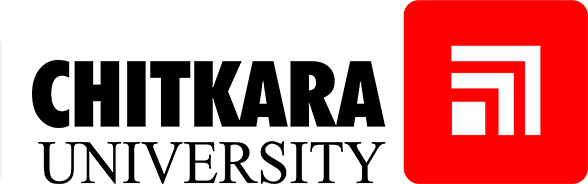
**Front End Engineering- II**

Project Report

Semester- 4 (Batch- 2022)

**Social Media Dashboard**



### Supervised By: Submitted By:

Mrs. Baljit Kaur Jaivardhan Khanna- 429

Jashan Jot Kaur- 436

Jasmine Rathor- 450

**Department of Computer Science and Engineering Chitkara University Institute of Engineering & Technology,**

**Chitkara University, Punjab.**

**Table of Content**

|  |  |  |
| --- | --- | --- |
| **SR. NO.** | **SECTION** | **PAGE NO.** |
| **1.)** | Introduction:   * Background * Objective * Significance | **1-2** |
| **2.)** | Problem statement:   * Problem Definition * Software Requirements * Hardware Requirements | **3** |
| **3.)** | Proposed design/Methodology:   * Home.js * Posts.js * AddPost.js | **4-7** |
| **4.)** | Result | **8-9** |

**Introduction**

The project is aimed at replicating the core functionality and user experience of Instagram, a popular social media platform for sharing photos and videos. The project is made by implementing React JS. Asynchronous fetch methods we have retrieved data from the firebase database and displayed it on our website by integrating it with html and CSS and rendering all that React JS.

**Background:**

In an era where social media has become integral to personal branding, business marketing, and public relations, the need for efficient management tools is more critical than ever. A Social Media Dashboard is an essential tool designed to help users manage multiple social media accounts, streamline their online activities, and derive actionable insights from their social media presence.

**Objective:**

The Instagram clone project is an exciting and challenging endeavour that provides valuable insights into the development of modern social media applications. By replicating and building upon Instagram's successful features, this project not only enhances technical skills but also explores the dynamics of user engagement and community building in the digital age. The end result will be a comprehensive social media platform that embodies the essence of Instagram while offering opportunities for innovation and customization.

**Significance:**

The significance of making a Social Media Dashboard lies in its ability to centralize management, enable data-driven decision-making, enhance audience engagement, support strategic planning, optimize resources, provide a competitive edge, and offer scalability and flexibility. Whether for individual users, businesses, or organizations, a Social Media Dashboard is an indispensable tool for navigating the complexities of modern social media management and achieving sustained success in the digital landscape.

**Problem Statement and Requirements:**

**Definition:**

This project aims to show the use of firebase in order to fetch metadata of provided username from our locally hosted database and display that info on a HTML page rendered page through React. The primary goal is to create a user-friendly and visually appealing platform that allows user to share moments, connects with friends and explore trending content seamlessly.

**Software Requirements:**

The software requirements for the following movie recommendation system encompass the following components:

* Programming Language: JavaScript, CSS, React JS.
* Libraries/Framework:
  + - React.js: Core library for building user interfaces. Reacts component-based architecture makes it suitable for creating dynamic and interactive dashboards.
    - Firebase Authentication: A comprehensive solution for adding authentication to your app. It supports email/password, social login providers, and more.
    - Node.js: A JavaScript runtime built on Chrome's V8 JavaScript engine, often used with Express.js for backend services.
* Integrated Development Environments (IDE):
* Version Control:

**Hardware Requirements:**

The hardware requirements for the movie recommendation system are relatively modest and can vary based on the factors such as dataset and user traffic. The basic hardware setup includes:

* Processor
* Memory RAM
* Storage
* Network connectivity

**Proposed Design/Methodology:**

**1.)Home.js**

* The Home component is the main component that renders the entire home page of the application.
* It uses the useState hook to manage various state variables such as open, openSignin, username, email, password, user, and posts.
* It also uses the useEffect hook to handle side effects, such as listening for changes in the authentication state and fetching posts from the Firebase Firestore database.
* The component renders the app header, modals for user registration and login, and conditionally renders the AddPost component and the list of posts using the Posts component.

