**CMS REST API Documentation**

**Overview**

A basic Content Management System (CMS) implemented as a Spring Boot REST API. This service allows two types of users:

* **Admin**: Create, read, update, delete articles and upload images.
* **Guest**: Read articles and view images.

All development is done locally using:

* Java 17 source compatibility but runs on Java 23 via VS Code Java runtime
* Spring Boot 3.5.0
* MySQL (via XAMPP & phpMyAdmin) creating the database named cmsdb, the tables are created and updated on their own through the project
* Maven for build/test/run
* VS Code with **Java Extension Pack** (Microsoft) and **Java Auto Config Extension Pack** (Pleiades) are important for running this project, most important being that although this projects Java version is written as Java 17 it was run using VS code Java runtime for Java 23.

**Tech Stack**

| **Layer** | **Technology** |
| --- | --- |
| Language | Java 17 (compiled/run on 23) |
| Framework | Spring Boot 3.5.0 |
| Build & Dependency | Maven |
| Database | MySQL (phpMyAdmin/XAMPP) |
| Authentication | JWT (HS512) |
| IDE & Extensions | VS Code + Microsoft Java Extension Pack + Pleiades Java Auto Config |
| Documentation | Swagger UI & OpenAPI |

**Local Environment Setup**

1. **Install Java 17** JDK (source) and Java 23 runtime.
2. **Install MySQL/XAMPP** and start Apache & MySQL via XAMPP control panel.
3. **Create database** cmsdb in phpMyAdmin (<http://localhost/phpmyadmin>).
4. **Clone & open** the project in VS Code.
5. **Enable VS Code extensions**:
   * **Java Extension Pack** by Microsoft
   * **Java Auto Config Extension Pack** by Pleiades
6. **Configure application.properties** under src/main/resources:
7. spring.datasource.url=jdbc:mysql://localhost:8000/cmsdb
8. spring.datasource.username=YOUR\_DB\_USER
9. spring.datasource.password=YOUR\_DB\_PASS
10. spring.jpa.hibernate.ddl-auto=update

**Running & Testing**

Use your terminal in project root:

# Run all unit tests on embedded H2

mvn clean test

# Build & package, runs tests as well

mvn clean install

# Launch the application

mvn spring-boot:run

Once the project is running, you can access:

* **Swagger UI**: <http://localhost:8080/swagger-ui/index.html#/>
* **OpenAPI JSON**: <http://localhost:8080/v3/api-docs>

For testing the endpoints and interacting with the features of the project

**API Endpoints**

**Authentication**

| **Method** | **URL** | **Body** | **Response** | **Role** |
| --- | --- | --- | --- | --- |
| POST | /api/auth/login | { "username":"admin","password":"password" } | { token, username, role } | Admin |

**Articles**

| **Method** | **URL** | **Body** | **Response** | **Role** |
| --- | --- | --- | --- | --- |
| GET | /api/articles | — | Article[] | Admin, Guest |
| GET | /api/articles/{id} | — | Article | Admin, Guest |
| POST | /api/articles | **multipart/form-data**– data (JSON): { "title","content" }– image (file) | Article | Admin |
| PUT | /api/articles/{id} | **multipart/form-data**– data (JSON): { "title","content" }– image (file) | Article | Admin |
| DELETE | /api/articles/{id} | — | 200 OK | Admin |

**Images**

| **Method** | **URL** | **Body** | **Response** | **Role** |
| --- | --- | --- | --- | --- |
| POST | /api/images | **multipart/form-data**– image (file) | Image | Admin |
| GET | /api/images/{id} | — | raw binary image | Admin, Guest |

The POST /api/articles endpoint needs a JSON input in the Data text field and then to upload an image from the local directory when testing in either Swagger UI or Postman. For example data: {"title":"….","content":"…….."}. The same goes for the PUT endpoint so that it updates the article chosen by id and in the POST /api/images endpoint it needs an image to be uploaded from the computers local directory to the SQL database. It is also important to note that many of these endpoints need confirmation or authentication that an admin user is testing them on Swagger UI or Postman the admin credentials might be asked for authentication, if that happens then use the credentials for the admin user (username: admin, password: password) seeded by the DataInitializer class to be able to run and test the endpoints. These credentials are seeded in the database by the DataInitializer class can also be used by the POST /api/auth/login endpoint and it will show the token, also the database saves the password of all users in a 512 bit hash so there no security violations when running.

**Data Model**

* **User**: id, username (unique), passwordHash, role
* **Article**: id, title, content, timestamps, linked Image
* **Image**: id, filename, filepath, upload timestamp

**Development Process**

1. **Data Layer**: Defined JPA entities & Spring Data repositories for User, Article and Image
2. **Service Layer**: Business logic for articles & images.
3. **Controller Layer**: REST endpoints; multipart support for image file uploads.
4. **Security**: JWT Auth filter and CustomUserDetailsService.
5. **Initialization**: DataInitializer which seeds two users (admin/guest) into the SQL database.
6. **Swagger**: Auto-generated OpenAPI via Springdoc.
7. **Testing**: Unit tests for services & controllers (MockMvc).

**Resources Used:**

* Spring Boot & Spring Data JPA docs
* Lombok dependency for not having repetition of always writing getters, setters and constructors for many classes such as the entity classes.
* H2 database dependency and the MockMvc for testing
* Spring Security & JWT tutorials and dependencies
* Swagger/OpenAPI with springdoc
* Maven & VS Code Java extension guides