Secure Task Management System

Full-Stack Assessment Project

A modern, role-based task management application built with NestJS, Angular, and TypeScript

Author: Hema Sri Puppala

Organization: TurboVets

Date: February 2026

Technical Documentation
Version 1.0

Contents

Abstract

This document provides comprehensive technical documentation for the Secure Task Management System, a full-stack web application developed as an assessment project for TurboVets. The system implements enterprise-grade features including JWT authentication, role-based access control (RBAC), and a modern Kanban-style task board with drag-and-drop functionality.

The application is built using a monorepo architecture with Nx, featuring a NestJS backend and Angular frontend. It demonstrates best practices in software engineering, security, and user experience design.

Key Features

- Secure JWT-based authentication
- Three-tier role-based access control (OWNER, ADMIN, VIEWER)
- Complete CRUD operations for task management
- Drag-and-drop Kanban board interface
- Organization hierarchy for multi-tenant support
- Comprehensive audit logging
- Responsive design with dark mode support

1 Introduction

1.1 Project Overview

The Secure Task Management System is a modern web application designed to facilitate team collaboration through efficient task organization and management. The system provides a secure, scalable platform for organizations to manage their workflows with fine-grained access control and comprehensive audit trails.

1.2 Objectives

The primary objectives of this project are:

1. Implement a secure authentication system using industry-standard JWT tokens
2. Provide role-based access control with three distinct permission levels
3. Create an intuitive user interface with drag-and-drop functionality
4. Ensure data integrity and security through proper validation and authorization
5. Maintain comprehensive audit logs for compliance and security monitoring
6. Deliver a responsive, accessible interface that works across all devices

1.3 Scope

This project encompasses:

- Backend API development with NestJS
- Frontend application development with Angular
- Database design and implementation with SQLite
- Authentication and authorization mechanisms
- User interface design and implementation
- Testing and quality assurance
- Documentation and deployment guidelines

2 Technology Stack

2.1 Backend Technologies

Technology	Purpose
NestJS	Backend framework
TypeORM	Object-Relational Mapping
SQLite	Database
Passport.js	Authentication middleware
JWT	Token-based authentication
bcrypt	Password hashing
class-validator	Input validation

Table 1: Backend Technology Stack

2.2 Frontend Technologies

Technology	Purpose
Angular 18+	Frontend framework
TypeScript	Programming language
TailwindCSS	Utility-first CSS framework
Angular CDK	Component development kit
RxJS	Reactive programming

Table 2: Frontend Technology Stack

2.3 Development Tools

- **Nx:** Monorepo management and build system
- **Jest:** Testing framework
- **ESLint:** Code linting and quality
- **Prettier:** Code formatting
- **Git:** Version control

3 System Architecture

3.1 High-Level Architecture

The application follows a three-tier architecture pattern:

1. **Presentation Layer:** Angular frontend application
2. **Application Layer:** NestJS backend API
3. **Data Layer:** SQLite database with TypeORM

3.2 Backend Architecture

The backend is organized into modular components:

- **AuthModule:** Handles authentication and JWT token generation
- **UsersModule:** Manages user data and operations
- **TasksModule:** Implements task CRUD operations
- **OrganizationsModule:** Manages organizational hierarchy
- **AuditModule:** Tracks and logs user actions

3.3 Frontend Architecture

The frontend follows Angular best practices with:

- **Core Module:** Services, guards, and interceptors
- **Shared Module:** Reusable components and directives
- **Feature Modules:** Page-specific components
- **Routing Module:** Navigation and route guards

3.4 Communication Flow

1. User interacts with Angular frontend
2. HTTP requests are sent to NestJS backend
3. JWT token is attached via HTTP interceptor
4. Backend validates token and checks permissions
5. Database operations are performed via TypeORM
6. Response is sent back to frontend
7. UI is updated with new data

4 Database Design

4.1 Entity Relationship Model

The database consists of four main entities:

1. **Organization:** Represents a company or team
2. **User:** System users with roles and credentials
3. **Task:** Work items with status and assignments
4. **AuditLog:** Records of user actions

4.2 Entity Schemas

4.2.1 Organization Entity

```
CREATE TABLE organization (
    id VARCHAR(36) PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    createdAt DATETIME DEFAULT CURRENT_TIMESTAMP,
    updatedAt DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

Listing 1: Organization Table Schema

4.2.2 User Entity

```
CREATE TABLE user (
    id VARCHAR(36) PRIMARY KEY,
    username VARCHAR(255) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL,
    email VARCHAR(255),
    role ENUM('OWNER', 'ADMIN', 'VIEWER') NOT NULL,
    organizationId VARCHAR(36) NOT NULL,
    createdAt DATETIME DEFAULT CURRENT_TIMESTAMP,
    updatedAt DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (organizationId) REFERENCES organization(id)
);
```

Listing 2: User Table Schema

4.2.3 Task Entity

