

# Rapport du Projet de développement d'application



## RetroChat

**Nom des étudiants :** Noam Boutounas & Evan Lebeau

**Date de soumission :** Décembre 2024

**Cours :** Développement d'Application

**Enseignant :** R.Tomzcak

---

## Résumé

RetroChat est un projet informatique visant à développer une application de chat en ligne inspirée des interfaces « oldschool » telles que MSN, tout en intégrant des



fonctionnalités modernes et épurées. L'objectif principal est de créer une plateforme de communication en temps réel permettant aux utilisateurs de discuter, envoyer des messages vocaux, effectuer des appels vidéo, partager des photos et vidéos, et personnaliser leur expérience à travers des pseudos et des couleurs associées à chaque utilisateur. Pour ce projet on utilisera différents logiciels pour coder comme le HTML, CSS, JavaScript, Node.js et WebSocket, pour assurer une interaction fluide et réactive entre les utilisateurs.

---

## **SOMMAIRE**

### **I] Table des Matières**

### **II] Partie Commune**

#### **1) Présentation**

- a) Présentation du contexte
- b) Description générale du projet
- c) Objectifs et Finalités du projet
- d) Importance et justification du projet
- e) Définition précise de ce qui est inclus ou exclu du projet
- f) Limites et Contraintes du projet

#### **2) Analyse des besoins et exigence fonctionnelles**

- a) Description détaillée des besoins du logiciel
- b) Rédaction du cahier des charges fonctionnelles
- c) Spécifications fonctionnelles
- d) Spécifications non fonctionnelles
- e) Contraintes techniques

#### **3) Cahier des charges : Sécurité – Cybersécurité**

##### **I] Analyse des Risques de Sécurité**

- a) évaluer les menaces spécifiques au type de projet
- b) identifier les données sensibles qui seront manipulées et les conséquences potentielles de leur compromission.

##### **II] Conformité et Réglementations :**

- a) Examiner les lois et normes de sécurité applicables
  - RGPD pour les données personnelles

- la norme OWASP pour les applications web

### **III] Définition des Exigences de Sécurité :**

a) Par exemple , exiger l'authentification forte , le chiffrement des données sensibles , ou des contrôles d'accès spécifiques.

### **IV] Identification des Interactions Externes :**

a) Analyser les interactions du projet avec des services tiers ou des API externes , qui peuvent être des points d'entrées pour les attaques.

b) Identifier les contrôles à mettre en place pour sécuriser ces échanges

### **V] Solutions envisagées**

#### **4) Phases, planification et répartition des tâches**

a) Découpage du projet en différentes phases ou étapes

b) Estimation de la durée de chaque phase

c) Répartition équitable suivant les compétences de chacun

d) Jalons clés et dates importantes

#### **5) Budget et ressources :**

a) Estimation du coût total du projet

b) Détail des ressources nécessaires (humaines , matérielles, financières)

#### **6) Critères d'acceptation :**

a) Conditions à remplir pour que le projet soit considéré comme terminé

b) Méthodes de validation et de vérification

#### **7) Risques et plan de gestion des risques**

a) Identification des risques potentiels

b) Stratégies pour atténuer ou gérer ces risques.

### **III]Partie Personnelle (une par étudiant)**

#### **IV] Suite et fin de partie générale**

##### **1) Conclusion et perspectives :**

a) Bilan du projet

b) Axes d'amélioration et fonctionnalités futures

##### **2) Annexes :**

- a) Code source (ou un lien vers un dépôt Git)
  - b) Captures d'écrans
  - c) Résultats détaillés des tests
  - d) Tout autre élément jugé pertinent
- 3) Bibliographie , sitographie et références :**
- a) Sources , articles , livres , outils et technologies utilisés.
- 4) Explication Globale du code**

## Table des matières

RetroChat.....	1
Résumé.....	1
SOMMAIRE.....	2
I] Table des Matières.....	2
Table des matières.....	4
Première partie : Partie commune.....	6
1. Présentation.....	6
Présentation du contexte.....	6
Description générale du projet.....	6
Objectifs et finalités du projet.....	6
Importance et justification du projet.....	6
Définition précise de ce qui est inclus et exclu du projet.....	7
Limites et contraintes du projet.....	8
2. Analyse des besoins et exigences fonctionnelles.....	8
Description détaillée des besoins du logiciel.....	8
Rédaction du cahier des charges fonctionnelles.....	8
Spécifications fonctionnelles.....	8
Description des interactions utilisateur.....	8
Scénarios d'utilisation ou cas d'utilisation.....	9
Spécifications non fonctionnelles.....	10
Exigences liées à la performance, la sécurité, etc.....	10
Contraintes techniques.....	10
Plateformes et technologies à utiliser ou à éviter.....	10
Compatibilité avec d'autres systèmes ou logiciels.....	11
3.1 Analyse des Risques de Sécurité.....	11
3.2 Conformité et Réglementations.....	12
3.3 Exigences de Sécurité.....	12
3.4 Sécuriser les Services Externes.....	13
3.5 Solutions simples envisagées.....	13
3. Phases, planification et répartition des tâches.....	13
Découpage du projet en différentes phases ou étapes.....	13
Répartition équitable suivant les compétences de chacun(e).....	14
5. Critères d'acceptation.....	15
Conditions à remplir pour que le projet soit considéré comme terminé.....	15

Méthodes de validation et de vérification.....	15
6. Risques et plan de gestion des risques.....	15
Identification des risques potentiels.....	15
Stratégies pour atténuer ou gérer ces risques.....	15
II. Partie personnelle : Noam Boutounas.....	16
1. Introduction.....	16
2. Fin de l'Analyse.....	16
3. Conception.....	17
a) Conception générale.....	17
b) Conception détaillée.....	17
4. Implémentation.....	18
5. Tests unitaires et validation.....	18
6. Tests d'intégration (collectif).....	18
7. Discussion.....	19
II. Partie personnelle : Evan Lebeau.....	20
1. Introduction.....	20
2. Fin de l'Analyse.....	20
3. Conception.....	21
a) Conception générale.....	21
b) Conception détaillée.....	21
4. Implémentation.....	22
5. Tests unitaires et validation.....	22
6. Tests d'intégration (collectif).....	22
7. Discussion.....	23
III. Suite et fin de partie générale.....	23
1. Conclusion et perspectives.....	23
2. Annexes.....	24
3. Bibliographie, sitographie et références.....	25

---

# Première partie : Partie commune

## 1. Présentation

### Présentation du contexte

Avec l'évolution rapide des technologies de communication, les plateformes de messagerie instantanée sont devenues indispensables pour les interactions quotidiennes. Cependant, malgré les nombreuses options disponibles, il existe une demande pour des applications de chat qui allient nostalgie et modernité, offrant une expérience utilisateur unique.



