Social Media Project-Twitter inspired

Github: https://github.com/skavvathas/Social-Media-App-React.js

The project has been developed using the Node.js environment.

Therefore, it is necessary for the user to have Node.js installed on their computer. Additionally, the project utilizes React.js for the frontend and Express.js for the backend.

The database that was used is MongoDB, where all user and post data is stored. It's important to note that CSS was used for the user interface design. Once the files are downloaded, the user can run the project locally. To do this, open a terminal in both the server and client folders and execute the command npm i in each of them. This will install all the required node modules.

Furthermore, the user needs to connect their MongoDB database in the application. This can be done by modifying the server.js file in the server folder. The specific cluster used by the editor is not provided for safety reasons.

In the project, JWT tokens are used for user authentication. To enable user authentication in the app, the user should create a file named .env. In this file, they should add a secret key by writing SECRET=<key> and specify the port as PORT=4000. The .env file used by the editor is not provided for obvious security reasons.

Once these configurations are complete, the user can run the application by typing npm start in both terminals. The server will be running on localhost:4000 and the client on localhost:3000.

The project is a simple Social Media application (Twitter based/inspired). The user can create his personalized profile by registering in the application. Also, if the user is logged out, he can login by giving the proper username and password. The user, inside the application, can post his text and an image, he can follow others users in order to see their post, and also he can unfollow them. Moreover, he can search the users of the app and personalize his profile with a profile photo.

server file:

We have 2 schemas: userSchema and postSchema. The userSchema is saved in the server/models/userModel.js file and the postSchema is saved in the server/models/postModel.js .

userSchema:

```
const userSchema = new mongoose.Schema({
  firstName: {
    type: String,
    required: true,
  },
  lastName: {
    type: String,
    required: true,
  },
  email: {
    type: String,
    required: true,
    unique: true
  },
  username:  {
    type: String,
    required: true,
  },
  password:  {
    type: String,
    required: true,
  },
  imageUrl: {
    type: String,
    default: "defaultUser.png"
  },
  following: [{ type: String }],
});
```

postSchema:

```
const postSchema = new mongoose.Schema({
    user_id: {
        type: String,
        required: true
    },
    username: {
        type: String,
        required: true
    },
    post: {
        type: String,
        required: true
    },
    likes: {
        type: Number,
        default: 0
    },
    userImage: {
        type: String,
        default: "defaultUser.png"
    },
    imageFile: {
        type: String,
        default: ""
    }
}, { timestamps: true });
```

APIs:

- **/api/posts:** All the routes for this API is in the server/routes/posts.js file:

    1. **/allPosts** : get all the posts from all the users. It uses the **getAllPostFromAllUsers** function from the **postController.js** file.

2. **/allFollowPosts** : get the post only from the users that you follow. It uses the **getAllPostFromTheFollowingUsers** function from **postController.js** file.

3. **/ (get)** : Get all posts from a specific user. It uses the **getPosts** function from the **postController.js** file.

4. **/:id (get)** : Get a single post. It uses the **getPost** function from the **postController.js** file.

5. **/ (post)**: Create a new post. It uses the **createPost** function from the **postController.js** file.

6. **/:id (delete)** : Delete a specific post. It uses the **deletePost** function from the **postController.js** file.

7. **/:id (updatePost)** : It uses the **updatePost** function from **postController.js** file.

8. **/user/:id** : Get the posts from a specific user, other from the currently logged in user. It uses the **getPostsByUsername** function from the **postController.js** file.

9. **/uplike/:id** : Increase the number of likes. It uses the **upLikes** function from the **postController.js** file.

10. **/downlike/:id** : Decrease the number of likes. It uses the **downLikes** function from the **postController.js** file.

11. **/edit** : Edit a post of the user logged in. It uses the **editPosts** function from the **postController.js** file.

- **/api/user :** All the routes for this API is in the server/routes/posts.js file:
    1. **/login** : Login of user in the application. It uses the **loginUser** from the file **userController.js** .

    2. **/register** : Register a new user in the application. It uses the **registerUser** from the file **userController.js**.

