Social Media Application

Submitted for the course :

SmartBridge- Modern Application Development (Java Spring Boot)

Submitted By:

Tirthankar Chakraborty(20MIS0206)

S Pooja (20MIS0037)

Piyush Rana(20MIS0410)

Anjali Jha(20MIS0124)

CONTENTS

1.Introduction

1.1) Overview
1.2)Purpose

2.Literature Survey

2.1)Existing Problem

2.2)Proposed Solution

3.Theoretical Analysis

3.1)Block Diagram

3.2)Hardware and Software Specification

4.Experimental Investigations

5. Flowchart

6.Result

7.Advantages and Disadvantages

8.Applications

9.Conclusion

10.Future Scope

11.Bibliography

Introduction

Overview

This project provides a simple solution to create a platform for connecting people all over the world with ease and securely in the form of a simple social media application. Users of the application would be able to post about their interests and other users would be able to show their appreciation to the post by sharing likes and comments. Similarly stories about the user can also be shared on the app by the user .Users can connect to other users by means of “following” them , which would be reflected in the users’ accounts as following and other users who are following the user would be reflected in the users’ account as followers.

It follows a loosely coupled architecture, combining Java Spring Boot for the backend API and React for the frontend web application. mySQL serves as the database to store movie data and user reviews. Future deployment using Docker and Kubernetes ensures scalability and reliability.

Purpose

The purpose of a social media app like our social media app is to provide a platform for users to share and discover visual content, primarily through photos and videos. It allows users to create a profile, connect with others, and engage with content by liking, commenting, and sharing. Our Social media app aims to foster creativity, self-expression, and social interaction, enabling users to showcase their lives, interests, and talents, while also staying connected with friends, celebrities, brands, and communities of shared interests. It serves as a medium for communication, entertainment, inspiration, and building relationships through visual storytelling in a visually-centric and engaging environment.

Literature Review

Existing problems

As the user base grows, social media platforms need to ensure their infrastructure can handle the increasing load of users, data, and interactions. Scaling the app's servers, databases, and networking systems to maintain performance and responsiveness can be a significant technological challenge.

User Privacy and Data security is also major concern that revolves around the usage of social media apps

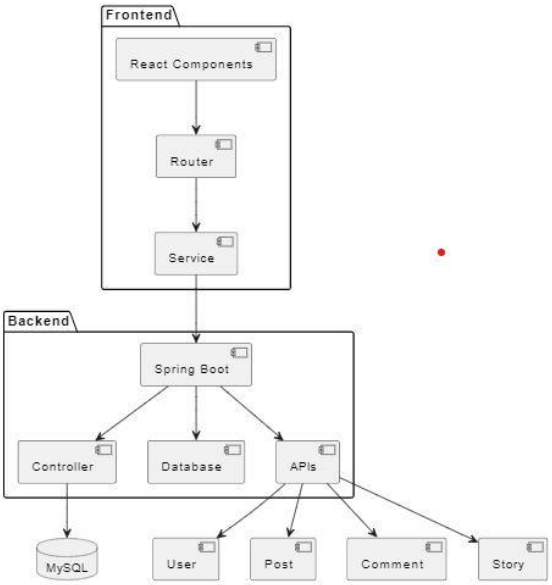
Proposed Solution

To address the existing problem, our project proposes the development of a social media application. Our solution aims to provide a centralized platform that offers the users to connect with their peers and loved ones in a digital form enabling self-expression and social interaction . By combining Java Spring Boot for the backend API, React for the frontend web application, and mySQL as the database, our proposed solution provides a seamless and user-friendly experience.

Through our proposed solution we intend to diminish the existing problems by promoting scalability and security using spring security and Docker and Kubernetes.

