**Name : Mansi Sanjay Ghule**

**Email : mansi.ghule1698@gmail.com**

**Phone No : +91 7745078471**

# Assignment 2: Java Backend

- Explanation of Endpoints:

First, we have to create a maven project in STS.
For that we have to add some dependencies in our project. We will create the war file using spring initializer and there we will add our dependencies like :

1. spring-boot-starter-data-jpa
2. `spring-boot-starter-security`
3. `spring-boot-starter-web`
4. `spring-boot-devtools`
5. `mysql-connector-java`
6. `spring-boot-starter-tomcat`

Then we will import that project in our STS IDE.

- First we have to configure our application.properties file.
- In that we will write the database related code and file upload code.

```
server.port=8080
server.tomcat.uri-encoding=utf-8

#database configuration
spring.datasource.url=jdbc:mysql://localhost:3309/socialmedia_db?serverTimezone=UTC&useUnicode
=yes&characterEncoding=UTF-8&rewriteBatchedStatements=true
spring.datasource.username=root
spring.datasource.password=Mysql@1698
```

```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true

## Multipart config
        spring.servlet.multipart.enabled=true
        spring.servlet.multipart.file-size-threshold=2KB
        spring.servlet.multipart.max-file-size=200MB
        spring.servlet.multipart.max-request-size=215MB
```

1. Create Account

Now directly we will go to 1st end i.e. Create Account

- Spring Security:
  WebSecurityConfigurerAdapter is the crux of our security implementation. It provides HttpSecurity configurations to configure cors, csrf, session management, rules for protected resources. We can also extend and customize the default configuration that contains the elements below.

  UserDetailsService interface has a method to load User by username and returns a UserDetails object that Spring Security can use for authentication and validation.

  Suppose that we want to prevent unauthorized users from viewing the greeting page at /hello. As it is now, if visitors click the link on the home page, they see the greeting with no barriers to stop them. You need to add a barrier that forces the visitor to sign in before they can see that page.

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter{

    @Autowired
    private CustomUserDetailService customUserDetailService;


    @Override
    protected void configure(HttpSecurity http) throws Exception {
```

```
        // TODO Auto-generated method stub

        http
                .csrf().disable()// by default it is enable so to disable this have to write
this
                .requestMatchers()
                .antMatchers("/newuser")
                .and()
                .authorizeRequests()
                .anyRequest()
                .authenticated()
                .and()
                .httpBasic();



    }



      @Override
      protected void configure(AuthenticationManagerBuilder auth) throws Exception {

            auth.userDetailsService(customUserDetailService).passwordEncoder(passwordEncoder()
);

      }


      @Override
        @Bean
        public AuthenticationManager authenticationManagerBean() throws Exception {
            return super.authenticationManagerBean();
        }


    @Bean
    public BCryptPasswordEncoder passwordEncoder() {

          return new BCryptPasswordEncoder(11);
      }



}
```

UserDetails contains necessary information (such as: username, password, authorities) to build an Authentication object.

- Controller receives and handles request.
- Create the models:

We're gonna have 1 table in database i.e user.

```
@Entity
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int userId;

    @NonNull
    @Column(unique = true)
    private String userName;


    @Column(unique = true)
    private String userEmail;


    private String userPassword;

    private String userRole;
```

- UserService :
  The service is used to perform various operations(e.g. CRUD) on the database. The service file is used to write business logic in the code. Nowadays getting data is not enough we need to apply the preprocessing on data. The methods of service class will be useful to extract the data from the database holding specific conditions and rules.

```
-
-   @Service
```

```
-   public class UserService {
-
-       @Autowired
-       private UserRepo userRepo;
-
-
-
-       public User addUser(User user) {
-
-           User newUser = userRepo.save(user);
-
-           return newUser;
-       }
-
-       public List<User> getAllUsers(){
-
-           List<User> userList = userRepo.findAll();
-
-           return userList;
-       }
-
-       public Boolean existsByUserName(String userName) {
-
-           Boolean user = userRepo.existsByUserName(userName);
-
-           return user;
-
-       }
-
-       public Boolean existsByUserPassword(String userPassword) {
-
-           Boolean user = userRepo.existsByUserPassword(userPassword);
-
-           return user;
-
-       }
-
-
-       public User findByUserId(Integer userId) {
-
-           User findByUserId = userRepo.findByUserId(userId);
-
-           return findByUserId;
-
-       }
```