### Description générale du projet

RetroChat est conçu pour recréer l'ambiance des anciens messagers instantanés tels que MSN, tout en intégrant des fonctionnalités plus modernes. L'application permettra aux utilisateurs de communiquer en temps réel via des messages écrits, vocaux, appels vidéo, et de partager des photos. De plus, elle offrira des options de personnalisation telles que la création de pseudos et l'association de couleurs aux messages en fonction des utilisateurs, la possibilité de personnaliser ses photos de profils ou encore sa biographie.

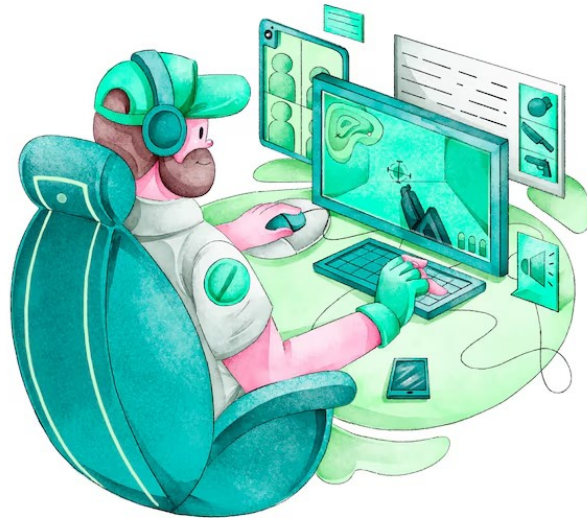


### Objectifs et finalités du projet

- **Créer une interface utilisateur intuitive et nostalgique** inspirée des anciennes interfaces rappelant notamment MSN.
- **Intégrer des fonctionnalités modernes** telles que les messages vocaux, les appels vidéo et le partage de photos ou de vidéos.
- **Assurer une communication en temps réel** correcte et sécurisée entre les utilisateurs.
- **Permettre une personnalisation poussée** du profil de l'utilisateur.

## Importance et justification du projet

RetroChat répond à une demande spécifique en combinant l'esthétique rétro pour les plus anciens, avec les exigences modernes de communication qui se fait plus fréquemment de nos jours. Ce projet permet non seulement de revoir une interface familière mais aussi d'explorer l'intégration de technologies actuelles dans un cadre nostalgique, offrant ainsi une expérience d'utilisateur unique ravivant les souvenirs pour les plus anciens et la découverte de quelque chose nouveau et universel pour les autres.



## Définition précise de ce qui est inclus et exclu du projet



### Inclus :

- Chat écrit en temps réel.
- Messages vocaux.
- Appels audio et vidéo.
- Partage de photos et vidéos.
- Personnalisation via pseudonymes et couleurs de messages.



### Exclus :

- Fonctionnalités similaires à Instagram telles que les posts et les stories.
- Scrolling infini comme sur TikTok.

### Limites et contraintes du projet

- **Compatibilité multiplateforme** : Assurer le fonctionnement sur différents navigateurs et appareils.
- **Sécurité des données** : Protéger les communications et les informations personnelles des utilisateurs.
- **Mémoire et Performances** : Gérer un grand nombre d'utilisateurs simultanément sans dégradation des performances.

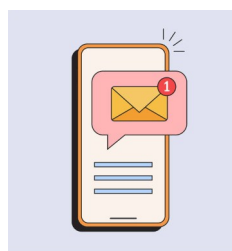
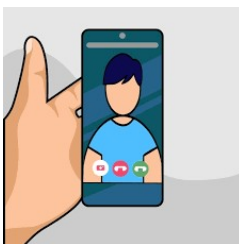
## 2. Analyse des besoins et exigences fonctionnelles

### Description détaillée des besoins du logiciel

RetroChat doit offrir une plateforme de communication en temps réel, facile à utiliser et sécurisée. Les utilisateurs doivent pouvoir créer un compte, personnaliser leur profil, envoyer et recevoir des messages écrits, vocaux, effectuer des appels audio et vidéo, et partager des photos ou vidéos. L'interface doit être intuitive et esthétiquement plaisante, reflétant une ambiance rétro tout en étant moderne.

### Rédaction du cahier des charges fonctionnelles

- **Inscription et authentification des utilisateurs.**
- **Création et personnalisation du profil** (pseudo, photo de profil, couleur des messages).
- **Envoi et réception de messages écrits en temps réel.**
- **Messages vocaux** : Enregistrement et écoute.
- **Appels audio et vidéo.**
- **Partage de photos et vidéos.**
- **Interface utilisateur épurée et rétro.**
- **Gestion des contacts et listes d'amis.**
- **Notifications en temps réel** pour les nouveaux messages.





## Spécifications fonctionnelles

### Description des interactions utilisateur

- **Inscription** : L'utilisateur crée un compte en fournissant un pseudo, une adresse email et un mot de passe.
- **Connexion** : L'utilisateur se connecte avec ses identifiants pour accéder à la plateforme.
- **Chat en direct** : Envoi et réception de messages instantanés.
- **Messages vocaux** : Enregistrement et envoi de messages vocaux.
- **Appels** : Initiation et réception d'appels audio et vidéo.
- **Partage de médias** : Upload et visualisation de photos et vidéos.
- **Personnalisation** : Choix de la couleur des messages et mise à jour du profil , que ce soit la photo ou la biographie.



