Kamil Gabrysiewicz	Index: 95400	Grupa: 2.1
Magisterskie Semestr 2	Programowanie aplikacji internetowych w JEE	

Memstagram: Instagram for Memes

An Instagram-like app where users can post memes, follow other users, and view memes of other users. No likes or comments, just follows. This project implements a backend API using Spring Boot 3.4.1 and Java 23.

Features

- User registration and login.
- Profile management (username, bio, profile picture, password, email).
- Upload memes (image URL, description).

- Memes management (Update, Delete)
- Follow and unfollow other users, see who follows.
- View memes from other users at their profile.

Technologies Used

- Backend: Spring Boot 3.4.1 (Java 23)
- Database: MySQL (Dockerized)
- Authentication: JWT (JSON Web Token)
- ORM: Spring Data JPA
- Security: Spring Security
- Dependency Management: Maven
- Frontend:
  - React: 18.3.1
  - Vite: 6.0.5
  - Bootstrap: 5.3.3

Project Structure Here is simplified view of the project structure.

```
.
├── README.md
│   └── Documentation
├── database
│   └── Data
├── docker
│   └── docker-compose.yaml
├── frontend
│   └── Vite+React
└── memstagram
│   └── Spring
```

Here is more complexed view of Spring packages, services, models [...]

```
.
├── MemstagramApplication.java
├── config
│   └── SecurityConfig.java
├── controller
│   ├── AuthController.java
│   ├── FollowController.java
│   ├── MemeController.java
│   └── UserController.java
├── dto
│   ├── FollowedDto.java
│   ├── LoginRequestDto.java
│   ├── MemeDto.java
│   ├── UnfollowRequest.java
│   └── UserRegistrationDto.java
├── exception
│   ├── GlobalExceptionHandler.java
│   ├── ResourceAlreadyExistsException.java
│   ├── ResourceNotFoundException.java
│   └── UserNotFoundException.java
├── model
│   ├── Follow.java
│   ├── Meme.java
│   └── User.java
├── repository
│   ├── FollowRepository.java
│   ├── MemeRepository.java
│   └── UserRepository.java
├── service
│   ├── FollowService.java
│   ├── MemeService.java
│   └── UserService.java
└── util
    ├── JwtAuthenticationFilter.java
    └── JwtUtil.java
```

Here is how react with its components and assets is organized:

```
.
├── App.css
├── App.jsx
├── assets
│   ├── default_profile_image.jpeg
│   ├── no_memes.jpg
│   └── react.svg
├── components
│   ├── Feed.css
│   ├── Feed.jsx
│   ├── Login.jsx
│   ├── MemeEdit.css
│   ├── MemeEdit.jsx
│   ├── MemeManage.css
│   ├── MemeManage.jsx
│   ├── NavigationBar.css
│   ├── NavigationBar.jsx
│   ├── Profile.css
│   ├── Profile.jsx
│   ├── ProfileEdit.jsx
│   ├── ProfileFollowers.jsx
│   ├── ProfileFollowing.jsx
│   ├── ProfileMemes.jsx
│   ├── Register.jsx
│   ├── UploadMeme.jsx
│   └── Welcome.jsx
├── index.css
├── main.jsx
├── pages
│   ├── AuthPage.css
│   ├── AuthPage.jsx
│   └── ProfilePage.jsx
└── styles
```

Database The project uses a MySQL database running in Docker. Here is the database schema:

```
+-------------------+---------------------+------+-----+---------+----------------+
| Field             | Type                | Null | Key | Default | Extra          |
+-------------------+---------------------+------+-----+---------+----------------+
| created_at        | datetime(6)         | YES  |     | NULL    |                |
| id                | bigint              | NO   | PRI | NULL    | auto_increment |
| updated_at        | datetime(6)         | YES  |     | NULL    |                |
| username          | varchar(48)         | NO   | UNI | NULL    |                |
| bio               | varchar(128)        | YES  |     | NULL    |                |
| email             | varchar(255)        | NO   | UNI | NULL    |                |
| password          | varchar(255)        | NO   |     | NULL    |                |
| profile_image_url | varchar(255)        | YES  |     | NULL    |                |
| role              | enum('ADMIN','USER') | NO   |     | NULL    |                |
+-------------------+---------------------+------+-----+---------+----------------+

+-------------+--------------+------+-----+---------+----------------+
| Field       | Type         | Null | Key | Default | Extra          |
+-------------+--------------+------+-----+---------+----------------+
| created_at  | datetime(6)  | YES  |     | NULL    |                |
| id          | bigint       | NO   | PRI | NULL    | auto_increment |
| updated_at  | datetime(6)  | NO   |     | NULL    |                |
| user_id     | bigint       | NO   | MUL | NULL    |                |
| description | varchar(255) | YES  |     | NULL    |                |
| image_url   | varchar(255) | NO   |     | NULL    |                |
+-------------+--------------+------+-----+---------+----------------+

+-------------+-------------+------+-----+---------+----------------+
| Field       | Type        | Null | Key | Default | Extra          |
+-------------+-------------+------+-----+---------+----------------+
| created_at  | datetime(6) | YES  |     | NULL    |                |
| followed_id | bigint      | NO   | MUL | NULL    |                |
| follower_id | bigint      | NO   | MUL | NULL    |                |
| id          | bigint      | NO   | PRI | NULL    | auto_increment |
| updated_at  | datetime(6) | YES  |     | NULL    |                |
+-------------+-------------+------+-----+---------+----------------+
```

API Endpoints

Backend server releases an API at localhost:8080/api, all endpoint except (JWT FREE) require valid JWT to access them.

User Endpoints:

- GET /api/test: Endpoint for API test
- (JWT FREE) POST /api/auth/register: Register a new user.
- (JWT FREE) POST /api/auth/login: Log in and get a JWT token.
- GET /api/users: Returns all users.
- GET /api/user/id/{id}: Returns only given user.
- GET /api/user/username/{username}: Returns only given user.
- PUT /api/user/{id}: Allows to update user data.
- DELETE /api/user/{id}: Delets user, only ADMIN can use it.

Meme Endpoints:

- POST /memes/upload: Upload a new meme.
- GET /api/memes: Gets all memes .
- GET /api/meme/{id}: Get meme by id.
- GET /api/memes/username/{username}: Returns memes of a user.
- GET /api/memes/count/username/{username}: Return how many memes user has.
- GET /api/memes/username/{username}/{from}/{to}: Returns memes of a {user} with id between {from} {to}
- POST /api/meme: Upload a meme.
- PUT /api/meme/{id}: Edits a meme.
- DELETE /api/meme/{id}: Deletes a meme.

Follow Endpoints:

- POST /api/follow: Allows user1 to follow user2.
- GET /api/followers/{id}: Returns followers of given user.
- GET /api/following/{id}: Returns users which are followed by given user
- GET /api/followers/username/{username}: Returns followers of given user.
- GET /api/following/username/{username}: Returns users which are followed by given user.
- DELETE /api/unfollow: Allows user1 to unfollow user2

How to run:

Database, inside docker directory run these (--build might be needed):

```
docker-compose up -d
```

Spring Backend, its better to use terminal to avoid IDE errors that can happen very often. I used only these to run the backend, these commands are for Windows Powershell. Should run the app at localhost:8080:

```
.\mvnw clean
.\mvnw package install
.\mvnw spring-boot:run
```

Frontend, inside frontend directory. Make sure that dependencies are installed:

```
npm install
```

Should run app at localhost:3000:

```
npm run dev
```