To test the **Social Media Application** REST API using **Postman**, follow the steps below. I'll provide examples for each module, including sample requests and expected responses.

**1. Start the Spring Boot Application**

* Run the SocialMediaApplication.java class to start the Spring Boot application.
* Ensure the application is running on the default port 8080.

**2. Install and Open Postman**

* Download and install Postman from [here](https://www.postman.com/downloads/).
* Open Postman and create a new workspace (optional).

**3. Test Each Endpoint with Sample Data**

Below are examples of how to test each endpoint using Postman. Replace {id} with actual IDs from your database.

**A. Users**

1. **Create a User**
   * **Method**: POST
   * **URL**: http://localhost:8080/api/users
   * **Body (Raw - JSON)**:

json

Copy

{

"username": "john\_doe",

"password": "password123",

"email": "john.doe@example.com",

"profilePicture": "base64\_encoded\_image"

}

* + **Response**: The created user with an ID.

1. **Get All Users**
   * **Method**: GET
   * **URL**: http://localhost:8080/api/users
   * **Response**: List of all users.
2. **Get User by ID**
   * **Method**: GET
   * **URL**: http://localhost:8080/api/users/{id}
   * **Response**: The user with the specified ID.
3. **Update a User**
   * **Method**: PUT
   * **URL**: http://localhost:8080/api/users/{id}
   * **Body (Raw - JSON)**:

json

Copy

{

"username": "jane\_doe",

"password": "newpassword123",

"email": "jane.doe@example.com",

"profilePicture": "new\_base64\_encoded\_image"

}

* + **Response**: The updated user.

1. **Delete a User**
   * **Method**: DELETE
   * **URL**: http://localhost:8080/api/users/{id}
   * **Response**: HTTP 204 (No Content).

**B. Posts**

1. **Create a Post**
   * **Method**: POST
   * **URL**: http://localhost:8080/api/posts
   * **Body (Raw - JSON)**:

json

Copy

{

"content": "This is my first post!",

"user": {

"userID": 1

}

}

* + **Response**: The created post with an ID.

1. **Get All Posts**
   * **Method**: GET
   * **URL**: http://localhost:8080/api/posts
   * **Response**: List of all posts.
2. **Get Post by ID**
   * **Method**: GET
   * **URL**: http://localhost:8080/api/posts/{id}
   * **Response**: The post with the specified ID.
3. **Update a Post**
   * **Method**: PUT
   * **URL**: http://localhost:8080/api/posts/{id}
   * **Body (Raw - JSON)**:

json

Copy

{

"content": "This is an updated post!",

"user": {

"userID": 1

}

}

* + **Response**: The updated post.

1. **Delete a Post**
   * **Method**: DELETE
   * **URL**: http://localhost:8080/api/posts/{id}
   * **Response**: HTTP 204 (No Content).

**C. Comments**

1. **Create a Comment**
   * **Method**: POST
   * **URL**: http://localhost:8080/api/comments
   * **Body (Raw - JSON)**:

json

Copy

{

"commentText": "Great post!",

"post": {

"postID": 1

},

"user": {

"userID": 1

}

}

* + **Response**: The created comment with an ID.

1. **Get All Comments**
   * **Method**: GET
   * **URL**: http://localhost:8080/api/comments
   * **Response**: List of all comments.
2. **Get Comment by ID**
   * **Method**: GET
   * **URL**: http://localhost:8080/api/comments/{id}
   * **Response**: The comment with the specified ID.
3. **Update a Comment**
   * **Method**: PUT
   * **URL**: http://localhost:8080/api/comments/{id}
   * **Body (Raw - JSON)**:

json

Copy

{

"commentText": "Updated comment!",

"post": {

"postID": 1

},

"user": {

"userID": 1

}

}

* + **Response**: The updated comment.

1. **Delete a Comment**
   * **Method**: DELETE
   * **URL**: http://localhost:8080/api/comments/{id}
   * **Response**: HTTP 204 (No Content).

**D. Likes**

1. **Create a Like**
   * **Method**: POST
   * **URL**: http://localhost:8080/api/likes
   * **Body (Raw - JSON)**:

json

Copy

{

"post": {

"postID": 1

},

"user": {

"userID": 1

}

}

* + **Response**: The created like with an ID.

