Architecture**: Frontend built with reusable Vue.js components

## 4. Component Design

### 4.1 Backend Components

#### 4.1.1 Controller Layer

* **AuthController**: Handles user authentication, registration, and profile management
* **UserController**: Manages user operations and role assignments
* **RoleController**: Handles role and permission management

#### 4.1.2 Service Layer

* **AuthService**: Business logic for authentication, password encryption, and Google OAuth
* **UserService**: User management business logic
* **RoleService**: Role and permission management logic

#### 4.1.3 Data Access Layer

* **UserMapper**: Database operations for user entities
* **RoleMapper**: Database operations for role entities
* **PermissionMapper**: Database operations for permission entities

### 4.2 Frontend Components

#### 4.2.1 Core Components

* **Layout**: Main application layout with navigation
* **Router**: Vue Router configuration with route guards
* **Store**: Vuex state management for user authentication and application state

#### 4.2.2 Feature Components

* **Login/Register**: User authentication interfaces
* **Dashboard**: Main application dashboard
* **AI Assistant**: WebSocket-based AI consultation interface
* **User Management**: Admin interface for user operations

## 5. Database Design

### 5.1 Entity Relationship Diagram

┌─────────────────┐ ┌─────────────────┐ ┌─────────────────┐  
│ Users │ │ User\_Roles │ │ Roles │  
├─────────────────┤ ├─────────────────┤ ├─────────────────┤  
│ id (PK) │◄──►│ user\_id (FK) │◄──►│ id (PK) │  
│ username │ │ role\_id (FK) │ │ name │  
│ password\_hash │ │ created\_at │ │ description │  
│ email │ └─────────────────┘ │ created\_at │  
│ full\_name │ └─────────────────┘  
│ age │ │  
│ gender │ ▼  
│ phone │ ┌─────────────────┐ ┌─────────────────┐  
│ avatar │ │Role\_Permissions │ │ Permissions │  
│ google\_id │ ├─────────────────┤ ├─────────────────┤  
│ login\_type │ │ role\_id (FK) │◄──►│ id (PK) │  
│ is\_active │ │ permission\_id │ │ name │  
│ created\_at │ │ created\_at │ │ description │  
│ updated\_at │ └─────────────────┘ │ resource │  
└─────────────────┘ │ action │  
 │ │ created\_at │  
 │ └─────────────────┘  
 ▼  
┌─────────────────┐ ┌─────────────────┐  
│ j\_body │ │ detail │  
├─────────────────┤ ├─────────────────┤  
│ id (PK) │ │ id (PK) │  
│ user\_id (FK) │ │ sport\_type │  
│ bloodLipid │ │ disease │  
│ heart\_rate │ │ method │  
│ vision │ │ notes │  
│ sleep\_duration │ └─────────────────┘  
│ sleep\_quality │  
│ smoking │ ┌─────────────────┐  
│ drinking │ │ sport\_info │  
│ exercise │ ├─────────────────┤  
│ food\_types │ │ id (PK) │  
│ water\_consumption│ │ user\_id (FK) │  
│ created\_at │ │ sport\_type │  
│ updated\_at │ │ duration │  
└─────────────────┘ │ calories │  
 │ created\_at │  
 └─────────────────┘

### 5.2 Table Specifications

#### 5.2.1 Users Table

* **Primary Key**: id (INTEGER, AUTO\_INCREMENT)
* **Unique Constraints**: username, email, google\_id
* **Check Constraints**: gender IN (‘MALE’, ‘FEMALE’, ‘OTHER’), login\_type IN (‘PASSWORD’, ‘GOOGLE’)

#### 5.2.2 Roles Table

* **Primary Key**: id (INTEGER, AUTO\_INCREMENT)
* **Unique Constraints**: name
* **Default Roles**: ADMIN, USER, DOCTOR

#### 5.2.3 Permissions Table

* **Primary Key**: id (INTEGER, AUTO\_INCREMENT)
* **Unique Constraints**: name
* **Structure**: resource + action based permissions