**Modal Component (from Material-UI)**

* The Modal component is used to display the user registration and login forms in a modal dialog.
* It is imported from the @material-ui/core package and is used twice in the Home component, once for the signup modal and once for the signin modal.
* The visibility of the modals is controlled by the open and openSignin state variables.
* Button Component (from Material-UI):
* The Button component is used for various buttons in the Home component, such as the signup, signin, and logout buttons.
* It is imported from the @material-ui/core package and is used to provide a consistent and styled button appearance.
* Input Component (from Material-UI):
* The Input component is used for the input fields in the user registration and login forms.
* It is imported from the @material-ui/core package and is used to capture user input for the username, email, and password fields.

**Posts Component:**

* The Posts component is a custom component that represents an individual post in the list of posts.
* It receives props such as postId, user, userName, caption, and imageURL to display the details of each post.
* The Posts component is rendered within the Home component using the map function to iterate over the posts state variable.

**AddPost Component:**

* The AddPost component is a custom component that allows authenticated users to add new posts.
* It is conditionally rendered in the Home component based on whether the user is authenticated and has a display name.
* The AddPost component receives the username prop from the Home component to associate the new post with the authenticated user.
* Firebase Integration:
* The code imports the db and auth objects from the ../firebase module, which are used to interact with the Firebase Firestore database and authentication service.
* The signUp function is used to create a new user account using Firebase authentication and update the user's profile with the provided username.
* The signIn function is used to authenticate the user with the provided email and password using Firebase authentication.
* The useEffect hook is used to listen for changes in the authentication state and fetch posts from the Firebase Firestore database.
* Styling:
* The code uses the makeStyles function from Material-UI to define custom styles for the modals.
* The getModalStyle function is used to set the positioning of the modals.
* The useStyles hook is used to access the custom styles defined using makeStyles.

Overall, the **Home.js** serves as the main entry point of the application, handling user authentication, displaying modals for user registration and login, rendering the list of posts, and allowing authenticated users to add new posts. It utilizes various components from Material-UI for styling and user interface elements, and integrates with Firebase for authentication and database functionality.

**3.)AddPost.js**

* The AddPost component is a functional component that allows users to add a new post to the application.
* It receives the username prop, which represents the username of the user adding the post.
* It uses the **useState** hook to manage state variables such as image, caption, and progress.
* The component renders an input field for selecting an image file, a text field for entering the caption, a progress bar to display the upload progress, and a button to initiate the post upload.
* When the user selects an image file, the **handleChange** function is triggered, which updates the image state with the selected file.
* When the user clicks the "*ADD POST*" button, the handleUpload function is triggered, which initiates the image upload process to Firebase Storage and adds a new post to the Firebase Firestore database.
* The **TextField** component is used for the caption input field.
* It is imported from the @material-ui/core package and receives props such as id, label, variant, onChange, and value to customize its appearance and behavior.
* The **onChange** prop is used to update the caption state whenever the user types in the text field.
* The **Button component** is used for the "ADD POST" button.
* It is imported from the @material-ui/core package and receives props such as variant, color, and onClick to customize its appearance and behavior.
* The **onClick** prop is used to trigger the handleUpload function when the button is clicked.
* **Progress Bar** component renders a progress bar using the <progress> element to display the upload progress.
* The progress state variable is used to set the value of the progress bar, which is updated during the image upload process.

Overall, the **AddPost.js** provides a user interface for adding a new post to the application. It allows users to select an image file, enter a caption, and upload the post to Firebase Storage and Firestore. The component utilizes Material-UI components such as TextField and Button for input and button elements, respectively. It also integrates with Firebase to handle image uploads and store post data in the database.

