HMS T&L System

Usage Documentation
Group 15

JAVA CAN'T SEE SHARP

Front to Back, We're on Track

Deliverable Two: Back-End

(CMPG323)

Group Members:

41425626 – Z. Hansa – (Group Leader)

37833707 – S. Ndobela

42304954 – S. Nhlapo

35928654 – T Gumbi

Submission date: 27/09/2024

Table of Contents

System Overview

The backend system is a Node.js application using Express, designed to manage courses, assignments, videos, and user roles (Admins, Lecturers, and Students). The system integrates with Azure SQL Database for data storage and Azure Blob Storage for video management. Swagger is used for API documentation, and Docker is used to containerize the application.

Technology Stack

- Frontend: React (not included in this documentation)
- Backend: Node.js with Express
- Database: Azure SQL Database
- File Storage: Azure Blob Storage
- Video Compression: ffmpeg
- API Documentation: Swagger
- Authentication: JWT
- Containerization: Docker

Backend Structure

The backend code structure for our project (excluding node_modules) contains the following components:

env	This file stores environment variables such as API keys, database connection strings, and JWT secrets, ensuring sensitive data is not hard-coded.
config/	This folder contains configuration files for different aspects of the application, such as database configurations or environment settings.
controllers/	Contains the business logic for handling requests. Each controller typically corresponds to a different resource (e.g., users, assignments, videos) and manages how data is processed, validated, and sent as a response.
middlewares/	This folder contains middleware functions that intercept requests before they reach the controller. Our Middleware includes authentication (JWT validation), error handling, and logging.
package-lock.json & package.json	These files describe the project's dependencies and scripts. They include all the npm packages being used (e.g., Express, JWT, Swagger) and versions, allowing reproducibility of the environment.
routes/	This folder contains the route definitions for the API. It maps HTTP requests to the appropriate controllers and endpoints. For instance, routes could define the different operations for assignments, submissions, or videos.
server.js	This is the main entry point of the backend application. It initializes the server, connects to the database, and starts listening for incoming requests.
swaggerConfig.js	This file configures Swagger, ensuring the API documentation is generated dynamically based on the code structure and routes.

Analysis in Relation to the Spec Doc:

- **Environment Configuration**: The use of an .env file aligns well with security best practices outlined in the spec document, ensuring sensitive information like database credentials and JWT secrets are not exposed.
- **API Layer (Controllers and Routes):** The structure separating controllers and routes follows a typical MVC (Model-View-Controller) architecture, which allows for easy scalability and separation of concerns. This is essential for managing a complex system with multiple entities like users, assignments, videos, and feedback.
- **Middleware:** The addition of a middlewares folder includes the use of role-based access control (e.g., Admin, Lecturer, Student) as discussed in the documentation, likely enforced through JWT validation or other security layers.
- **Swagger Documentation:** The swagger folder and swaggerConfig.js show a focus on clear API documentation, which is crucial for both internal and external developers to understand and test the backend API, as outlined in your API documentation.

Dependencies

Before setting up the project, ensure you have the following tools installed on your machine:

Dependency	**Version**
@azure/storage-blob	12.24.0
axios	1.7.7
bcryptjs	2.4.3
cors	2.8.5
dotenv	16.4.5
express	4.19.2
fluent-ffmpeg	2.1.3
json2csv	6.0.0-alpha.2
jsonwebtoken	9.0.2
mssql	11.0.1
multer	1.4.5-lts.1
streamifier	0.1.1
swagger-jsdoc	6.2.8
swagger-ui-express	5.0.1
uuid	10.0.0

Installation and Setup

1. Clone the repository:

```
git clone <https://github.com/zainaaazz/FullStackWebApplication>

cd HMS SYSTEM
```

2. Install dependencies:

```
npm install
```

Configuration

1. Environment Variables

Create a `.env` file in the root directory and add the following configuration values:

```
# Database Configuration
DATABASE_USER=<Your-Database-Username>
DATABASE_PASSWORD=<Your-Database-Password>
DATABASE_SERVER=<Your-Database-Server>
DATABASE_NAME=<Your-Database-Name>

# JWT Secret Key
JWT_SECRET=<Your-Secret-Key>

# Server Port
PORT=3000
```

Ensure that these values correspond to your Azure SQL Database credentials and JWT secret for secure authentication.

Database Setup and Migrations

1. Setting Up the Database:

The project is configured to connect to an Azure SQL database. Ensure that you have created an Azure SQL Database instance and have the connection string in your .env file.

2. Running Migrations:

The system uses a migration tool to ensure the database schema is properly set up. To run migrations:

```
npx sequelize-cli db:migrate
```

Running the Application Locally

1. Starting the Server Locally: To start the backend server locally, use:

```
npm start
```

2. The backend server will be available at:

```
http://localhost:3000
```

3. Accessing Swagger API Documentation:

Once the server is running, you can view and interact with the API documentation through Swagger at:

```
http://localhost:3000/api-docs
```

Authentication and JWT Usage

All protected endpoints require a valid JWT token for access. After logging in through the /auth/login endpoint, include the token in the Authorization header in the following format:

```
Authorization: Bearer <your-token>
```

Docker Setup and Container Builds

If you prefer to run the system in a Docker container, follow these steps:

Building the Docker Container

Build the Docker container with the following command:

```
docker build -t hms-backend .
```

Running the Docker Container

Once built, run the container using:

```
docker run -p 3000:3000 hms-backend
```

This maps the container's internal port 3000 to the local port 3000 on your machine.

https://nwu-hms-g6fjckard7fggqar.southafricanorth-01.azurewebsites.net/api-docs/

Testing

To run tests (if applicable), use the following command:

```
npm test
```

Troubleshooting

- Port conflicts: If port 3000 is already in use, modify the port in the .env file.
- Database connection issues: Ensure your .env file contains the correct credentials for the Azure SQL database.
- JWT errors: Make sure the JWT_SECRET value is set correctly in the .env file and matches the token signing configuration used by the application.