Software Requirements Specification

Contents

Client Portal Web Application

Version: 1.0
Date: October 26, 2025
Project: Client Portal Web App
Repository: mah-creator/Client-Portal-Web-App

---

1. Introduction

1.1 Purpose

This SRS document describes the functional and non-functional requirements for the Client Portal Web Application. The system enables service providers to interact with their clients, manage tasks/deliverables, exchange files, and communicate through a structured dashboard.

1.2 Scope

The Client Portal is a full-stack web application consisting of:

- **Frontend**: React-based SPA (Single Page Application) with TypeScript
- **Backend**: ASP.NET Core Web API with C#
- **Database**: SQLite (demo/development), extensible to other databases
- **Real-time Communication**: SignalR for notifications

1.3 Definitions, Acronyms, and Abbreviations

- **SRS**: Software Requirements Specification
- **API**: Application Programming Interface
- **JWT**: JSON Web Token
- **SPA**: Single Page Application
- **EF Core**: Entity Framework Core
- **SignalR**: Real-time communication library

1.4 References

- Repository: https://github.com/mah-creator/Client-Portal-Web-App
- .NET 8 SDK Documentation
- React 18+ Documentation

1.5 Overview

This document is organized into sections covering system overview, functional requirements, non-functional requirements, system architecture, and data requirements.

---

2. Overall Description

2.1 Product Perspective

The Client Portal is a standalone web application designed to facilitate communication and project management between service providers and their clients. It provides role-based dashboards and features tailored to three user types: Admins, Freelancers, and Customers.

2.2 Product Functions

The system provides the following core functionalities:

1. User authentication and authorization
2. Role-based access control (Admin, Freelancer, Customer)
3. Project management
4. Task/deliverable tracking
5. File upload and management
6. Real-time notifications
7. User profile management
8. Dashboard with statistics and analytics

2.3 User Classes and Characteristics

2.3.1 Admin

- Full system access
- User management capabilities
- System configuration
- View all projects and tasks
- Access to comprehensive analytics

2.3.2 Freelancer

- Manage assigned projects
- Update task statuses
- Upload deliverables
- Communicate with clients
- View project-specific analytics

2.3.3 Customer

- View project progress
- Download deliverables
- Provide feedback on tasks
- Communicate with service providers
- View personal project statistics

2.4 Operating Environment

- **Client-side**: Modern web browsers (Chrome, Firefox, Safari, Edge)
- **Server-side**: Cross-platform (.NET 8 runtime)
- **Database**: SQLite (development), SQL Server/PostgreSQL (production-ready)
- **Deployment**: Can be hosted on Windows/Linux servers, cloud platforms (Azure, AWS)

2.5 Design and Implementation Constraints

- Must use HTTPS for production deployment
- JWT tokens expire after 60 minutes (configurable)
- File uploads stored in `wwwroot/uploads` directory
- Maximum file upload size determined by server configuration
- CORS configured for React development server

2.6 Assumptions and Dependencies

- Users have stable internet connectivity
- Modern browser support (ES6+)
- .NET 8 SDK installed for backend
- Node.js and npm for frontend development

---

3. System Features and Requirements

3.1 User Authentication and Authorization

3.1.1 Description The system implements secure user authentication using JWT tokens and role-based authorization.

3.1.2 Functional Requirements

FR-AUTH-001: User Login

- **Priority**: High
- **Description**: Users shall be able to log in using email and password
- **Input**: Email address, password
- **Process**: Validate credentials, generate JWT token
- **Output**: JWT token, user profile data, token expiration time
- **Implementation**: `AuthController.Login()`, `TokenService.GenerateToken()`

FR-AUTH-002: User Registration

- **Priority**: High
- **Description**: New users shall be able to create accounts
- **Input**: Email, name, password, role (Admin/Freelancer/Customer)

- **Process**: Validate email uniqueness, hash password, create user record
- **Output**: User account created, automatic login
- **Implementation**: `AuthController.Signup()`

FR-AUTH-003: Password Management

- **Priority**: Medium
- **Description**: Users shall be able to change their passwords
- **Input**: Current password, new password
- **Process**: Verify current password, hash new password, update database
- **Output**: Success/failure confirmation
- **Implementation**: `AuthController.ChangePassword()`

FR-AUTH-004: Token-Based Authentication

- **Priority**: High
- **Description**: API endpoints shall be protected using JWT bearer tokens
- **Implementation**: JWT Bearer Authentication middleware, `[Authorize]` attribute

FR-AUTH-005: Role-Based Access Control

- **Priority**: High
- **Description**: System shall restrict access based on user roles
- **Roles**: Admin, Freelancer, Customer
- **Implementation**: Role claims in JWT, authorization policies

3.2 Project Management

3.2.1 Description The system provides comprehensive project management capabilities for organizing work between service providers and clients.

