Full Stack Development with MERN

Frontend Development Report

Date	17/7/2024
Team ID	SWTID1719938419
Project Name	Social Media App (MERN)
Maximum Marks	

Project Title: Social Media App (MERN)

Date: 17/7/2024

Prepared by: ALL 4 MEMBERS IN MY TEAM

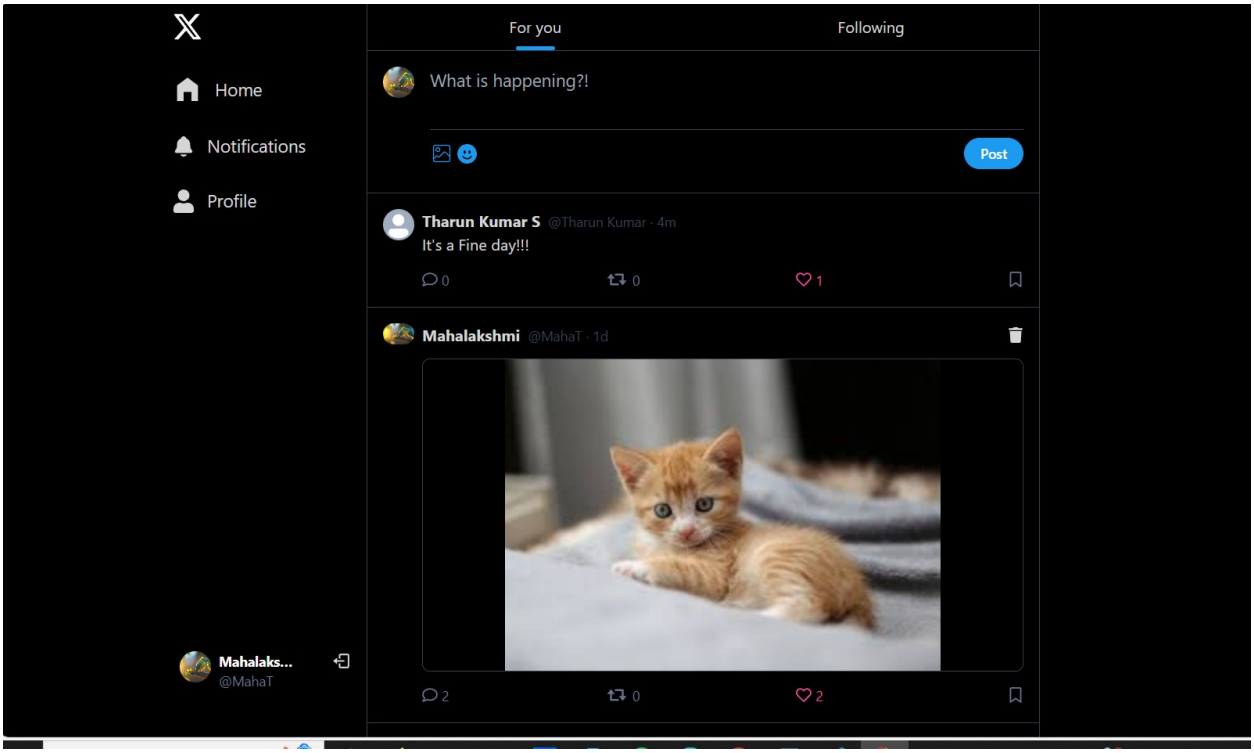
Objective

This is used to serve as a template or starting point for building a social media application. This project likely includes various features commonly found in social media platforms, such as user authentication, profile management, posting, commenting, and possibly real-time updates or notifications.

Technologies Used

- **Frontend Framework:** React.js
- **State Management:** using React's Context API
- **UI Framework/Libraries:** Material-UI(MUI)
- **Api Libraries:** Axios

Project Structure



Frontend Project Structure

The client directory contains the frontend part of the social media application. Here are the main components:

1. **.gitignore:** Specifies which files and directories should be ignored by Git.
2. **package-lock.json:** Automatically generated file that describes the exact version tree that was installed to ensure consistent installs.
3. **package.json:** Lists the project dependencies, scripts, and other configurations.
4. **public:** Contains static files that are directly served by the web server. This usually includes the index.html file, which is the entry point of the React application.
5. **src:** Contains the source code of the React application, including components, styles, and other assets.

Key Components

1. **App.js**

•Purpose:

o This file is responsible for the overall layout and routing of your application. It sets up the structure and navigation between different pages.

•Responsibilities:

o Importing and configuring the routing logic using react-router-dom. o Defining the main layout components such as headers, footers, or sidebars if they are part of the overall structure. o Wrapping the application with any global context providers or state management tools, if used.

2. **/components**

•Purpose:

o This directory contains reusable UI components that can be used across various parts of the application to maintain consistency and modularity.

•Responsibilities:

o Each file or sub-directory within /components represents a reusable piece of UI, such as buttons, modals, forms, etc. oThese components are designed to be used in multiple places throughout the application to ensure a consistent look and feel.

3. **/pages**

•Purpose:

oThis directory contains the different pages of your web application. Each file or sub-directory represents a different view or screen.

•Responsibilities:

o Defining the layout and logic specific to each page. oThese components often use the reusable components defined in the /components directory to build the page UI.

Routing

Routing in App.js

It uses React Router to handle routing. This allows different components to be rendered based on the URL path. This setup is typically done in the App.js file.

Main Routes

1. **/home -** Landing Page oPurpose: The landing page is the entry point of your application. It typically includes an overview of the app, welcome messages, and navigation to other sections. o Component: Home
    a. Example Content: The Home component might display a welcome banner, some introductory information about the app, and links or buttons to sign up or log in.
2. **/dashboard -** User Dashboard o Purpose: The dashboard is where users can view their data and statistics. It usually provides a personalized experience, showing relevant information based on the user's activities and preferences. oComponent: Dashboard
    a. Example Content: The Dashboard component might display user-specific data like recent activity, statistics, charts, and quick links to other features.
3. **/profile -** User Profile Management o Purpose: The profile management page allows users to update their personal information, change their settings, and manage their account details.
    a. Component: Profile

    b. Example Content: The Profile component might include forms for updating user information, changing passwords, and other settings related to the user account. Example Implementation in App.js

Explanation

1. Router: The Router component wraps the application and provides the routing context.
2. Switch: The Switch component ensures that only one route is rendered at a time.

It checks each route in order and renders the first one that matches the current URL.

3. Route: Each Route component maps a URL path to a specific component. For example, the path /home renders the Home component.

State Management (If Applicable)

1. **State Management**: Achieved using React's Context API.
2. **Context Setup**: Create a context and provider to manage the state.
3. **Provider Usage**: Wrap the application with the provider in App.js.
4. **Context Consumption**: Use the context in components to access and update state. This approach ensures that state is managed centrally and can be easily accessed and modified from any component in the application, making the code more organized and maintainable.

Integration with Backend

The frontend communicates with the backend APIs hosted on a specified backend URL. Key endpoints include:

GET /api/data: Retrieves data for display.

POST /api/user/login: Handles user authentication..

User Interface (UI) Design

1. The UI design follows a clean, modern, and responsive design principle. This ensures that the application is user-friendly, aesthetically pleasing, and works well across different devices and screen sizes.

2. Implemented using Material-UI (MUI), a popular React UI framework that provides pre-built, customizable components adhering to Material Design guidelines. This helps in quickly building a consistent and visually appealing user interface.

Third-Party Integrations (If any) NO.