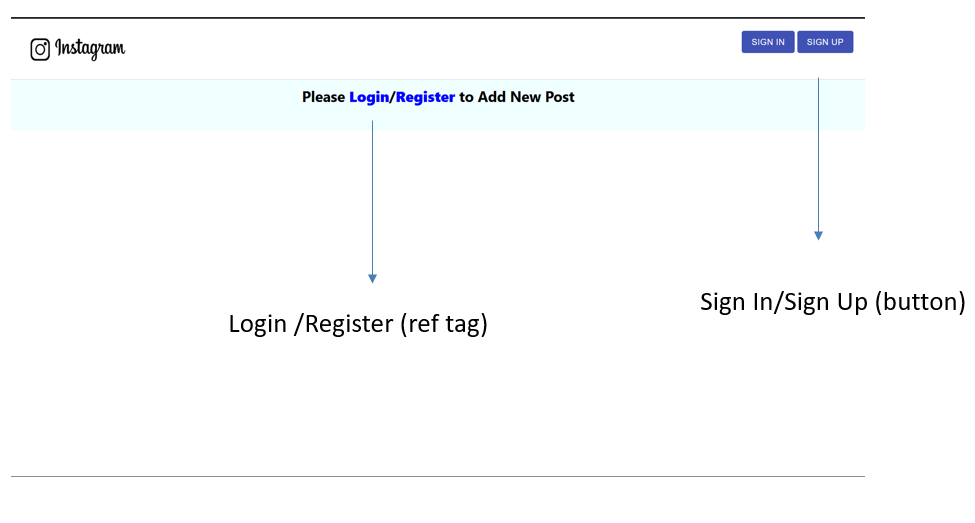
**2.)Posts.js**

* The Posts component is a functional component that represents an individual post in the application.
* It receives props such as postId, user, userName, caption, and imageURL to display the details of each post.
* It uses the **useState** hook to manage state variables such as comments, newComment, editComment, commentID, and show.
* It also uses the **useEffect** hook to fetch comments for the specific post from the Firebase Firestore database whenever the postId changes.
* The component renders the post header with the user's avatar and username, the post image, caption, and the list of comments.
* It provides functionality to add new comments, edit existing comments, and delete comments based on user permissions.
* The **Avatar component** is used to display the user's avatar in the post header.
* It is imported from the @material-ui/core package and receives the alt and src props to set the alternative text and image source, respectively.
* The **Button** component is used for the "POST" and "update" buttons in the comment form.
* It is imported from the @material-ui/core package and receives the disabled prop to control the button's disabled state based on the presence of a comment.
* The **EditIcon** component is used to represent the edit icon for comments.
* It is imported from the @mui/icons-material/Edit package and is rendered conditionally based on the user's permission to edit the comment.
* When clicked, it triggers the handleEdit function to set the state for editing a specific comment.
* The **DeleteForeverIcon** component is used to represent the delete icon for the entire post.
* It is imported from the @mui/icons-material/DeleteForever package and is rendered conditionally based on the user's permission to delete the post.
* When clicked, it triggers the deletion of the post from the Firebase Firestore database.
* The **DeleteOutlineIcon** component is used to represent the delete icon for individual comments.
* It is imported from the @mui/icons-material/DeleteOutline package and is rendered conditionally based on the user's permission to delete the comment.
* When clicked, it triggers the deletion of the specific comment from the Firebase Firestore database.