#### 5.2.4 Health Data Tables

##### j\_body Table (Personal Health Metrics)

| Field | Type | Length | Description | Constraints |
| --- | --- | --- | --- | --- |
| id | INTEGER | - | Primary Key | PK, AUTO\_INCREMENT |
| user\_id | INTEGER | - | User Reference | FK to users.id |
| bloodLipid | FLOAT | - | Blood Lipid Level | - |
| heart\_rate | DECIMAL | 10,0 | Heart Rate (BPM) | - |
| vision | INTEGER | 10 | Vision Score | - |
| sleep\_duration | DECIMAL | 10,0 | Sleep Duration (hrs) | - |
| sleep\_quality | TEXT | - | Sleep Quality Rating | - |
| smoking | TINYINT | 1 | Smoking Status | 0/1 Boolean |
| drinking | TINYINT | 1 | Drinking Status | 0/1 Boolean |
| exercise | TINYINT | 1 | Exercise Status | 0/1 Boolean |
| food\_types | TEXT | - | Preferred Food Types | - |
| water\_consumption | DECIMAL | 10,0 | Daily Water Intake (L) | - |

##### sport\_info Table (Exercise Records)

| Field | Type | Length | Description | Constraints |
| --- | --- | --- | --- | --- |
| id | INTEGER | - | Primary Key | PK, AUTO\_INCREMENT |
| user\_id | INTEGER | - | User Reference | FK to users.id |
| sport\_type | VARCHAR | 255 | Exercise Type | - |
| duration | DECIMAL | 10,2 | Duration (minutes) | - |
| calories | INTEGER | - | Calories Burned | - |
| created\_at | TIMESTAMP | - | Record Date | DEFAULT CURRENT\_TIMESTAMP |

##### detail Table (Exercise Guidelines)

| Field | Type | Length | Description | Constraints |
| --- | --- | --- | --- | --- |
| id | INTEGER | - | Primary Key | PK, AUTO\_INCREMENT |
| sport\_type | VARCHAR | 255 | Exercise Type | - |
| disease | VARCHAR | 255 | Contraindicated Diseases | - |
| method | VARCHAR | 255 | Exercise Method | - |
| notes | VARCHAR | 255 | Precautions | - |

## 6. API Design

### 6.1 Authentication Endpoints

| Method | Endpoint | Description | Request Body | Response |
| --- | --- | --- | --- | --- |
| POST | /api/auth/register | User registration | {username, password, email, fullName} | User object |
| POST | /api/auth/login | User login | {username, password} | User object |
| POST | /api/auth/google-login | Google OAuth login | {googleToken, email, name} | User object |
| PUT | /api/auth/change-password | Change password | {oldPassword, newPassword} | Success message |
| PUT | /api/auth/update-profile | Update profile | User fields | Updated user |

### 6.2 Health Data Management Endpoints

| Method | Endpoint | Description | Request Body | Response |
| --- | --- | --- | --- | --- |
| GET | /api/health/{userId} | Get user health data | - | Health metrics object |
| POST | /api/health/{userId} | Create health record | Health data fields | Created record |
| PUT | /api/health/{userId} | Update health record | Updated fields | Updated record |
| DELETE | /api/health/{userId} | Delete health record | - | Success message |

### 6.3 Exercise Management Endpoints

| Method | Endpoint | Description | Request Body | Response |
| --- | --- | --- | --- | --- |
| GET | /api/exercise/{userId} | Get exercise records | - | List of exercise records |
| POST | /api/exercise/{userId} | Log exercise activity | Exercise data | Created record |
| GET | /api/exercise/guidelines | Get exercise guidelines | - | List of guidelines |
| GET | /api/exercise/guidelines/{type} | Get specific guidelines | - | Exercise guidelines |

### 6.4 User Management Endpoints

| Method | Endpoint | Description | Parameters | Response |
| --- | --- | --- | --- | --- |
| GET | /api/users/{id} | Get user details | User ID | User object |
| GET | /api/users/{id}/roles | Get user roles | User ID | List of roles |
| GET | /api/users/{id}/permissions | Get user permissions | User ID | List of permissions |
| POST | /api/users/{id}/roles/{roleId} | Assign role | User ID, Role ID | Success message |
| DELETE | /api/users/{id}/roles/{roleId} | Remove role | User ID, Role ID | Success message |

