**Social media platform**

**P**roject Requirements and Architecture

Tech Stack

**Frontend:** Next.js for server-side rendering and better SEO, with context API or Redux for state management.

**Back-end:**Node.js with Express.js for handling API requests.

**Database**: MongoDB for storing user, post, and notification data. Optionally, Redis can be used to cache frequent data like trending posts.

4. \*\***Real-Time Updates:** Web-socket (using libraries like Socket.io) to enable instant updates for notifications, direct messaging, and other real-time interactions.

### Key Features and Requirements

**1. \*\*User Authentication and Profile Creation\*\***

- \*\***Requirements:\*\***

- User registration and login using email/password or social login (e.g., Google).

- JWT-based authentication for secure session handling.

- Profile creation page where users can set profile pictures, bio, etc.

- \***\*Implementation:\*\***

- Create API routes for registration and login.

- Use MongoDB to store user information (e.g., username, email, bio).

- Implement profile update functionality, including media upload for profile images.

2. \***\*Post, Like, Share, and Comment Functionalities\*\***

- \*\***Requirements:**

- Users can create, edit, delete posts (text, images, and videos).

- Functionality to like, share, and comment on posts.

- \*\*Implementation:\*\*

- MongoDB will store each post with fields like post content, author, likes, comments, and timestamps.

- Use WebSocket to push real-time updates to users when someone likes or comments on their posts.

- Comment replies and threaded conversations can enhance engagement.

3. **\*\*Real-Time Notifications\*\***

**- \*\*Requirements:\*\***

- Notifications for likes, comments, follows, and shares.

- Real-time push notifications.

- \*\*Implementation:\*\*

- Set up a `notifications` collection in MongoDB to store notification data.

- Use WebSocket to emit notifications to users.

- Allow users to mark notifications as read/unread.

4. **\*\*Explore Feature (Trending Posts or Users)\*\***

**- \*\*Requirements:\*\***

- Page showing trending posts or users based on likes, shares, or recent activity.

- \*\*Implementation:\*\*

- Create an aggregation pipeline in MongoDB to fetch top-performing posts.

- Caching frequent queries with Redis can improve performance on this feature.

**5. \*\*Media Uploads (Images, Videos)\*\***

- \*\*Requirements:\*\*

- Users should be able to upload images and videos as part of their posts.

- \*\*Implementation:\*\*

- Integrate a service like Cloudinary or Amazon S3 for media storage.

- On the frontend, handle file uploads and display media previews before posting.

**6. \*\*Direct Messaging Feature\*\***

- **\*\*Requirements:\*\***

**- Real-time messaging system where users can chat with each other.**

**- \*\*Implementation:\*\***

**- Create a `messages` collection in MongoDB for storing message history.**

**- Use WebSocket (Socket.io) for real-time chat updates.**

**- Display online/offline status for users to indicate availability.**

**### Backend API Structure**

**1. \*\*Authentication API\*\***

- \*\*Endpoints:\*\* `/api/register`, `/api/login`, `/api/logout`, `/api/profile`

- \*\*Functions:\*\* User registration, login/logout, profile management

2. \*\*Post API\*\*

- \*\*Endpoints:\*\* `/api/posts`, `/api/posts/:id`, `/api/posts/:id/like`, `/api/posts/:id/comment`

- \*\*Functions:\*\* CRUD operations for posts, like and comment on posts

3. \*\*Notification API\*\*

- **\*\*Endpoints:\*\* `/api/notifications`, `/api/notifications/:id/read`**

**- \*\*Functions:\*\* Fetch and mark notifications as read**

**4. \*\*Messaging API\*\***

- \*\*Endpoints:\*\* `/api/messages/:userId`, `/api/messages`

- \*\*Functions:\*\* Send and fetch messages

**5. \*\*Explore API\*\***

- \*\*Endpoints:\*\* `/api/explore`

- \*\*Functions:\*\* Fetch trending posts and users

**Database Models**

1. **\*\*User Model\***\*

- Fields: `username`, `email`, `password`, `bio`, `profileImage`, `followers`, `following`

**2. \*\*Post Model\*\***

- Fields: `authorId`, `content`, `media`, `likes`, `comments`, `shares`, `createdAt`

**3. \*\*Comment Model\*\***

- Fields: `postId`, `authorId`, `content`, `likes`, `createdAt`

4. **\*\*Notification Model\*\***

- Fields: `userId`, `type` (e.g., like, comment, follow), `message`, `isRead`, `createdAt`

5. \*\*Message Model\*\*

- Fields: `senderId`, `receiverId`, `message`, `createdAt`, `status`

For otp and email twilio firebase