3.2.2 Functional Requirements

FR-PROJ-001: Create Project

- **Priority**: High
- **Description**: Authorized users shall be able to create new projects
- **Input**: Title, description, due date, owner ID, status
- **Process**: Validate input, create project record
- **Output**: Project created with unique ID
- **Implementation**: `ProjectsController`, `Projects` table

FR-PROJ-002: View Projects

- **Priority**: High
- **Description**: Users shall view projects based on their role
- **Admin**: All projects
- **Freelancer**: Assigned projects
- **Customer**: Projects they own or are members of

- **Implementation**: Role-based filtering in `ProjectsController`

FR-PROJ-003: Update Project

- **Priority**: High
- **Description**: Authorized users shall be able to update project details
- **Input**: Project ID, updated fields
- **Process**: Validate permissions, update record
- **Output**: Updated project data

FR-PROJ-004: Project Status Management

- **Priority**: High
- **Description**: Projects shall have trackable statuses
- **Statuses**: Pending, In Progress, Completed, On Hold
- **Implementation**: `ProjectStatus` enum

FR-PROJ-005: Project Members

- **Priority**: Medium
- **Description**: Projects shall support multiple team members
- **Implementation**: `ProjectMembers` table with project-user relationships

3.3 Task Management

3.3.1 Description Tasks (deliverables) are work items within projects that can be assigned, tracked, and completed.

3.3.2 Functional Requirements

FR-TASK-001: Create Task

- **Priority**: High
- **Description**: Users shall be able to create tasks within projects
- **Input**: Title, description, project ID, due date, assigned user
- **Process**: Validate project exists, create task record
- **Output**: Task created with unique ID
- **Implementation**: `TasksController`, `Tasks` table

FR-TASK-002: Update Task Status

- **Priority**: High
- **Description**: Users shall be able to update task statuses
- **Statuses**: Pending, In Progress, In Review, Completed, Cancelled
- **Implementation**: `TaskStatus` enum, status update endpoints

FR-TASK-003: Assign Tasks

- **Priority**: High
- **Description**: Tasks can be assigned to specific users
- **Input**: Task ID, user ID

- **Process**: Validate user exists, update assignment
- **Implementation**: `AssignedTo` field in Tasks table

FR-TASK-004: Task Comments

- **Priority**: Medium
- **Description**: Users shall be able to comment on tasks
- **Input**: Task ID, comment text
- **Process**: Create comment record linked to task
- **Output**: Comment added with timestamp
- **Implementation**: `Comments` table

FR-TASK-005: Task Filtering and Search

- **Priority**: Medium
- **Description**: Users shall be able to filter and search tasks
- **Filters**: Status, priority, assigned user, date range
- **Implementation**: `TaskAggregationController`

3.4 File Management

3.4.1 Description The system supports file uploads, storage, and downloads for project-related documents and deliverables.

3.4.2 Functional Requirements

FR-FILE-001: Upload Files

- **Priority**: High
- **Description**: Users shall be able to upload files to projects/tasks
- **Input**: File data, project/task ID, file metadata
- **Process**: Save file to server, create database record
- **Output**: File stored, file ID returned
- **Implementation**: `FileService`, `Files` table, `wwwroot/uploads` storage

FR-FILE-002: Download Files

- **Priority**: High
- **Description**: Authorized users shall be able to download files
- **Process**: Verify permissions, serve file
- **Output**: File stream
- **Implementation**: `FileService.GetFileAsync()`

FR-FILE-003: File Metadata

- **Priority**: Medium
- **Description**: System shall track file metadata
- **Metadata**: Original filename, size, MIME type, upload date, uploader
- **Implementation**: `Files` table

FR-FILE-004: Recent Files View

- **Priority**: Low
- **Description**: Users shall see recently uploaded files
- **Implementation**: Frontend `useRecentFiles` hook

3.5 Real-Time Notifications

3.5.1 Description The system provides real-time notifications for important events using SignalR.

3.5.2 Functional Requirements

FR-NOTIF-001: Real-Time Push Notifications

- **Priority**: Medium
- **Description**: Users shall receive real-time notifications
- **Events**: Task assignments, status changes, new comments, file uploads
- **Implementation**: SignalR `NotificationHub`, `NotificationService`

