Documentation Technique ChatApp

1. Présentation du Projet

Le projet ChatApp est une application web conçue pour permettre aux utilisateurs d'échanger en temps réel au sein de salles thématiques, en fonction de leurs intentions relationnelles (amitié, rencontres, mariage, etc.). Le frontend de l'application est développé en React.js, offrant une interface utilisateur moderne, intuitive et responsive. Le backend repose sur Node.js avec Express.js, assurant la gestion des API, de l'authentification, des profils utilisateurs et de la modération. Les données des utilisateurs, des messages et des salles sont stockées dans une base de données MySQL, garantissant robustesse et cohérence. Enfin, la communication en temps réel est assurée grâce à Socket.IO, permettant un échange instantané de messages au sein des différentes salles.

2. Architecture de l'Application

2.1 Schéma Général

[React (Client)] ⇄ [Express (API REST + WebSocket)] ⇄ [MySQL (Base de données)]

- Client (React) : SPA avec routage, formulaires, gestion du state, appels API.
- Serveur (Express.js) : API REST pour les ressources (utilisateurs, salles, messages), WebSocket via Socket.IO pour les messages temps réel.
- Base de données (MySQL) : stockage des utilisateurs, messages, salles, blocages, etc.

3. Structure de Projet

3.1 Backend (Express)

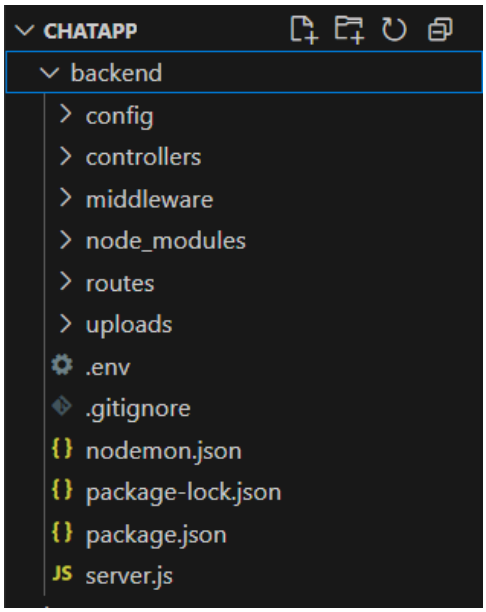

Figure 1 : structure du backend (express)

Le backend de l'application ChatApp est une application Node.js, construite avec le framework Express.js. Sa structure de dossiers est organisée de manière modulaire pour séparer les différentes responsabilités (separation of concerns), suivant une architecture inspirée du modèle MVC (Modèle-Vue-Contrôleur).

- config/ : Contient les fichiers de configuration de l'application. On y trouve la configuration de la base de données (URI de connexion, options), les clés d'API, ou d'autres paramètres globaux.
- controllers/ : Héberge la logique métier de l'application. Chaque fichier de ce dossier est responsable de traiter les requêtes entrantes pour une ressource spécifique, d'interagir avec la base de données (via les modèles) et de renvoyer une réponse au client.
- middleware/ : Contient les fonctions intermédiaires (middlewares). Ces fonctions s'exécutent entre la réception de la requête et son traitement par le contrôleur final.
- node_modules/ : Répertoire généré automatiquement par npm. Il contient toutes les dépendances (bibliothèques et frameworks) du projet.
- routes/ : Définit les points d'entrée (endpoints) de l'API. Chaque fichier de route mappe une URL à une fonction spécifique dans un contrôleur.
- uploads/ : Dossier destiné à stocker les fichiers téléversés par les utilisateurs, comme les avatars, les images partagées dans le chat ou d'autres documents.
- .env : Fichier contenant les variables d'environnement. Il stocke des informations sensibles ou spécifiques à l'environnement (développement, production) comme les identifiants de base de données, les clés secrètes pour les tokens, le port du serveur, etc.
- .gitignore : Fichier de configuration pour Git qui spécifie les fichiers et dossiers à ne pas suivre. Il inclut node_modules/, .env, les logs, et les fichiers de builds.
- nodemon.json : Fichier de configuration pour l'utilitaire nodemon. Il définit des options spécifiques pour le redémarrage automatique du serveur lors du développement.
- package-lock.json : Fichier généré automatiquement qui verrouille les versions exactes de chaque dépendance. Il garantit que tous les développeurs et les environnements de déploiement installent exactement les mêmes versions des paquets, évitant ainsi les conflits.