```
-
-
-
-   }
```

- Repository contains UserRepo to work with Database, will be imported into Controller.

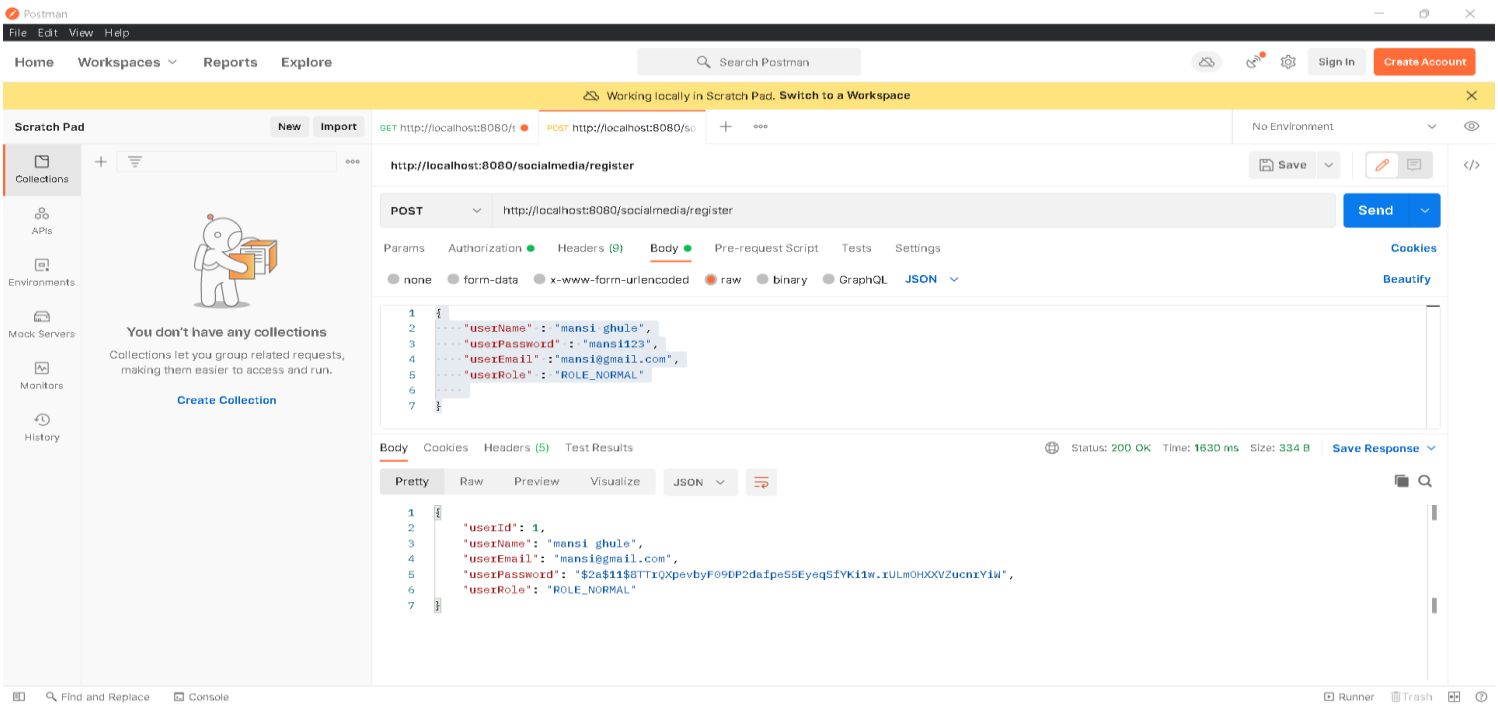
```
public interface UserRepo extends JpaRepository<User, Integer> {


    public User findByUserName(String userName);

    public Boolean existsByUserName(String userName);

    public Boolean existsByUserPassword(String userPassword);

    public User findByUserId(Integer userId);
// getter setter

}
```

- Create Spring Rest Controllers :

```
-   @RestController
-   @RequestMapping("/socialmedia")
-   public class UserController {
-
-        @Autowired
-           private AuthenticationManager authenticationManager;
-
-
-       @Autowired
-       private UserService userService;
-
-       @Autowired
-       private BCryptPasswordEncoder passowrdEncoder;
-
-
-       @PostMapping("/register")
-       public User addUser(@RequestBody User user) {
```

```
-
-           User newUser = new User();
-
-           newUser.setUserName(user.getUserName());
-
-           newUser.setUserEmail(user.getUserEmail());
-
-           newUser.setUserPassword(passowrdEncoder.encode(user.getUserPassword()));
-
-           newUser.setUserRole(user.getUserRole());
-
-           User addUser = userService.addUser(newUser);
-
-           return addUser;
-       }
-
```

- **Screen Shot Of Register User in Postman:**




2. Login :

   For login /login url is used.

For login PostMapping is used because we are sending the data using JSON format.

Here, we will send the username ans userpassword through JSON raw data.

If the user exists with that username and password then the user is successfully logged in.

But this is possible only because of authenticationManager

- The CustomerUserDetailsService code will be as follows:

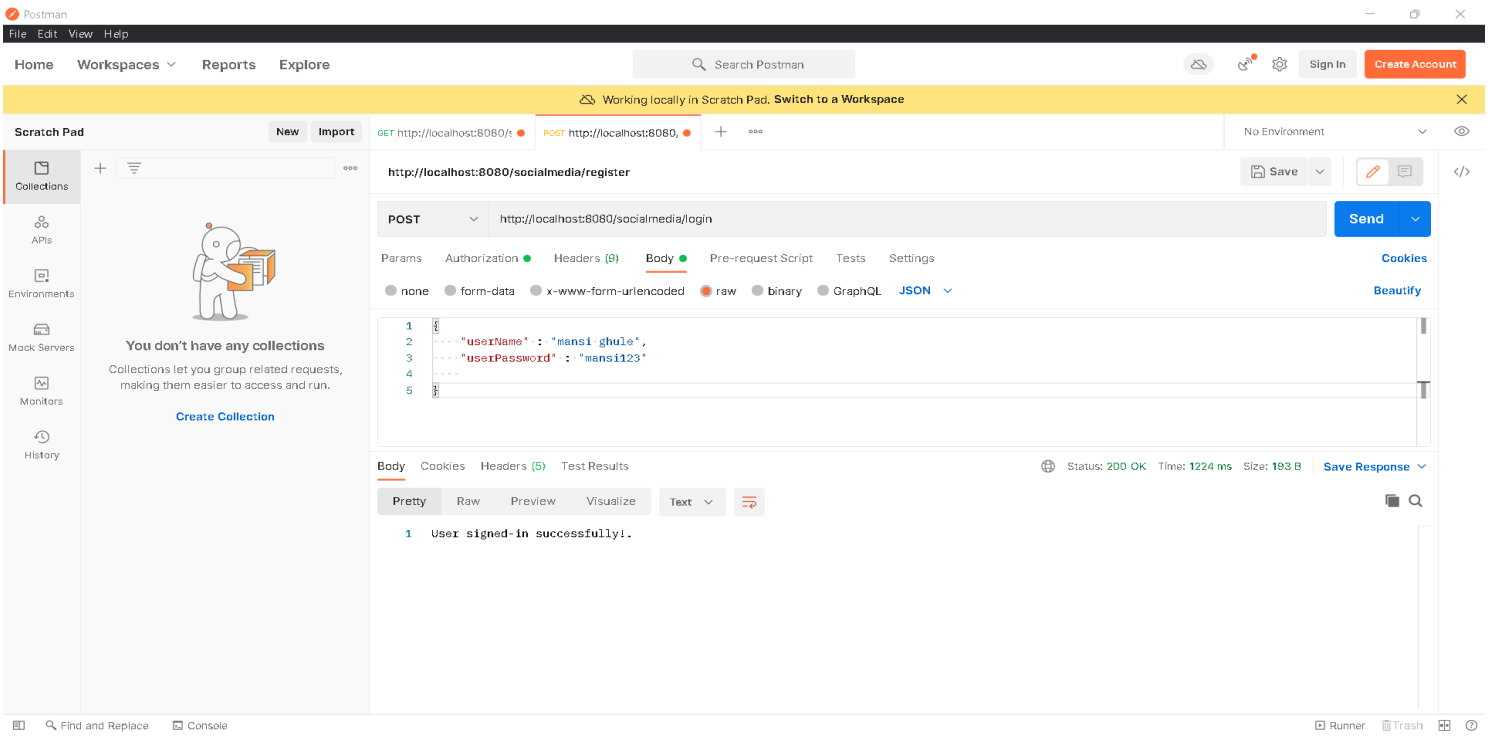
```
-   @Service
-   public class CustomUserDetailService implements UserDetailsService {
-
-
-       @Autowired
-       private UserRepo userRepo;
-
-
-       public CustomUserDetailService(UserRepo userRepo) {
-
-           this.userRepo = userRepo;
-
-       }
-
-
-
-
-         @Override public UserDetails loadUserByUsername(String userName) throws
    UsernameNotFoundException {
-             // TODO Auto-generated method stub
-
-             System.out.println("loadbyuser");
-
-         User user = userRepo.findByUserName(userName);
-
-
-
-         if(user == null) {
-
-             throw new UsernameNotFoundException("No Such User");
-         }
-
-         return new CustomUserDetails(user); }
-
```