### Scénarios d'utilisation ou cas d'utilisation

#### 1. Envoi d'un message écrit :

- L'utilisateur ouvre une conversation.
- Tape un message et appuie sur envoyer.
- Le message apparaît instantanément chez le destinataire avec la couleur attribuée.

#### 2. Enregistrement d'un message vocal :

- L'utilisateur appuie sur le bouton d'enregistrement.
- Enregistre le message et l'envoie.
- Le destinataire reçoit et peut écouter le message.

### 3. Initiation d'un appel vidéo :

- L'utilisateur sélectionne un contact.
- Appuie sur le bouton d'appel vidéo.
- L'autre utilisateur reçoit une notification et accepte l'appel.



### Spécifications non fonctionnelles

Exigences liées à la performance, la sécurité, etc.

- **Performance** : Temps de réponse rapide pour l'envoi et la réception des messages.
- **Sécurité** : Chiffrement des communications, protection des données personnelles.
- **Compatibilité** : Fonctionnement sur les principaux navigateurs web et appareils mobiles.



## Contraintes techniques

Plateformes et technologies à utiliser ou à éviter

À utiliser :

- **HTML, CSS, JavaScript** pour le développement front-end.
- **Node.js** pour le serveur back-end.
- **WebSocket** pour la communication en temps réel.

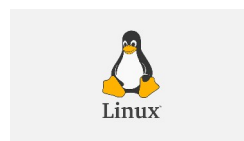


À éviter :

- Technologies non compatibles avec WebSocket ou qui ne supportent pas la communication en temps réel de manière efficace.

Compatibilité avec d'autres systèmes ou logiciels

RetroChat doit être compatible avec les navigateurs web modernes (Chrome, Firefox, Safari, Edge) et les systèmes d'exploitation courants (Windows, macOS, Linux, Android, iOS).

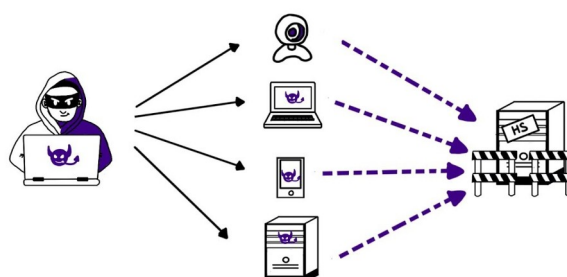


## Sécurité – Cybersécurité du projet

### 3.1 Analyse des Risques de Sécurité

Pour éviter que RetroChat ne devienne une cible facile pour les hackers, nous avons identifié les principaux risques et les données sensibles :

- **Quels sont les risques ?**
  - Les comptes utilisateurs peuvent être piratés si les mots de passe sont faibles.
  - Les messages ou photos peuvent être interceptés si la connexion n'est pas sécurisée.
  - Une attaque peut bloquer le fonctionnement du site (attaque DDoS).



**DDOS**

- **Quelles données sont importantes ?**

- Les informations personnelles des utilisateurs (pseudo, e-mail, photo de profil).
- Les messages envoyés (texte, audio, photos).

### 3.2 Conformité et Réglementations

Pour respecter la loi et protéger les utilisateurs, nous devons :

- Suivre les règles du **RGPD** (protection des données personnelles) :
  - Demander aux utilisateurs leur consentement avant de collecter leurs données.
  - Permettre aux utilisateurs de supprimer ou modifier leurs données.
- Suivre les **recommandations de sécurité pour les sites web** (comme OWASP) :
  - Empêcher les attaques classiques, comme les vols de mot de passe.
  - Vérifier que les messages envoyés ne contiennent pas de code dangereux.



---

### 3.3 Exigences de Sécurité

Pour sécuriser RetroChat, nous avons décidé de mettre en place :

1. **Un bon système d'authentification :**

- Obliger les utilisateurs à choisir des mots de passe solides.
- Ajouter une option pour envoyer un code de sécurité (par SMS ou e-mail) pour protéger leur compte.

2. **Protéger les échanges :**

- Utiliser une connexion sécurisée (**HTTPS**) pour éviter que les hackers ne puissent lire les messages.

- Crypter les messages pour que seul l'utilisateur puisse les voir.
3. **Limiter les accès :**
- Donner des droits différents aux utilisateurs normaux et aux administrateurs (personnes qui gèrent le site).
- 

### 3.4 Sécuriser les Services Externes

Notre application utilisera des services externes (comme le stockage en ligne pour les photos et vidéos). Ces services peuvent aussi être des points faibles. Pour éviter les problèmes :

- On devra vérifier toutes les données qui arrivent depuis ces services.
  - On utilisera des **clés de sécurité** pour connecter RetroChat à ces services (comme un mot de passe secret entre nous et le service).
- 

### 3.5 Solutions simples envisagées

Pour que RetroChat soit sûr tout en restant simple à utiliser :

1. **HTTPS** : Tous les échanges entre l'utilisateur et le site seront sécurisés.
2. **Chiffrement** : Les messages seront « cryptés » pour qu'ils soient lisibles uniquement par ceux à qui ils sont destinés.
3. **Mots de passe** : On proposera des règles simples pour choisir un bon mot de passe (ex : minimum 8 caractères, avec des chiffres et des lettres).
4. **Vérifications régulières** : Avant de lancer RetroChat, on vérifiera si tout est bien sécurisé (test de sécurité).



## 3. Phases, planification et répartition des tâches

Découpage du projet en différentes phases ou étapes

1. **Phase de conception :**
  - Définition des fonctionnalités.
  - Création des maquettes et prototypes.

- Élaboration de l'architecture logicielle.

## 2. Phase de développement :

- Développement front-end (HTML, CSS, JavaScript).
- Développement back-end (Node.js, WebSocket).
- Intégration des fonctionnalités de communication en temps réel.