FR-NOTIF-002: Notification Persistence

- **Priority**: Medium
- **Description**: Notifications shall be stored for later retrieval
- **Implementation**: `Notifications` table

FR-NOTIF-003: Notification Center

- **Priority**: Low
- **Description**: Users shall access a notification history
- **Implementation**: Frontend notification components

3.6 User Profile Management

3.6.1 Description Users can view and update their profile information.

3.6.2 Functional Requirements

FR-PROFILE-001: View Profile

- **Priority**: Medium
- **Description**: Users shall view their profile information
- **Data**: Name, email, role, avatar, bio, phone
- **Implementation**: `UserController.GetProfile()`, `Profile` table

FR-PROFILE-002: Update Profile

- **Priority**: Medium
- **Description**: Users shall update their profile information
- **Input**: Updated profile fields
- **Process**: Validate and save changes
- **Implementation**: `UserController.UpdateProfile()`

FR-PROFILE-003: Avatar Upload

- **Priority**: Low
- **Description**: Users shall upload profile avatars
- **Implementation**: File upload integrated with profile

3.7 Dashboard and Analytics

3.7.1 Description Role-specific dashboards provide statistics and insights.

3.7.2 Functional Requirements

FR-DASH-001: User Statistics

- **Priority**: Medium
- **Description**: Users shall view role-specific statistics
- **Admin Stats**: Total users, projects, tasks, system activity
- **Freelancer Stats**: Assigned tasks, completion rate, deadlines
- **Customer Stats**: Project progress, deliverables, pending items
- **Implementation**: `UserController.GetUserStats()`

FR-DASH-002: Dashboard Widgets

- **Priority**: Medium
- **Description**: Dashboards shall display relevant widgets
- **Widgets**: Recent projects, task lists, file uploads, notifications
- **Implementation**: Frontend dashboard components (`CustomerDashboard.tsx`, etc.)

---

4. External Interface Requirements

4.1 User Interfaces

4.1.1 Login Page

- Email and password input fields
- "Demo Login" button for testing
- Responsive design for mobile and desktop
- Professional branding with gradient backgrounds

4.1.2 Dashboard Pages

- Role-specific layouts (Admin, Freelancer, Customer)
- Navigation header with user profile and logout
- Sidebar navigation for main sections
- Content area with widgets and data tables

4.1.3 Project/Task Management Pages

- List views with filtering and sorting
- Detail views for individual items
- Forms for creating/editing
- Status badges and progress indicators

4.1.4 File Management Interface

- Upload interface with drag-and-drop
- File list with metadata
- Download and preview capabilities

4.2 Hardware Interfaces

- No specific hardware interfaces required
- Standard web server hardware sufficient

4.3 Software Interfaces

4.3.1 Frontend-Backend Communication

- **Protocol**: HTTPS/HTTP
- **Format**: JSON
- **Authentication**: JWT Bearer tokens in Authorization header
- **Base URL**: Configurable (default: `http://localhost:56545`)

4.3.2 Database Interface

- **ORM**: Entity Framework Core
- **Provider**: SQLite (development), extensible to SQL Server/PostgreSQL
- **Connection String**: Configurable via `appsettings.json`

4.3.3 Real-Time Communication

- **Protocol**: WebSocket (SignalR)
- **Hub Endpoint**: `/hubs/notifications`
- **Authentication**: JWT token via query string

4.4 Communication Interfaces

4.4.1 HTTP API Endpoints

- **Authentication**: `/api/auth/login`, `/api/auth/signup`, `/api/auth/change-password`
- **Projects**: `/api/projects/*`
- **Tasks**: `/api/tasks/*`
- **Files**: `/api/files/*`
- **Users**: `/api/users/*`
- **Admin**: `/api/admin/*`

4.4.2 CORS Configuration

- Allows requests from React development server
- Configurable allowed origins
- All headers and methods permitted for allowed origins

---

5. Non-Functional Requirements

5.1 Performance Requirements

NFR-PERF-001: Response Time

- API endpoints shall respond within 2 seconds under normal load
- Real-time notifications shall be delivered within 1 second

NFR-PERF-002: Concurrent Users

- System shall support at least 100 concurrent users
- Database connection pooling for efficiency

NFR-PERF-003: File Upload Performance

- File uploads up to 50MB shall complete within reasonable time based on connection speed
- Chunked upload support for large files (future enhancement)

