1. Overview

- **Application Name:** Web Chat Application
- **Primary Technologies:**
  - **Front-end:** React
  - **Back-end:** Laravel (PHP)
  - **Database:** PostgreSQL
  - **Real-time communication:** WebSockets (e.g., Laravel WebSockets or Pusher)
- **Hosting/Deployment:**
  - React: Render (static site or Node service)
  - Laravel: Docker + Render (web service)
  - PostgreSQL: Managed PostgreSQL instance on Render

Purpose: Provide a user-friendly, secure, and real-time chat system supporting:

1. **Private Chats** between different users.
2. **Personal Chat** (sending messages to oneself, often used as a quick note-taking feature).
3. **File Sharing** (images, videos, documents).
4. **File Previewing** within the chat interface.
5. **User Profile Management** (edit username, avatar, bio, etc.).

---

2. Requirements

2.1 Functional Requirements

1. **User Registration & Login**
   - Secure registration with email verification (optional).
   - Login with email/password or third-party (optional).
2. **Chat Management**
   - Users can see a list of ongoing private chats.
   - Users can initiate a new private chat by selecting another user from a directory/list.
   - Personal (self) chat is available to store personal notes.
3. **Messages**
   - Send text messages.
   - Real-time messaging updates via WebSockets (no page refresh needed).
   - Mark messages as read/unread.
4. **File Attachments**
   - Upload images, videos, or documents in chat.
   - Basic file preview (if supported by the file type).
   - Restrict file size to avoid large, unwanted uploads.
   - Properly store files in a designated file storage (e.g., local storage or cloud like S3).

5. **User Profile**
    - View and edit own profile details: name, profile picture, status/bio.
    - Profile picture upload and storage.
    - Ability to change password.
6. **Search**
    - Search users by name or email to initiate chat

---

3. Technology Stack

1. **Front-End (React)**
    - React (hooks and functional components).
    - WebSocket client (native WebSocket, Socket.IO client, or Pusher JS).
    - UI libraries (Material UI, Ant Design, or Bootstrap) for a consistent design.
2. **Back-End (Laravel)**
    - Laravel 10+ (or latest stable).
    - Laravel WebSockets package (beyondcode/laravel-websockets) or Pusher integration for real-time.
    - Laravel built-in file storage system for attachments.
3. **Database (PostgreSQL)**
    - Hosted on Render.
    - Use standard Laravel migrations.
4. **Deployment**
    - **React**: Deployed as a static front-end on Render or as a Node service.
    - **Laravel**: Containerized with Docker, then deployed on Render.
    - **PostgreSQL**: Managed instance on Render.

---

4. Features Breakdown

4.1 Private Chats

- **Endpoint**: `POST /api/chats` to create a new private chat.
- **Database**:
    - `chats` table has two user references (user1_id, user2_id).
    - Check if a chat already exists between these two users before creating a new one.

4.2 Personal Chat

- A special "chat" record where `user1_id` and `user2_id` are the same (representing user -> self).

4.3 Sending & Receiving Messages

- **Endpoint**: `POST /api/messages`
    - Fields: `chat_id`, `sender_id`, `message_text`, `attachment_id` (optional).
    - Broadcast the new message to chat participants via WebSockets.

4.4 File Attachments

- **Endpoint**: `POST /api/attachments`
    - Store file in `storage/app/public` or cloud (depending on config).
    - Return `attachment_id` or file URL.
- **messages** table can reference `attachment_id`.
- The front-end displays a preview for supported file types.

4.5 User Profile Editing

- **Endpoint**: `PUT /api/users/{id}`
    - Change username, avatar, or other details.
- Avatars or profile pictures are also managed as file uploads to unify approach.

4.6 WebSockets / Real-Time

- **Server**: Use **Laravel WebSockets** package or Pusher:
    1. Install and configure the WebSockets package in Laravel.
    2. Create event classes (e.g., `MessageSent`) to broadcast when a new message is saved.
- **Client**:
    1. Establish connection using a WS library.
    2. Subscribe to channels (e.g., `chat.{chatId}`).
    3. Update local state upon receiving new messages.

---

5. Database Schema (Simplified)

Below is a possible schema using **PostgreSQL**.

1. **users**
    - `id` (PK, UUID or auto-increment)
    - `name` (string)
    - `email` (string, unique)
    - `password` (string)
    - `avatar_url` (string, nullable)
    - `bio` (text, nullable)
    - `created_at`, `updated_at` (timestamps)
2. **chats**
    - `id` (PK, auto-increment)

    - `user1_id` (FK -> users.id)
    - `user2_id` (FK -> users.id)
    - `created_at`, `updated_at`
3. **messages**
    - `id` (PK, auto-increment)
    - `chat_id` (FK -> chats.id)
    - `sender_id` (FK -> users.id)
    - `content` (text, nullable if it's only an attachment)
    - `attachment_id` (FK -> attachments.id, nullable)
    - `read_at` (datetime, nullable)
    - `created_at`, `updated_at`
4. **attachments**
    - `id` (PK, auto-increment)
    - `file_path` (string)
    - `file_type` (string)
    - `created_at`, `updated_at`

---

6. Summary

This specification outlines a **simple but robust real-time chat application** using React on the front-end, Laravel on the back-end, and PostgreSQL as the database. File attachments, user profiles, personal/self chat, and real-time message delivery over WebSockets form the core functionalities. Hosting on Render for both front-end and back-end, with Docker for Laravel, ensures a straightforward and scalable deployment setup.