- package.json : Le fichier manifeste du projet Node.js. Il contient les métadonnées du projet (nom, version), la liste des dépendances (dependencies et devDependencies) et les scripts personnalisés (npm start, npm run dev, npm test).
- server.js : Le point d'entrée principal de l'application backend. C'est ce fichier qui initialise le serveur web, configure les middlewares globaux, connecte les routes, se connecte à la base de données et met le serveur en écoute sur un port défini.

3.2 Frontend (React)

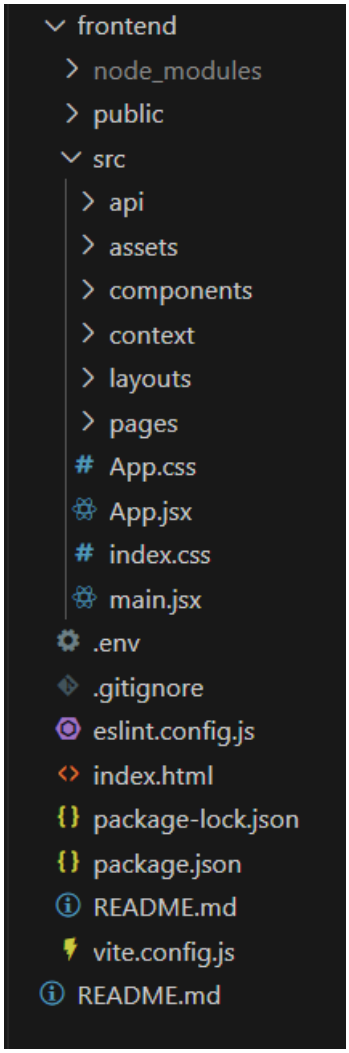

Figure 2: Structure du frontend

Le frontend de l'application est une Single Page Application (SPA) développée avec la bibliothèque React. Le projet est configuré et optimisé à l'aide de l'outil de build moderne Vite. La structure des dossiers est conçue pour être modulaire, évolutive et facile à maintenir.

- node_modules/ : Contient toutes les dépendances du projet (React, bibliothèques de composants, etc.), gérées par npm ou yarn. Ce dossier est ignoré par le contrôle de version.
- public/ : Contient les fichiers statiques qui ne sont pas traités par le processus de build de Vite. Ils sont servis tels quels.
- src/ : Le cœur de l'application. Ce répertoire contient tout le code source qui sera compilé et optimisé par Vite.
    - api/ : Centralise la logique de communication avec le backend. On y trouve des fonctions qui effectuent des requêtes HTTP (avec axios ou fetch) vers les points d'entrée de l'API.
    - assets/ : Contient les ressources statiques qui sont importées directement dans les composants, comme les images, les polices de caractères ou les icônes SVG. Contrairement au dossier public, ces fichiers sont traités et optimisés par Vite.
    - components/ : Répertoire fondamental dans une application React. Il contient des composants d'interface utilisateur (UI) réutilisables. L'objectif est de créer des briques logicielles indépendantes pour construire l'interface.
    - context/ : Utilisé pour la gestion d'état global avec l'API Context de React. Permet de partager des données (comme l'état d'authentification de l'utilisateur, le thème de l'application) à travers l'arborescence des composants sans avoir à passer des props à chaque niveau ("prop drilling").
    - layouts/ : Contient des composants qui définissent la structure générale des pages.
    - pages/ : Contient les composants qui représentent les différentes pages de l'application. Chaque page assemble divers composants et layouts pour former une vue complète.
    - App.css : Feuille de style CSS pour le composant principal App.jsx.
    - App.jsx : Le composant racine de l'application React. C'est lui qui orchestre le routage et affiche les différentes pages et layouts.
    - index.css : Fichier CSS global pour l'ensemble de l'application. Iil contient les styles de base, les variables CSS, et les "resets".

- main.jsx : Le point d'entrée de l'application. C'est ce script qui "monte" le composant App dans le DOM, typiquement dans la balise du fichier index.html.
- .env : Fichier pour les variables d'environnement côté clien. Avec Vite, les variables est préfixées par VITE_ pour être accessibles dans le code.
- .gitignore : Spécifie les fichiers et dossiers à ignorer par Git (ex: node_modules/, dist/, .env).
- eslint.config.js : Fichier de configuration pour ESLint, un outil d'analyse de code statique qui aide à maintenir une qualité de code cohérente et à détecter les erreurs potentielles.
- index.html : Le seul fichier HTML de l'application. Il sert de "coquille" dans laquelle l'application React est injectée par Vite.
- package-lock.json : Verrouille les versions des dépendances pour garantir une installation cohérente sur toutes les machines.
- package.json : Fichier manifeste du projet. Liste les dépendances et les scripts (npm run dev, npm run build).
- README.md : Fichier de documentation principal du projet, expliquant comment l'installer et le lancer.
- vite.config.js : Fichier de configuration pour Vite. Permet de personnaliser le serveur de développement, le processus de build, de configurer des plugins, des proxys pour les requêtes API, etc.