## 3. Phase de tests :

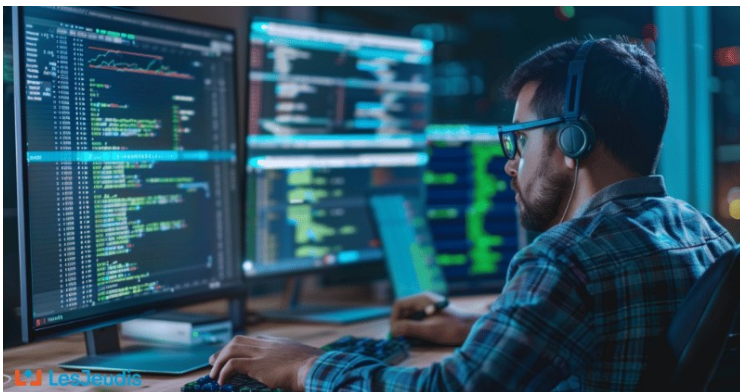
- Tests unitaires.
- Tests d'intégration.
- Tests de performance et de sécurité.

## 4. Phase de déploiement :

- Configuration du serveur.
- Déploiement de l'application en ligne.
- Surveillance et maintenance post-déploiement.

### Répartition équitable suivant les compétences de chacun(e)

- **Développeurs front-end** : Création de l'interface utilisateur, stylisation avec CSS, intégration des fonctionnalités interactives avec JavaScript.
- **Développeurs back-end** : Mise en place du serveur Node.js, gestion des WebSocket, implémentation des fonctionnalités de communication en temps réel.
- **Designer** : Conception de la partie esthétique et confort de l'application
- **Testeurs** : Élaboration des scénarios de test, réalisation des tests unitaires et d'intégration.



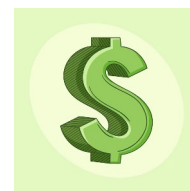
### Matérielles :

- Serveurs pour hébergement

- Ordinateurs et équipements de développement



- **Financières :**
  - Budget pour les logiciels, licences et hébergement



## 5. Critères d'acceptation

### Conditions à remplir pour que le projet soit considéré comme terminé

- Toutes les fonctionnalités définies dans le cahier des charges sont implémentées et opérationnelles.
- L'application fonctionne de manière fluide et sans bugs majeurs sur les plateformes ciblées.
- Les tests de performance et de sécurité sont passés avec succès.
- L'interface utilisateur est conforme aux maquettes et offre une expérience utilisateur optimale.
- La documentation complète du projet est fournie.

### Méthodes de validation et de vérification

- **Revue de code** : Vérification par les pairs des segments de code étape par étape.
- **Tests utilisateurs** : Recueillir les retours des utilisateurs finaux pour améliorer l'expérience.
- **Audit de sécurité** : Vérification des mesures de protection des données et des communications.

## 6. Risques et plan de gestion des risques

### Identification des risques potentiels

- **Retards dans le développement** : Difficultés techniques ou sous-estimation du temps nécessaire.

- **Problèmes de sécurité** : Vulnérabilités dans les communications ou la gestion des données.
- **Incompatibilité des navigateurs** : L'application ne fonctionne pas correctement sur certains navigateurs.
- **Défaillance du serveur** : Pannes ou interruptions de service.

### Stratégies pour atténuer ou gérer ces risques

- **Gestion du temps** : Suivi rigoureux de la planification et ajustements en cas de retard.
- **Sécurité renforcée** : Implémentation de bonnes pratiques de sécurité et réalisation d'audits réguliers.
- **Tests de compatibilité** : Assurer une couverture étendue des tests sur différents navigateurs et appareils.
- **Redondance des serveurs** : Mise en place de serveurs de secours pour garantir la disponibilité

## II. Partie personnelle : Noam Boutounas

### 1. Introduction

- **Rappel des tâches à réaliser** :

Mon rôle principal dans ce projet a été de travailler sur la **personnalisation**, le **design** et le **stylisme de l'interface utilisateur et de RetroChat en général**. J'ai utilisé CSS et JavaScript pour garantir une expérience utilisateur fluide, esthétique et agréable. J'ai aussi travaillé avec mon coéquipier sur l'intégration des fonctionnalités développées par lui pour que tout fonctionne harmonieusement.

- **Positionnement** :

En complément de son travail sur la structure globale (HTML, Node.js et WebSocket), j'ai conçu les éléments visuels et interactifs comme les boutons, les couleurs, le fait de pouvoir écrire... Je me suis assuré que l'application soit agréable à utiliser et respecte les principes modernes d'ergonomie.

### 2. Fin de l'Analyse

- **Identification des Scénarios et Cas d'Utilisation** :

Mon analyse s'est concentrée sur les **besoins utilisateurs en termes de design**. Par exemple :

- **Scénario 1** : Un utilisateur modifie son pseudo et change la couleur associée à ses messages dans une interface intuitive.
- **Scénario 2** : Un utilisateur interagit avec une fenêtre de chat où les couleurs des messages varient selon l'expéditeur.
- **Scénario 3** : Un utilisateur navigue sur une interface moderne, qui s'adapte aux écrans mobiles et ordinateurs.



- **Recherche de solutions :**

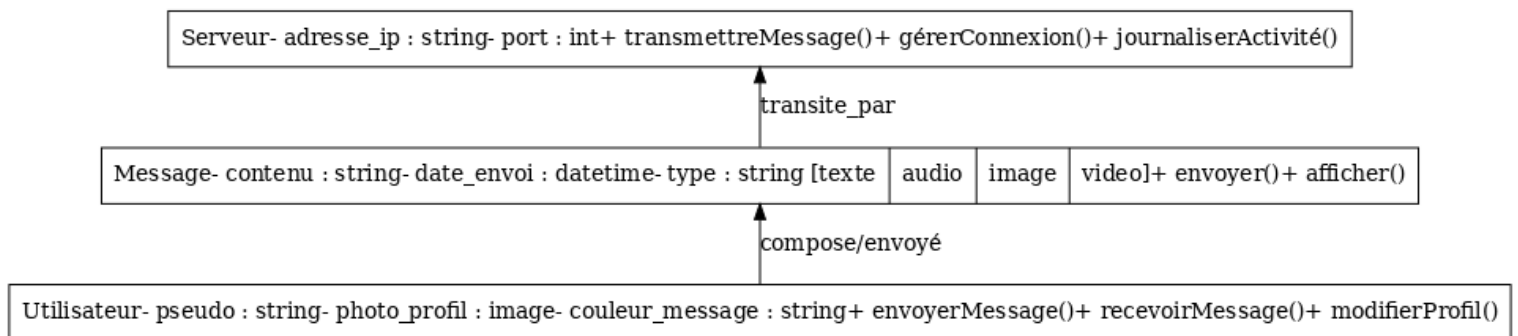
Pour répondre à ces besoins, j'ai exploré et comparé plusieurs outils :

- **Framework CSS :** Nous avons opté pour une approche native avec du CSS pur pour garder le contrôle total sur le style.
- **Bibliothèques JS :** J'ai intégré quelques fonctionnalités dynamiques comme l'animation des messages avec JavaScript.
- **Théorie des couleurs et UX design :** J'ai suivi des recommandations pour choisir des palettes de couleurs adaptées aux interfaces numériques, mais également de suivre et observer les meilleures combinaisons de couleurs possibles.