Theoretical Analysis

Block Diagram



Hardware / Software designing

Hardware Requirements

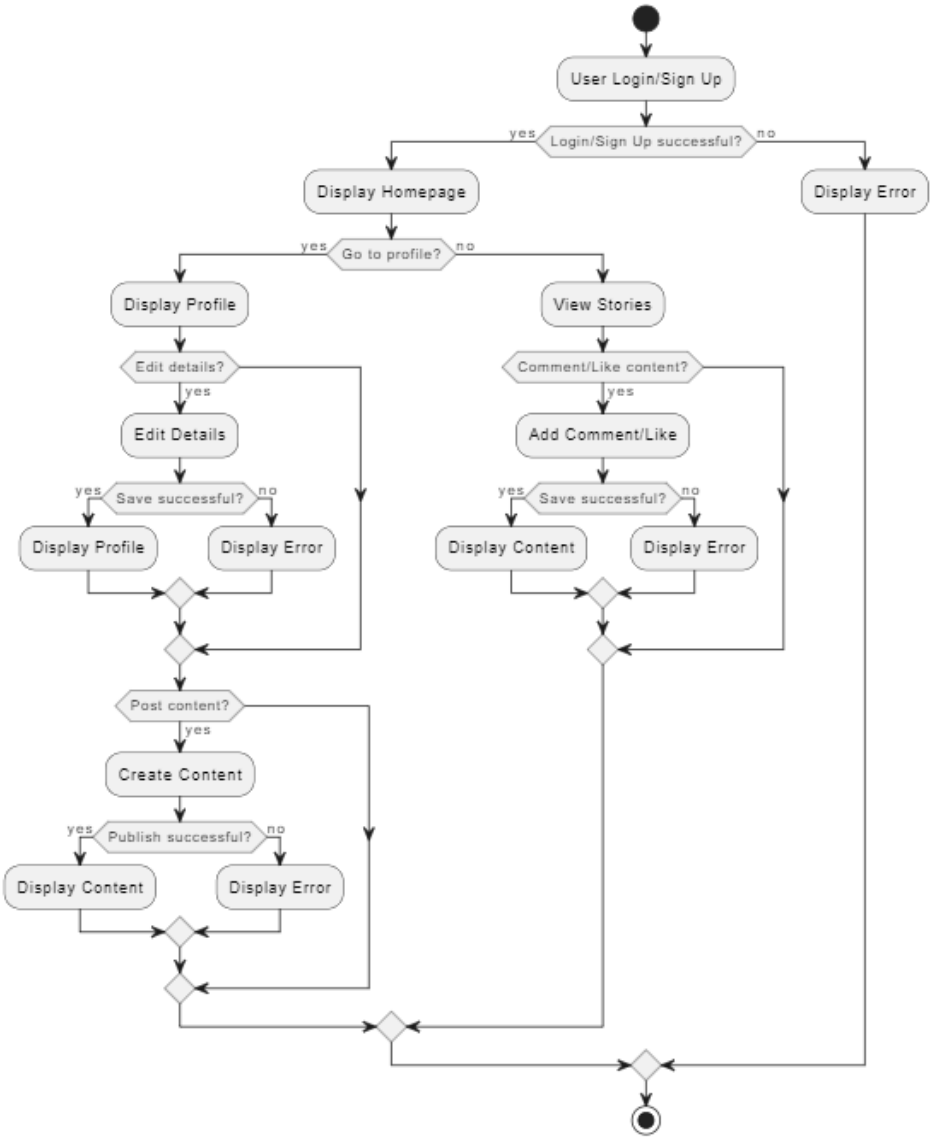
Software Requirements

1. JDK
2. Maven
3. spring tool suite
4. mysql
5. Bootstrap
6. React
7. Axios

Experimental Investigations

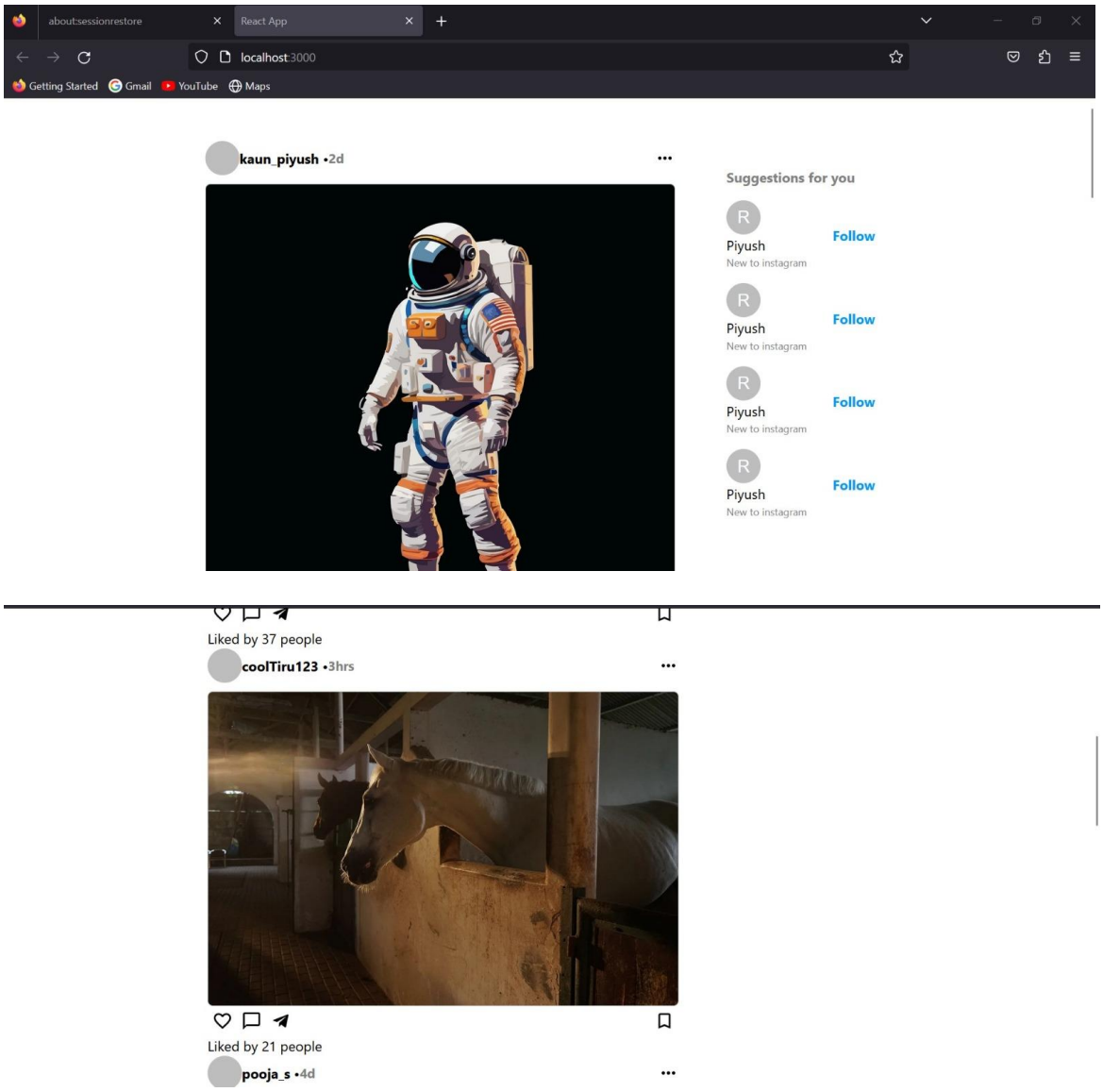
Analysis or the investigation made while working on the solution.