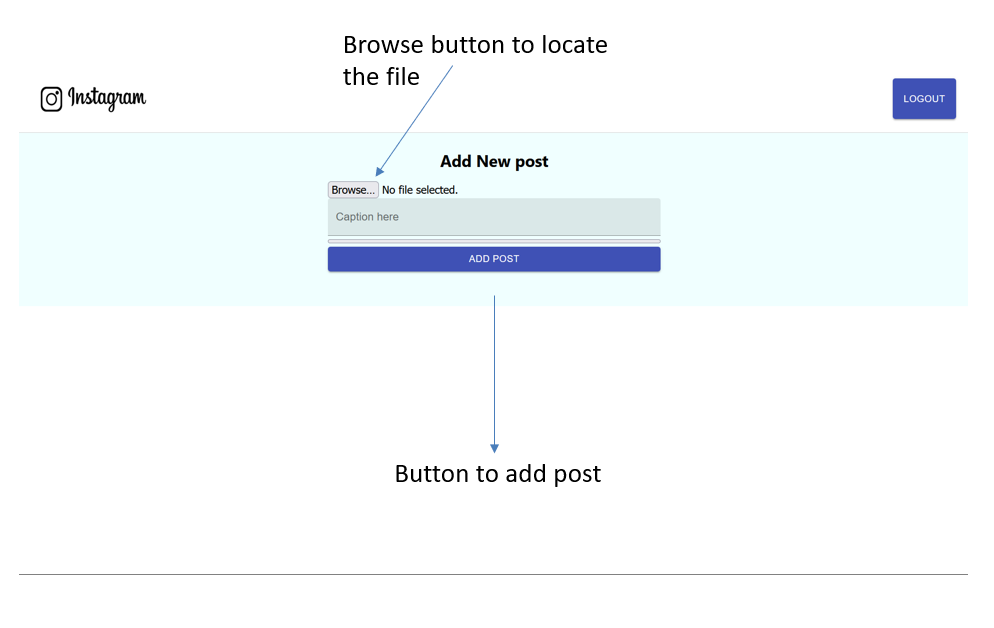
Overall, the **Posts.js** is responsible for rendering an individual post with its details, comments, and user interactions. It utilizes various components from Material-UI and Material-UI Icons for styling and iconography. The component integrates with Firebase Firestore to fetch and manipulate post and comment data in real-time.

**Result**

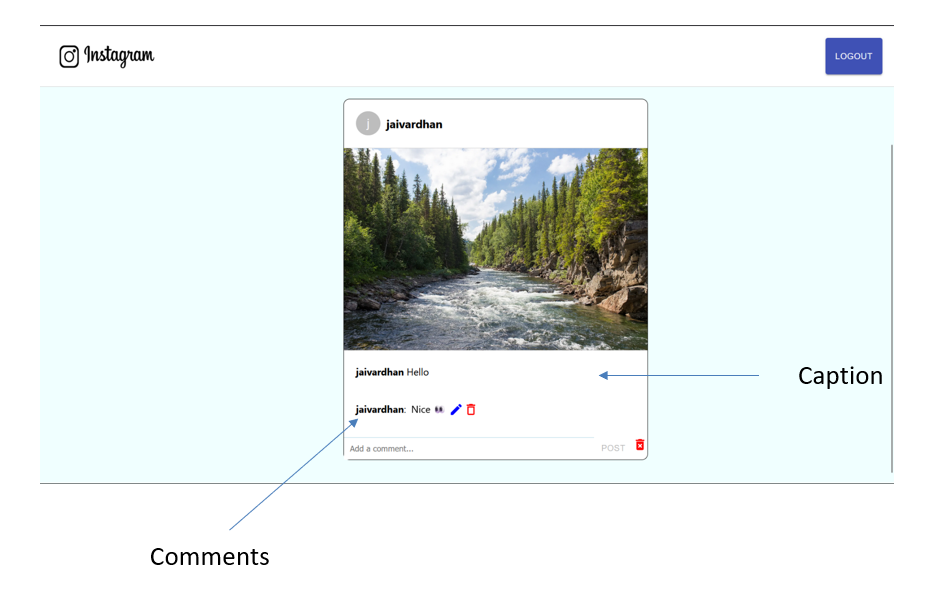
**1.)**

****

**2.)**

****

**3.)**

****