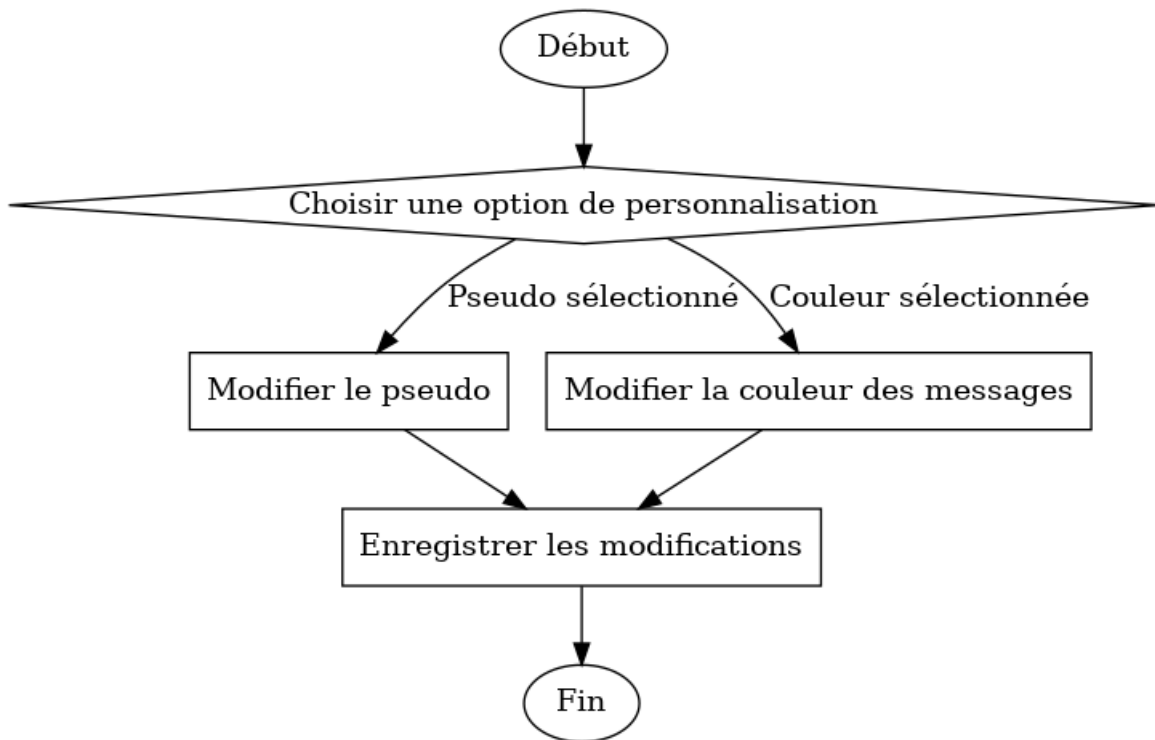
### 3. Conception

#### a) Conception générale

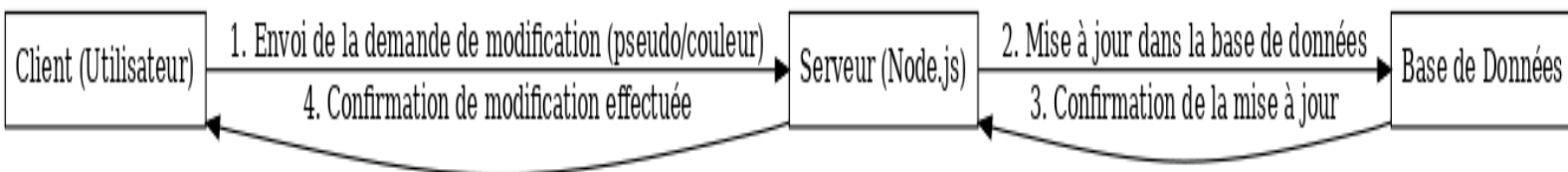
- **Visualiser la structure et le flux de l'application :**
  - **Frontend :** Un design épuré, et moderne mais qui rappelle toujours l'ancien style MSN basé sur une grille avec CSS.
  - **Interactions dynamiques :** Utilisation de JavaScript pour des animations, la gestion des couleurs des messages, et l'adaptation de l'interface en temps réel.
- **Diagrammes UML liés à mon rôle :**



- Un **diagramme d'activité** montrant comment l'utilisateur interagit avec les fonctions de personnalisation (changement de pseudo, couleurs des messages).



- Un **diagramme de séquence** illustrant les interactions entre les événements côté client (JavaScript) et les mises à jour côté serveur.

