API DOCUMENTATION

1. Overview

Develop a social media application with the following features :

a. User sign-up
b. User login
c. Send friend request
d. Accept or reject friend request
e. Create a post (text max 256 characters)
f. Like post
g. List posts

Creating a Restful Api for each feature.
Using a MongoDb Database for storing User information , Post created by Users.
MongoDb database consists of two collections:

- User(name ,email ,password ,postIds[ ] ,friends[ ] ,friendRequestSent[ ] ,friendRequestReceived[ ])
- Post(date, content ,likes[ ])

2. Base URL

https://www.application.com/

3. Authentication

Using JWT tokens for authentication and authorisation.

4. Endpoints

a. User sign-up

URL : /sign-up
Method : POST
Request Header :

```
{
        token
 }
```

Request Body :

```
{
        "email" : ommankar@gmail.com ,
        "name" : Om mankar ,
        "password" : pass@word
}
```

Response Body :

```
{
      "Message" : msg depending on controller written ,
}
```

Status Code : 200 (Status Ok) , 400 (Bad Request) , 500 (Internal Server Error).

Description :
The user will send the email, name, and password in the request body to the server in JSON format. On the server side, we will use middleware to parse the JSON data. Next, we will validate the email address format and the password. Next, we will check whether the user already exists. If it doesn't exist, we will hash the password and store user details in the database. We will generate a JWT token, which we will send in res.cookie() to the client.

---

b. User login

URL : /login
Method : POST
Request Header :

```
{
      token
}
```

Request Body :

```
{
    "email" : ommankar@gmail.com ,
    "password" : pass@word
}
```

Response Body :

```
{
  "Message" : msg depending on controller written ,
  "Data" : {
            userId : [generated by db],
            name : [received form db],
            email : [received from db],
            postIds : [Array received from db],
            friends : [Array received from db],
            freindRequestSend : [Array received from db]
            friendRequestReceived[ ] :[Array received
            from db]
          }
}
```

Status Code : 200 (Status Ok) , 400 (Bad Request) , 500 (Internal Server Error).

Description :
The user will send an email and password in the request body to the server in JSON format. On the server side we will use middleware to parse the JSON data, will use middleware to authorize the user by checking the token, Next we will first validate the email address format, next we will check whether the user exists or not, if it doesn't exist then will send a response message user doesn’t have an account. Otherwise, we will send the user details in the response body to the client side. We will generate a JWT token, which we will send in res.cookie() to the client.

---

c. *Send friend request*

URL : /sendRequest
Method : POST
Request Header :

```
{
        token
 }
```

Request Body :

```
{
      “userId” : “userId”,
      “userIdOfFreind” : “id”
}
```

Response Body :

```
{
  “Message” : msg depending on controller written ,
}
```

Status Code : 200 (Status Ok) , 400 (Bad Request) , 500 (Internal Server Error).

Description :
The user will add his user ID and friend’s user ID in the request body to the server in JSON format. On the server side, we will use middleware to parse the JSON data, next use middleware to authorize the user by checking the token, then we will update the user’s freindRequestSend[] by appending the friend’s userId, and if successfully updated the user document will send the friend’s user Id to the client to render it in frontend.

d. *Accept or reject friend request*

URL : /requestStatus
Method : POST
Request Header :
```
{
        token
 }
```
Request Body :
```
{
      “userId” : “userId”,
      “userIdOfFreind” : “id”,
      “status” : boolean
}
```
Response Body :
```
{
  “Message” : msg depending on controller written ,
}
```

Status Code : 200 (Status Ok) , 400 (Bad Request) , 500 (Internal Server Error).

Description :
The user will add his user and friend’s user ID and status in the request body to the server in JSON format. On the server side we will use middleware to parse the JSON data, will use middleware to authorize the user by checking the token, Next depending on the status attribute we will update the user’s freindRequestReiceved[] and freinds[], and if successfully updated the user document will send the status to the client to render it in frontend.

e. *Create a post (text max 256 characters)*

URL : /createPost
Method : POST
Request Header :
```
{
        token
 }
```
Request Body :
```
{
      “userId” : “userId”,
      “post” : {
                content : “”
             }
}
```

Response Body :
```
{
  "Message" : msg depending on controller written,
}
```

Status Code : 200 (Status Ok) , 400 (Bad Request) , 500 (Internal Server Error).

Description :
The user will add his userId and Post content in the request body to the server in JSON format. On the server side we will use middleware to parse the JSON data, will use middleware to authorize the user by checking the token, Next will create a new document in the post collection in the database,first will append the post id to user's postId array and then will send the status to the client to render it in the frontend.

f. *Like post*

URL : /likePost
Method : POST
Request Header :{
```
    token
}
```
Request Body :
```
{
    "userId" : "userId",
    "postId" : "postId",
}
```
Response Body :
```
{
  "Message" : msg depending on controller written ,
}
```

Status Code : 200 (Status Ok) , 400 (Bad Request) , 500 (Internal Server Error).

Description :
The user will add his userId and PostId in the request body to the server in json format. On the server side we will use middleware to parse the json data, will use a middleware to authorize the user by checking the token, Next will append the userId to the Like array of the post document corresponding to the postId user then will send the status to the client to render it in the frontend.

g. *List posts*

URL : /listPost

Method : POST

Request Header :{
```
                token
            }
```
Request Body :{
```
                userID
            }
```

Response Body :
```
    {
      "Message" : msg depending on controller written ,
      Data :{
             }
      "pagination": {
           "total_records": ,
           "current_page": ,
           "total_pages": ,
           "next_page": ,
           "prev_page":
         }
    }
```

Status Code : 200 (Status Ok) , 400 (Bad Request) , 500 (Internal Server Error).

Description :
The user will add his userId in the request body so as to fetch posts made by that user. On the server side we will use a middleware to authorize the user by checking the token , Next will send all the posts corresponding to the postId stored in the user document to the client to render on frontend.

5. Validation

- email Id :
  - Validate the format of email Id.
- Password
  - limited to 8 characters
  - Should contain special characters and numbers

6. Security

Using jwt tokens for authorisation.
Using https for secure transfer of data over the network.

7. Database Design

Create two collection : User , Post

User : { name , email , password , postIds[ ] , friends[ ] , friendRequestSent[ ] , friendRequestReceived[ ] }	Post : { date, content , likes[ ] }