# Personal Health Management System (PHMS) - Software Test Document (STD)

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

## Table of Contents

1. [Introduction](#Xe3d0fc0bea9a42ce7605565d0964033d7f6ee47)
2. [Test Strategy](#Xc6773d02bd61f209635dc3a341770dc1cfb3e2e)
3. [Test Environment](#X97a63654cf3d9c799947406817e01977db1e4dc)
4. [Test Cases](#Xdcf0f5ef6a693433df8dc52c59c38a4ec467ec2)
5. [API Testing](#Xfa99393fa50537a10f39682a3761037ac315c43)
6. [Frontend Testing](#X636ccb257d262c5c6e394c23ba049445561abc3)
7. [Integration Testing](#X260c42987389d0cb4e32bd3e01144892ad37d23)
8. [Security Testing](#X89916509729d56715096295986fcbb3357e2161)
9. [Performance Testing](#Xf40b9d2ee04b9c0b585f49596c050ff875349b5)
10. [Acceptance Criteria](#Xb506786b8f22ab3dac105496323403a20ea394e)

## 1. Introduction

### 1.1 Purpose

This STD defines the comprehensive testing strategy, test cases, & procedures for the PHMS. It ensures system quality, reliability, & compliance with functional requirements.

### 1.2 Scope

Testing covers all system components including authentication, RBAC, health data management, exercise tracking, AI assistant functionality, & API endpoints. Both functional & non-functional testing are included.

### 1.3 Definitions & Abbreviations

* **STD**: Software Test Document
* **PHMS**: Personal Health Management System
* **TC**: Test Case
* **API**: Application Programming Interface
* **RBAC**: Role-Based Access Control
* **UI**: User Interface
* **CRUD**: Create, Read, Update, Delete
* **E2E**: End-to-End
* **UAT**: User Acceptance Testing

## 2. Test Strategy

### 2.1 Test Levels

* **Unit Testing**: Individual component validation
* **Integration Testing**: Component interaction verification
* **System Testing**: Complete system functionality validation
* **Acceptance Testing**: Business requirement compliance verification

### 2.2 Test Types

* **Functional Testing**: Feature & requirement validation
* **Security Testing**: Authentication, authorization, & data protection
* **Performance Testing**: Load, stress, & response time validation
* **Usability Testing**: UI/UX & accessibility validation
* **API Testing**: REST endpoint & data validation

### 2.3 Testing Framework

| Test Type | Description | Tools | Responsibility |
| --- | --- | --- | --- |
| Unit Testing | Individual component testing | Jest, JUnit | Developers |
| Integration Testing | Component interaction testing | Postman, Cypress | QA Team |
| Functional Testing | Feature requirement validation | Manual/Automated | QA Team |
| Security Testing | Vulnerability assessment | OWASP ZAP, Manual | Security Team |
| Performance Testing | Load and stress testing | JMeter, Lighthouse | Performance Team |

## 3. Test Environment

### 3.1 Development Environment

* **OS**: Windows 10/11, macOS, Ubuntu
* **Browsers**: Chrome 90+, Firefox 88+, Safari 14+, Edge 90+
* **Node.js**: Version 14.x or higher
* **Java**: JDK 8 or higher
* **Database**: SQLite (file-based)

### 3.2 Test Data Management

* **Test Database**: Separate SQLite database for testing
* **Test Users**: Predefined user accounts for different roles
* **Mock Data**: Generated test data for health records
* **Data Reset**: Automated test data cleanup between test runs

### 3.3 Test Infrastructure

* **CI/CD Pipeline**: Automated testing on code commits
* **Test Reporting**: Automated test result generation
* **Bug Tracking**: Integration with issue tracking system

## 4. Test Cases

### 4.1 Authentication Test Cases

#### TC-AUTH-001: User Registration

**Objective**: Verify user can register with valid credentials  
**Preconditions**: User is on registration page  
**Test Steps**: 1. Enter valid username, email, password, and full name 2. Click “Register” button 3. Verify success message is displayed 4. Verify user is redirected to login page

**Expected Result**: User account is created successfully  
**Priority**: High

#### TC-AUTH-002: User Login

**Objective**: Verify user can login with valid credentials  
**Preconditions**: User account exists in database  
**Test Steps**: 1. Enter valid username and password 2. Click “Login” button 3. Verify user is redirected to dashboard 4. Verify user session is established