```
-
-   }
```

- Controller code of login :

```
3.  @PostMapping("/login")
4.      public ResponseEntity<String> authenticateUser(@RequestBody User user){
5.
6.          //System.out.println("password"
    +passowrdEncoder.encode(user.getUserPassword()));
7.
8.
9.
10.
11.         if(userService.existsByUserName(user.getUserName()))
12.         {
13.
14.             //if(userService.existsByUserPassword(user.getUserPassword()))
15.
16.
17.             Authentication authentication = authenticationManager.authenticate(new
    UsernamePasswordAuthenticationToken(
18.                         user.getUserName(), user.getUserPassword()));
19.
20.                 SecurityContextHolder.getContext().setAuthentication(authentication);
21.                 return new ResponseEntity<>("User signed-in successfully!.",
    HttpStatus.OK);
22.
23.
24.         }else {
25.             return new ResponseEntity<>("Username is already taken!",
    HttpStatus.BAD_REQUEST);
26.         }
27.
28.
29.
30.     }
31.
32.
```

- **ScreenShot Of Login User in Postman:**




3. Post an image in the user's own account:

The model defines the structure/format in which the data has to be stored in the database. It represents the java object carrying the data.

- create Post.java Model class:

```java
@Entity
public class Posts {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int postId;

    @ManyToOne
    @JoinColumn(name="userId")
    private User user;


    @Column(columnDefinition="TEXT")
    private String postImg;
```

```
    @Column(columnDefinition="TIMESTAMP DEFAULT CURRENT_TIMESTAMP")
    private LocalDateTime postDate;
// getter setter
```

- Configuring Server and File Storage Properties in application.properties :

```
-   ## Multipart config
-           spring.servlet.multipart.enabled=true
-           spring.servlet.multipart.file-size-threshold=2KB
-           spring.servlet.multipart.max-file-size=200MB
-           spring.servlet.multipart.max-request-size=215MB
```

- Now we will create the FileUploadHelper  class which will upload the file to our specific location using REST API.

```
@Component
public class FileUpload {

    // public final String UPLOAD_DIR =
"C:\\Users\\ADMIN\\Downloads\\SocialMediaApplication\\src\\main\\resources\\static\\image";

     public final String UPLOAD_DIR = new
ClassPathResource("static/image/").getFile().getAbsolutePath();


     public FileUpload() throws IOException{

     }


     public boolean fileUpload(MultipartFile file) {

         boolean f = false;


        try {
            /*
             * InputStream inputStream = file.getInputStream(); byte data[]=new
             * byte[inputStream.available()]; inputStream.read(data);
             *
             * //write FileOutputStream fos= new
```

```
            * FileOutputStream(UPLOAD_DIR+File.separator+file.getOriginalFilename());
            *
            * fos.write(data); fos.flush(); fos.close();
            */

           Files.copy(file.getInputStream(),
Paths.get(UPLOAD_DIR+File.separator+file.getOriginalFilename()),
StandardCopyOption.REPLACE_EXISTING);

            f=true;


       } catch (IOException e) {
           // TODO Auto-generated catch block
           e.printStackTrace();
       }


       return f;
    }
```

- We will add this code to postService class to save our post to that particular user who is logged in.
- For that we will pass the userId of the user to save that post to that specific user using url in postman.