1. **Get All Likes**
   * **Method**: GET
   * **URL**: http://localhost:8080/api/likes
   * **Response**: List of all likes.
2. **Get Like by ID**
   * **Method**: GET
   * **URL**: http://localhost:8080/api/likes/{id}
   * **Response**: The like with the specified ID.
3. **Delete a Like**
   * **Method**: DELETE
   * **URL**: http://localhost:8080/api/likes/{id}
   * **Response**: HTTP 204 (No Content).

**E. Groups**

1. **Create a Group**
   * **Method**: POST
   * **URL**: http://localhost:8080/api/groups
   * **Body (Raw - JSON)**:

json

Copy

{

"groupName": "Developers",

"admin": {

"userID": 1

}

}

* + **Response**: The created group with an ID.

1. **Get All Groups**
   * **Method**: GET
   * **URL**: http://localhost:8080/api/groups
   * **Response**: List of all groups.
2. **Get Group by ID**
   * **Method**: GET
   * **URL**: http://localhost:8080/api/groups/{id}
   * **Response**: The group with the specified ID.
3. **Update a Group**
   * **Method**: PUT
   * **URL**: http://localhost:8080/api/groups/{id}
   * **Body (Raw - JSON)**:

json

Copy

{

"groupName": "Updated Group Name",

"admin": {

"userID": 1

}

}

* + **Response**: The updated group.

1. **Delete a Group**
   * **Method**: DELETE
   * **URL**: http://localhost:8080/api/groups/{id}
   * **Response**: HTTP 204 (No Content).

**F. Friends**

1. **Create a Friend**
   * **Method**: POST
   * **URL**: http://localhost:8080/api/friends
   * **Body (Raw - JSON)**:

json

Copy

{

"user": {

"userID": 1

},

"status": "PENDING"

}

* + **Response**: The created friend with an ID.

1. **Get All Friends**
   * **Method**: GET
   * **URL**: http://localhost:8080/api/friends
   * **Response**: List of all friends.
2. **Get Friend by ID**
   * **Method**: GET
   * **URL**: http://localhost:8080/api/friends/{id}
   * **Response**: The friend with the specified ID.
3. **Update a Friend**
   * **Method**: PUT
   * **URL**: http://localhost:8080/api/friends/{id}
   * **Body (Raw - JSON)**:

json

Copy

{

"user": {

"userID": 1

},

"status": "ACCEPTED"

}

* + **Response**: The updated friend.

1. **Delete a Friend**
   * **Method**: DELETE
   * **URL**: http://localhost:8080/api/friends/{id}
   * **Response**: HTTP 204 (No Content).

**G. Notifications**

1. **Create a Notification**
   * **Method**: POST
   * **URL**: http://localhost:8080/api/notifications
   * **Body (Raw - JSON)**:

json

Copy

{

"content": "You have a new friend request!",

"user": {

"userID": 1

}

}

* + **Response**: The created notification with an ID.

1. **Get All Notifications**
   * **Method**: GET
   * **URL**: http://localhost:8080/api/notifications
   * **Response**: List of all notifications.
2. **Get Notification by ID**
   * **Method**: GET
   * **URL**: http://localhost:8080/api/notifications/{id}
   * **Response**: The notification with the specified ID.
3. **Delete a Notification**
   * **Method**: DELETE
   * **URL**: http://localhost:8080/api/notifications/{id}
   * **Response**: HTTP 204 (No Content).

**H. Messages**

1. **Create a Message**
   * **Method**: POST
   * **URL**: http://localhost:8080/api/messages
   * **Body (Raw - JSON)**:

json

Copy

{

"messageText": "Hello!",

"sender": {

"userID": 1

},

"receiver": {

"userID": 2

}

}

* + **Response**: The created message with an ID.

1. **Get All Messages**
   * **Method**: GET
   * **URL**: http://localhost:8080/api/messages
   * **Response**: List of all messages.
2. **Get Message by ID**
   * **Method**: GET
   * **URL**: http://localhost:8080/api/messages/{id}
   * **Response**: The message with the specified ID.
3. **Delete a Message**
   * **Method**: DELETE
   * **URL**: http://localhost:8080/api/messages/{id}
   * **Response**: HTTP 204 (No Content).

