Micro Blog API Documentation

This document provides a reference for the Micro Blog API endpoints. It is intended to be used by frontend developers to build the client-side application.

The Development Environment

The installation strategy for a Node.js server involves using npm (Node Package Manager), as bundled with Node.js. The goal is to set up the project with all its external dependencies (like Express) correctly installed and managed. First, we need to install Node, per the operating system in question. See the official Node.js website for details: https://nodejs.org/. Our application uses Sequelize as an Object Relational Mapper (ORM), which supports various databases. We will need a database server installed and set up (this example employs MySQL).

Having successfully installed MySQL and Node, we can go to install credentials and configure the interfaces, within VSCode.

Ensure that the database credentialss are loaded into the .env environmental file, within the project root directory. For our example, copy the `.env.example` file, renaming it `.env`, then update the environment variables (DB_PASSWORD={"YOUR DB PASSWORD HERE"}).

Procedure	Command
Log in to the MySQL database (use predeclared credentials)	$: mysql -u root -
Set up the database	SQL> source db/schema.sql;
Quit out of MySQL	SQL> quit;
Verify Node installed and accessible from our project file	$: node -v
Initialise npm (creating package.json file).	$: npm init
Install dependencies (creating node_modules and package-lock.json)	$: npm install
Seed the database with test data	$: npm run seed
Start the server and run the application	|$: npm start

Based on our configuration files, the application frontend will now be accessible at http://localhost:3001

API Integration

All endpoints are hosted at the base URL specified below.

Base URL

http://localhost:3001/api

Authentication

All endpoints, unless otherwise noted, require a JSON Web Token (JWT) for authentication. The token must be included in the Authorization header of the request in the following format:

Authorization: Bearer <token>

The token is obtained from the successful login request and should be stored securely on the client side.

User Management

The API can handle new user registration, user login and logout, and retrieving information for both the currently logged-in user and other users by ID.

1. Register a New User

- **Method:** POST
- **Endpoint:** /users
- **Authentication:** None
- **Body:**
  - username: (string) The desired username.
  - email: (string) The user's email address.
  - password: (string) The user's password.

- **Example Request:**

  POST {{baseUrl}}/api/users
  Content-Type: application/json

  ```
  {
     "username": "newuser",
     "email": " newuser@example.com ",
     "password": "password123"
  }
  ```

- **Example Response (Success):**

  ```
  {
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
    "id": 1,
    "username": "newuser",
    "email": "newuser@example.com"
  }
  ```

2. Login User

- **Method:** POST
- **Endpoint:** /users/login
- **Authentication:** None
- **Body:**
  - email: (string) The user's email.
  - password: (string) The user's password.

- **Example Request:**

  POST {{baseUrl}}/api/users/login

  Content-Type: application/json

  ```
  {
    "email": newuser@example.com ",
    "password": "password123"
  }
  ```

- **Example Response (Success):**

  ```
  {
   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
   "user": {
    "id": 1,
    "username": "newuser"
   }
  }
  ```

3. Get Logged-in User Info

- **Method:** GET
- **Endpoint:** /users/me
- **Authentication:** Required

- **Example Request:**

  GET {{baseUrl}}/api/users/me
  Authorization: Bearer {{authToken}}

- **Example Response (Success):**

```
{
  "id": 1,
  "username": "newuser",
  "email": "newuser@example.com",
  "posts": [...],
  "comments": [...]
}
```

4. Get a Single User by ID

- **Method:** GET
- **Endpoint:** /users/:id
- **Authentication:** None

- **Example Request:**

```
GET {{baseUrl}}/api/users/{{registeredUserId}}
```

- **Example Response (Success):**

```
{
  "id": 1,
  "username": "newuser",
  "followerCount": 5
}
```

5. Get All Users

- **Method:** GET
- **Endpoint:** /users
- **Authentication:** Required

- **Example Request:**

```
GET {{baseUrl}}/api/users
Authorization: Bearer {{authToken}}
```

- **Example Response (Success):**

```
[
 { "id": 1, "username": "user1" },
 { "id": 2, "username": "user2" }
]
```

Category Management

6. Create a New Category

- **Method:** POST
- **Endpoint:** /categories
- **Authentication:** Required
- **Body:**
  - category_name: (string) The name of the new category.
- **Example Response (Success):**

```
{
 "id": 1,
 "category_name": "Technology"
}
```

7. Get All Categories

- **Method:** GET
- **Endpoint:** /categories
- **Authentication:** Required
- **Example Response (Success):**

```
[
 { "id": 1, "category_name": "Technology" },
 { "id": 2, "category_name": "Sports" }
]
```

Post Management

Users can **create**, **update**, and **delete** their own posts. They can also retrieve all posts or a specific post by its ID.