```
-   @Autowired
-       private PostRepo postRepo;
-
-       @Autowired
-       private UserRepo userRepo;
-
-
-       public Posts savePost(Posts posts) {
-
-           Posts savedPosts = postRepo.save(posts);
-
-           return savedPosts;
-       }
```

- We will add our defined method to access the post of specific user i.e the userID which we will pass in our url.

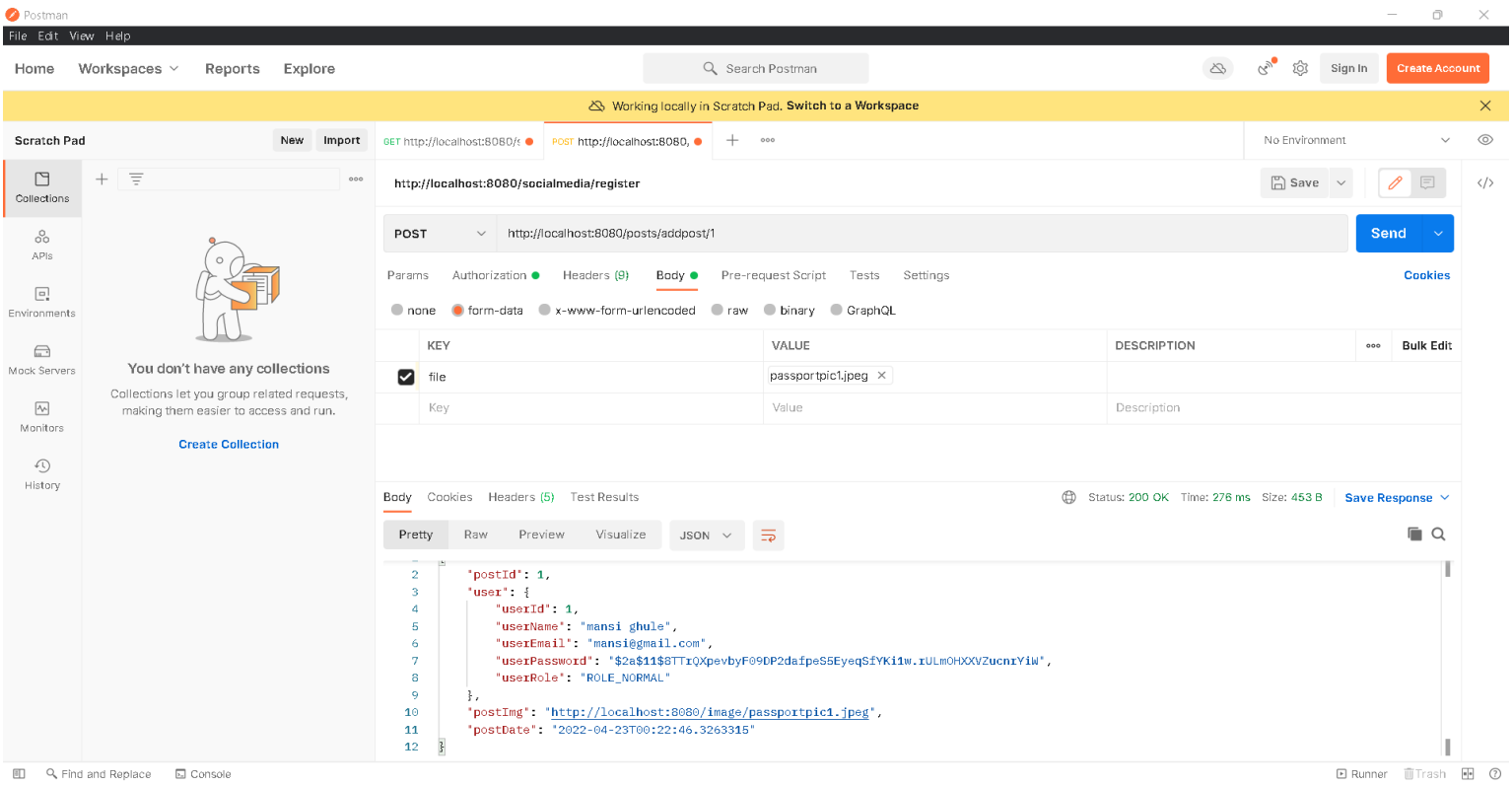
```java
public interface PostRepo extends JpaRepository<Posts, Integer>{

    /* public List<Posts> findAllPostByUserId() */


    public List<Posts> findByUser(User user);

}
```

- To save the save we will use Postmapping in PostController class.
- And there we will upload the data i.e the file that will be the jpeg image to our database.

```java
-       @PostMapping("/addpost/{userId}")
-       public ResponseEntity<?> addPosts(@RequestParam("file") MultipartFile
    file,@PathVariable("userId") Integer userId){
-
-           System.out.println(file.getOriginalFilename());
-           System.out.println(file.getSize());
-           System.out.println(file.getContentType());
-
-           try {
-
-
-               //
-               if(file.isEmpty())
-               {
-               return ResponseEntity.status (HttpStatus.
    INTERNAL_SERVER_ERROR).body("Request must contain file");
-               }
-
-               if(!file.getContentType().equals("image/jpeg"))
-               {
-               return ResponseEntity.status (HttpStatus.
    INTERNAL_SERVER_ERROR).body("Only jpeg allowed");
-               }
-
-               boolean fileUploaded = upload.fileUpload(file);
-
-               if(fileUploaded) {
-
-                   LocalDateTime localDateTime = LocalDateTime.now();
-
```