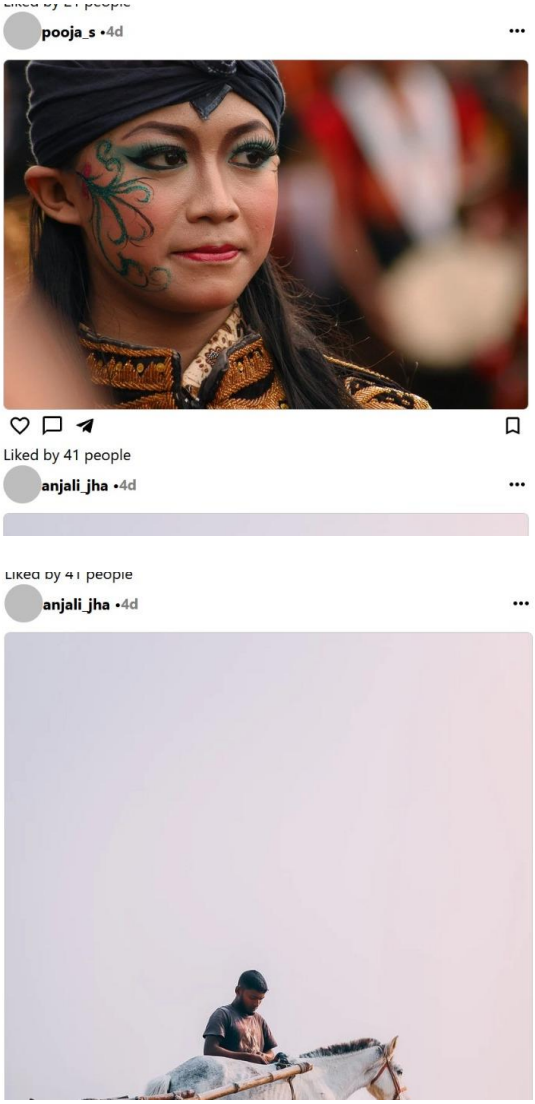
Flowchart



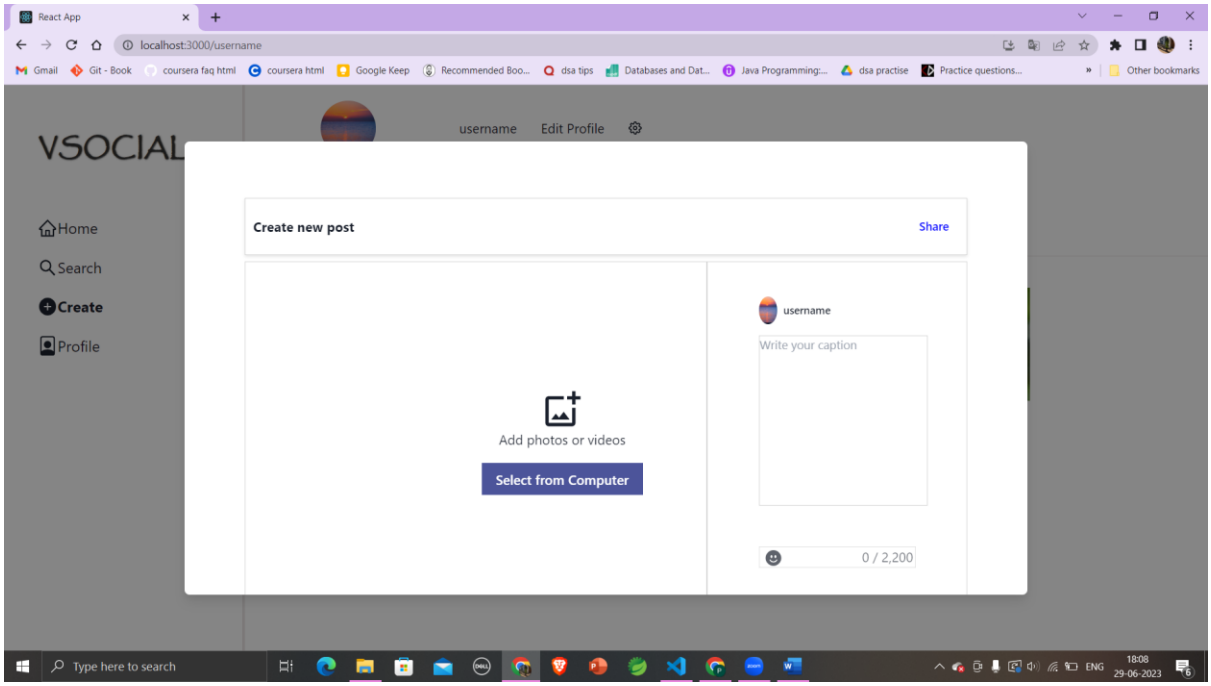
Results

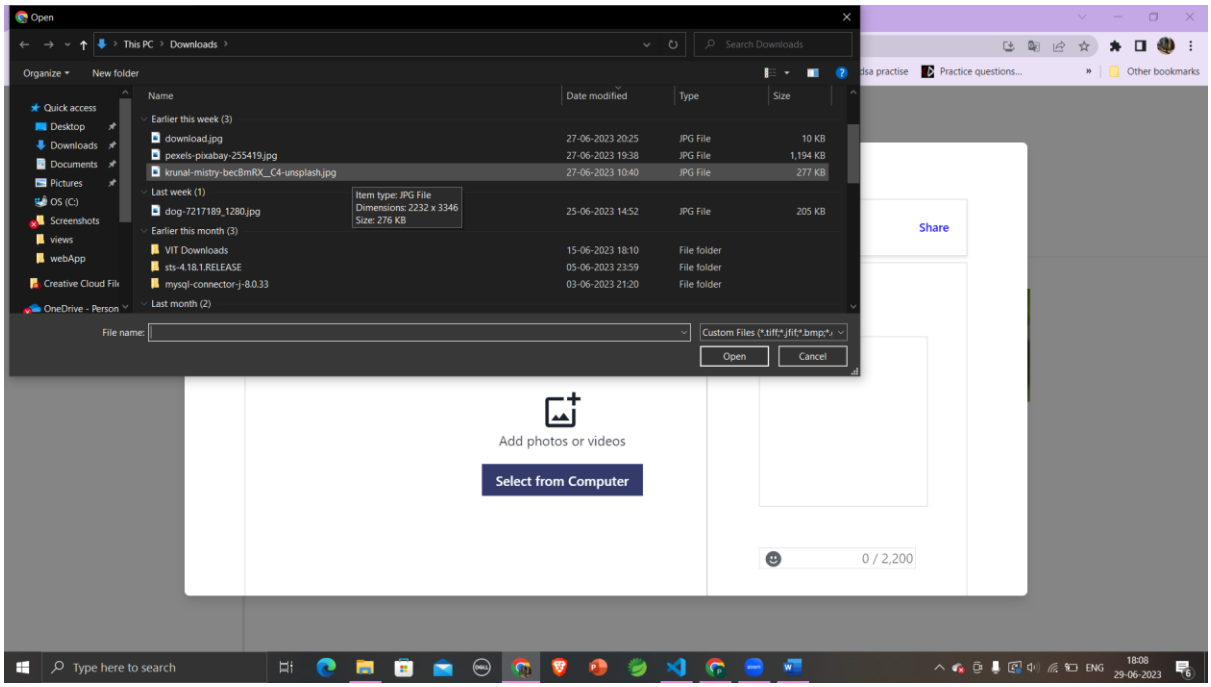
Home page:

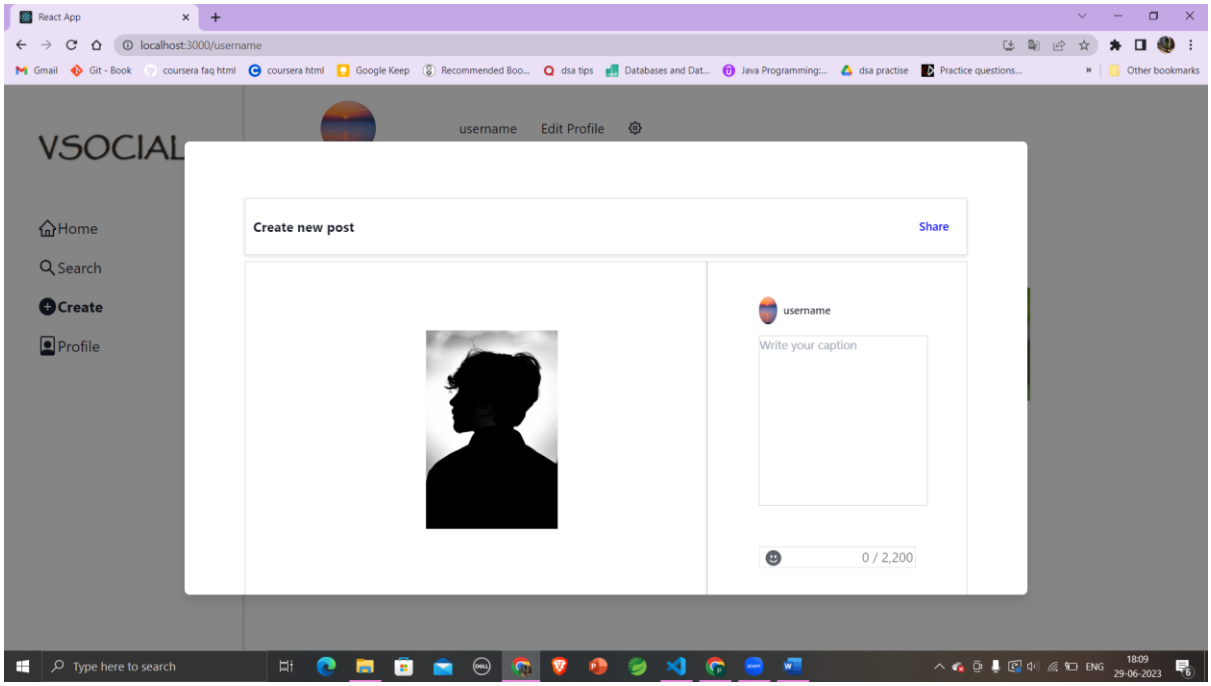


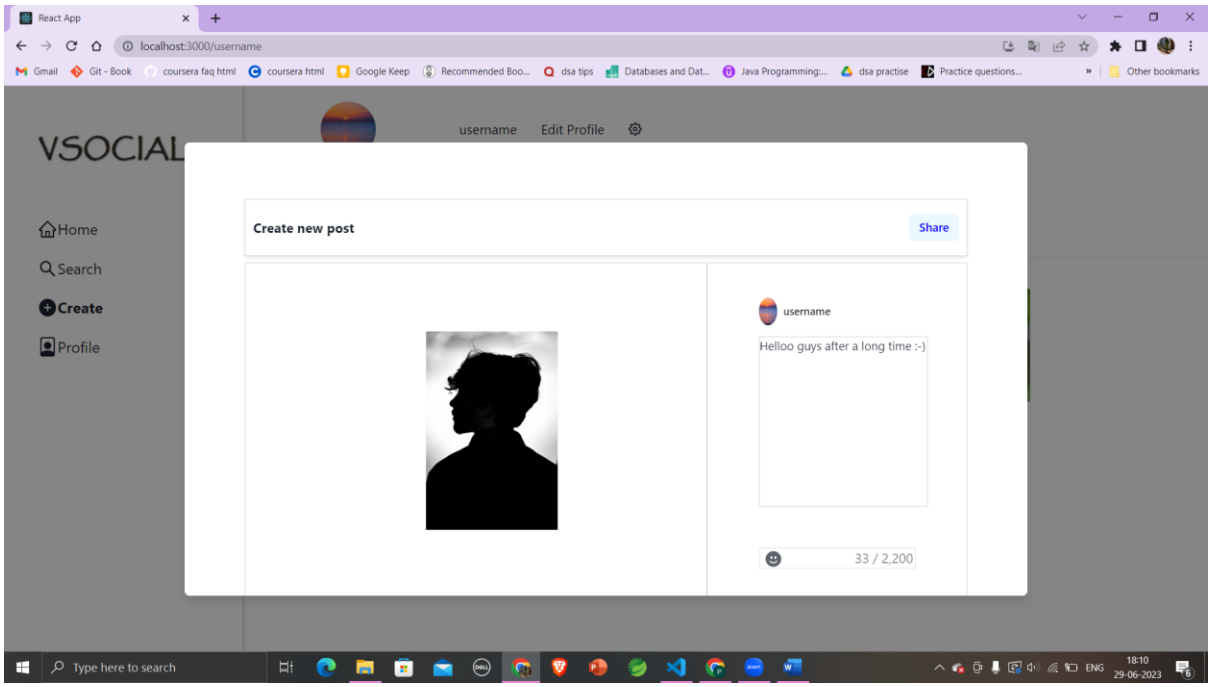


Posting a new photo/video:

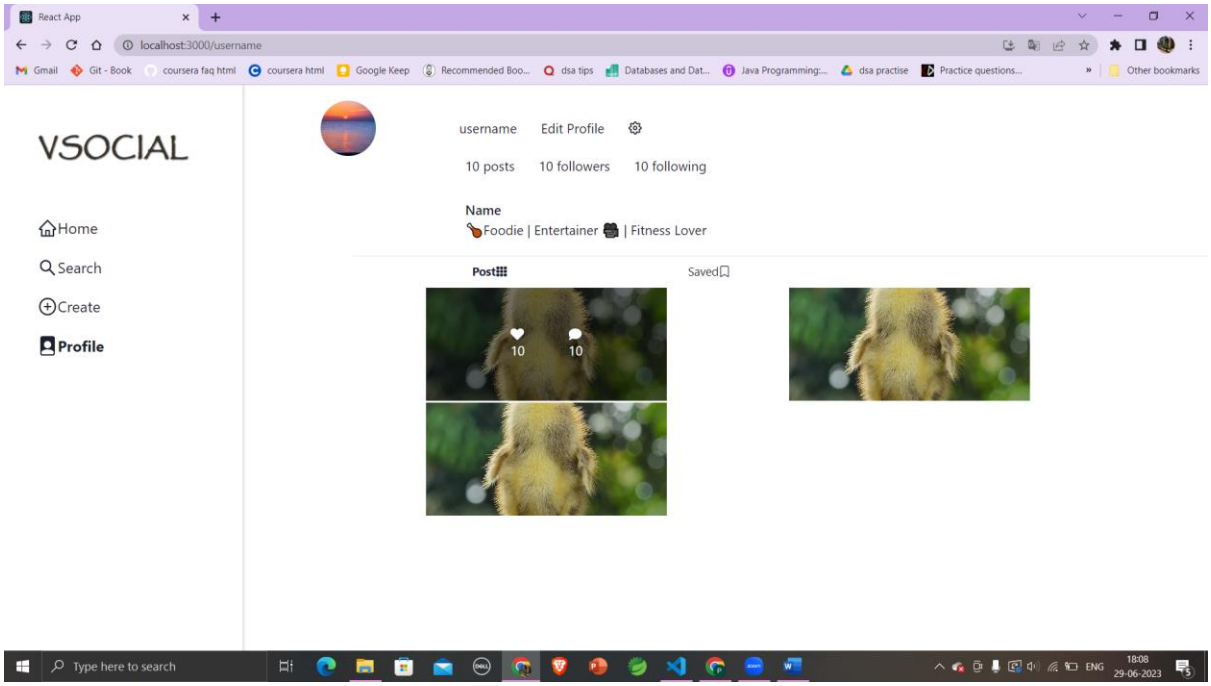

React App
localhost:3000/username
VSOCIAL
username
Edit Profile
Home
Search
Create
Profile
Create new post
Share
username
Write your caption
Add photos or videos
Select from Computer
0 / 2,200
Type here to search
ENG
18:08
29-06-2023


Open
This PC
Downloads
Search Downloads
Organize
New folder
Quick access
Desktop
Downloads
Documents
Pictures
OS (C:)
Screenshots
views
webApp
Creative Cloud Fil
OneDrive - Person
Name
Date modified
Type
Size
Earlier this week (3)
download.jpg
27-06-2023 20:25
JPG File
10 KB
pexels-pixabay-255419.jpg
27-06-2023 19:38
JPG File
1,194 KB
krunal-mistry-becBmRX__C4-unsplash.jpg
27-06-2023 10:40
JPG File
277 KB
Last week (1)
dog-7217189_1280.jpg
25-06-2023 14:52
JPG File
205 KB
Item type: JPG File
Dimensions: 2232 x 3346
Size: 276 KB
Earlier this month (3)
VIT Downloads
15-06-2023 18:10
File folder
sts-4.18.1.RELEASE
05-06-2023 23:59
File folder
mysql-connector-j-8.0.33
03-06-2023 21:20
File folder
Last month (2)
File name:
Custom Files (*.tiff;*.jfif;*.bmp;*.
Open
Cancel
Share
Add photos or videos
Select from Computer
0 / 2,200
Type here to search
ENG
18:08
29-06-2023


React App
localhost:3000/username
VSOCIAL
username
Edit Profile
Home
Search
Create
Profile
Create new post
Share
username
Write your caption
0 / 2,200
Type here to search
ENG
18:09
29-06-2023

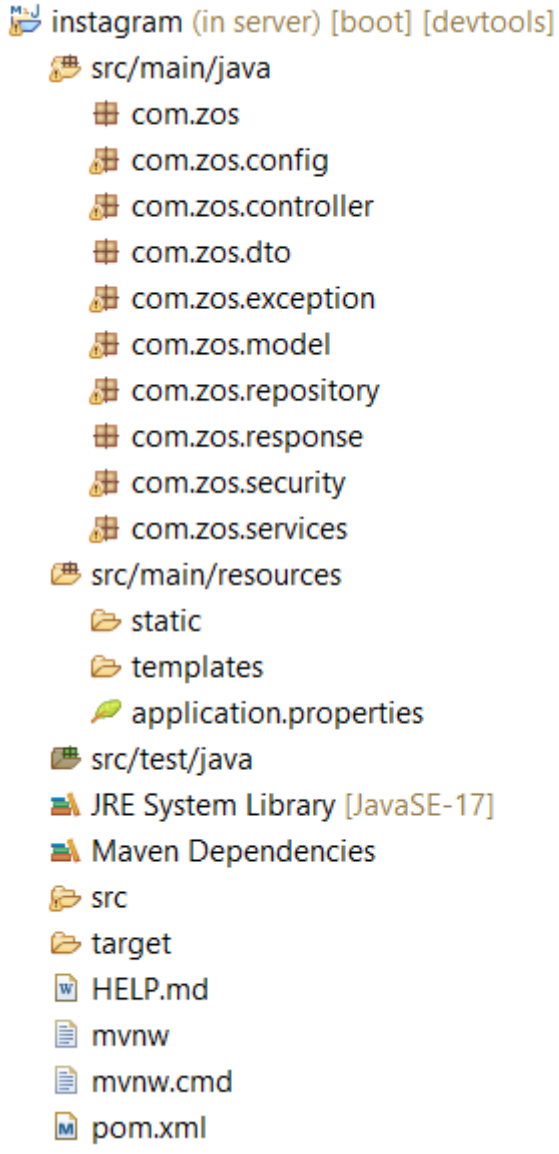

React App
localhost:3000/username
VSOCIAL
username
Edit Profile
Home
Search
Create
Profile
Create new post
Share
username
Helloo guys after a long time :-)
33 / 2,200
Type here to search
ENG
18:10
29-06-2023