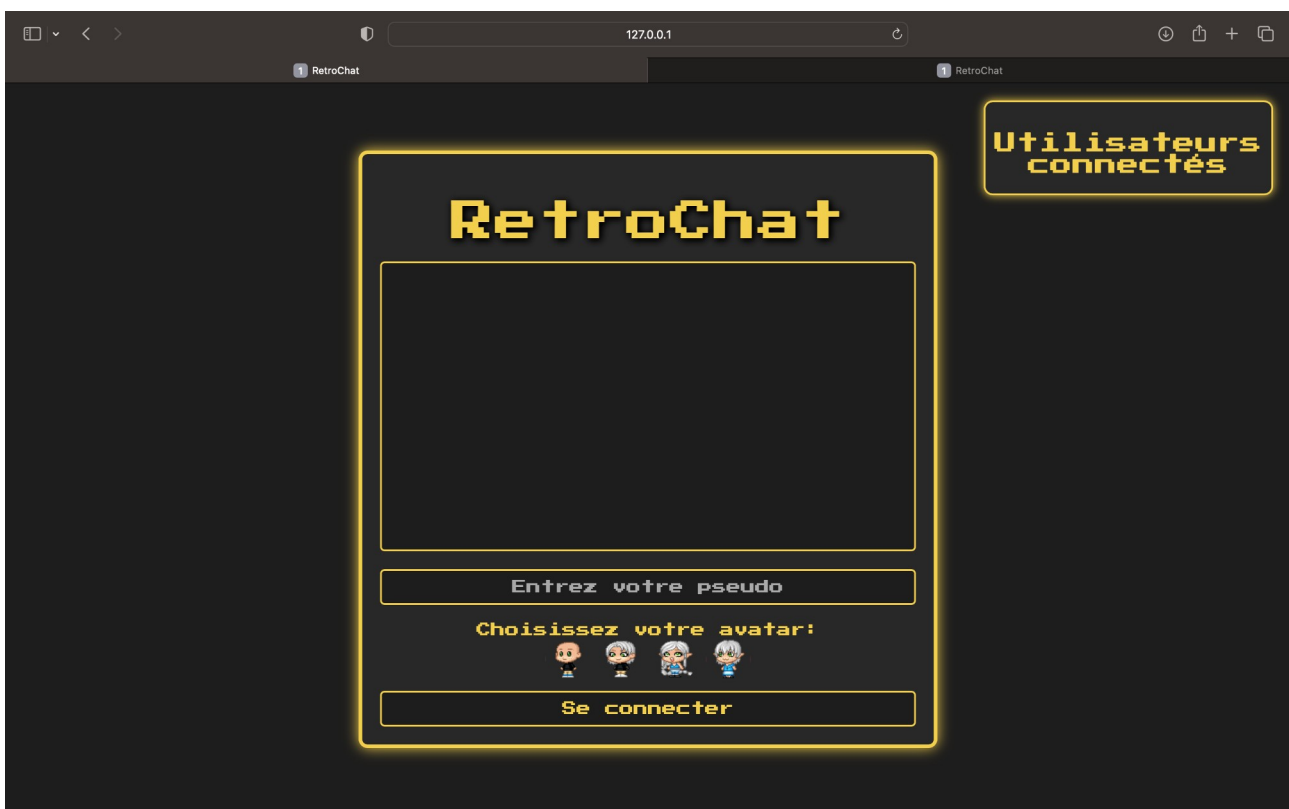


- **Interface utilisateur :**

- **Présentation de l'interface :**

L'application RetroChat se compose de :

1. Une liste des utilisateurs connectés sur la droite.
2. Une zone de chat principale au centre.
3. Des options de personnalisation accessibles en dessous du centre ( on entre notre pseudo , on choisit notre avatar , on peut se connecter...)



- **Justification des choix de design :**

J'ai choisi une **typographie simple et lisible**, des **contrastes élevés** pour l'accessibilité, et des couleurs jaunes et noir avec des teintes et nuances de gris qui se complètent parfaitement .

On a également une police d'écriture un peu pixelisé qui rappelle l'ancienne époque «Retro» que j'ai voulu donner à ce projet.

- **Accessibilité et ergonomie :**

L'interface est optimisée pour être utilisable sur mobile et ordinateur, avec des boutons de grande taille , et des fonctionnalités accessibles au clavier.

#### b) Conception détaillée

- **Conception des Composants Frontend :**

- **Composant de personnalisation utilisateur :**

- Permet à l'utilisateur de modifier son pseudo, son avatar, et la couleur de ses messages.
- Utilisation de CSS variables pour implémenter les changements en temps réel.

- **Composant de chat :**

- Affichage des messages avec des animations légères (JavaScript).
- Stylistique des messages pour inclure des sons lorsque l'utilisateur reçoit un message (comme une sorte de notification)

- **Design :**

- Média CSS pour adapter la mise en page aux écrans de différentes tailles.

- **Difficultés rencontrées et solutions apportées :**

- **Problème :** Gestion des couleurs personnalisées sans affecter la lisibilité.

- **Solution :** Limiter la palette de couleurs disponibles et ajuster automatiquement le contraste avec la couleur de fond.

- **Problème :** Intégration des animations sans ralentir les performances.

- **Solution :** Utilisation de transitions CSS au lieu de JavaScript lourd pour les effets visuels.
- 

## 4. Implémentation

- **Détails techniques de la réalisation :**

- **CSS :**

- Utilisation de variables CSS pour les couleurs dynamiques.

- Grille CSS pour organiser la disposition des éléments (contact, messages, etc.).
  - **JavaScript :**
    - Gestion des événements : clics, survols et choix de personnalisation.
    - Script pour synchroniser les choix de personnalisation avec le backend (via des requêtes WebSocket).
  - **Problèmes rencontrés et solutions adoptées :**
    - Lors de l'implémentation des personnalisations en temps réel, il y avait un problème de synchronisation avec le serveur. Nous avons résolu cela en ajoutant un délai de mise à jour côté client pour éviter les conflits.
- 

## 5. Tests unitaires et validation

- **Stratégie de tests :**
    - Tester chaque fonctionnalité de personnalisation (pseudo, couleur, avatar) individuellement.
    - Vérifier l'affichage correct des messages sur différentes tailles d'écran.
  - **Résultats et interprétation :**

Les tests ont montré que les fonctionnalités sont fiables et les animations fluides. Cependant, des bugs mineurs liés à la compatibilité mobile ont été identifiés et corrigés.
- 

## 6. Tests d'intégration (collectif)

Avec mon coéquipier, nous avons testé l'intégration de la personnalisation au système global. Nous avons travaillé ensemble pour synchroniser les changements de style avec les interactions serveur.