```
CREATE TABLE task (
    id VARCHAR(36) PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    description TEXT,
    status ENUM('OPEN', 'IN_PROGRESS', 'DONE') NOT NULL,
    organizationId VARCHAR(36) NOT NULL,
```

```
    createdById VARCHAR(36) NOT NULL,
    assignedToId VARCHAR(36),
    createdAt DATETIME DEFAULT CURRENT_TIMESTAMP,
    updatedAt DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (organizationId) REFERENCES organization(id),
    FOREIGN KEY (createdById) REFERENCES user(id),
    FOREIGN KEY (assignedToId) REFERENCES user(id)
);
```

Listing 3: Task Table Schema

4.3 Relationships

- One Organization has many Users (1:N)
- One Organization has many Tasks (1:N)
- One User creates many Tasks (1:N)
- One User is assigned to many Tasks (1:N)
- Users can have manager-subordinate relationships (M:N)

5 Authentication and Authorization

5.1 Authentication Flow

1. User submits credentials (username and password)
2. Backend validates credentials against hashed password
3. If valid, JWT token is generated with user information
4. Token is returned to frontend
5. Frontend stores token in localStorage
6. Token is attached to all subsequent API requests

5.2 JWT Token Structure

The JWT token contains:

```
{
  "sub": "user-uuid",
  "username": "admin",
  "role": "ADMIN",
  "organizationId": "org-uuid",
  "iat": 1234567890,
  "exp": 1234571490
}
```

Listing 4: JWT Payload

5.3 Role-Based Access Control

5.3.1 Role Hierarchy

Role	Permissions
OWNER	Full system access, user management, all task operations
ADMIN	Create and manage tasks, assign tasks to subordinates, view team analytics
VIEWER	Read-only access to tasks and analytics

Table 3: Role Permissions Matrix

5.3.2 Permission Enforcement

Permissions are enforced at multiple levels:

- **Backend Guards:** RolesGuard checks user role before allowing access
- **Frontend Guards:** Route guards prevent unauthorized navigation

- **UI Directives:** *appHasRole directive hides unauthorized UI elements
- **API Validation:** Ownership checks ensure users can only modify their data

6 Features Implementation

6.1 Task Management

6.1.1 Create Task

Users with ADMIN or OWNER roles can create tasks through a modal form:

```
createTask() {
  if (this.createTaskForm.valid) {
    this.taskService.create(this.createTaskForm.value)
      .subscribe(() => {
        this.loadTasks();
        this.closeCreateModal();
      });
  }
}
```

Listing 5: Create Task Implementation

6.1.2 Update Task

Tasks can be updated via:

- Edit modal for title, description, and assignee
- Drag-and-drop for status changes

```
updateTask() {
  if (this.editTaskForm.valid && this.selectedTask) {
    const updateData = {
      title: this.editTaskForm.value.title,
      description: this.editTaskForm.value.description,
      assigneeId: this.editTaskForm.value.assigneeId
    };

    this.taskService.update(this.selectedTask.id, updateData)
      .subscribe(() => {
        this.loadTasks();
        this.closeEditModal();
      });
  }
}
```

Listing 6: Update Task Implementation

6.1.3 Delete Task

Tasks can be deleted with confirmation:

```
deleteTask() {
  if (this.selectedTask &&
      confirm('Are you sure you want to delete this task?')) {
    this.taskService.delete(this.selectedTask.id)
      .subscribe(() => {
```

```
        this.loadTasks();
        this.closeEditModal();
      });
  }
}
```

Listing 7: Delete Task Implementation

6.2 Drag-and-Drop Functionality

The Kanban board uses Angular CDK for drag-and-drop:

```
drop(event: CdkDragDrop<Task[]>, status: string) {
  if (event.previousContainer === event.container) {
    moveItemInArray(
      event.container.data,
      event.previousIndex,
      event.currentIndex
    );
  } else {
    transferArrayItem(
      event.previousContainer.data,
      event.container.data,
      event.previousIndex,
      event.currentIndex
    );
    const task = event.container.data[event.currentIndex];
    this.taskService.update(task.id, { status })
      .subscribe();
  }
}
```

Listing 8: Drag-and-Drop Handler

6.3 Analytics Dashboard

The analytics component provides:

- Total task count
- Completion rate calculation
- Task distribution by status
- Tasks by assignee
- Recent activity feed

7 Security Considerations

7.1 Password Security

- Passwords are hashed using bcrypt with salt rounds
- Plain text passwords are never stored
- Password validation enforces minimum requirements

7.2 Token Security

- JWT tokens have expiration times
- Tokens are signed with a secret key
- Tokens are validated on every API request
- Refresh token mechanism can be implemented

7.3 Input Validation

- All inputs are validated using class-validator
- SQL injection prevention through TypeORM parameterization
- XSS protection through Angular's built-in sanitization
- CORS configuration restricts allowed origins

7.4 Audit Logging

All user actions are logged with:

- User ID and username
- Action performed
- Resource accessed
- Timestamp
- Request method and endpoint

8 API Documentation

8.1 Authentication Endpoints

8.1.1 POST /api/auth/login

Description: Authenticate user and receive JWT token

Request Body:

```
{
  "username": "admin",
  "password": "password"
}
```

Response:

```
{
  "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "user": {
    "id": "uuid",
    "username": "admin",
    "role": "ADMIN"
  }
}
```

8.2 Task Endpoints

Method	Endpoint	Description
GET	/api/tasks	Get all tasks for user's organization
GET	/api/tasks/:id	Get specific task by ID
POST	/api/tasks	Create new task (ADMIN/OWNER only)
PATCH	/api/tasks/:id	Update task (ADMIN/OWNER only)
DELETE	/api/tasks/:id	Delete task (ADMIN/OWNER only)

Table 4: Task API Endpoints

8.3 User Endpoints

Method	Endpoint	Description
GET	/api/users	Get all users (ADMIN/OWNER only)
GET	/api/users/:id	Get specific user by ID
POST	/api/users	Create new user (OWNER only)
PATCH	/api/users/:id	Update user (OWNER only)
DELETE	/api/users/:id	Delete user (OWNER only)

Table 5: User API Endpoints

9 Installation and Setup

9.1 Prerequisites

- Node.js v18 or higher
- npm or pnpm package manager
- Git version control

9.2 Installation Steps

1. Clone the repository:

```
git clone <repository-url>
cd HPuppala-18023069-8027-4aa5-99dc-07eaf624ab5f

```

2. Install dependencies:

```
npm install

```

3. Start the backend:

```
npx nx serve api

```

4. Start the frontend (in a new terminal):

```
npx nx serve dashboard

```

5. Access the application at http://localhost:4200

9.3 Default Credentials

Username	Password	Role
owner	password	OWNER
admin	password	ADMIN
demo	password	VIEWER

Table 6: Default User Accounts

10 Testing

10.1 Unit Testing

Run backend unit tests:

```
npx nx test api
```

Run frontend unit tests:

```
npx nx test dashboard
```

10.2 Test Coverage

Generate coverage reports:

```
npx nx test api --coverage
npx nx test dashboard --coverage
```

10.3 End-to-End Testing

E2E tests can be added using Cypress or Playwright for comprehensive testing of user workflows.

11 Conclusion

11.1 Project Summary

The Secure Task Management System successfully demonstrates the implementation of a modern, full-stack web application with enterprise-grade features. The project showcases:

- Secure authentication and authorization mechanisms
- Clean, maintainable code architecture
- Modern UI/UX design principles
- Comprehensive security measures
- Scalable database design
- Professional documentation

11.2 Future Enhancements

Potential improvements for future iterations:

1. Real-time updates using WebSockets
2. Email notifications for task assignments
3. File attachments for tasks
4. Advanced search and filtering
5. Export functionality (PDF, CSV)
6. Mobile native applications
7. Integration with third-party services
8. Advanced analytics and reporting

11.3 Lessons Learned

Key takeaways from this project:

- Importance of proper security implementation
- Value of modular architecture
- Benefits of TypeScript for type safety
- Effectiveness of monorepo structure
- Significance of user experience design

References

1. NestJS Documentation. https://docs.nestjs.com/
2. Angular Documentation. https://angular.io/docs
3. TypeORM Documentation. https://typeorm.io/
4. JWT Introduction. https://jwt.io/introduction
5. TailwindCSS Documentation. https://tailwindcss.com/docs
6. Nx Documentation. https://nx.dev/
7. Angular CDK. https://material.angular.io/cdk/categories
8. OWASP Security Guidelines. https://owasp.org/

A Project Structure

```
.
        apps/
            api/                    # NestJS Backend
                src/
                    auth/           # Authentication module
                    users/          # User management
                    tasks/          # Task management
                    organizations/  # Organization module
                    audit/          # Audit logging
                test/               # Tests
            dashboard/              # Angular Frontend
                src/
                    app/
                        core/       # Services, guards
                        pages/      # Components
                        shared/     # Shared code
                    assets/         # Static files
                tailwind.config.js
        libs/
            data/                   # Shared interfaces
            auth/                   # Auth utilities
        docs/                     # Documentation
        db.sqlite                 # Database
        README.md
```

B Environment Variables

```
# Database
DATABASE_PATH=./db.sqlite

# JWT
JWT_SECRET=your-secret-key-here
JWT_EXPIRATION=1d

# Server
PORT=3000
NODE_ENV=development

# Frontend
API_URL=http://localhost:3000
```

Listing 9: .env Configuration

C Glossary

API Application Programming Interface

CDK Component Development Kit

CRUD Create, Read, Update, Delete

JWT JSON Web Token

ORM Object-Relational Mapping

RBAC Role-Based Access Control

REST Representational State Transfer

SPA Single Page Application

UUID Universally Unique Identifier