Profile page:



BACKEND:

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	
caption	varchar(255)	YES		NULL	
created_at	datetime(6)	YES		NULL	
image	varchar(255)	NO		NULL	
location	varchar(255)	YES		NULL	
user_email	varchar(255)	YES		NULL	
user_id	int	YES		NULL	
name	varchar(255)	YES		NULL	
user_image	varchar(255)	YES		NULL	
user_username	varchar(255)	YES		NULL	



```java
package com.zos;

import org.springframework.boot.SpringApplication;

@SpringBootApplication
public class InstagramApplication {

	public static void main(String[] args) {
		SpringApplication.run(InstagramApplication.class, args);
	}

}
```

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v3.0.2)

2023-06-28T18:33:32.286+05:30  INFO 13608 --- [  restartedMain] com.zos.InstagramApplication             : Starting InstagramApplication using Java 17.0.7 wi
2023-06-28T18:33:32.328+05:30  INFO 13608 --- [  restartedMain] com.zos.InstagramApplication             : No active profile set, falling back to 1 default p
2023-06-28T18:33:32.574+05:30  INFO 13608 --- [  restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : Devtools property defaults active! Set 'spring.dev
2023-06-28T18:33:32.575+05:30  INFO 13608 --- [  restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : For additional web related logging consider settin
2023-06-28T18:33:35.522+05:30  INFO 13608 --- [  restartedMain] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFA
2023-06-28T18:33:35.808+05:30  INFO 13608 --- [  restartedMain] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 253 ms
2023-06-28T18:33:38.217+05:30  INFO 13608 --- [  restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 5454 (http)
2023-06-28T18:33:38.261+05:30  INFO 13608 --- [  restartedMain] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2023-06-28T18:33:38.262+05:30  INFO 13608 --- [  restartedMain] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/10.1.5]
2023-06-28T18:33:38.686+05:30  INFO 13608 --- [  restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2023-06-28T18:33:38.689+05:30  INFO 13608 --- [  restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization complet
2023-06-28T18:33:39.376+05:30  INFO 13608 --- [  restartedMain] o.hibernate.jpa.internal.util.LogHelper  : HHH000204: Processing PersistenceUnitInfo [name: d
2023-06-28T18:33:39.630+05:30  INFO 13608 --- [  restartedMain] org.hibernate.Version                    : HHH000412: Hibernate ORM core version 6.1.6.Final
2023-06-28T18:33:40.406+05:30  WARN 13608 --- [  restartedMain] org.hibernate.orm.deprecation            : HHH90000021: Encountered deprecated setting [javax
2023-06-28T18:33:40.954+05:30  INFO 13608 --- [  restartedMain] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
2023-06-28T18:33:42.449+05:30  INFO 13608 --- [  restartedMain] com.zaxxer.hikari.pool.HikariPool        : HikariPool-1 - Added connection com.mysql.cj.jdbc.
2023-06-28T18:33:42.456+05:30  INFO 13608 --- [  restartedMain] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
2023-06-28T18:33:42.705+05:30  INFO 13608 --- [  restartedMain] SQL dialect                              : HHH000400: Using dialect: org.hibernate.dialect.My
2023-06-28T18:33:45.610+05:30  INFO 13608 --- [  restartedMain] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000490: Using JtaPlatform implementation: [org.
2023-06-28T18:33:45.650+05:30  INFO 13608 --- [  restartedMain] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persisten
```

PostServiceImplementation:

PostServiceI... | InstagramApp... | JwtValidatio... | AuthControll... | PostReposito... | UserReposito... | MessageRespo

```java
@Service
public class PostServiceImplementation implements PostService {

    @Autowired
    private UserService userService;

    @Autowired
    private PostRepository postRepo;

    @Autowired
    private UserRepository userRepo;




    @Override
    public Post createPost(Post post, Integer userId) throws UserException   {

        User user = userService.findUserById(userId);

        UserDto userDto=new UserDto();

        userDto.setEmail(user.getEmail());
        userDto.setUsername(user.getUsername());
        userDto.setId(user.getId());
        userDto.setName(user.getName());
        userDto.setUserImage(user.getImage());

        post.setUser(userDto);

        post.setCreatedAt(LocalDateTime.now());

            Post createdPost =postRepo.save(post);


        return createdPost;
    }


    @Override
    public List<Post> findPostByUserId(Integer userId) throws UserException {

```

```java

    @Override
    public List<Post> findPostByUserId(Integer userId) throws UserException {

        List<Post> posts=postRepo.findByUserId(userId);

        if(posts.size()>0) {
            return posts;
        }

        throw new UserException("This user don't have any post");
    }


    @Override
    public Post findePostById(Integer postId) throws PostException {
        Optional<Post> opt = postRepo.findById(postId);
        if(opt.isPresent()) {
            return opt.get();
        }
        throw new PostException("Post not exist with id: "+postId);
    }


    @Override
    public List<Post> findAllPost() throws PostException {
        List<Post> posts = postRepo.findAll();
        if(posts.size()>0) {
            return posts;
        }
        throw new PostException("Post Not Exist");
    }


    @Override
    public Post likePost(Integer postId, Integer userId) throws UserException, PostException  {
        // TODO Auto-generated method stub

        User user= userService.findUserById(userId);

        UserDto userDto=new UserDto();

        userDto.setEmail(user.getEmail());
```

```java
@Override
public Post unLikePost(Integer postId, Integer userId) throws UserException, PostException  {
    // TODO Auto-generated method stub

    User user= userService.findUserById(userId);
    UserDto userDto=new UserDto();

    userDto.setEmail(user.getEmail());
    userDto.setUsername(user.getUsername());
    userDto.setId(user.getId());
    userDto.setName(user.getName());
    userDto.setUserImage(user.getImage());

    Post post=findePostById(postId);
    post.getLikedByUsers().remove(userDto);



    return postRepo.save(post);
}


@Override
public String deletePost(Integer postId, Integer userId) throws UserException, PostException {
    // TODO Auto-generated method stub

    Post post =findePostById(postId);

