Web Chat - Specification

1. Overview

Application Name: Web Chat

Primary Technologies:

- **Front-end:** React, JavaScript, HTML5, CSS3, Bootstrap 5
- **Back-end:** Laravel (PHP)
- **Database:** PostgreSQL
- **Real-time Communication:** WebSockets (e.g., Laravel WebSockets or Pusher)
- **Hosting/Deployment:**
    - **React:** Render (static site)
    - **Laravel:** Docker + Render (Render does not support Laravel PHP directly, so Docker is used to run the backend.)
    - **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 for note-taking).
3. **File Sharing** (images, videos, documents).
4. **File Previewing** within the chat interface.
5. **User Profile Management** (edit username, avatar, bio, etc.).

Architecture:

- **Client-Server Architecture**
- **Single Page Application (SPA)**

Supported Browsers:

- Google Chrome
- Mozilla Firefox
- Microsoft Edge
- Safari

---

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.
    - Personal (self) chat 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 file type).
    - Restrict file size to avoid large uploads.
    - Store files in local storage or cloud (e.g., S3).
5. **User Profile**
    - View and edit 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

3.1 Front-End (React)

- React (hooks and functional components)
- WebSocket client (native WebSocket, Socket.IO client, or Pusher JS)
- UI libraries (Bootstrap or similar)
- HTML5, CSS3, JavaScript

3.2 Back-End (Laravel)

- Laravel 10+ (or latest stable)
- Laravel WebSockets package (beyondcode/laravel-websockets) or Pusher
- Laravel built-in file storage system

3.3 Database (PostgreSQL)

- Hosted on Render
- Use standard Laravel migrations

3.4 Deployment

- **React:** Deployed as a static front-end on Render.
- **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).
    - Prevent duplicate chats between the same users.

4.2 Personal Chat

- A special “chat” record where user1_id and user2_id are the same (self-chat).

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 (configurable).
    - Return attachment_id or file URL.
- messages table references 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.
- Profile pictures managed as file uploads.

4.6 WebSockets / Real-Time Messaging

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

---

5. Database Schema (Simplified)

5.1 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)

5.2 chats

- id (PK, auto-increment)
- user1_id (FK → users.id)
- user2_id (FK → users.id)
- created_at, updated_at

5.3 messages

- id (PK, auto-increment)
- chat_id (FK → chats.id)

- sender_id (FK → users.id)
- content (text, nullable if attachment only)
- attachment_id (FK → attachments.id, nullable)
- read_at (datetime, nullable)
- created_at, updated_at

5.4 attachments

- id (PK, auto-increment)
- file_path (string)
- file_type (string)
- created_at, updated_at

---

6. Timeline

Phase 1: The project theme and specification (Week 1-4)

- **Set up Git repository and deployment environments.**
- **Project theme.**
- **Specification.**

Phase 2: Frontend Development (Week 4-6)

- **Implement UI components with React.**
- **Implement chat list, messaging, and file previews.**
- **Profile editing.**

Phase 3: Backend Development (Week 6-9)

- **Set up Laravel, PostgreSQL, and WebSockets.**
- **Implement user authentication and chat models.**
- **Configure file storage.**

Phase 4: Integration & Deployment (Week 9-11)

- **Connect frontend and backend.**
- **Test API endpoints and WebSocket connections.**
- **Deploy on Render and perform debugging.**

Phase 5: Final Testing & Presentation (Week 12-13)

- **Unit and integration testing.**

- **Optimize performance and security.**
- **Present the project and collect feedback.**

---

7. 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.