5.2 Security Requirements

NFR-SEC-001: Authentication

- Passwords shall be hashed using ASP.NET Identity PasswordHasher
- JWT tokens shall use HMAC-SHA256 signing

NFR-SEC-002: Authorization

- All API endpoints (except auth) shall require valid JWT
- Role-based access control enforced at controller level

NFR-SEC-003: Data Protection

- HTTPS required for production deployments
- Sensitive configuration (JWT keys) stored in environment variables or secure vaults

NFR-SEC-004: Input Validation

- All user inputs validated on both client and server
- Protection against SQL injection via parameterized queries (EF Core)
- XSS prevention through proper output encoding

NFR-SEC-005: Password Policy

- Minimum password length: 6 characters (configurable)
- Password complexity requirements enforceable

5.3 Reliability Requirements

NFR-REL-001: Availability

- System shall have 99% uptime during business hours
- Graceful degradation when real-time features unavailable

NFR-REL-002: Data Integrity

- Database transactions for critical operations
- Referential integrity enforced via foreign keys

NFR-REL-003: Error Handling

- Comprehensive error logging
- User-friendly error messages
- API errors returned with appropriate HTTP status codes

5.4 Maintainability Requirements

NFR-MAIN-001: Code Quality

- TypeScript for type safety on frontend
- C# with strong typing on backend
- Separation of concerns (Controllers, Services, Data layers)

NFR-MAIN-002: Documentation

- API endpoints documented via Swagger/OpenAPI
- README files for setup instructions
- Inline code comments for complex logic

NFR-MAIN-003: Database Migrations

- EF Core migrations for schema version control
- Rollback capability for database changes

5.5 Portability Requirements

NFR-PORT-001: Cross-Platform Backend

- .NET 8 supports Windows, Linux, macOS
- Database abstraction allows switching providers

NFR-PORT-002: Browser Compatibility

- Support for Chrome, Firefox, Safari, Edge (latest 2 versions)
- Responsive design for mobile and tablet devices

5.6 Usability Requirements

NFR-USE-001: User Interface

- Intuitive navigation with consistent design patterns
- Responsive design using Tailwind CSS
- Accessibility features (WCAG 2.1 compliance goal)

NFR-USE-002: Learning Curve

- Demo mode for testing without signup
- Clear labeling and helpful error messages
- Role-appropriate interfaces

---

6. System Architecture

6.1 Technology Stack

6.1.1 Frontend (76.3% TypeScript)

- **Framework**: React 18+ with TypeScript
- **Build Tool**: Vite
- **State Management**: React hooks, Context API
- **HTTP Client**: Axios (via api-client)
- **UI Components**: Custom components with Tailwind CSS
- **Icons**: Lucide React
- **Forms**: React Hook Form (inferred)
- **Notifications**: Toast system

6.1.2 Backend (20.8% C#)

- **Framework**: ASP.NET Core 8.0
- **ORM**: Entity Framework Core
- **Authentication**: JWT Bearer with Microsoft.AspNetCore.Authentication.JwtBearer
- **Real-time**: SignalR
- **API Documentation**: Swagger/OpenAPI
- **Password Hashing**: ASP.NET Core Identity PasswordHasher

6.1.3 Database

- **Development**: SQLite (file: `clientportal.db`)
- **Production-Ready**: SQL Server, PostgreSQL (via EF Core provider swap)

6.2 Architecture Pattern

- **Frontend**: Component-based architecture with hooks
- **Backend**: Layered architecture

- **Controllers**: Handle HTTP requests/responses
- **Services**: Business logic (TokenService, FileService, NotificationService, ProjectService)
- **Data**: Database context and models
- **DTOs**: Data transfer objects for API contracts

6.3 Database Schema

6.3.1 Core Tables

1. **Users**: User accounts and authentication
   - Id (PK), Email, PasswordHash, Name, Role, IsSuspended, CreatedAt
2. **Profile**: Extended user information
   - Id (PK, FK to Users), Bio, AvatarUrl, Phone, Company, Website
3. **Projects**: Project records
   - Id (PK), Title, Description, OwnerId, DueDate, Status, CreatedAt
4. **Tasks**: Work items within projects
   - Id (PK), Title, Description, ProjectId (FK), AssignedTo (FK), Status, Priority, DueDate, CreatedAt
5. **Files**: Uploaded files
   - Id (PK), FileName, FilePath, MimeType, Size, ProjectId, TaskId, UploadedBy (FK), UploadedAt
6. **Comments**: Task comments
   - Id (PK), UserId (FK), TaskId (FK), Body, CreatedAt
7. **ProjectMembers**: Project team membership
   - ProjectId (FK), UserId (FK), Role, JoinedAt
8. **Notifications**: User notifications
   - Id (PK), UserId (FK), Message, Type, IsRead, CreatedAt