8. Create a New Post

- **Method:** POST
- **Endpoint:** /posts
- **Authentication:** Required
- **Body:**
  - title: (string) The post title.
  - content: (string) The post content.
  - category_id: (integer) The ID of the category the post belongs to.

- **Example Request:**

  POST {{baseUrl}}/api/posts
  Content-Type: application/json
  Authorization: Bearer {{authToken}}

  ```
  {
    "title": "testuser: My First API Post Title",
    "content": "testuser: This is the content of my first post created via REST Client.",
    "category_id": {{categoryId}}
  }
  ```

- **Example Response (Success):**

  ```
  {
   "id": 1,
   "title": "My First Post",
   "content": "This is the content.",
   "category_id": 1,
   "user_id": 1
  }
  ```

9. Get All Posts

- **Method:** GET
- **Endpoint:** /posts
- **Authentication:** Required
- **Query Parameter:** category_id (optional, integer) - Filters posts by category.

- **Example Request:**

  GET {{baseUrl}}/api/posts
  Authorization: Bearer {{authToken}}

- **Example Response (Success):**

```
[
 {
  "id": 1,
  "title": "First Post",
  "user": { "id": 1, "username": "user1" },
  "category": { "id": 1, "category_name": "Technology" }
 },
 ...
]
```

10. Get a Single Post by ID

- **Method:** GET
- **Endpoint:** /posts/:id
- **Authentication:** Required

- **Example Request:**

  GET {{baseUrl}}/api/posts/{{postId}}
  Authorization: Bearer {{authToken}}

- **Example Response (Success):**

```
{
 "id": 1,
 "title": "First Post",
 "content": "This is the content.",
 "user": { "id": 1, "username": "user1" },
 "category": { "id": 1, "category_name": "Technology" },
 "comments": [
  { "id": 1, "comment_text": "Great post!", "CommentAuthor": { "id": 2, "username": "user2" } }
 ],
 "likers": [
  { "id": 2, "username": "user2" }
 ]
}
```

11. Update a Post by ID

- **Method:** PUT
- **Endpoint:** /posts/:id
- **Authentication:** Required (and ownership of the post)
- **Body:** (optional fields to update)
  - title: (string)
  - content: (string)
  - category_id: (integer)

- **Example Request:**

  PUT {{baseUrl}}/api/posts/{{postId}}
  Content-Type: application/json
  Authorization: Bearer {{authToken}}

  ```
  {
    "title": "Updated API Post Title",
    "content": "This is the updated content of my post.",
    "category_id": {{categoryId}}
  }
  ```

- **Example Response (Success):**

  ```
  {
    "message": "Post updated successfully."
  }
  ```

12. Delete a Post by ID

- **Method:** DELETE
- **Endpoint:** /posts/:id
- **Authentication:** Required (and ownership of the post)

- **Example Request:**

  DELETE {{baseUrl}}/api/posts/{{postId}}
  Authorization: Bearer {{authToken}}

- **Example Response (Success):**

```
{
  "message": "Post deleted successfully."
}
```

Comment Management

Users have the ability to **create**, **update**, and **delete** their own comments on posts.

13. Create a New Comment

- **Method:** POST
- **Endpoint:** /comments
- **Authentication:** Required
- **Body:**
  - comment_text: (string) The comment content.
  - post_id: (integer) The ID of the post to comment on.