4. Base de Données (MySQL)

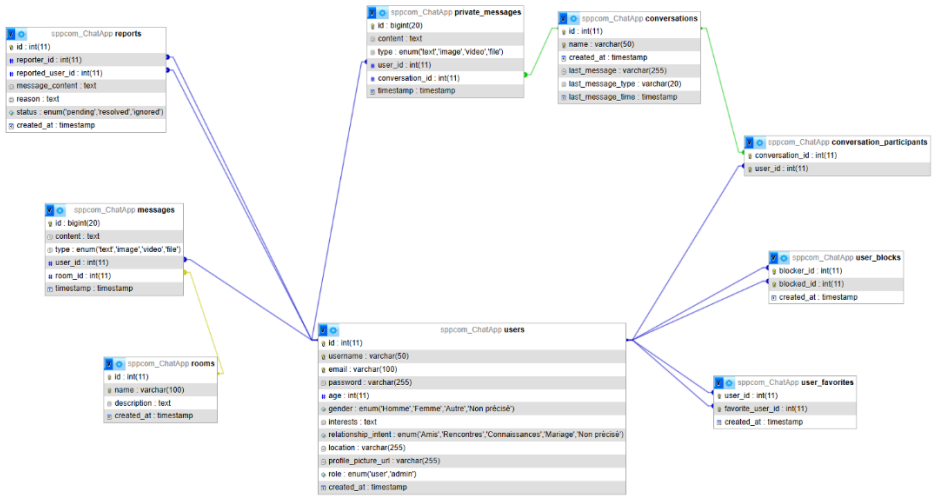

Figure 3 : Schéma de la base de donnée

Le schéma de la base de données de l'application CHATAPP est conçu pour gérer les utilisateurs, leurs interactions sociales, ainsi que deux systèmes de messagerie distincts : des salons de discussion publics (rooms) et des conversations privées (conversations). La structure est centrée autour de la table users.

- **Table 1 : users**

C'est la table centrale de l'application. Elle stocke toutes les informations relatives au profil et à l'authentification des utilisateurs.

- **Objectif :** Gérer les comptes utilisateurs et leurs données de profil.
- **Colonnes Clés :**
    - id : (Clé primaire) Identifiant numérique unique pour chaque utilisateur.
    - username, email : Identifiants uniques pour la connexion et la communication.
    - password : Mot de passe de l'utilisateur, qui doit être stocké sous forme de hash sécurisé.
    - gender, age, interests, relationship_intent, location : Champs de profil utilisés pour le matching et la découverte d'autres utilisateurs. Les champs enum standardisent les options possibles.
    - role : (enum('user', 'admin')) Définit les permissions de l'utilisateur au sein de l'application.
    - created_at : Horodatage de la création du compte.
- **Table 2 : conversations**
    - **Objectif :** Représente une session de chat privée.
    - **Colonnes Clés :**
        - id : (Clé primaire) Identifiant unique de la conversation.
        - last_message_varchar, last_message_type, last_message_time : Colonnes de dénormalisation pour optimiser les performances. Elles permettent d'afficher un aperçu de la dernière activité dans la liste des conversations sans avoir à joindre la table private_messages.
- **Table 3 : conversation_participants (Table de Jonction)**
    - **Objectif :** Associer les utilisateurs aux conversations. Elle met en place une relation **plusieurs-à-plusieurs (Many-to-Many)** entre les tables users et conversations.

- **Colonnes Clés :**
  - conversation_id : (Clé étrangère) Fait référence à conversations.id.
  - user_id : (Clé étrangère) Fait référence à users.id.

- **Table 4 : private_messages**
  - **Objectif :** Stocker chaque message envoyé dans une conversation privée.
  - **Colonnes Clés :**
    - id : (Clé primaire) Identifiant unique du message.
    - content : Le contenu textuel du message.
    - type : (enum('text','image','video','file')) Spécifie le type de contenu du message.
    - user_id : (Clé étrangère) L'expéditeur du message, référence users.id.
    - conversation_id : (Clé étrangère) La conversation à laquelle ce message appartient, référence conversations.id.