**Expected Result**: User is successfully logged in  
**Priority**: High

#### TC-AUTH-003: Google OAuth Login

**Objective**: Verify Google OAuth integration works correctly  
**Preconditions**: Google OAuth is configured  
**Test Steps**: 1. Click “Login with Google” button 2. Complete Google authentication flow 3. Verify user account is created/linked 4. Verify user is redirected to dashboard

**Expected Result**: User is authenticated via Google OAuth  
**Priority**: Medium

#### TC-AUTH-004: Invalid Login Credentials

**Objective**: Verify system handles invalid login attempts  
**Preconditions**: User is on login page  
**Test Steps**: 1. Enter invalid username or password 2. Click “Login” button 3. Verify error message is displayed 4. Verify user remains on login page

**Expected Result**: Login fails with appropriate error message  
**Priority**: High

### 4.2 User Management Test Cases

#### TC-USER-001: View User Profile

**Objective**: Verify user can view their profile information  
**Preconditions**: User is logged in  
**Test Steps**: 1. Navigate to profile page 2. Verify all user information is displayed correctly 3. Verify sensitive information is properly masked

**Expected Result**: User profile is displayed accurately  
**Priority**: Medium

#### TC-USER-002: Update User Profile

**Objective**: Verify user can update their profile information  
**Preconditions**: User is logged in  
**Test Steps**: 1. Navigate to profile edit page 2. Modify user information fields 3. Click “Save” button 4. Verify success message is displayed 5. Verify changes are persisted

**Expected Result**: User profile is updated successfully  
**Priority**: Medium

#### TC-USER-003: Change Password

**Objective**: Verify user can change their password  
**Preconditions**: User is logged in with password authentication  
**Test Steps**: 1. Navigate to change password page 2. Enter current password and new password 3. Click “Change Password” button 4. Verify success message is displayed 5. Verify user can login with new password

**Expected Result**: Password is changed successfully  
**Priority**: High

### 4.3 Role-Based Access Control Test Cases

#### TC-RBAC-001: Admin Role Permissions

**Objective**: Verify admin users have full system access  
**Preconditions**: User with admin role is logged in  
**Test Steps**: 1. Verify access to user management features 2. Verify access to role management features 3. Verify access to system configuration 4. Verify ability to assign/remove user roles

**Expected Result**: Admin has full system access  
**Priority**: High

#### TC-RBAC-002: Doctor Role Permissions

**Objective**: Verify doctor users have appropriate access  
**Preconditions**: User with doctor role is logged in  
**Test Steps**: 1. Verify access to patient health data (read) 2. Verify access to health data entry (write) 3. Verify restricted access to user management 4. Verify restricted access to system configuration

**Expected Result**: Doctor has appropriate limited access  
**Priority**: High

#### TC-RBAC-003: Regular User Permissions

**Objective**: Verify regular users have limited access  
**Preconditions**: User with user role is logged in  
**Test Steps**: 1. Verify access to own health data only 2. Verify ability to update own profile 3. Verify no access to other users’ data 4. Verify no access to administrative functions

**Expected Result**: User has appropriate limited access  
**Priority**: High

### 4.4 AI Assistant Test Cases

#### TC-AI-001: WebSocket Connection

**Objective**: Verify AI assistant WebSocket connection works  
**Preconditions**: User is logged in and on AI assistant page  
**Test Steps**: 1. Navigate to AI assistant page 2. Verify WebSocket connection is established 3. Verify connection status indicator shows “connected” 4. Verify reconnection on connection loss

**Expected Result**: WebSocket connection is stable  
**Priority**: Medium

#### TC-AI-002: Send Message to AI

**Objective**: Verify user can send messages to AI assistant  
**Preconditions**: WebSocket connection is established  
**Test Steps**: 1. Type message in input field 2. Click “Send” button or press Enter 3. Verify message appears in chat history 4. Verify AI response is received and displayed