    User user=userService.findUserById(userId);
    System.out.println(post.getUser().getId()+" ------ "+user.getId());
    if(post.getUser().getId().equals(user.getId())) {
        System.out.println("inside delete");
        postRepo.deleteById(postId);

    return "Post Deleted Successfully";
    }


    throw new PostException("You Dont have access to delete this post");

}
```

```java
@Override
public List<Post> findAllPostByUserIds(List<Integer> userIds) throws PostException, UserException {


    List<Post> posts= postRepo.findAllPostByUserIds(userIds);

    if(posts.size()==0) {
        throw new PostException("No Post Available of your followings");
    }


    return posts;
}


@Override
public String savedPost(Integer postId, Integer userId) throws PostException, UserException {

    Post post=findePostById(postId);
    User user=userService.findUserById(userId);
    if(!user.getSavedPost().contains(post)) {
        user.getSavedPost().add(post);
        userRepo.save(user);
    }



    return "Post Saved Successfully";
}


@Override
public String unSavePost(Integer postId, Integer userId) throws PostException, UserException {
    Post post=findePostById(postId);
    User user=userService.findUserById(userId);

    if(user.getSavedPost().contains(post)) {
        user.getSavedPost().remove(post);
        userRepo.save(user);
    }

    return "Post Remove Successfully";
}
```

UserServiceImplementation:

```java

@Service
public class UserServiceImplementation implements UserService {

    @Autowired
    private UserRepository repo;

    @Autowired
    private PasswordEncoder passwordEncoder;

//  @Autowired
//  private PostService postService;

    @Autowired
    private JwtTokenProvider jwtTokenProvider;

    @Override
    public User registerUser(User user) throws UserException {

        System.out.println("registered user ------ ");

        Optional<User> isEmailExist = repo.findByEmail(user.getEmail());

        if (isEmailExist.isPresent()) {
            throw new UserException("Email Already Exist");
        }

        Optional<User> isUsernameTaken=repo.findByUsername(user.getUsername());

        if(isUsernameTaken.isPresent()) {
            throw new UserException("Username Already Taken");
        }

        if(user.getEmail()== null || user.getPassword()== null || user.getUsername()==null || user.getName()==null) {
            throw new UserException("email,password and username are required");

        }
```

```java

        String encodedPassword=passwordEncoder.encode(user.getPassword());

        User newUser=new User();

        newUser.setEmail(user.getEmail());
        newUser.setPassword(encodedPassword);
        newUser.setUsername(user.getUsername());
        newUser.setName(user.getName());

        return repo.save(newUser);

    }


    @Override
    public User findUserById(Integer userId) throws UserException {

        Optional<User> opt =repo.findById(userId);

        if(opt.isPresent()) {
            return opt.get();
        }

        throw new UserException("user not found with userid :"+userId);
    }




    @Override
    public String followUser(Integer reqUserId, Integer followUserId) throws UserException {
        User followUser=findUserById(followUserId);
        User reqUser=findUserById(reqUserId);

        UserDto follower=new UserDto();
        follower.setEmail(reqUser.getEmail());
        follower.setUsername(reqUser.getUsername());
        follower.setId(reqUser.getId());
        follower.setName(reqUser.getName());
        follower.setUserImage(reqUser.getImage());

```

```
        UserDto following=new UserDto();
        following.setEmail(followUser.getEmail());
        following.setUsername(followUser.getUsername());
        following.setId(followUser.getId());
        following.setName(followUser.getName());
        following.setUserImage(followUser.getImage());


        followUser.getFollower().add(follower);
        reqUser.getFollowing().add(following);

        repo.save(followUser);
        repo.save(reqUser);

        return "you are following "+followUser.getUsername();
    }


    @Override
    public String unfollowUser(Integer reqUserId, Integer unfollowUserId) throws UserException {


        User unfollowUser=findUserById(unfollowUserId);

        System.out.println("unfollow user ---- "+unfollowUser.toString());
        System.out.println("unfollow user's follower"+unfollowUser.getFollower().toString());

        User reqUser=findUserById(reqUserId);

        UserDto unfollow=new UserDto();
        unfollow.setEmail(reqUser.getEmail());
        unfollow.setUsername(reqUser.getUsername());
        unfollow.setId(reqUser.getId());
        unfollow.setName(reqUser.getName());
        unfollow.setUserImage(reqUser.getImage());


        UserDto following=new UserDto();
        following.setEmail(unfollowUser.getEmail());
        following.setUsername(unfollowUser.getUsername());
        following.setId(unfollowUser.getId());
        following.setName(unfollowUser.getName());
        following.setUserImage(unfollowUser.getImage());
```

```
        unfollowUser.getFollower().remove(unfollow);

        repo.save(reqUser);

//      User user= userService.findUserById(userId);
//      UserDto userDto=new UserDto();
//
//      userDto.setEmail(user.getEmail());
//      userDto.setUsername(user.getUsername());
//      userDto.setId(user.getId());
//
//      Post post=findePostById(postId);
//      post.getLikedByUsers().remove(userDto);

        return "you have unfollow "+unfollowUser.getUsername();


    }


    @Override
    public User findUserProfile(String token) throws UserException {

        token=token.substring(7);

        JwtTokenClaims jwtTokenClaims = jwtTokenProvider.getClaimsFromToken(token);

        String username = jwtTokenClaims.getUsername();

        Optional<User> opt = repo.findByEmail(username);

        if(opt.isPresent()) {


            return opt.get();

        }

        throw new UserException("user not exist with email : "+username);
```

```
    @Override
    public User findUserByUsername(String username) throws UserException {

        Optional<User> opt=repo.findByUsername(username);

        if(opt.isPresent()) {
            User user=opt.get();
            return user;
        }

        throw new UserException("user not exist with username "+username);
    }


    @Override
    public List<User> findUsersByUserIds(List<Integer> userIds) {
        List<User> users= repo.findAllUserByUserIds(userIds);

        return users;
    }


    @Override
    public List<User> searchUser(String query) throws UserException {
        List<User> users=repo.findByQuery(query);
        if(users.size()==0) {
            throw new UserException("user not exist");
        }
        return users;
    }