**4. Verify Responses**

* Check the **Status Code** (e.g., 200 for success, 201 for created, 204 for deleted).
* Verify the **Response Body** for the expected data.

**5. Save Requests in Postman**

* Save each request in a Postman collection for future use.

**Database Schema**

**1. Users Table**

Stores user information.

sql

Copy

CREATE TABLE users (

userID INT AUTO\_INCREMENT PRIMARY KEY,

username VARCHAR(255) NOT NULL UNIQUE,

password VARCHAR(255) NOT NULL,

email VARCHAR(255) NOT NULL UNIQUE,

profile\_picture BLOB

);

**2. Posts Table**

Stores posts created by users.

sql

Copy

CREATE TABLE posts (

postID INT AUTO\_INCREMENT PRIMARY KEY,

userID INT NOT NULL,

content TEXT NOT NULL,

postTimestamp TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (userID) REFERENCES users(userID) ON DELETE CASCADE

);

**3. Comments Table**

Stores comments on posts.

sql

Copy

CREATE TABLE comments (

commentID INT AUTO\_INCREMENT PRIMARY KEY,

postID INT NOT NULL,

userID INT NOT NULL,

comment\_text TEXT NOT NULL,

commentTimestamp TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (postID) REFERENCES posts(postID) ON DELETE CASCADE,

FOREIGN KEY (userID) REFERENCES users(userID) ON DELETE CASCADE

);

**4. Likes Table**

Stores likes on posts.

sql

Copy

CREATE TABLE likes (

likeID INT AUTO\_INCREMENT PRIMARY KEY,

postID INT NOT NULL,

userID INT NOT NULL,

likeTimestamp TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (postID) REFERENCES posts(postID) ON DELETE CASCADE,

FOREIGN KEY (userID) REFERENCES users(userID) ON DELETE CASCADE

);

**5. Groups Table**

Stores groups and their admin.

sql

Copy

CREATE TABLE groups (

groupID INT AUTO\_INCREMENT PRIMARY KEY,

groupName VARCHAR(255) NOT NULL,

adminID INT NOT NULL,

FOREIGN KEY (adminID) REFERENCES users(userID) ON DELETE CASCADE

);

**6. Friends Table**

Stores friend relationships between users.

sql

Copy

CREATE TABLE friends (

friendID INT AUTO\_INCREMENT PRIMARY KEY,

userID INT NOT NULL,

friendUserID INT NOT NULL,

status ENUM('PENDING', 'ACCEPTED', 'REJECTED') DEFAULT 'PENDING',

FOREIGN KEY (userID) REFERENCES users(userID) ON DELETE CASCADE,

FOREIGN KEY (friendUserID) REFERENCES users(userID) ON DELETE CASCADE

);

**7. Notifications Table**

Stores notifications for users.

sql

Copy

CREATE TABLE notifications (

notificationID INT AUTO\_INCREMENT PRIMARY KEY,

userID INT NOT NULL,

content TEXT NOT NULL,

notificationTimestamp TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (userID) REFERENCES users(userID) ON DELETE CASCADE

);

**8. Messages Table**

Stores messages between users.

sql

Copy

CREATE TABLE messages (

messageID INT AUTO\_INCREMENT PRIMARY KEY,

senderID INT NOT NULL,

receiverID INT NOT NULL,

message\_text TEXT NOT NULL,

messageTimestamp TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (senderID) REFERENCES users(userID) ON DELETE CASCADE,

FOREIGN KEY (receiverID) REFERENCES users(userID) ON DELETE CASCADE

);

**Relationships Between Tables**

1. **Users → Posts**: A user can create many posts (users.userID → posts.userID).
2. **Users → Comments**: A user can write many comments (users.userID → comments.userID).
3. **Users → Likes**: A user can like many posts (users.userID → likes.userID).
4. **Users → Groups**: A user can be an admin of many groups (users.userID → groups.adminID).
5. **Users → Friends**: A user can have many friends (users.userID → friends.userID and friends.friendUserID).
6. **Users → Notifications**: A user can have many notifications (users.userID → notifications.userID).
7. **Users → Messages**: A user can send and receive many messages (users.userID → messages.senderID and messages.receiverID).
8. **Posts → Comments**: A post can have many comments (posts.postID → comments.postID).
9. **Posts → Likes**: A post can have many likes (posts.postID → likes.postID).