- **Example Request:**

```
POST {{baseUrl}}/api/comments
Content-Type: application/json
Authorization: Bearer {{authToken}}

{
    "comment_text": "This is a post comment, from REST Client! Test User 1",
    "post_id": {{postId}}
}
```

- **Example Response (Success):**

```
{
  "id": 1,
  "comment_text": "This is a great post!",
  "post_id": 1,
  "user_id": 2
}
```

14. Update a Comment

- **Method:** PUT
- **Endpoint:** /comments/:id
- **Authentication:** Required (and ownership of the comment)
- **Body:**
  - comment_text: (string) The updated comment content.

- **Example Request:**

  PUT {{baseUrl}}/api/comments/{{commentId}}
  Content-Type: application/json
  Authorization: Bearer {{authToken}}

  ```
  {
    "comment_text": "This comment has been updated from the REST Client!"
  }
  ```

- **Example Response (Success):**

  ```
  {
    "message": "Comment updated successfully."
  }
  ```

15. Delete a Comment

- **Method:** DELETE
- **Endpoint:** /comments/:id
- **Authentication:** Required (and ownership of the comment)

- **Example Request:**

  DELETE {{baseUrl}}/api/comments/{{commentId}}
  Authorization: Bearer {{authToken}}

- **Example Response (Success):**

  ```
  {
    "message": "Comment deleted successfully."
  }
  ```

Like Management

The API includes endpoints for a user to **like** or **unlike** a post.

16. Like a Post

- **Method:** POST
- **Endpoint:** /likes
- **Authentication:** Required
- **Body:**
  - post_id: (integer) The ID of the post to like.

- **Example Request:**

  POST {{baseUrl}}/api/likes
  Content-Type: application/json
  Authorization: Bearer {{authToken}}

  ```
  {
    "post_id": {{postId}}
  }
  ```

- **Example Response (Success):**

  ```
  {
    "message": "Post liked successfully."
  }
  ```

17. Unlike a Post

- **Method:** DELETE
- **Endpoint:** /likes/:id
- **Authentication:** Required

- **Example Request:**

  DELETE {{baseUrl}}/api/likes/{{postId}}
  Authorization: Bearer {{authToken}}

- **Example Response (Success):**

  ```
  {
    "message": "Post unliked successfully."
  }
  ```

Follow Management

Users can **follow** and **unfollow** other users, which updates the follower count on the followed user's profile.

18. Follow a User

- **Method:** POST
- **Endpoint:** /follows
- **Authentication:** Required
- **Body:**
  - followed_id: (integer) The ID of the user to follow.

- **Example Request:**

  POST {{baseUrl}}/api/follows
  Content-Type: application/json
  Authorization: Bearer {{testUser2Token}}

  ```
  {
    "followed_id": {{registeredUserId}}
  }
  ```

- **Example Response (Success):**

  ```
  {
    "message": "User followed successfully."
  }
  ```

19. Unfollow a User

- **Method:** DELETE
- **Endpoint:** /follows/:id
- **Authentication:** Required

- **Example Request:**

  DELETE {{baseUrl}}/api/follows/{{registeredUserId}}
  Authorization: Bearer {{testUser2Token}}

- **Example Response (Success):**

  ```
  {
    "message": "User unfollowed successfully."
  }
  ```

Feed and Search

The API provides a personalized feed that shows posts from the users a person is following, as well as a search function to find posts by keyword.

20. Get Personalised Feed

- **Method:** GET
- **Endpoint:** /feed
- **Authentication:** Required

- **Example Request:**

  ```
  GET {{baseUrl}}/feed
  Authorization: Bearer {{testUser2Token}}
  ```

- **Example Response (Success):**

  ```
  [
    { "id": 1, "title": "Post from a user I follow...", "user": { "id": 2, "username": "user2" } },
    { "id": 2, "title": "My own post...", "user": { "id": 1, "username": "myuser" } }
  ]
  ```

21. Search for Posts

- **Method:** GET
- **Endpoint:** /search
- **Authentication:** Required
- **Query Parameter:** q (string) - The search term.

- **Example Request:**

  ```
  GET {{baseUrl}}/api/search?q=Web
  Authorization: Bearer {{testUser2Token}}
  ```

- **Example Response (Success):**

  ```
  [
    { "id": 1, "title": "A Searchable Title about Web Development", "content": "This post contains content about JavaScript and Node.js..." }
  ]
  ```

Test Plan

An API test plane is held in the main project folder at: micro_blog_api_test.http