**Expected Result**: Message exchange works correctly  
**Priority**: Medium

### 4.5 Functional Test Cases (Based on System Requirements)

#### TC-FUNC-001: User Login Validation

**Objective**: Verify login functionality with various input combinations  
**Test Data & Expected Results**:

| Test Case | Username | Password | Expected Result | Status |
| --- | --- | --- | --- | --- |
| Valid Login | 001 | 001 | Admin login successful | ✓ Pass |
| Invalid Password | 001 | 123 | Error: “Password incorrect” | ✓ Pass |
| Invalid Username | 123 | 002 | Error: “Username not found” | ✓ Pass |
| Empty Fields | - | - | Error: “Required fields missing” | ✓ Pass |

**Priority**: High  
**Test Type**: Black-box testing

#### TC-FUNC-002: Password Change Functionality

**Objective**: Verify password change with validation  
**Test Data & Expected Results**:

| Test Case | Current Password | New Password | Confirm Password | Expected Result | Status |
| --- | --- | --- | --- | --- | --- |
| Valid Change | 001 | 123 | 123 | “Password changed successfully” | ✓ Pass |
| Wrong Current | 666 | 123 | 123 | Error: “Current password incorrect” | ✓ Pass |
| Mismatch Confirm | 001 | 123 | 333 | Error: “Passwords do not match” | ✓ Pass |

**Priority**: Medium  
**Test Type**: Black-box testing

#### TC-FUNC-003: Health Data Management

**Objective**: Verify CRUD operations for health records  
**Preconditions**: User is logged in with appropriate permissions  
**Test Steps**: 1. Create new health record with valid data 2. Read/retrieve existing health records 3. Update health record fields 4. Delete health record 5. Verify data persistence across sessions

**Expected Result**: All CRUD operations work correctly  
**Priority**: High

#### TC-FUNC-004: Exercise Record Management

**Objective**: Verify exercise logging and tracking  
**Test Data**:

{  
 "sport\_type": "Running",  
 "duration": 30,  
 "calories": 250,  
 "date": "2025-01-15"  
}

**Expected Result**: Exercise record saved and retrievable  
**Priority**: Medium

## 5. API Testing

### 5.1 Authentication API Tests

#### API-AUTH-001: POST /api/auth/register

**Test Data**:

{  
 "username": "testuser",  
 "password": "TestPass123!",  
 "email": "test@example.com",  
 "fullName": "Test User"  
}

**Expected Response**: 201 Created with user object  
**Error Cases**: 400 for duplicate username/email, 400 for invalid data

#### API-AUTH-002: POST /api/auth/login

**Test Data**:

{  
 "username": "testuser",  
 "password": "TestPass123!"  
}

**Expected Response**: 200 OK with user object and session token  
**Error Cases**: 401 for invalid credentials, 400 for missing data

### 5.2 User Management API Tests

#### API-USER-001: GET /api/users/{id}

**Preconditions**: Valid user ID and authentication  
**Expected Response**: 200 OK with user object  
**Error Cases**: 404 for non-existent user, 403 for unauthorized access

#### API-USER-002: POST /api/users/{id}/roles/{roleId}

**Preconditions**: Admin authentication, valid user and role IDs  
**Expected Response**: 200 OK with success message  
**Error Cases**: 403 for non-admin user, 404 for invalid IDs

### 5.3 Health Data API Tests

#### API-HEALTH-001: GET /api/health/{userId}

**Preconditions**: Valid authentication, user permissions  
**Expected Response**: 200 OK with health data array  
**Error Cases**: 403 for unauthorized access, 404 for invalid user

#### API-HEALTH-002: POST /api/health/{userId}

**Test Data**:

{  
 "bloodPressure": "120/80",  
 "heartRate": 72,  
 "weight": 70.5,  
 "height": 175,  
 "bloodSugar": 5.5,  
 "sleepDuration": 8,  
 "exerciseMinutes": 30  
}

**Expected Response**: 201 Created with health record  
**Error Cases**: 400 for invalid data, 403 for unauthorized access

### 5.4 Exercise Management API Tests

#### API-EXERCISE-001: GET /api/exercise/{userId}

**Preconditions**: Valid authentication  
**Expected Response**: 200 OK with exercise records  
**Error Cases**: 403 for unauthorized access

#### API-EXERCISE-002: POST /api/exercise/{userId}

**Test Data**:

{  
 "sportType": "Running",  
 "duration": 30,  
 "calories": 250,  
 "notes": "Morning jog in park"  
}