    @Override
    public User updateUserDetails(User updatedUser, User existingUser) throws UserException {

        if(updatedUser.getEmail()!= null) {
            existingUser.setEmail(updatedUser.getEmail());
        }
        if(updatedUser.getBio()!=null) {
            existingUser.setBio(updatedUser.getBio());
        }
        if(updatedUser.getName()!=null) {
            existingUser.setName(updatedUser.getName());
```

Advantages & Disadvantages

Advantages

1.Responsive and Interactive User Interface: React's efficient rendering and virtual DOM enable a smooth and responsive user interface. Users can enjoy fast page load times, seamless transitions, and interactive features, enhancing their overall experience while using the app.

2.Real-Time Updates: React's ability to handle real-time updates makes it well-suited for social media apps. Users can receive instant notifications, updates, and new content without needing to manually refresh the page, keeping them engaged and informed.

3.Enhanced User Experience: The combination of React's component-based architecture and Spring Boot's rapid development capabilities can result in an app with a user-friendly and intuitive interface. Users can navigate easily, discover content, interact with posts, and connect with others, creating an enjoyable experience.

4.Security and Data Protection: With Spring Boot as the backend, user data can be stored securely, and appropriate measures can be implemented to protect user privacy. Robust authentication mechanisms and secure data handling practices can help instill user confidence and trust in the app.

5.Cross-Platform Accessibility: React's capability to build mobile-friendly interfaces ensures that the social media app can be accessed across various devices and platforms. Users can use the app on their desktops, smartphones, and tablets, providing flexibility and convenience.

6.Performance and Speed: React's efficient rendering and Spring Boot's optimized backend can contribute to improved performance and faster response times. Users can enjoy a snappy and lag-free experience while browsing, uploading content, or interacting with the app's features.

Disadvantages

1.Learning Curve and Familiarity: If users are not accustomed to social media apps built with React and Spring Boot, they may require some time to understand the app's interface, navigation, and functionalities. This learning curve can initially present challenges for new users.

2.Compatibility and Technical Requirements: Users may need to ensure their devices meet the technical requirements for running the social media app smoothly. Compatibility issues or older devices may result in reduced performance or limited functionality.

3.Occasional Bugs and Updates: Like any software, social media apps can encounter bugs or require periodic updates to introduce new features or fix issues. Users may occasionally experience glitches, crashes, or temporary disruptions during updates.

4.Privacy Concerns: Users should remain vigilant about their privacy and data security while using any social media app. It is important to review and understand the app's privacy settings, data collection practices, and take appropriate measures to protect personal information.

5.Dependency on Internet Connectivity: Social media apps heavily rely on internet connectivity. Users may experience limitations in accessing or using the app in areas with poor network coverage or during network outages.

Applications

1. Connecting with Friends and Family: Social media apps provide a platform for users to connect and stay in touch with their friends and family, regardless of geographical distances. It enables users to share updates, photos, and videos, and engage in real-time conversations, fostering stronger relationships and reducing feelings of isolation.

2. Sharing Life Moments: Social media apps allow users to share significant moments and experiences from their lives with their network. Users can post photos and videos, share their achievements, travel experiences, special occasions, and everyday moments, creating a digital diary of memories.

3. Discovering and Exploring Interests: Users can explore their interests and discover new content through social media apps. They can follow accounts, hashtags, and communities related to their hobbies, passions, or areas of interest. This facilitates the discovery of relevant and engaging content, such as articles, tutorials, art, music, and more.

4. Professional Networking and Career Growth: Social media apps offer opportunities for professional networking and career advancement. Users can connect with industry professionals, join relevant groups and communities, and showcase their skills and expertise. It can serve as a platform for job hunting, recruitment, and expanding professional networks.

5. Accessing News and Information: Social media apps can be a source of news and information. Users can follow news outlets, journalists, and influencers to stay updated on

current events, trends, and topics of interest. However, users should exercise critical thinking and verify the credibility of sources to avoid misinformation.

6. Support and Community Engagement: Social media apps enable users to find and join communities of shared interests, causes, or support groups. It offers a platform for users to engage in discussions, seek advice, provide support, and connect with like-minded individuals who can relate to their experiences.

7. Creative Expression and Inspiration: Social media apps provide a space for creative expression and inspiration. Users can share their artwork, photography, writing, and other forms of creativity, and receive feedback and encouragement from their peers. They can also explore and draw inspiration from the creative works of others.

8. Promoting Businesses and Entrepreneurship: Social media apps offer opportunities for businesses and entrepreneurs to promote their products, services, or personal brands. Users can create business profiles, showcase their offerings, engage with customers, and leverage the app's advertising and marketing features to reach a wider audience.

Conclusion

We were able to successfully create a social media application by using react.js and spring boot in order to provide a smooth and fruitful user experience .

Future Scope

Personalization and AI-driven Recommendations: The future of social media lies in personalized content and recommendations. By utilizing machine learning algorithms and artificial intelligence (AI), social media apps can analyze user preferences, behavior, and interactions to deliver more tailored and relevant content, ensuring a highly personalized user experience.

Improved Privacy and Data Security: With growing concerns around privacy and data security, social media apps will need to continue enhancing their privacy measures. Future iterations can implement advanced encryption techniques, granular privacy controls, and transparent data handling practices to provide users with greater control over their personal information.

Enhanced Social Features: Social media apps can continue to evolve by introducing new and innovative social features. For example, incorporating live streaming, group video calls, collaborative content creation, and interactive gaming can foster more meaningful connections and engagement among users

Bibliography

1."Pro Spring Boot 2: An Authoritative Guide to Building Microservices, Web, and Enterprise Applications" by Felipe Gutierrez

2. "Spring Boot in Action" by Craig Walls.

3. "React Up and Running: Building Web Applications" by Stoyan Stefanov

4. Official Documentation: The official documentation for Spring Boot (https://spring.io/projects/spring-boot) and React (https://reactjs.org/docs/)

5. Spring Boot and React tutorial series by Amigoscode

6. Spring Boot and React Tutorials by JavaGuides

7. Ma, Meng, et al. "Light-weight and scalable hierarchical-MVC architecture for cloud web applications." 2019 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2019 5th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom). IEEE, 2019.

8. Shah, Jay, and Dushyant Dubaria. "Building modern clouds: using docker, kubernetes & Google cloud platform." 2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC). IEEE, 2019.