SRS Analysis

SRS Analysis: Todo Application

1. Introduction

1.1 Purpose

This Software Requirements Specification (SRS) document outlines the functional and non-functional requirements for the Todo Application. The document serves as a comprehensive guide for the development team to understand the client's expectations and requirements.

1.2 Scope

The Todo Application is designed to allow users to manage personal task lists with different permission levels based on user roles. The system will support multiple user types, task management operations, and user management functions with appropriate access controls.

1.3 Definitions, Acronyms, and Abbreviations

- **SRS:** Software Requirements Specification
- **CRUD:** Create, Read, Update, Delete
- **JWT:** JSON Web Token
- **API:** Application Programming Interface

1.4 References

- Requirements document provided by the client
- Industry-standard security protocols for user authentication
- Best practices for scalable application architecture

1.5 Overview

The Todo Application will be a multi-user system with role-based permissions. It will allow users to manage their own todos, while administrators will have additional capabilities to manage users. The system is designed with scalability in mind to accommodate future features and tools.

2. Overall Description

2.1 Product Perspective

The Todo Application is a standalone system designed to be extensible for future integrations. It will serve as a foundation for a larger ecosystem of productivity tools.

2.2 Product Functions

- User authentication (registration, login)
- Todo management (create, view, update, delete)
- User management (create, view, update, delete)
- Role-based access control

2.3 User Classes and Characteristics

1. **Super Admin:** Has complete system access, including user management and all administrative functions
2. **Admin:** Can manage regular users and view their own todos
3. **User:** Can manage only their own todos

2.4 Operating Environment

- Web-based application
- Compatible with modern web browsers
- Backend services accessible via REST API and GraphQL

2.5 Design and Implementation Constraints

- Implementation using a monorepo architecture
- Strict data isolation between users
- Scalable infrastructure to accommodate future growth
- Adherence to security best practices for authentication and data protection

2.6 Assumptions and Constraints

- Users will have internet connectivity
- The system will be designed for high availability
- User data must be protected with appropriate encryption and access controls
- Performance should remain consistent as the user base grows

3. System Requirements

3.1 Functional Requirements

3.1.1 User Registration

- **Description:** Allow new users to create an account in the system
- **Input:** Email, password, name, additional profile information
- **Output:** User account creation confirmation
- **Process:** Validate input data, check for existing email, hash password, create user record
- **Constraints:** Email must be unique, password must meet security requirements

3.1.2 User Login

- **Description:** Authenticate existing users to access the system
- **Input:** Email, password
- **Output:** Authentication token, user profile information
- **Process:** Validate credentials against stored data, create JWT token, log login attempt
- **Constraints:** Implement account lockout after multiple failed attempts, session timeout

3.1.3 Create Todo

- **Description:** Allow users to create new todo items
- **Input:** Todo title, description (optional)

- **Output:** Created todo record
- **Process:** Validate input, create todo item associated with current user
- **Constraints:** User must be authenticated

3.1.4 View Todos

- **Description:** Allow users to view their own todo items
- **Input:** Filter parameters (optional), pagination parameters (optional)
- **Output:** List of user's todo items
- **Process:** Query todos associated with current user ID
- **Constraints:** Users can only view their own todos, not those of others

3.1.5 Update Todo

- **Description:** Allow users to modify existing todo items
- **Input:** Todo ID, updated fields (title, description, status, etc.)
- **Output:** Updated todo record
- **Process:** Validate ownership, update todo fields
- **Constraints:** Users can only update their own todos

3.1.6 Delete Todo

- **Description:** Allow users to remove todo items
- **Input:** Todo ID
- **Output:** Deletion confirmation
- **Process:** Validate ownership, mark todo as deleted or remove from database
- **Constraints:** Users can only delete their own todos

3.1.7 Create User

- **Description:** Allow admins and super admins to create new user accounts
- **Input:** User details (email, name, password), role assignment
- **Output:** Created user record
- **Process:** Validate input, create user with specified role

- **Constraints:** Only admins and super admins can create users, super admins can create any role, admins can only create regular users

3.1.8 View Users

- **Description:** Allow admins and super admins to view user accounts
- **Input:** Filter parameters (optional), pagination parameters (optional)
- **Output:** List of user accounts based on admin level access
- **Process:** Query user database with role-based restrictions
- **Constraints:** Regular users cannot view other users, admins can view all users

3.1.9 Update User

- **Description:** Allow admins and super admins to modify user accounts
- **Input:** User ID, updated fields (email, name, role, etc.)
- **Output:** Updated user record
- **Process:** Validate admin permissions, update user fields
- **Constraints:** Super admins can update any user including other admins, admins can only update regular users, role escalation restrictions apply