**Sample Data**

**Users Table**

sql

Copy

INSERT INTO users (username, password, email) VALUES

('john\_doe', 'password123', 'john.doe@example.com'),

('jane\_doe', 'password456', 'jane.doe@example.com');

**Posts Table**

sql

Copy

INSERT INTO posts (userID, content) VALUES

(1, 'This is my first post!'),

(2, 'Hello, world!');

**Comments Table**

sql

Copy

INSERT INTO comments (postID, userID, comment\_text) VALUES

(1, 2, 'Great post!'),

(2, 1, 'Nice!');

**Likes Table**

sql

Copy

INSERT INTO likes (postID, userID) VALUES

(1, 2),

(2, 1);

**Groups Table**

sql

Copy

INSERT INTO groups (groupName, adminID) VALUES

('Developers', 1),

('Designers', 2);

**Friends Table**

sql

Copy

INSERT INTO friends (userID, friendUserID, status) VALUES

(1, 2, 'ACCEPTED'),

(2, 1, 'ACCEPTED');

**Notifications Table**

sql

Copy

INSERT INTO notifications (userID, content) VALUES

(1, 'You have a new friend request!'),

(2, 'Your post was liked!');

**Messages Table**

sql

Copy

INSERT INTO messages (senderID, receiverID, message\_text) VALUES

(1, 2, 'Hello Jane!'),

(2, 1, 'Hi John!');

**Database Diagram**

Here’s a simplified ERD (Entity-Relationship Diagram) for the schema:

Copy

+------------+ +------------+ +------------+

| Users | | Posts | | Comments |

+------------+ +------------+ +------------+

| userID (PK)|<------| userID (FK)|<------| userID (FK)|

| username | | postID (PK)| | commentID |

| password | | content | | commentText|

| email | | timestamp | | timestamp |

| profilePic | +------------+ +------------+

+------------+

| +------------+ +------------+

+----------| Likes | | Messages |

| +------------+ +------------+

| | likeID (PK)| | messageID |

| | postID (FK)| | senderID |

| | userID (FK)| | receiverID |

| | timestamp | | messageText|

| +------------+ | timestamp |

| +------------+

|

| +------------+ +------------+

+----------| Friends | | Notifications|

| +------------+ +------------+

| | friendID | | notificationID|

| | userID (FK)| | userID (FK) |

| | friendUserID| | content |

| | status | | timestamp |

| +------------+ +------------+

|

| +------------+

+----------| Groups |

+------------+

| groupID (PK)|

| groupName |

| adminID (FK)|

+------------+

Angular Code

Below is the **Angular** code for the **Social Media Application** that integrates with the Spring Boot REST API endpoints. The Angular application will have components, services, and models for each module.

**1. Angular Project Setup**

1. **Install Angular CLI** (if not already installed):

bash

Copy

npm install -g @angular/cli

1. **Create a new Angular project**:

bash

Copy

ng new social-media-app

cd social-media-app

1. **Generate Components and Services**:

bash

Copy

ng generate component users

ng generate component posts

ng generate component comments

ng generate component likes

ng generate component groups

ng generate component friends

ng generate component notifications

ng generate component messages

ng generate service services/user

ng generate service services/post

ng generate service services/comment

ng generate service services/like

ng generate service services/group

ng generate service services/friend

ng generate service services/notification

ng generate service services/message

1. **Install HttpClientModule**:
   * Open src/app/app.module.ts and add HttpClientModule:

typescript

Copy

import { HttpClientModule } from '@angular/common/http';

@NgModule({

imports: [

HttpClientModule

]

})

export class AppModule { }

**2. Models**

Create models for each module in the src/app/models folder.

