**MERN-Stack Project –** SOCIOPEDIA (Social-Media Website)

Our social media app, designed for seamless connectivity and engagement, offers users a dynamic platform to connect with friends and share their experiences. Users can easily build their social circle by adding friends, fostering a sense of community. The app ensures a personalized experience, allowing users to like, comment, and interact with posts from their network. The intuitive interface facilitates effortless navigation, enabling users to effortlessly add their own posts and curate their profiles with a collection of shared moments. A secure and user-friendly login and signup process, implemented through JSON Web Tokens (JWT), ensures the protection of user data and privacy.

**Frontend Overview:**

1. **Tech Stack**

**Front-end -** React, MUI (Material UI)

**State-Management** – React-Redux-toolkit

1. **React:**

Library Choice: The frontend is built using React, a declarative and efficient JavaScript library for building user interfaces. React allows for the creation of reusable UI components.

1. **Authentication Components:**

Login and Registration Forms: React components for user authentication, including login and registration forms. These components interact with the backend authentication API.

1. **User Profile Components:**

Profile Display: Components for displaying user information, including the user's profile picture, username, bio, and other relevant details.

Edit Profile Form: A form to edit and update the user's profile information.

1. **Feed Components:**

Post Feed: Components responsible for displaying posts in a user's feed, including images, captions, likes, and comments.

Create Post Form: A form for users to create and upload posts, including images.

1. **Friend Management Components:**

Friends List: Displays the user's list of friends with links to their profiles.

Add Friend Component: Allows users to send friend requests and manage their friends.

1. **Interactive Components:**

Like and Comment Buttons: Interactive components for liking posts and adding comments.

Notification Alerts: Components to display notifications for new friend requests, post likes, and comments.

1. **Routing:**

React Router: Implements client-side routing to navigate between different pages and components of the application without refreshing the entire page.

1. **State Management:**

React Context or Redux: Manages the state of the application, ensuring efficient data flow between components. State management is crucial for handling user authentication, post data, and friend lists.

1. **Responsive Design:**

Media Queries: Ensures a responsive design that adapts to various screen sizes and devices, providing a consistent user experience.

1. **Integration with Backend APIs:**

Axios or Fetch: Integration with backend APIs for user authentication, fetching posts, managing friends, and other interactions.

1. **Error Handling:**

Display Error Messages: Components for displaying error messages in case of failed operations, such as unsuccessful login attempts or failed post uploads.

**Backend Overview:**

1. **Tech Stack**

**Backend -** Node, Express

**Database -** Mongo Db

**Authentication -** JWT

1. **Node.js and Express.js:**

Node.js serves as the runtime, and Express.js is the web application framework for handling HTTP requests and responses.

1. **MongoDB and Mongoose:**

MongoDB is used as the NoSQL database to store data, while Mongoose serves as the ODM (Object Data Modeling) library to define schemas and interact with MongoDB.

1. **Models :**
2. **User Model:**

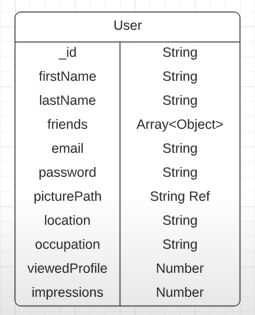
* **Schema Definition:**

Defines the structure of the User entity, including fields such as username, email, password (hashed), profilePicture, bio, etc.

* **Mongoose Model:**

Represents the User entity in the application code.

Manages CRUD operations related to users.



1. **Post Model:**

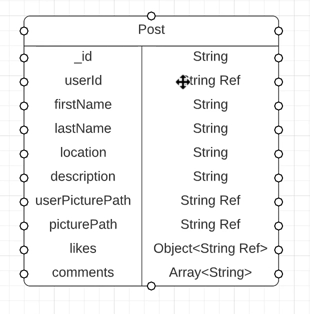
* **Schema Definition:**

Describes the structure of a post, including fields like userId (reference to the user who created the post), image, caption, likes, comments, createdAt, etc.

* **Mongoose Model:**

Represents the Post entity in the application code.

Handles CRUD operations related to posts.



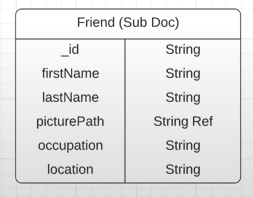
1. **Friend Model:**

* **Schema Definition:**

Represents the relationship between users, including fields like userA, userB, status (pending, accepted, etc.).

* **Mongoose Model:**

Manages friend-related operations, such as sending and accepting friend requests.



1. **Backend APIs**
2. **User Authentication API:**

/api/auth/register (POST): Allows users to register by providing a username, email, and password.

/api/auth/login (POST): Authenticates users using their credentials and returns a JWT upon successful login.

/api/auth/me (GET): Retrieves the user's profile information based on the JWT token.

1. **Post API:**

/api/posts (GET): Fetches a list of posts from the user's friends for the feed.

/api/posts/:postId (GET): Retrieves a specific post by ID.

/api/posts (POST): Creates a new post, allowing users to upload images and add captions.

/api/posts/:postId (PUT): Updates an existing post, e.g., editing captions.

/api/posts/:postId (DELETE): Deletes a post.

1. **Friend Management API:**

/api/friends/send-request (POST): Sends a friend request to another user.

/api/friends/accept-request (POST): Accepts a friend request from another user.

/api/friends/list (GET): Retrieves the list of friends for a user.

Authentication Middleware:

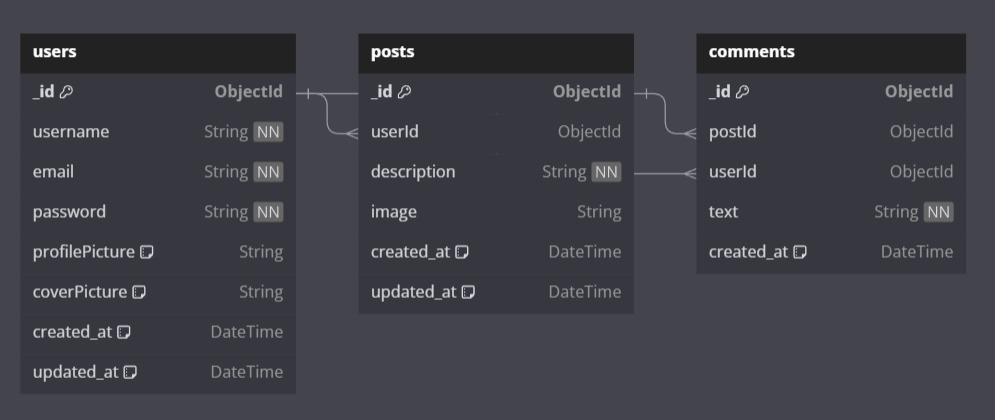
1. **JWT Verification:**

Middleware to verify and decode JWTs attached to protected routes.

Ensures that only authenticated users have access to certain endpoints.

Controllers:

**Flow Diagram :**



**Deployment :** on vercel