3. **/edit** : Edit the user with username: prevUsername. It uses the **editUser** from the file **userController.js** .

4. **/getUserByUsername** : Get the user with username: username. It uses the **getUserByUsernameUser** from the file **userController.js** .

- **/api/friends :** All the routes for this API is in the server/routes/posts.js file
    1. **/allusers** : Get the user with username: username. It uses the **allUsers** from the file **friendsController.js** .

    2. **/follow** : Follow the user with id: followingUserId. It uses the **followUser** from the file **friendsController.js** .

    3. **/unfollow** : Unfollow the user with id: followingUserId . It uses the **unfollowUser** from the file **friendsController.js** .

    4. **/isfollowing** :Check if the User with username: username follows the user with username: followingUser . It uses the **isFollowing** from the file **friendsController.js** .

    5. **/getfollowing** : Get all the users that the user follows. It uses the **getFollowing** from the file **friendsController.js** .

- **/api/uploaded :** All the routes for this API is in the server/routes/posts.js file
    1. /upload-image : It uses the **uploadFunc** from the file **uploadController.js**.

    2. /upload-image-post/:postId : It uses the **uploadFuncPost** from the file **uploadController.js**.

    3. /getImage : It uses the **getImage** from the file **uploadController.js**.

    4. /getImageById/:id : It uses the **getImageById** from the file **uploadController.js**.

client file:

App.js : This file includes the connections between the routes of the app and the pages that are displayed. The connections are:

- path="/" -> Select.js
- path="/login" -> Login.js
- path="/register" -> Register.js
- path="/home" -> Home.js
- path="/profile" -> Profile.js
- path="/friends" -> Followers.js
- path="/post" -> Compose.js
- path="/users" -> Users.js
- path="/followers" -> Followers.js
- path="/search" -> Search.js
- path="/home/:id" -> User.js
- path="/*" -> NoPage.js

The files in the **pages** folder :

- **Login.js** : The file has the code that is displayed in the "*/login"* route. Also, the ***useLogin*** hook is called when the credentials are submitted, in order to check if the user exists in the database.

- **Register.js** : The file has the code that is displayed in the "*/register*" route. The hook ***useRegister*** is called in order to create a new user.

- **Home.js** : The file has the code that is displayed in the "/*home*" route. The components ***Navbar***, ***SinglePost***, ***FollowBar*** are imported in the file. The user here can see basic information of his profile (the posts of other users, search bar, the navbar of the app).
  API api/posts/allFollowPosts route is called in order to return all the posts from the users that are being followed.

- **Profile.js** : The file has the code that is displayed in the "/*profile*" route. The user here sees his profile's basic information (image, username, name) and can update his information.

- **Users.js** : The file has the code that is displayed in the "/*users*" route. Here all the users of the app can be seen.

- **Compose.js** : The file has the code that is displayed in the "/*post*" route. Here the user can create his post. The file calls the api/posts API route in order to create the post in the database.

- **User.js**: The file has the code that is displayed in the "*/home/:id*" route. Here the user can see another user profile. The file calls the api/posts/user API route in order to get the posts of this specific user. The "*/:id*" part in the "*/home/:id*" give us the username of the user whose profile we want to display.

The files in the **components** folder :

- **FollowBar.js** : Here search bar displayed alongside with all the users that are in the database. /api/friends/allusers API route is called**.**

- **Navbar.js** : Includes the buttons that link the routes. Here the user has the chance to navigate ,through the buttons that are displayed, in the whole app. Also includes a logout button.

- **SearchBar.js**: Here the user can search for a different user. /api/friends/allusers API route is called for this reason.

- **User.js** : This file includes the information that makes a user special. For example, it shows the image of the user (/api/uploaded/image is called) and the name of the user. Also displays a button for follow and unfollow of the user.
- **SinglePost.js** : This file contains the implementation (text, image, likes) of the UI of every post.

The files in the **context** folder :
- **AuthContext.js** : This code defines a React context named AuthContext and a context provider component named AuthContextProvider.