```
-                    System.out.println(localDateTime);
-                    User findByUserId = userService.findByUserId(userId);
-
-                    Posts savePost = new Posts();
-
-                    savePost.setUser(findByUserId);
-                    savePost.setPostDate(localDateTime);
-                    savePost.setPostImg(ServletUriComponentsBuilder.fromCurrentContextPath
    ().path("/image/").path(file.getOriginalFilename()).toUriString());
-
-                    Posts savedPost = postService.savePost(savePost);
-
-                   // return ResponseEntity.ok("file uploaded successfully..!");
-
-                   // return
    ResponseEntity.ok(ServletUriComponentsBuilder.fromCurrentContextPath().path("/image/").
    path(file.getOriginalFilename()).toUriString());
-                   return ResponseEntity.ok(savedPost);
-
-                }
-
-
-            }catch (Exception e) {
-               // TODO: handle exception
-                e.printStackTrace();
-           }
-
-
-            return ResponseEntity.status(HttpStatus.NOT_MODIFIED).body("Something went
    wrong..!Please try again..");
-        }
```

- **ScreenShot of adding post i.e image to an User in Postman:**




4. Get all images uploaded by the current user:

- To get all the images posted by the user .First, we have to add some function to get all the post by userId which we will provide using GETMapping in postman.

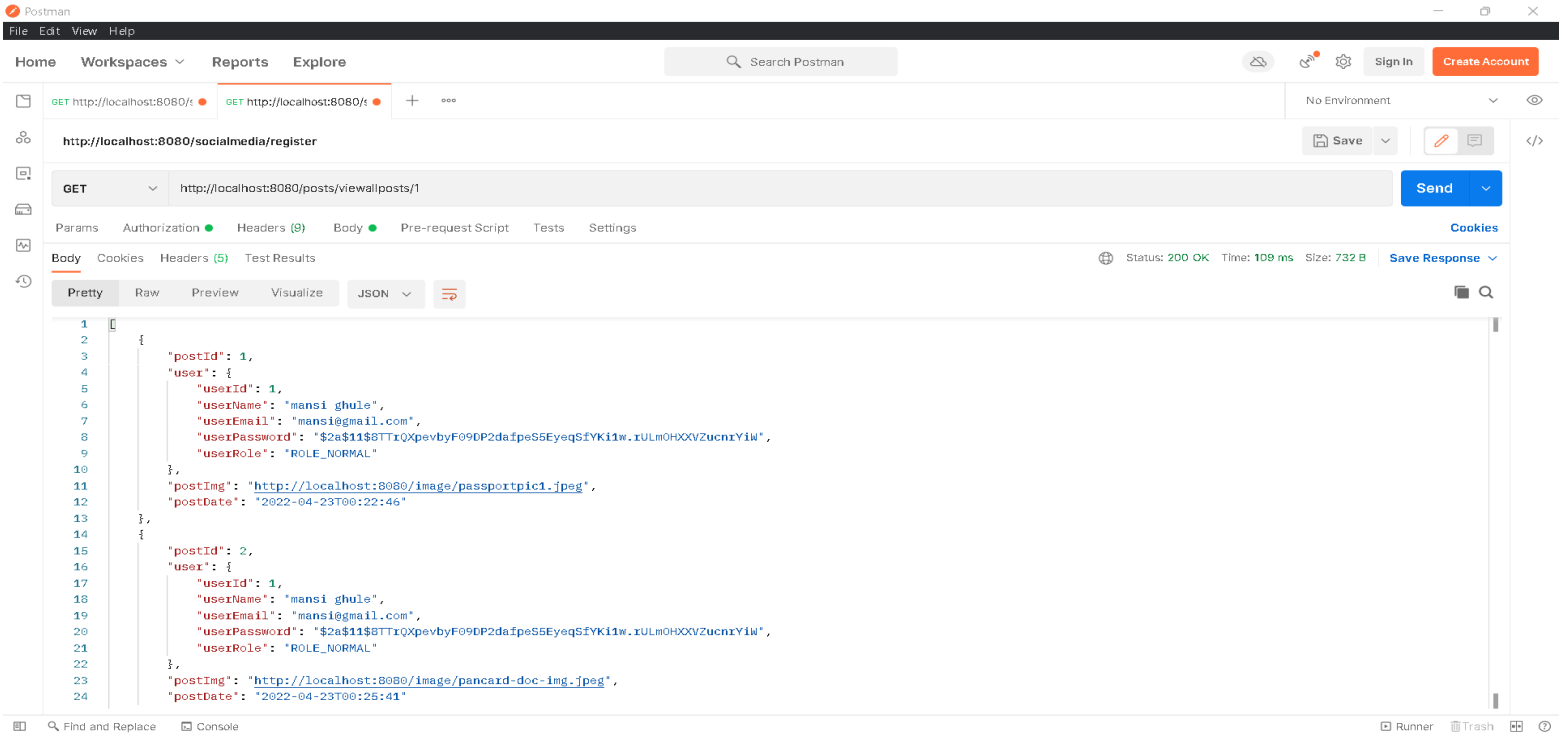
```
-       public List<Posts> getAllPostByUser(Integer userId){
-
-           User findByUserId = userRepo.findByUserId(userId);
-
-           List<Posts> postList = postRepo.findByUser(findByUserId);
-
-           return postList;
-       }
-
```

- Then in postsController we will add the POSTMapping to get the all post by user. Here we will use the function getAllPostByUser() and there we will pass the userId as a parameter.

```
-   @GetMapping("/viewallposts/{userId}")
```

```
-       public ResponseEntity<List<Posts>> getAllPostByUser(@PathVariable("userId") Integer
    userId){
-
-           List<Posts> postListByUser = postService.getAllPostByUser(userId);
-
-           return ResponseEntity.ok(postListByUser);
-       }
```

- **Screenshot of getting all post of an User in Postman:**




5. Show a list of all followers and following

- For the follwers and following we have to create new model class i.e FollowersData.