**User Model**

typescript

Copy

export interface User {

userID: number;

username: string;

password: string;

email: string;

profilePicture: string; // Base64 encoded image

}

**Post Model**

typescript

Copy

export interface Post {

postID: number;

userID: number;

content: string;

postTimestamp: string;

}

**Comment Model**

typescript

Copy

export interface Comment {

commentID: number;

postID: number;

userID: number;

commentText: string;

commentTimestamp: string;

}

**Like Model**

typescript

Copy

export interface Like {

likeID: number;

postID: number;

userID: number;

likeTimestamp: string;

}

**Group Model**

typescript

Copy

export interface Group {

groupID: number;

groupName: string;

adminID: number;

}

**Friend Model**

typescript

Copy

export interface Friend {

friendID: number;

userID: number;

friendUserID: number;

status: 'PENDING' | 'ACCEPTED' | 'REJECTED';

}

**Notification Model**

typescript

Copy

export interface Notification {

notificationID: number;

userID: number;

content: string;

notificationTimestamp: string;

}

**Message Model**

typescript

Copy

export interface Message {

messageID: number;

senderID: number;

receiverID: number;

messageText: string;

messageTimestamp: string;

}

**3. Services**

Create services to interact with the backend API.

**User Service**

typescript

Copy

import { Injectable } from '@angular/core';

import { HttpClient } from '@angular/common/http';

import { Observable } from 'rxjs';

import { User } from '../models/user.model';

@Injectable({

providedIn: 'root'

})

export class UserService {

private apiUrl = 'http://localhost:8080/api/users';

constructor(private http: HttpClient) {}

getAllUsers(): Observable<User[]> {

return this.http.get<User[]>(this.apiUrl);

}

createUser(user: User): Observable<User> {

return this.http.post<User>(this.apiUrl, user);

}

updateUser(userID: number, user: User): Observable<User> {

return this.http.put<User>(`${this.apiUrl}/${userID}`, user);

}

deleteUser(userID: number): Observable<void> {

return this.http.delete<void>(`${this.apiUrl}/${userID}`);

}

}

**Post Service**

typescript

Copy

import { Injectable } from '@angular/core';

import { HttpClient } from '@angular/common/http';

import { Observable } from 'rxjs';

import { Post } from '../models/post.model';

@Injectable({

providedIn: 'root'

})

export class PostService {

private apiUrl = 'http://localhost:8080/api/posts';

constructor(private http: HttpClient) {}

getAllPosts(): Observable<Post[]> {

return this.http.get<Post[]>(this.apiUrl);

}

createPost(post: Post): Observable<Post> {

return this.http.post<Post>(this.apiUrl, post);

}

updatePost(postID: number, post: Post): Observable<Post> {

return this.http.put<Post>(`${this.apiUrl}/${postID}`, post);

}

deletePost(postID: number): Observable<void> {

return this.http.delete<void>(`${this.apiUrl}/${postID}`);

}

}

**Comment Service**

typescript

Copy

import { Injectable } from '@angular/core';

import { HttpClient } from '@angular/common/http';

import { Observable } from 'rxjs';

import { Comment } from '../models/comment.model';

@Injectable({

providedIn: 'root'

})

export class CommentService {

private apiUrl = 'http://localhost:8080/api/comments';

constructor(private http: HttpClient) {}

getAllComments(): Observable<Comment[]> {

return this.http.get<Comment[]>(this.apiUrl);

}

createComment(comment: Comment): Observable<Comment> {

return this.http.post<Comment>(this.apiUrl, comment);

}

updateComment(commentID: number, comment: Comment): Observable<Comment> {

return this.http.put<Comment>(`${this.apiUrl}/${commentID}`, comment);

}

deleteComment(commentID: number): Observable<void> {

return this.http.delete<void>(`${this.apiUrl}/${commentID}`);

}

}

**Like Service**

typescript

Copy

import { Injectable } from '@angular/core';

import { HttpClient } from '@angular/common/http';

import { Observable } from 'rxjs';

import { Like } from '../models/like.model';

@Injectable({

providedIn: 'root'

})

export class LikeService {

private apiUrl = 'http://localhost:8080/api/likes';

constructor(private http: HttpClient) {}

getAllLikes(): Observable<Like[]> {

return this.http.get<Like[]>(this.apiUrl);

}

createLike(like: Like): Observable<Like> {

return this.http.post<Like>(this.apiUrl, like);

}

deleteLike(likeID: number): Observable<void> {

return this.http.delete<void>(`${this.apiUrl}/${likeID}`);