3.1.10 Delete User

- **Description:** Allow admins and super admins to remove user accounts
- **Input:** User ID
- **Output:** Deletion confirmation
- **Process:** Validate admin permissions, handle user data (todos)
- **Constraints:** Super admins can delete any user including admins, admins can only delete regular users

3.1.11 Update Own Profile

- **Description:** Allow all users to update their own profile information
- **Input:** Updated profile details (name, email, password, etc.)
- **Output:** Updated user profile

- **Process:** Validate input, update user record
- **Constraints:** Users can only update their own profiles regardless of role

3.2 Non-Functional Requirements

3.2.1 Performance

- **Response Time:** API responses should be delivered within 200ms under normal load
- **Scalability:** System should support at least 10,000 concurrent users without degradation
- **Transactions Throughput:** Handle at least 1,000 transactions per second

3.2.2 Security

- **Authentication:** JWT-based token authentication with secure password handling
- **Data Protection:** Encryption of sensitive data in transit and at rest
- **Authorization:** Role-based permission system with fine-grained access control

3.2.3 Reliability

- **Uptime:** 99.9% availability (less than 9 hours of downtime per year)
- **Data Integrity:** Consistent data state with appropriate validation and error handling

3.2.4 Usability

- **API Documentation:** Comprehensive documentation for all endpoints

3.2.5 Maintainability

- **Modularity:** Separation of concerns with clear boundaries between components
- **Code Quality:** Adherence to coding standards, automated linting and formatting
- **Testing:** Comprehensive test coverage (unit, integration, end-to-end)

3.2.6 Portability

- Cross-platform compatibility for web interfaces
- Containerized deployment for consistent environment across platforms

3.3 Interface Requirements

3.3.1 REST API

- **Endpoints:**
    - `POST /auth/register`: Register new user
    - `POST /auth/login`: Authenticate user
    - `GET /users`: List users (admin access)
    - `GET /users/:id`: Get user details
    - `POST /users`: Create user (admin access)
    - `PUT /users/:id`: Update user
    - `DELETE /users/:id`: Delete user (admin access)
    - `GET /todos`: List current user's todos
    - `GET /todos/:id`: Get todo details
    - `POST /todos`: Create new todo
    - `PUT /todos/:id`: Update todo
    - `DELETE /todos/:id`: Delete todo

3.3.2 Format

- JSON for request and response bodies
- Standard HTTP status codes for responses
- Authentication via Bearer token in Authorization header

4. System Architecture

4.1 Monorepo Architecture

4.1.1 Apps

- `api`: REST API server
- `cli`: Command-line interface
- `web`: Web frontend
- `admin`: Admin dashboard

4.1.2 Packages

- **core**: Business logic (entities, use cases, repository interfaces)
  - User entity and repositories
  - Todo entity and repositories
  - Authentication services
  - Permission logic
- **database**: Database abstractions and adapters
  - PostgreSQL adapter with Prisma
  - In-memory adapter for testing
- **shared**: Utilities
  - Logger
  - Error handling
  - Validation
- **types**: Shared TypeScript types
  - DTO definitions
  - Entity interfaces
  - Enum types for roles and permissions

4.1.3 Tools

- Turborepo for monorepo management
- TypeScript for type safety
- ESLint for code quality
- Jest for testing
- Prettier for code formatting

4.2 Technology Stack

4.2.1 Backend

- Node.js runtime environment
- Express for REST AP
- TypeScript for type safety

4.2.2 Database

- PostgreSQL via Prisma ORM

4.2.3 Authentication

- JWT (jsonwebtoken) for authentication tokens
- bcrypt for password hashing
- Role-based permission middleware

4.2.6 Deployment

- Docker containers
- GitHub Actions for CI/CD
- AWS or Heroku for hosting

4.3 Data Model

4.3.1 User

- `id` : UUID
- `email` : String (unique)
- `password` : String (hashed)
- `name` : String
- `role` : Enum (USER, ADMIN, SUPER_ADMIN)
- `createdAt` : DateTime
- `updatedAt` : DateTime

4.3.2 Todo

- `id`: UUID
- `title`: String
- `description`: String (optional)
- `completed`: Boolean
- `userId`: UUID (foreign key)
- `createdAt`: DateTime
- `updatedAt`: DateTime

4.4 Appendices

4.4.1 Use Case Details

- **User Registration**: New users can create accounts with email/password
- **Todo Creation**: Users can create new todo items with title and optional description
- **User Management**: Admins can create, view, update, and delete users
- **Todo Privacy**: Users can only see their own todos, even if the user deleted his todo.

4.4.2 Assumptions

- The system will initially focus on core functionality before expanding to additional tools
- User data isolation is a critical requirement
- The application will need to scale horizontally as user base grows

4.4.3 Constraints

- Data privacy regulations must be followed
- User data must be completely isolated
- System must be designed for future expansion with minimal refactoring