SUNSTONE

“Social Media Website And Application”

Task - 2

SUNSTONE

Social Media Website And Application

- This document serves as an introductory guide to social media websites and applications. It aims to provide readers with a comprehensive understanding of these digital platforms, their functions, and their significance in contemporary society.

LMS Username	Name	Batch
au310520205057	Sentamil kumaran B	04
au310520205053	Santhosh B	04
au310520205056	Selva kumar K	04
au310520205047	Rakesh R	04

Task - 2

Create UI and implement various components using react

- Split design into components and Higher order Components
- Define structure of the components
- Set the basic ui components with dummy data

Integrate the APIs to frontend to ensure the dynamic feature of website

- Point base api to the severs base url
- Design api calls for each element
- Handle errors in the output
- Render output of apis to different low level components
- Secure content of post apisx

Evaluation Metric:

- 100% Completion of the above tasks

Learning Outcome

- Developing complicated UI using react components
- Using props drilling and context to pass variables
- Getting familiar with different type of api calls

Create UI and implement using react

➢ Folder Structure:

```
src
|-- components
|   |-- app
|   |   |-- app.js
|   |-- common
|   |   |-- app header.js
|   |-- post
|   |   |-- New post.js
|   |-- User
|   |   |-- login.js
|-- higherOrderComponents
|   |-- withDataFetching.js
|-- App.js
|-- index.js
```

➢ Component Definitions:

❑ App Component (App.js):

```
// components/App/App.js
import React from 'react';
import "./App.css";
class App extends Component {  state = {  };
if (this.state.isAuthenticated) {};
return ();
}
}
export default withRouter(App);
```

❑ Common Component (app header.js):

```
// components/common/common.js
import React from 'react';
class AppHeader extends Component {
 let menuItems = [ ];
return ( );
 }
}
export default withRouter(AppHeader);
```

❑ Post Component (NewPost.js):

```
// components/new Post/new Post.js
import React from 'react';
import React, { Component } from "react";import "./newpost.css";
import {  Button,  Modal,  Upload,  Icon,  notification,  Spin,  Input,  Row,  Col};
class NewPost extends Component {
state = {   visible: false,    loading: false,    imageUrl: null,    caption: "",
uploading: false  };
return ();
}}
export default NewPost;
```

User Component (login.js):

```
// components/login/login.js
import React, { Component } from "react";import "./login.css";
import { Form, Input, Button, Icon, Row, Col, notification } from "antd";
import { Link } from "react-router-dom";
import { ACCESS_TOKEN } from "../../common/constants";
import { login } from "../../util/ApiUtil";

class Login extends Component {
state = {};
componentDidMount = () => {    if (this.props.isAuthenticated) {
 this.props.history.push("/");
 }
};
class LoginForm extends Component ();
}
}
render() {
);  }}
export default Login;
```

Higher Order Component (withDataFetching.js):

```
// higherOrderComponents/withDataFetching.js
import React, { useState, useEffect } from 'react';

const withDataFetching = (WrappedComponent, dataSource) => {
  return () => {
    const [data, setData] = useState([]);
    const [loading, setLoading] = useState(true);
    const [error, setError] = useState('');

    useEffect(() => {
      const fetchData = async () => {
        try {
          const response = await fetch(dataSource);
          const result = await response.json();
          setData(result);
        } catch (error) {
          setError('Error fetching data');
        } finally {
          setLoading(false);
        }
```

```
      fetchData();
    }, [dataSource]);

    return <WrappedComponent data={data} loading={loading} error={error}
/>;
  };
};

export default withDataFetching;
```

App Component (App.js):

// App.js

```
class App extends Component {  state = {
 currentUser: null,    isAuthenticated: false,    isLoading: false  };
componentDidMount = () => {
 this.loadCurrentUser();  };
handleLogin = () => {
this.loadCurrentUser();
 this.props.history.push("/");  };
render() {    if (this.state.isLoading) {      return <LoadingIndicator />;    }    let
layoutHeader;
if (this.state.isAuthenticated) {
  layoutHeader = (        <Affix offsetTop={0}>          <Header>
<AppHeader              isAuthenticated={this.state.isAuthenticated}
   currentUser={this.state.currentUser}
onGetUserPosts={this.handleGetUserPosts}
 {...this.props}           />         </Header>       </Affix>      );    }
return ();
}
}
export default withRouter(App);
```

Integrate the API’s to front end to ensure the dynamic feature of website

To integrate APIs into our React frontend, you can use the fetch function or a library like Axios. Below, I'll show you how to modify the components to fetch data from your API endpoints, handle errors, and render the output.