}

}

**Group Service**

typescript

Copy

import { Injectable } from '@angular/core';

import { HttpClient } from '@angular/common/http';

import { Observable } from 'rxjs';

import { Group } from '../models/group.model';

@Injectable({

providedIn: 'root'

})

export class GroupService {

private apiUrl = 'http://localhost:8080/api/groups';

constructor(private http: HttpClient) {}

getAllGroups(): Observable<Group[]> {

return this.http.get<Group[]>(this.apiUrl);

}

createGroup(group: Group): Observable<Group> {

return this.http.post<Group>(this.apiUrl, group);

}

updateGroup(groupID: number, group: Group): Observable<Group> {

return this.http.put<Group>(`${this.apiUrl}/${groupID}`, group);

}

deleteGroup(groupID: number): Observable<void> {

return this.http.delete<void>(`${this.apiUrl}/${groupID}`);

}

}

**Friend Service**

typescript

Copy

import { Injectable } from '@angular/core';

import { HttpClient } from '@angular/common/http';

import { Observable } from 'rxjs';

import { Friend } from '../models/friend.model';

@Injectable({

providedIn: 'root'

})

export class FriendService {

private apiUrl = 'http://localhost:8080/api/friends';

constructor(private http: HttpClient) {}

getAllFriends(): Observable<Friend[]> {

return this.http.get<Friend[]>(this.apiUrl);

}

createFriend(friend: Friend): Observable<Friend> {

return this.http.post<Friend>(this.apiUrl, friend);

}

updateFriend(friendID: number, friend: Friend): Observable<Friend> {

return this.http.put<Friend>(`${this.apiUrl}/${friendID}`, friend);

}

deleteFriend(friendID: number): Observable<void> {

return this.http.delete<void>(`${this.apiUrl}/${friendID}`);

}

}

**Notification Service**

typescript

Copy

import { Injectable } from '@angular/core';

import { HttpClient } from '@angular/common/http';

import { Observable } from 'rxjs';

import { Notification } from '../models/notification.model';

@Injectable({

providedIn: 'root'

})

export class NotificationService {

private apiUrl = 'http://localhost:8080/api/notifications';

constructor(private http: HttpClient) {}

getAllNotifications(): Observable<Notification[]> {

return this.http.get<Notification[]>(this.apiUrl);

}

createNotification(notification: Notification): Observable<Notification> {

return this.http.post<Notification>(this.apiUrl, notification);

}

deleteNotification(notificationID: number): Observable<void> {

return this.http.delete<void>(`${this.apiUrl}/${notificationID}`);

}

}

**Message Service**

typescript

Copy

import { Injectable } from '@angular/core';

import { HttpClient } from '@angular/common/http';

import { Observable } from 'rxjs';

import { Message } from '../models/message.model';

@Injectable({

providedIn: 'root'

})

export class MessageService {

private apiUrl = 'http://localhost:8080/api/messages';

constructor(private http: HttpClient) {}

getAllMessages(): Observable<Message[]> {

return this.http.get<Message[]>(this.apiUrl);

}

createMessage(message: Message): Observable<Message> {

return this.http.post<Message>(this.apiUrl, message);

}

deleteMessage(messageID: number): Observable<void> {

return this.http.delete<void>(`${this.apiUrl}/${messageID}`);

}

}

**4. Components**

**Users Component**

typescript

Copy

import { Component, OnInit } from '@angular/core';

import { UserService } from '../services/user.service';

import { User } from '../models/user.model';

@Component({

selector: 'app-users',

templateUrl: './users.component.html',

styleUrls: ['./users.component.css']

})

export class UsersComponent implements OnInit {

users: User[] = [];

constructor(private userService: UserService) {}

ngOnInit(): void {

this.userService.getAllUsers().subscribe(data => {

this.users = data;

});

}

}

**Users Component Template**

html

Copy

<h2>Users</h2>

<ul>

<li \*ngFor="let user of users">

{{ user.username }} - {{ user.email }}

</li>

</ul>

Run HTML

**5. Routing**

Add routes for each module in src/app/app-routing.module.ts:

typescript

Copy

const routes: Routes = [

{ path: 'users', component: UsersComponent },

{ path: 'posts', component: PostsComponent },

{ path: 'comments', component: CommentsComponent },

{ path: 'likes', component: LikesComponent },

{ path: 'groups', component: GroupsComponent },

{ path: 'friends', component: FriendsComponent },

{ path: 'notifications', component: NotificationsComponent },

{ path: 'messages', component: MessagesComponent }

];