```
@Entity
public class FollowersData {


    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int followId;
```

```
    private int userId;

    @ManyToOne
    @JoinColumn(name = "followers_id")
    private User user;


    private String following;


    public FollowersData() {
        super();
        // TODO Auto-generated constructor stub
    }


    public FollowersData(int followId, int userId, User user, String following) {
        super();
        this.followId = followId;
        this.userId = userId;
        this.user = user;
        this.following = following;
    }
// getter setter
```

- To get all the follwers and following of the user.
- We have to add our defined method declaration in FollowersDataRepo.
- Using Navtive Query we will fetch the records i.e the following and followers of the specific user.

```
-   public interface FollowersDataRepo extends JpaRepository<FollowersData, Integer>{
-
-
-       public List<FollowersData> findByUserId(int userId);
-
-
-       @Query("From FollowersData f where f.userId= ?1 And f.following ='true'")
-       public List<FollowersData> getAllFollowing(int userId);
-
```

```
-           @Query("from FollowersData f where user.userId = ?1 or userId = ?1 and
    following = 'true'")
   public List<FollowersData> getknown(int uid);
-
-   }
```

- Then we will add this 2 methods in FollowerDataService and there using FollowersDataRepo we will access that methods which we defined in repo class.

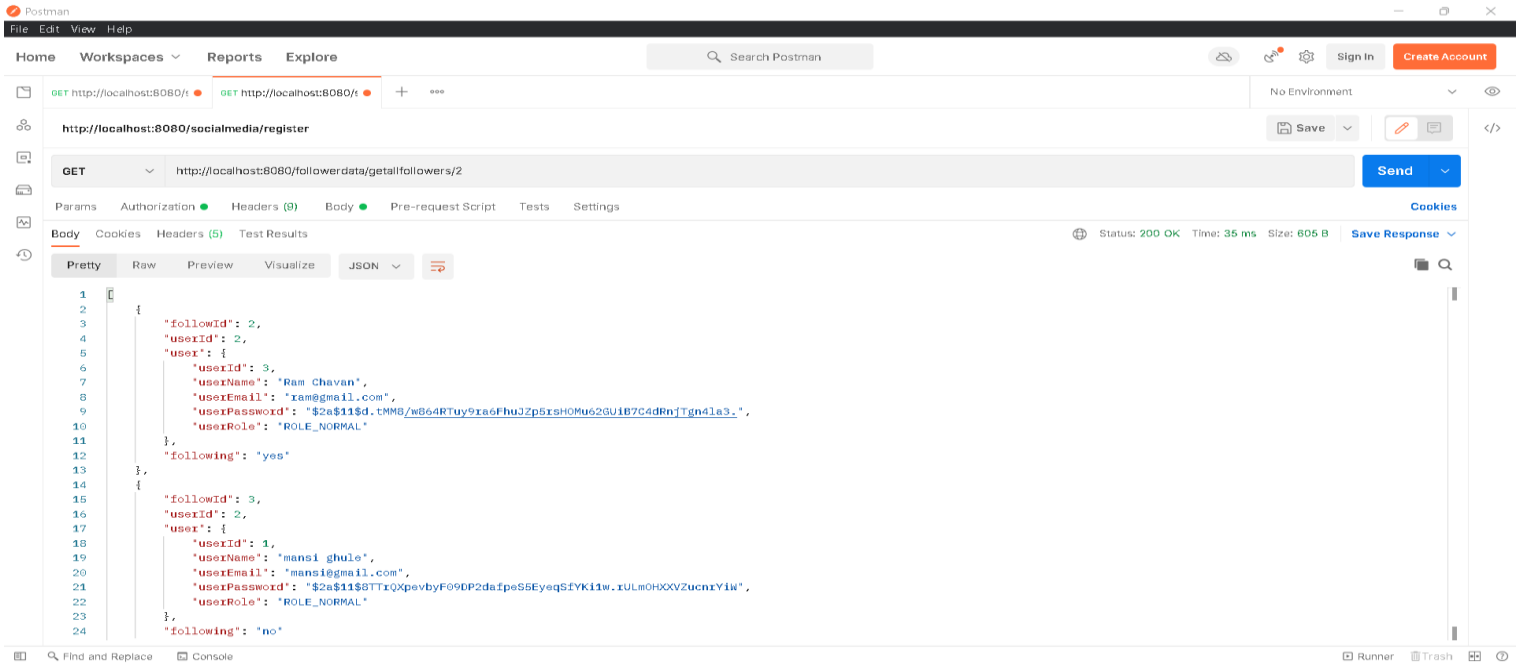
```
@Service
public class FollowersDataService {

    @Autowired
    private FollowersDataRepo dataRepo;



    public List<FollowersData> getAllFollowers(int userId){


        List<FollowersData> followersList = dataRepo.findByUserId(userId);

        return followersList;