---

## 7. Discussion

- **Retour sur les choix effectués :**

Le choix d'un design épuré et personnalisable a permis de répondre aux attentes des utilisateurs ciblés. Cependant, des améliorations pourraient être apportées à la gestion des avatars et à l'ajout de thèmes préconfigurés.
- **Limites du logiciel actuel :**

Les personnalisations sont limitées aux couleurs et pseudos. Une version future pourrait inclure des thèmes graphiques complets. Le logiciel dans sa globalité est encore restreint et n'exploite pas son plein potentiel.
- **Comparaison avec d'autres solutions existantes :**

Par rapport à des applications comme Discord ou WhatsApp, RetroChat se distingue par sa simplicité et sa personnalisation intuitive.

## II. Partie personnelle : Evan Lebeau

### 1. Introduction

- **Rappel des tâches à réaliser :**

Mon rôle principal dans ce projet a été de **concevoir et mettre en place les bases techniques et structurelles de l'application RetroChat**. J'ai travaillé sur la création de **l'interface utilisateur de base**, la mise en place d'un **serveur Node.js**, et la gestion des **communications en temps réel entre utilisateurs** via WebSocket. De plus, j'ai initié et configuré le dépôt GitHub pour le transfert de données et d'informations entre moi et mon collègue.

- **Positionnement :**

J'ai pris en charge les **fondations techniques** de l'application, permettant de poser une structure solide sur laquelle les fonctionnalités et les éléments de personnalisation ont été ajoutés par mon coéquipier.

---

### 2. Fin de l'Analyse

- **Identification des Scénarios et Cas d'Utilisation :**

Mon analyse s'est focalisée sur les **besoins fonctionnels et techniques** de l'application, par exemple :

- **Scénario 1 :** Un utilisateur envoie un message via l'interface, le serveur le transmet à un autre utilisateur en temps réel.
- **Scénario 2 :** Un utilisateur se connecte au serveur, et son statut est partagé avec les autres utilisateurs connectés.
- **Scénario 3 :** Les messages échangés s'affichent en temps réel avec des métadonnées comme le pseudo, l'avatar...

- **Recherche de solutions :**

J'ai exploré plusieurs solutions techniques avant de choisir les outils et approches :

- **Serveur Node.js :** Simple d'utilisation, rapide, adapté pour gérer les WebSockets.
  - **WebSocket :** J'ai choisi WebSocket pour une communication en temps réel efficace et pareil, simple à mettre en place
  - **JavaScript :** J'ai choisi JavaScript pour quelques fonctionnalités de base de notre Chat comme le fait de pouvoir envoyer un message ou d'inscrire notre pseudo
  - **HTML :** Pour la création globalr de l'interface utilisateur et du chat
  - **Gestion du dépôt GitHub :** Permet de partager facilement le code et de travailler en collab.
-

### 3. Conception

#### a) Conception générale

- **Visualiser la structure et le flux de l'application :**
    - **Partie Backend (avec Node.js) :** Pour la gestion des connexions et des échanges de messages entre les clients.
    - **Partie Frontend (HTML) :** Une interface utilisateur basique avec une zone de texte pour les messages et une liste des utilisateurs connectés.
  - **Diagrammes UML liés à mon rôle :**
    - **Diagramme de séquence** montrant la communication entre le client, le serveur Node.js, et les utilisateurs.
    - **Diagramme de déploiement** représentant le serveur central Node.js et les interactions client-serveur.
  - **Architecture générale du logiciel :**
    - L'application est basée sur une architecture **client-serveur** :
      - **Client :** HTML pour l'interface de base, enrichi par du JavaScript pour les interactions...
      - **Serveur :** Node.js gère les connexions et assure une communication en temps réel via WebSocket.
  - **Les difficultés rencontrées et les solutions apportées :**
    - **Problème :** Synchronisation des messages en cas de multiples connexions.
      - **Solution :** J'ai implémenté une gestion efficace des identifiants de session côté serveur pour éviter les conflits, ce qui m'a prit pas mal de temps sur le projet
    - **Problème :** Mise en place du dépôt GitHub pour collaborer efficacement.
      - **Solution :** Configuration de branches distinctes pour éviter les conflits entre les modifications.
- 

#### b) Conception détaillée

- **Architecture Logicielle Détaillée :**
  - **Serveur Node.js :**
    - Crée un serveur WebSocket pour gérer les connexions en temps réel.
    - Stocke temporairement les sessions des utilisateurs pour une gestion fluide., style chat moderne.
  - **Client HTML/JavaScript :**
    - Gère l'envoi et la réception des messages via WebSocket.
    - Affiche les messages reçus dans l'interface utilisateur.
- **Conception des Composants :**

- **Composant serveur :**
    - Écoute les connexions entrantes et gère les messages sortants.
    - Maintient une liste des utilisateurs connectés pour le partage des statuts.
  - **Composant client :**
    - Interface utilisateur avec zone de texte pour les messages et liste des contacts.
    - Envoie des requêtes au serveur et affiche les réponses en temps réel.
- 

## 4. Implémentation

- **Détails techniques de la réalisation :**
    - **Node.js :**
      - Mise en place d'un serveur avec le module ws pour gérer les WebSockets.
      - Gestion des connexions et des échanges de messages entre clients.
    - **HTML :**
      - Création d'une interface utilisateur simple avec une zone de chat et une liste d'utilisateurs.
    - **GitHub :**
      - Initialisation du dépôt, gestion des commits, et création des branches pour un travail collaboratif fluide.
  - **Problèmes rencontrés et solutions adoptées :**
    - **Défis avec les WebSockets :** Lors des tests initiaux, les messages ne se propageaient pas correctement entre plusieurs utilisateurs.
      - **Solution :** J'ai optimisé la gestion des sockets en associant chaque utilisateur à un identifiant unique et le serveur propage le message à tous les clients.
- 

## 5. Tests unitaires et validation

- **Stratégie de tests :**
    - Tester la connectivité entre le client et le serveur.
    - Vérifier que les messages sont correctement envoyés et reçus par plusieurs utilisateurs connectés simultanément.
  - **Résultats et interprétation :**

Les tests ont confirmé que les messages s'affichaient instantanément chez tous les utilisateurs connectés, sans pertes ni décalages majeurs.
- 

## 6. Tests d'intégration (collectif)

En collaboration avec mon coéquipier, nous avons intégré les fonctionnalités de personnalisation (pseudo et couleurs des messages) dans l'infrastructure du serveur. Cela a nécessité des ajustements pour transmettre correctement ces données en temps réel.