**6. Run the Application**

1. Start the Spring Boot backend.
2. Run the Angular application:

bash

Copy

ng serve

1. Open http://localhost:4200 in your browser.

-------------------using boot strap

Below is the **Angular** code for the **Social Media Application** integrated with **Bootstrap** for styling. The application includes components, services, and models for each module, and it uses Bootstrap for a responsive and modern UI.

**1. Install Bootstrap**

1. **Install Bootstrap**:

bash

Copy

npm install bootstrap

1. **Add Bootstrap CSS**:
   * Open angular.json and add the Bootstrap CSS file to the styles array:

json

Copy

"styles": [

"src/styles.css",

"node\_modules/bootstrap/dist/css/bootstrap.min.css"

]

1. **Install Bootstrap Icons** (optional):

bash

Copy

npm install bootstrap-icons

* + Add Bootstrap Icons to angular.json:

json

Copy

"styles": [

"src/styles.css",

"node\_modules/bootstrap/dist/css/bootstrap.min.css",

"node\_modules/bootstrap-icons/font/bootstrap-icons.css"

]

**2. Update Components with Bootstrap**

**Users Component Template**

html

Copy

<div class="container mt-5">

<h2 class="mb-4">Users</h2>

<table class="table table-striped">

<thead>

<tr>

<th>ID</th>

<th>Username</th>

<th>Email</th>

<th>Actions</th>

</tr>

</thead>

<tbody>

<tr \*ngFor="let user of users">

<td>{{ user.userID }}</td>

<td>{{ user.username }}</td>

<td>{{ user.email }}</td>

<td>

<button class="btn btn-primary btn-sm me-2">Edit</button>

<button class="btn btn-danger btn-sm">Delete</button>

</td>

</tr>

</tbody>

</table>

</div>

Run HTML

**Posts Component Template**

html

Copy

<div class="container mt-5">

<h2 class="mb-4">Posts</h2>

<div class="card mb-3" \*ngFor="let post of posts">

<div class="card-body">

<h5 class="card-title">Post ID: {{ post.postID }}</h5>

<p class="card-text">{{ post.content }}</p>

<small class="text-muted">Posted by User ID: {{ post.userID }}</small>

</div>

</div>

</div>

Run HTML

**Comments Component Template**

html

Copy

<div class="container mt-5">

<h2 class="mb-4">Comments</h2>

<div class="card mb-3" \*ngFor="let comment of comments">

<div class="card-body">

<h5 class="card-title">Comment ID: {{ comment.commentID }}</h5>

<p class="card-text">{{ comment.commentText }}</p>

<small class="text-muted">Posted by User ID: {{ comment.userID }}</small>

</div>

</div>

</div>

Run HTML

**Likes Component Template**

html

Copy

<div class="container mt-5">

<h2 class="mb-4">Likes</h2>

<ul class="list-group">

<li class="list-group-item" \*ngFor="let like of likes">

Like ID: {{ like.likeID }} - Post ID: {{ like.postID }} - User ID: {{ like.userID }}

</li>

</ul>

</div>

Run HTML

**Groups Component Template**

html

Copy

<div class="container mt-5">

<h2 class="mb-4">Groups</h2>

<table class="table table-striped">

<thead>

<tr>

<th>ID</th>

<th>Group Name</th>

<th>Admin ID</th>

<th>Actions</th>

</tr>

</thead>

<tbody>

<tr \*ngFor="let group of groups">

<td>{{ group.groupID }}</td>

<td>{{ group.groupName }}</td>

<td>{{ group.adminID }}</td>

<td>

<button class="btn btn-primary btn-sm me-2">Edit</button>

<button class="btn btn-danger btn-sm">Delete</button>

</td>

</tr>

</tbody>

</table>

</div>

Run HTML

**Friends Component Template**

html

Copy

<div class="container mt-5">

<h2 class="mb-4">Friends</h2>

<ul class="list-group">

<li class="list-group-item" \*ngFor="let friend of friends">

Friend ID: {{ friend.friendID }} - User ID: {{ friend.userID }} - Status: {{ friend.status }}