6.3.2 Enumerations

- **Role**: Admin, Freelancer, Customer
- **ProjectStatus**: Pending, InProgress, Completed, OnHold
- **TaskStatus**: Pending, InProgress, InReview, Completed, Cancelled
- **TaskPriority**: Low, Medium, High, Critical

---

7. Data Requirements

7.1 Data Dictionary

7.1.1 User Entity

Field	Type	Required	Description
Id	string (GUID)	Yes	Unique identifier
Email	string	Yes	User email (unique)

Field	Type	Required	Description
PasswordHash	string	Yes	Hashed password
Name	string	No	Display name
Role	enum	Yes	User role
IsSuspended	boolean	Yes	Account status
CreatedAt	DateTime	Yes	Registration timestamp

7.1.2 Project Entity

Field	Type	Required	Description
Id	string (GUID)	Yes	Unique identifier
Title	string	Yes	Project name
Description	string	No	Project details
OwnerId	string	Yes	FK to Users
DueDate	DateTime	No	Project deadline
Status	enum	Yes	Current status
CreatedAt	DateTime	Yes	Creation timestamp

7.1.3 Task Entity

Field	Type	Required	Description
Id	string (GUID)	Yes	Unique identifier
Title	string	Yes	Task name
Description	string	No	Task details
ProjectId	string	Yes	FK to Projects
AssignedTo	string	No	FK to Users
Status	enum	Yes	Current status
Priority	enum	Yes	Task priority
DueDate	DateTime	No	Task deadline
CreatedAt	DateTime	Yes	Creation timestamp

7.2 Data Constraints

DC-001: Email Uniqueness

- User emails must be unique across the system

DC-002: Role Validation

- User role must be one of: Admin, Freelancer, Customer

DC-003: Referential Integrity

- All foreign keys must reference existing records

- Cascade delete behavior configured for dependent entities

DC-004: Date Validation

- DueDate must be in the future when set
- CreatedAt automatically set to UTC timestamp

7.3 Data Migration and Seeding

DM-001: Initial Schema

- Migration: `20250918192701_InitialSchema.cs`
- Creates all core tables

DM-002: Schema Modifications

- Migration: `20250919135952_SlightModifications.cs`
- Updates to existing schema

DM-003: Demo Data Seeding

- `DbSeeder.SeedAsync()` creates demo users and data
- Executed on application startup

---

8. Appendices

8.1 Setup Instructions

8.1.1 Backend Setup

```
cd api
dotnet restore
dotnet tool install --global dotnet-ef --version 8.0.0
dotnet ef database update
dotnet run
```

8.1.2 Frontend Setup

```
cd app
npm install
npm run dev
```

8.2 Configuration

8.2.1 Backend Configuration (appsettings.json)

- **ConnectionStrings:DefaultConnection**: Database connection

- **Jwt:Key**: Secret key for JWT signing
- **Jwt:Issuer**: Token issuer
- **Jwt:Audience**: Token audience
- **Jwt:ExpiryMinutes**: Token lifetime (default: 60)
- **ReactClientUrl**: CORS allowed origin

8.2.2 Frontend Configuration (.env)

- **REACT_APP_API_URL**: Backend API base URL

8.3 API Authentication Flow

1. User submits credentials to `/api/auth/login`
2. Backend validates credentials
3. Backend generates JWT token with user claims
4. Frontend stores token and user data
5. Frontend includes token in Authorization header for subsequent requests
6. Backend validates token on protected endpoints
7. Token expires after configured time, requiring re-authentication

8.4 Demo Mode

The system includes demo authentication mode:

- Accepts any email/password combination
- Automatically creates user account if not exists
- Useful for testing and demonstrations
- Configured via `DemoAuth` setting

8.5 Future Enhancements

- ☐ Advanced search and filtering
- ☐ Email notifications
- ☐ Calendar integration
- ☐ Time tracking
- ☐ Invoice generation
- ☐ Multi-factor authentication
- ☐ Advanced reporting and analytics
- ☐ Mobile applications (iOS/Android)
- ☐ Integration with third-party tools (Slack, Jira, etc.)
- ☐ Customizable workflows
- ☐ Automated backups