- **Table 5 : rooms**
  - **Objectif :** Définir les salons de discussion publics.
  - **Colonnes Clés :**
    - id : (Clé primaire) Identifiant unique du salon.
    - name, description : Informations descriptives sur le salon.

- **Table 6 : messages**
  - **Objectif :** Stocker chaque message envoyé dans un salon public.
  - **Colonnes Clés :**
    - id : (Clé primaire) Identifiant unique du message.
    - user_id : (Clé étrangère) L'expéditeur du message, référence users.id.
    - room_id : (Clé étrangère) Le salon auquel ce message appartient, référence rooms.id.

- **Table 7 : user_favorites**
  - **Objectif :** Gérer une liste de "favoris" ou "d'amis" pour chaque utilisateur.

- **Colonnes Clés :**
  - user_id : (Clé étrangère) L'utilisateur qui ajoute un favori.
  - favorite_user_id : (Clé étrangère) L'utilisateur qui est ajouté en favori.

- **Table 8 : user_blocks**
  - **Objectif :** Gérer la fonctionnalité de blocage entre utilisateurs.
  - **Colonnes Clés :**
    - blocker_id : (Clé étrangère) L'utilisateur qui initie le blocage.
    - blocked_id : (Clé étrangère) L'utilisateur qui est bloqué.

- **Table 9 : reports**
  - **Objectif :** Stocker les signalements faits par les utilisateurs à des fins de modération.
  - **Colonnes Clés :**
    - reporter_id : (Clé étrangère) L'utilisateur qui a fait le signalement.
    - reported_user_id : (Clé étrangère) L'utilisateur qui est signalé.
    - message_content, reason : Le contexte et la raison du signalement.
    - status : (enum('pending','resolved','ignored')) Permet de suivre le traitement du signalement par les modérateurs.

5. Fonctionnalités Clés

5.1. Gestion des Utilisateurs et des Profils

Fonctionnalité	Description
Inscription sécurisée	Formulaire d'inscription avec validation des champs et hashage des mots de passe avec bcrypt.
Connexion / Authentification	Utilisation de JWT pour générer un token d'accès sécurisé après connexion.
Profil utilisateur	Informations personnelles modifiables : nom d'utilisateur, âge, sexe, photo, localisation (optionnelle), objectifs relationnels, centres d'intérêt.
Recherche d'utilisateurs	Système de recherche (par pseudo, intérêt ou intention relationnelle).
Blocage d'utilisateurs	Un utilisateur peut bloquer un autre pour éviter de recevoir ses messages.

5.2. Gestion des Salles Thématiques

Fonctionnalité	Description
Affichage dynamique des salles	Liste des salles avec titre, description et nombre d'utilisateurs connectés.
Création/Mise à jour/Suppression de salle	Accessible uniquement aux administrateurs via l'interface d'administration.
Salles thématiques	Catégories prédéfinies : Amis, Rencontres, Connaissances, Mariage, etc.
Rejoindre une salle	Les utilisateurs peuvent changer de salle librement en fonction de leurs intentions.

5.3. Tchat en Temps Réel (Socket.IO)

Fonctionnalité	Description
Tchat public par salle	Tous les utilisateurs d'une salle peuvent voir les messages diffusés en temps réel.
Affichage clair des messages	Chaque message affiche : pseudo, contenu et heure d'envoi.
Envoi de messages privés *(optionnel)*	Permet à deux utilisateurs dans une même salle d'échanger discrètement.
Support d'émojis *(optionnel)*	Les utilisateurs peuvent insérer des émojis dans les messages.
Indicateur de connexion/déconnexion	Affichage des notifications lors de l'entrée ou sortie d'un utilisateur (optionnel).

5.4. Interface Utilisateur (React)

Fonctionnalité	Description
Navigation fluide	SPA (Single Page App) avec React Router. Navigation entre salle, profil, recherche, etc.
Design responsive	Interface adaptée aux téléphones, tablettes et ordinateurs (media queries ou Tailwind CSS).
Interface de tchat moderne	Messages organisés, champs d'envoi clairs, avatars visibles, boutons intuitifs.