### 6.5 Role Management Endpoints

| Method | Endpoint | Description | Request Body | Response |
| --- | --- | --- | --- | --- |
| GET | /api/roles | List all roles | - | List of roles |
| GET | /api/roles/{id} | Get role details | - | Role object |
| POST | /api/roles | Create role | {name, description} | Created role |
| PUT | /api/roles/{id} | Update role | Role fields | Updated role |
| DELETE | /api/roles/{id} | Delete role | - | Success message |

## 7. User Interface Design

### 7.1 Frontend Architecture

* **Framework**: Vue.js 2.6.10 with Vue CLI
* **UI Library**: Element UI 2.13.2
* **State Management**: Vuex 3.1.0
* **Routing**: Vue Router 3.0.6
* **HTTP Client**: Axios 0.18.1

### 7.2 Key UI Components

#### 7.2.1 Authentication Flow

* Login page with username/password and Google OAuth options
* Registration form with validation
* Password change interface
* Profile update form

#### 7.2.2 Main Application

* Responsive layout with sidebar navigation
* Dashboard with health data overview
* AI Assistant chat interface with WebSocket connection
* User management interface (admin only)

### 7.3 Responsive Design

* Mobile-first approach using Element UI’s grid system
* Breakpoints: xs (<768px), sm (≥768px), md (≥992px), lg (≥1200px)
* Adaptive navigation and content layout

## 8. Security Design

### 8.1 Authentication & Authorization

* **Password Security**: BCrypt hashing with salt
* **Session Management**: Token-based authentication (planned JWT implementation)
* **Role-Based Access Control**: Three-tier permission system (Admin, Doctor, User)
* **OAuth Integration**: Google OAuth 2.0 for social login

### 8.2 Data Protection

* **Input Validation**: Server-side validation for all API endpoints
* **SQL Injection Prevention**: MyBatis parameterized queries
* **XSS Protection**: Vue.js built-in template escaping
* **CORS Configuration**: Controlled cross-origin resource sharing

### 8.3 Security Headers

* Content Security Policy (CSP)
* X-Frame-Options
* X-Content-Type-Options
* Strict-Transport-Security (HTTPS)

## 9. Technology Stack

### 9.1 Backend Technologies

* **Framework**: Spring Boot 2.x
* **Database**: SQLite with HikariCP connection pooling
* **ORM**: MyBatis 3.x
* **Security**: Spring Security (planned), BCrypt
* **Build Tool**: Maven
* **Java Version**: Java 8+

### 9.2 Frontend Technologies

* **Framework**: Vue.js 2.6.10
* **UI Library**: Element UI 2.13.2
* **Build Tool**: Vue CLI 4.4.4
* **Package Manager**: npm
* **Bundler**: Webpack (via Vue CLI)
* **CSS Preprocessor**: SCSS/Sass

### 9.3 Development Tools

* **Version Control**: Git
* **Code Quality**: ESLint, Babel
* **Testing**: Jest (unit testing framework)
* **Development Server**: Vue CLI Dev Server with hot reload

## 10. Deployment Architecture

### 10.1 Development Environment

* **Frontend**: Vue CLI Dev Server (port 8080)
* **Backend**: Spring Boot embedded Tomcat (port 8000)
* **Database**: SQLite file-based database
* **Proxy**: Vue CLI proxy for API calls during development

### 10.2 Production Environment (Planned)

* **Containerization**: Docker with docker-compose
* **Web Server**: Nginx for static file serving and reverse proxy
* **Application Server**: Spring Boot JAR deployment
* **Database**: SQLite or migration to PostgreSQL/MySQL
* **SSL/TLS**: Let’s Encrypt certificates

### 10.3 CI/CD Pipeline (Recommended)

* **Source Control**: Git-based workflow
* **Build Process**: Automated frontend build and backend compilation
* **Testing**: Automated unit and integration tests
* **Deployment**: Containerized deployment with health checks

## Conclusion

This Software Design Document provides a comprehensive overview of the Personal Health Management System’s architecture, components, and technical specifications. The system is designed with scalability, security, and maintainability in mind, following industry best practices and modern web development patterns.

The modular architecture allows for future enhancements and feature additions while maintaining system stability and performance. The role-based access control system ensures appropriate data access and system security for different user types.