❑ Update API URLs:

In our withDataFetching.js file, update the URLs to point to your server's base URL:

// higherOrderComponents/withDataFetching.js

```
package com.clone.instagram.instapostservice.api;
import com.clone.instagram.instapostservice.model.Post;
import com.clone.instagram.instapostservice.payload.ApiResponse;
import com.clone.instagram.instapostservice.payload.PostRequest;
import com.clone.instagram.instapostservice.service.PostService;
import lombok.extern.slf4j.Slf4j;import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.core.annotation.AuthenticationPrincipal;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.servlet.support.ServletUriComponentsBuilder;
```

```
import java.net.URI;
import java.security.Principal;
import java.util.List;@Slf4j@RestController
public class PostApi {    @Autowired    private PostService
postService;    @PostMapping("/posts")
public ResponseEntity<?> createPost(@RequestBody
PostRequest postRequest){
  log.info("received a request to create a post for image {}",
postRequest.getImageUrl());
  Post post = postService.createPost(postRequest);
URI location =
ServletUriComponentsBuilder            .fromCurrentContextPat
h().path("/posts/{id}")              .buildAndExpand(post.getId(
)).toUriList<Post> posts =
postService.postsByUsername(username);
 log.info("found {} posts for user {}", posts.size(),
username);
     return ResponseEntity.ok(posts);    }
@PostMapping("/posts/in")    public ResponseEntity<?>
findPostsByIdIn(@RequestBody List<String> ids)
{        log.info("retrieving posts for {} ids", ids.size());
 List<Post> posts = postService.postsByIdIn(ids);
  log.info("found {} posts", posts.size());
  return ResponseEntity.ok(posts);
}
}
```

❑ API Calls in Components:

Update RestaurantList.js and Cart.js to use the appropriate API endpoints:

// components/RestaurantList/RestaurantList.js

```
@RestController@Slf4jpublic class UserApi {
@Autowired    private UserService userService;
@PostMapping("/users/followers")
public ResponseEntity<?> follow(@RequestBody FollowRequest
request) {        log.info("received a follow request follow {}
following {}",              request.getFollower().getUsername(),
          request.getFollowing().getUsername());

userService.follow(              User.builder()                    .userI
d(request.getFollower().getId())                    .username(reque
st.getFollower().getUsername())                    .name(request.g
etFollower().getName())                  .profilePic(request.getFol
lower().getProfilePicture())
   .String message = String.format("user %s is following user
%s",              request.getFollower().getUsername(),
request.getFollowing().getUsername());
```

```
log.info(message);
return ResponseEntity.ok(new ApiResponse(true, message));    }
@GetMapping("/users/{username}/degree")
public ResponseEntity<?> findNodeDegree(@PathVariable String username) {

 log.info("received request to get node degree for {}", username);        return
ResponseEntity.ok(userService.findNodeDegree(username));
 }    @GetMapping("/users/{usernameA}/following/{usernameB}")    public
ResponseEntity<?> isFollwoing(
@PathVariable String usernameA,
@PathVariable String usernameB)
@PathVariable String username,
  @RequestParam(value = "page",
defaultValue = AppConstants.DEFAULT_PAGE_NUMBER) int page,
 @RequestParam(value = "size", defaultValue =
AppConstants.DEFAULT_PAGE_SIZE) int size)
 {
 return ResponseEntity.ok(userService.findPaginatedFollowers(username, page,
size));
  }
 @GetMapping("/users/{username}/following")
 public ResponseEntity<?> findFollowing(
@PathVariable String username) {
return ResponseEntity.ok(userService.findFollowing(username));
   }
}
```

❑ Handling Errors:

In the examples above, errors are handled by displaying an error message. You can customize the error handling based on your needs.

❑ Secure Content of Post APIs:

If you have POST APIs that require authentication or authorization, you may need to include authentication tokens in your requests. For example, using the fetch API:

```
const postData = async (url, data) => {
  const token = 'your_auth_token';

  const response = await fetch(url, {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      'Authorization': `Bearer ${token}`,
    },
    body: JSON.stringify(data),
  });

  return response.json();
};
```

```
// Example usage
const dataToPost = { /* your data */ };
postData(`${baseURL}/your-post-endpoint`, dataToPost)
  .then(response => console.log(response))
  .catch(error => console.error(error));
```

Submission
Github

https://github.comSandeSanthosh/
NMDSCET-04

Thank you!