**Expected Response**: 201 Created with exercise record  
**Error Cases**: 400 for invalid data

### 5.5 Role Management API Tests

#### API-ROLE-001: GET /api/roles

**Preconditions**: Admin authentication  
**Expected Response**: 200 OK with array of roles  
**Error Cases**: 403 for non-admin user

#### API-ROLE-002: POST /api/roles

**Test Data**:

{  
 "name": "NURSE",  
 "description": "Nursing staff role"  
}

**Expected Response**: 201 Created with role object  
**Error Cases**: 400 for duplicate name, 403 for non-admin user

## 6. Frontend Testing

### 6.1 Component Testing

#### FE-COMP-001: Login Component

**Test Cases**: - Render login form correctly - Validate input fields - Handle form submission - Display error messages - Navigate on successful login

#### FE-COMP-002: Dashboard Component

**Test Cases**: - Load user data on mount - Display health data summary - Handle navigation to different sections - Responsive layout on different screen sizes

#### FE-COMP-003: AI Assistant Component

**Test Cases**: - Establish WebSocket connection - Send and receive messages - Handle connection errors - Auto-scroll chat history - Display typing indicators

### 6.2 Integration Testing

#### FE-INT-001: Authentication Flow

**Test Scenario**: Complete user authentication workflow  
**Steps**: 1. User visits login page 2. Enters credentials and submits 3. System validates and creates session 4. User is redirected to dashboard 5. Navigation guards allow access to protected routes

#### FE-INT-002: Role-Based Navigation

**Test Scenario**: Navigation menu adapts to user role  
**Steps**: 1. Login as different user roles 2. Verify menu items match role permissions 3. Verify route guards prevent unauthorized access 4. Verify appropriate error messages for denied access

## 7. Integration Testing

### 7.1 Database Integration Tests

#### INT-DB-001: User CRUD Operations

**Objective**: Verify complete user lifecycle in database  
**Test Steps**: 1. Create new user via API 2. Retrieve user from database 3. Update user information 4. Verify changes are persisted 5. Delete user and verify removal

#### INT-DB-002: Role Assignment Integration

**Objective**: Verify role assignment affects user permissions  
**Test Steps**: 1. Create user with default role 2. Assign additional role via API 3. Verify user has combined permissions 4. Remove role and verify permission changes

### 7.2 Third-Party Integration Tests

#### INT-3P-001: Google OAuth Integration

**Objective**: Verify Google OAuth flow works end-to-end  
**Test Steps**: 1. Initiate Google OAuth flow 2. Complete authentication with Google 3. Verify user account creation/linking 4. Verify user session establishment 5. Verify subsequent logins work correctly

## 8. Security Testing

### 8.1 Authentication Security Tests

#### SEC-AUTH-001: Password Security

**Test Cases**: - Verify passwords are hashed with BCrypt - Verify password complexity requirements - Test password brute force protection - Verify session timeout functionality

#### SEC-AUTH-002: SQL Injection Prevention

**Test Cases**: - Test SQL injection in login forms - Test parameterized queries in MyBatis - Verify input sanitization - Test special characters in user input

#### SEC-AUTH-003: Cross-Site Scripting (XSS)

**Test Cases**: - Test script injection in form fields - Verify Vue.js template escaping - Test stored XSS in user profiles - Verify Content Security Policy headers

### 8.2 Authorization Security Tests

#### SEC-AUTHZ-001: Role-Based Access Control

**Test Cases**: - Verify users cannot access unauthorized endpoints - Test privilege escalation attempts - Verify role inheritance works correctly - Test permission boundary enforcement

#### SEC-AUTHZ-002: Data Access Control

**Test Cases**: - Verify users can only access own data - Test unauthorized data modification attempts - Verify admin access to all data - Test data isolation between users

## 9. Performance Testing

### 9.1 Load Testing

#### PERF-LOAD-001: API Endpoint Performance

**Objective**: Verify API performance under normal load  
**Test Configuration**: - Concurrent Users: 50 - Test Duration: 10 minutes - Ramp-up Time: 2 minutes