5.5. Modération et Sécurité

Fonctionnalité	Description
Modération par signalement *(optionnel)*	Les utilisateurs peuvent signaler un message inapproprié.
Bannissement des utilisateurs	Les admins peuvent bannir temporairement ou définitivement un utilisateur sans voir ses infos privées.

Fonctionnalité	Description
Logs de modération *(optionnel)*	Historique des utilisateurs bannis (raison, date, durée).
Filtrage de contenu inapproprié	Mots interdits remplacés ou bloqués automatiquement à l'envoi.

5.6. Performances et Scalabilité

Fonctionnalité	Description
Scalabilité backend	Architecture Express.js modulaire, facile à adapter à une base de données plus large.
Optimisation frontend	Chargement progressif des messages, lazy loading des composants lourds.
Gestion efficace des connexions Socket.IO	Reconnexion automatique, nettoyage à la déconnexion, gestion des rooms dynamiques.

6. Technologies Utilisées

ChatApp repose sur une architecture moderne Full Stack JavaScript, combinant React pour le frontend, Express.js pour le backend, et MySQL pour la base de données relationnelle.

Frontend : React.js

Technologie	Rôle
React.js	Bibliothèque JavaScript pour construire l'interface utilisateur (SPA). Utilisée pour le routage, le rendu conditionnel, les hooks (useState, useEffect) et la gestion d'état.
React Router	Pour la navigation dynamique entre les pages/salles sans rechargement complet.
Axios	Bibliothèque pour effectuer des appels API vers le backend Express.
Socket.IO Client	Pour se connecter au serveur WebSocket et gérer le tchat en temps réel.
React Context API ou Redux (optionnel)	Pour la gestion globale de l'état utilisateur (connexion, salle active, etc.).

Backend: Node.js + Express.js

Technologie	Rôle
Node.js	Environnement d'exécution JavaScript côté serveur.
Express.js	Framework léger pour créer l'API REST (inscription, connexion, gestion des salles et des profils).
JWT (JSON Web Token)	Pour l'authentification et la gestion sécurisée des sessions utilisateurs.
Bcrypt.js	Pour le hachage des mots de passe avant stockage en base de données.
Express Validator	Pour la validation et la sanitation des données côté serveur.
CORS	Middleware pour autoriser les requêtes du frontend vers l'API.
Socket.IO	Pour la communication en temps réel (envoi/réception de messages dans les salles).

Base de Données : MySQL

Composant	Rôle
MySQL	Système de gestion de base de données relationnelle pour stocker les utilisateurs, messages, salles, bans, etc.
MySQL2 (NPM)	Pilote Node.js performant pour se connecter à MySQL via Express.
Sequelize (optionnel)	ORM permettant de gérer les modèles et migrations en JavaScript.
phpMyAdmin	Outils d'administration et d'exploration de la base de données.

Outils et Environnement

Outil	Utilisation
Visual Studio Code	Éditeur de code avec extensions React, ESLint, Prettier.
Postman	Pour tester les routes API Express en local.
Git & GitHub	Pour le contrôle de version et l'hébergement du code.
Dotenv	Pour la gestion des variables d'environnement sensibles (.env).
Nodemon	Pour recharger automatiquement le serveur en développement.
Webpack / Vite / Create React App	Pour le bundling et le lancement de l'application React.

Déploiement

Plateforme	Rôle
Render	Hébergement du backend Express.
Render	Hébergement du frontend React.
cPanel	Hébergement de la base de données MySQL

7. Authentification

- JWT (JSON Web Token) pour sécuriser les routes.
- Bcrypt.js pour hacher les mots de passe.
- Middleware auth.js pour vérifier les jetons.

8. Sécurité

- Validation côté client + serveur
- Protection contre XSS / injection SQL avec express-validator
- CORS configuré
- Mots de passe hashés avec Bcrypt
- Gestion des utilisateurs bannis

9. Améliorations Futures

- Appel vocal et video
- Signalement de messages
- Notification en temps réel (Toast ou Push)
- Application mobile (React Native ou PWA)
- Thèmes sombres/clairs
- Historique de discussion par utilisateur

10. Maintenance

La maintenance de ChatApp comprend les tâches suivantes :

• Surveillance du serveur et de l'application.

• Correction des bogues.

• Mises à jour de sécurité.

• Mises à jour des fonctionnalités.

• Gestion de la base de données (par exemple, sauvegardes, optimisation).

• Gestion des logs.

11. Licence

12. Contact

Pour toute question ou demande de renseignements, veuillez contacter : ChatApp@gmail.com |+07 77 07 61 86