</li>

</ul>

</div>

Run HTML

**Notifications Component Template**

html

Copy

<div class="container mt-5">

<h2 class="mb-4">Notifications</h2>

<div class="card mb-3" \*ngFor="let notification of notifications">

<div class="card-body">

<h5 class="card-title">Notification ID: {{ notification.notificationID }}</h5>

<p class="card-text">{{ notification.content }}</p>

<small class="text-muted">User ID: {{ notification.userID }}</small>

</div>

</div>

</div>

Run HTML

**Messages Component Template**

html

Copy

<div class="container mt-5">

<h2 class="mb-4">Messages</h2>

<div class="card mb-3" \*ngFor="let message of messages">

<div class="card-body">

<h5 class="card-title">Message ID: {{ message.messageID }}</h5>

<p class="card-text">{{ message.messageText }}</p>

<small class="text-muted">From: {{ message.senderID }} | To: {{ message.receiverID }}</small>

</div>

</div>

</div>

Run HTML

**3. Add Navigation Bar**

**Navigation Component**

1. Generate a navigation component:

bash

Copy

ng generate component navigation

1. Update navigation.component.html:

html

Copy

<nav class="navbar navbar-expand-lg navbar-dark bg-dark">

<div class="container-fluid">

<a class="navbar-brand" href="#">Social Media App</a>

<button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarNav">

<span class="navbar-toggler-icon"></span>

</button>

<div class="collapse navbar-collapse" id="navbarNav">

<ul class="navbar-nav">

<li class="nav-item">

<a class="nav-link" routerLink="/users">Users</a>

</li>

<li class="nav-item">

<a class="nav-link" routerLink="/posts">Posts</a>

</li>

<li class="nav-item">

<a class="nav-link" routerLink="/comments">Comments</a>

</li>

<li class="nav-item">

<a class="nav-link" routerLink="/likes">Likes</a>

</li>

<li class="nav-item">

<a class="nav-link" routerLink="/groups">Groups</a>

</li>

<li class="nav-item">

<a class="nav-link" routerLink="/friends">Friends</a>

</li>

<li class="nav-item">

<a class="nav-link" routerLink="/notifications">Notifications</a>

</li>

<li class="nav-item">

<a class="nav-link" routerLink="/messages">Messages</a>

</li>

</ul>

</div>

</div>

</nav>

Run HTML

1. Add the navigation component to app.component.html:

html

Copy

<app-navigation></app-navigation>

<div class="container mt-4">

<router-outlet></router-outlet>

</div>

Run HTML

**4. Add Forms for Creating Data**

**User Form**

html

Copy

<div class="container mt-5">

<h2>Create User</h2>

<form (ngSubmit)="onSubmit()">

<div class="mb-3">

<label for="username" class="form-label">Username</label>

<input type="text" class="form-control" id="username" [(ngModel)]="user.username" name="username" required>

</div>

<div class="mb-3">

<label for="password" class="form-label">Password</label>

<input type="password" class="form-control" id="password" [(ngModel)]="user.password" name="password" required>

</div>

<div class="mb-3">

<label for="email" class="form-label">Email</label>

<input type="email" class="form-control" id="email" [(ngModel)]="user.email" name="email" required>

</div>

<button type="submit" class="btn btn-primary">Submit</button>

</form>

</div>

Run HTML

**User Form Logic**

typescript

Copy

import { Component } from '@angular/core';

import { UserService } from '../services/user.service';

import { User } from '../models/user.model';

@Component({

selector: 'app-user-form',

templateUrl: './user-form.component.html'

})

export class UserFormComponent {

user: User = {

userID: 0,

username: '',

password: '',

email: '',

profilePicture: ''

};

constructor(private userService: UserService) {}

onSubmit(): void {

this.userService.createUser(this.user).subscribe(response => {

console.log('User created:', response);

});

}

}

**5. Run the Application**

1. Start the Spring Boot backend.
2. Run the Angular application:

bash

Copy

ng serve

1. Open http://localhost:4200 in your browser.