**Acceptance Criteria**: - Response time < 2 seconds for 95% of requests - Error rate < 1% - Server CPU usage < 80%

#### PERF-LOAD-002: Database Performance

**Objective**: Verify database performance under load  
**Test Configuration**: - Concurrent Connections: 20 - Query Types: SELECT, INSERT, UPDATE - Test Duration: 15 minutes

**Acceptance Criteria**: - Query response time < 500ms for 95% of queries - No database connection timeouts - No data corruption

### 9.2 Stress Testing

#### PERF-STRESS-001: System Breaking Point

**Objective**: Determine system limits and failure modes  
**Test Configuration**: - Gradually increase load until system fails - Monitor system resources and response times - Identify bottlenecks and failure points

**Success Criteria**: - System degrades gracefully under stress - No data loss during high load - System recovers after load reduction

### 9.3 Frontend Performance

#### PERF-FE-001: Page Load Performance

**Objective**: Verify frontend performance metrics  
**Test Tools**: Google Lighthouse, WebPageTest

**Acceptance Criteria**: - First Contentful Paint < 2 seconds - Largest Contentful Paint < 4 seconds - Cumulative Layout Shift < 0.1 - Time to Interactive < 5 seconds

## 10. Acceptance Criteria

### 10.1 Functional Acceptance Criteria

#### AC-FUNC-001: User Authentication

* Users can register with valid credentials
* Users can login with username/password
* Users can login with Google OAuth
* Users can change their password
* Users can update their profile information
* Invalid login attempts are properly handled

#### AC-FUNC-002: Role-Based Access Control

* Admin users have full system access
* Doctor users have appropriate healthcare access
* Regular users have limited personal access
* Unauthorized access attempts are blocked
* Role assignments can be modified by admins

#### AC-FUNC-003: AI Assistant

* WebSocket connection is established reliably
* Users can send messages to AI assistant
* AI responses are received and displayed
* Connection errors are handled gracefully
* Chat history is maintained during session

### 10.2 Non-Functional Acceptance Criteria

#### AC-PERF-001: Performance Requirements

* API response time < 2 seconds for 95% of requests
* Page load time < 5 seconds on 3G connection
* System supports 100 concurrent users
* Database queries complete within 500ms
* Frontend achieves Lighthouse score > 90

#### AC-SEC-001: Security Requirements

* All passwords are encrypted with BCrypt
* SQL injection attacks are prevented
* XSS attacks are mitigated
* HTTPS is enforced in production
* User sessions expire after inactivity

#### AC-USAB-001: Usability Requirements

* Application is responsive on mobile devices
* Interface is accessible (WCAG 2.1 AA)
* Error messages are clear and helpful
* Navigation is intuitive and consistent
* Forms provide real-time validation feedback

### 10.3 Browser Compatibility

* Chrome 90+ (Windows, macOS, Android)
* Firefox 88+ (Windows, macOS)
* Safari 14+ (macOS, iOS)
* Edge 90+ (Windows)
* Mobile browsers (iOS Safari, Chrome Mobile)

## Test Execution Schedule

### Phase 1: Unit Testing (Week 1-2)

* Backend service unit tests
* Frontend component unit tests
* Code coverage analysis

### Phase 2: Integration Testing (Week 3-4)

* API integration tests
* Database integration tests
* Frontend-backend integration

### Phase 3: System Testing (Week 5-6)

* End-to-end workflow testing
* Security testing
* Performance testing

### Phase 4: Acceptance Testing (Week 7-8)

* User acceptance testing
* Business requirement validation
* Final bug fixes and retesting

## Test Deliverables

1. **Test Plan Document** - This document
2. **Test Case Specifications** - Detailed test cases with steps
3. **Test Execution Reports** - Results of test runs
4. **Bug Reports** - Issues found during testing
5. **Test Coverage Reports** - Code coverage analysis
6. **Performance Test Results** - Load and stress test outcomes
7. **Security Test Report** - Vulnerability assessment results
8. **Final Test Summary** - Overall testing conclusion and recommendations

## Conclusion

This Software Test Document provides comprehensive testing coverage for the Personal Health Management System. The testing approach ensures system reliability, security, and performance while validating all functional requirements. Regular test execution and continuous integration will maintain system quality throughout the development lifecycle.

The acceptance criteria defined in this document serve as the quality gates for system release, ensuring that the delivered product meets user expectations and business requirements.