    }



    public List<FollowersData> getAllFollowings(int userId){


        List<FollowersData> followingList = dataRepo.getAllFollowing(userId);
        return followingList;
    }
```

- Now in controller using the GETMapping we will get the list of followers and following .And we here for both we will specify the userId of the user in the url.

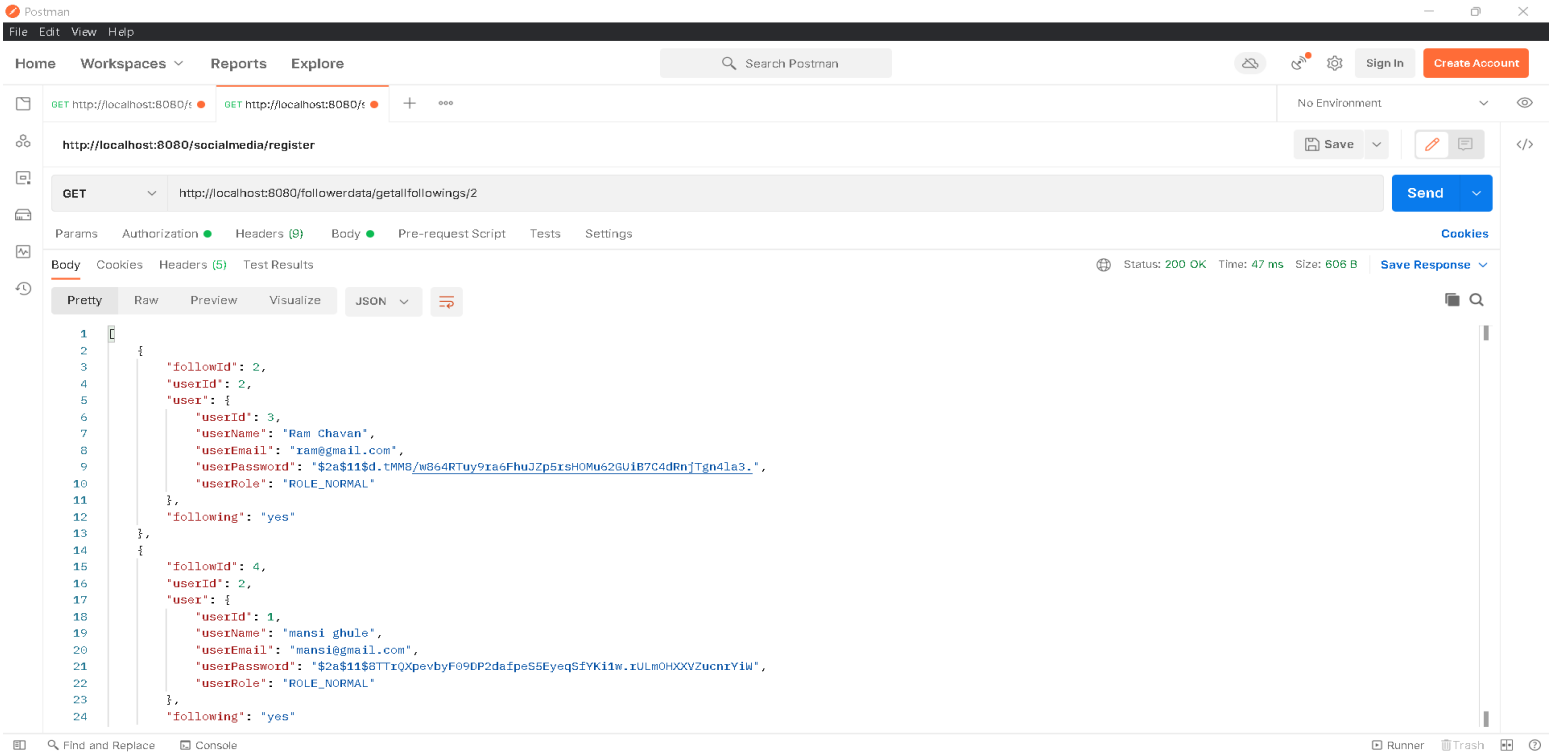
```
-   @RestController
-   @RequestMapping("/followerdata")
```

```
public class FollowDataController {


    @Autowired
    private FollowersDataService dataService;

    @GetMapping("/getallfollowers/{userId}")
    public List<FollowersData> getAllFollowers(@PathVariable("userId") int userId){


        List<FollowersData> allFollowersList = dataService.getAllFollowers(userId);

        return allFollowersList;
    }


    @GetMapping("/getallfollowings/{userId}")
    public ResponseEntity<List<FollowersData>> getAllFollowings(@PathVariable("userId")
int userId){


        List<FollowersData> allFollowingsList = dataService.getAllFollowings(userId);

        return ResponseEntity.ok(allFollowingsList);
    }
}
```

- **Followers List of an User in Postman :**




- **Following List of an User in Postman :**



6. Show a list of users a person might know. This could be done by showing the user his/her 2nd-degree connections:

- For this we define the method :

```
    @Query("from FollowersData f where user.userId = ?1 or userId = ?1 and
following = 'true'")
    public List<FollowersData> getknown(int uid);
```

Here, we are using customized query to get the You might know people list of the user.

- Now in FollwersDataService we will add this method to get all the list of You Might Know Pepole list of that user. And we will use that method which we define in FollowerdDataRepo .

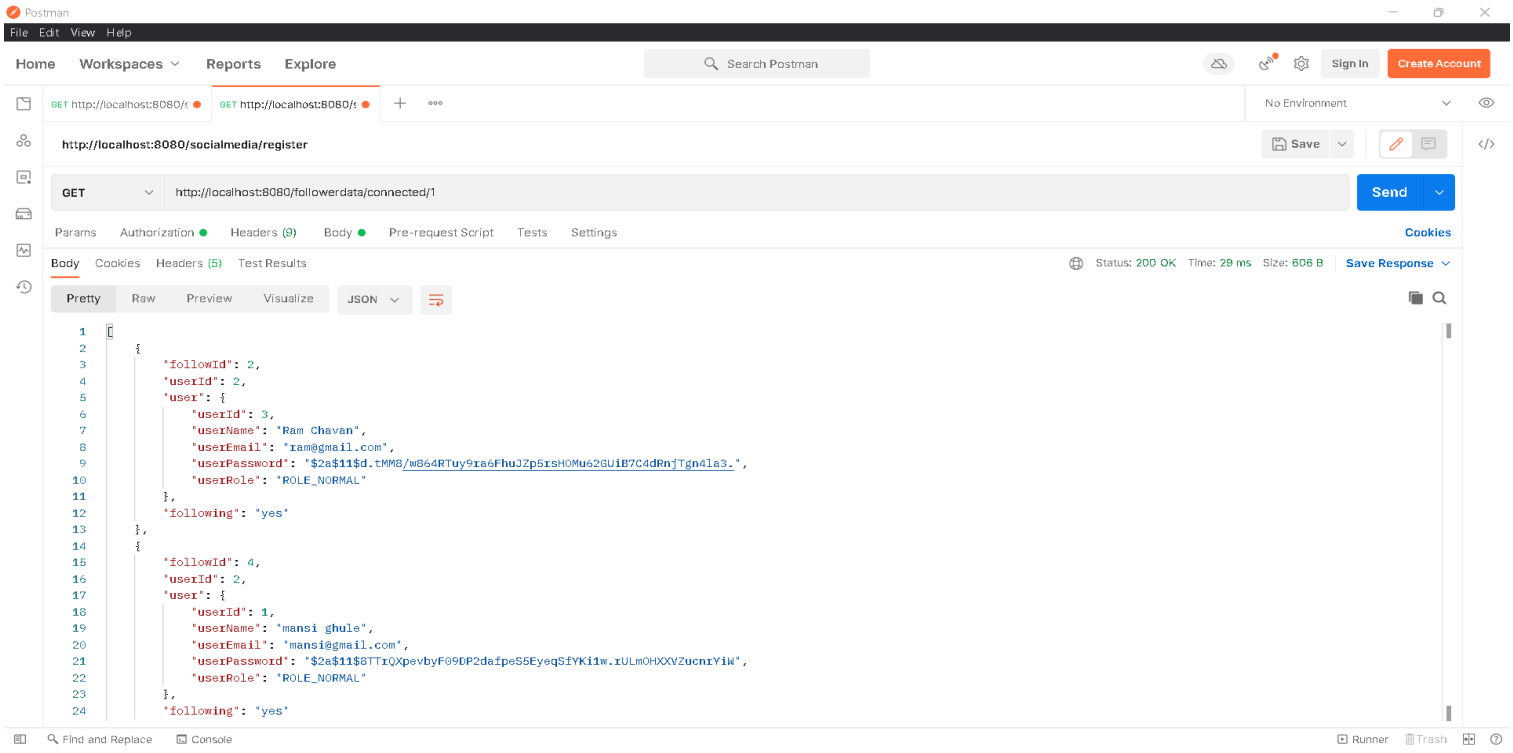
```
-       public List<FollowersData> getConnected(int userId){
-
-           List<FollowersData> getknown = dataRepo.getknown(userId);
-
-           return getknown;
-       }
```

- Using GETMapping and passing the userId as a parameter.

```
@GetMapping("/connected/{userId}")
    public ResponseEntity<?> getCon(@PathVariable("userId") int userId){
        List<FollowersData> following = dataService.getAllFollowings(userId);
        List<FollowersData> data = new ArrayList<FollowersData>();

        for (FollowersData followData : following) {


                List<FollowersData> getknown =
dataService.getConnected(followData.getUser().getUserId());
                for (FollowersData followData2 : getknown ) {
                    if(followData2.getUserId() == userId) {

                    }else {
                        data.add(followData2);
                    }
                }
```

```
        }

        return ResponseEntity.ok(data);
    }
```

- **Screenshot of You Might Know This People in postman :**




- Screenshot of database records :

MySQL 8.0 Command Line Client - Unicode

```
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use socialmedia_db;
Database changed
mysql> show tables ;
+--------------------------+
| Tables_in_socialmedia_db |
+--------------------------+
| followers_data           |
| hibernate_sequence       |
| posts                    |
| user                     |
+--------------------------+
4 rows in set (0.19 sec)

mysql> select * from user;
+---------+-------------------+-------------------+--------------------------------------------------------------+-------------+
| user_id | user_email        | user_name         | user_password                                                | user_role   |
+---------+-------------------+-------------------+--------------------------------------------------------------+-------------+
|       1 | mansi@gmail.com   | mansi ghule       | $2a$11$8TTrQXpevbyF09DP2dafpeS5EyeqSfYKi1w.rULmOHXXVZucnrYiW | ROLE_NORMAL |
|       2 | sayali@gmail.com  | sayali patil      | $2a$11$MBBFmtlnrzYFi65ryjGMlOsWrsaF4ryvy8n3mSKxYTqVvfPK4/p/i | ROLE_NORMAL |
|       3 | ram@gmail.com     | Ram Chavan        | $2a$11$d.tMM8/w864RTuy9ra6FhuJZp5rsHOMu62GUiB7C4dRnjTgn4la3. | ROLE_NORMAL |
|       4 | gayatri@gmail.com | gayatri chaudhari | $2a$11$WCvS1ZmPzsBMDSO5a/98AuAguenPTYjC9mnpISpl.7OTXPWzLP8g. | ROLE_NORMAL |
+---------+-------------------+-------------------+--------------------------------------------------------------+-------------+
4 rows in set (0.31 sec)

mysql> select * from posts;
+---------+---------------------+-----------------------------------------------+---------+
| post_id | post_date           | post_img                                      | user_id |
+---------+---------------------+-----------------------------------------------+---------+
|       1 | 2022-04-22 18:52:46 | http://localhost:8080/image/passportpic1.jpeg |       1 |
|       2 | 2022-04-22 18:55:41 | http://localhost:8080/image/pancard-doc-img.jpeg |       1 |
+---------+---------------------+-----------------------------------------------+---------+
2 rows in set (0.07 sec)

mysql> select * from followers_data;
+-----------+-----------+---------+-------------+
| follow_id | following | user_id | followers_id |
+-----------+-----------+---------+-------------+
|         1 | yes       |       1 |           2 |
|         2 | yes       |       2 |           3 |
|         3 | no        |       2 |           1 |
|         4 | yes       |       2 |           1 |
+-----------+-----------+---------+-------------+
4 rows in set (0.31 sec)

mysql>
```