---



## 7. Discussion

- **Retour sur les choix effectués :**

L'utilisation de Node.js et WebSocket s'est avérée être un choix judicieux pour répondre aux exigences en temps réel de l'application. GitHub a facilité la collaboration et le partage du code.

- **Limitations du logiciel actuel :**

Le système de gestion des utilisateurs est encore basique : il manque une authentification robuste et une persistance des données côté serveur. De plus il manque d'autres fonctionnalités comme pouvoir ajouter des gens en amis, pouvoir observer sa liste d'amis sur l'écran (à gauche), créer des groupes privés entre amis, envoyer des photos etc...

- **Comparaison avec d'autres solutions existantes :**

Contrairement à des solutions comme Firebase, notre approche offre une plus grande flexibilité dans la gestion du serveur, bien qu'elle soit plus complexe à configurer.

## III. Suite et fin de partie générale

### 1. Conclusion et perspectives

- **Bilan du projet :**

Le projet **RetroChat** nous a permis de concevoir un chat en ligne moderne inspiré de plateformes classiques comme MSN, tout en intégrant des fonctionnalités actuelles telles que la communication en temps réel et la personnalisation des messages. Tout ceci en gardant la touche «oldschool» que l'on voulait depuis le début. Nous avons réussi à établir une base technique solide grâce à l'utilisation de **Node.js**, **WebSocket**, et des langages web essentiels comme **HTML**, **CSS**, et **JavaScript**.

Ce projet a également renforcé nos compétences en **collaboration/communication** via GitHub. Malgré quelques limitations, l'application remplit ses objectifs principaux.

- **Axes d'amélioration et fonctionnalités futures :**

Pour le futur, nous proposons les améliorations et fonctionnalités suivantes :

1. **Authentification utilisateur sécurisée :**

- Ajout de systèmes de connexion avec des mots de passe ou des solutions OAuth (Google, Facebook).

2. **Persistance des données :**

- Implémentation d'une base de données (par exemple, **MySQL** ou **DB Browser**) pour stocker les historiques de messages, les profils utilisateurs, et les préférences.

3. **Amélioration de la personnalisation :**

- Ajout d'options avancées comme des thèmes personnalisables pour l'interface.
- Avatar personnalisé disponible

- Une musique que l'on peut choisir en fond
  - Des notifications (pop-up ou pas) que l'on peut choisir d'activer ou non , elles seraient accompagnés d'un son.
4. **Optimisation pour les appareils mobiles :**
    - Création d'une version ou d'une application mobile native.
  5. **Cybersécurité renforcée :**
    - Intégration de certificats SSL/TLS pour chiffrer toutes les communications.
  6. **Fonctionnalités multimédias :**
    - Ajout d'appels vidéo et vocaux.
  7. **Extensions sociales :**
    - Création de groupes de discussion ou de salons thématiques.
    - Ajout d'une liste d'amis
    - Suggestion d'amis possibles
- 

## 2. Annexes

### 1. Code source :

Le code source complet est accessible sur GitHub à cette adresse :

<https://github.com/BroAreYouGay/RetroChat/>

### 2. Captures d'écran :

Voici quelques captures illustrant le projet :

- **Interface utilisateur de base :**

Une simple boîte de discussion avec des messages affichés en temps réel, avec couleurs , pseudos et avatars personnalisés



### 3. Résultats détaillés des tests :

- **Tests unitaires :**

Tous les scénarios de communication (envoi/réception de messages, gestion des utilisateurs connectés) ont fonctionné sans bug majeur.

- **Tests d'intégration :**

L'intégration des fonctionnalités de personnalisation a nécessité des ajustements, mais les échanges entre les clients et le serveur se sont révélés stables.

### 4. Autres éléments pertinents :

- Notes de brainstorming et croquis des premières versions de l'interface:

Toute première version de l'interface utilisateur prototype :


## RetroChat - Utilisateur 1


Utilisateur 1: hello


Ensuite on a eu une deuxième version un peu plus évoluée :


User 1

Choisissez votre avatar:









Se connecter

Par la suite on a eu une troisième version où la personnalisation commençait à se mettre en place :



Et enfin la version dite finale où le style jaune et noir vintage est mis en place , les avatars sont créés à la mains par nos soins , les pseudos peuvent êtres changées à la guise de l'utilisateur ... :



```
serveur websocket en écoute sur ws://localhost:9999
```

### 3. Bibliographie, sitographie et références



#### Articles et documentation :

- Node.js Documentation : <https://nodejs.org>
- MDN Web Docs : HTML, CSS, and JavaScript guides : <https://developer.mozilla.org>
- WebSocket API : [https://developer.mozilla.org/en-US/docs/Web/API/WebSockets\\_API](https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API)

#### Technologies utilisées :

- **Node.js** pour la gestion du serveur.
- **WebSocket** pour la communication en temps réel.
- **HTML, CSS, et JavaScript** pour la création et la personnalisation de l'interface utilisateur.
- **GitHub** pour la gestion de version et la collaboration.



#### Outils et logiciels :

- **Visual Studio Code** ou VSCode pour l'édition du code.
- **Git** pour le suivi des modifications.
- **Draw.io** pour la création de diagrammes UML.



## IV] Explication globale du code

Tout d'abord on a commencé par la création globale de l'interface avec le titre , la création d'un champ pour rentrer un pseudo , les différentes sections et boutons :

```
<!-- formulaire pour entrer le pseudo et choisir un avatar -->
<form id="usernameForm" action="#" method="post">
  <!-- champ pour entrer le pseudo -->
  <input type="text" id="username" placeholder="Entrez votre pseudo" required>

  <!-- section pour choisir un avatar -->
  <div id="avatar-selection">
    <label for="avatar">Choisissez votre avatar:</label>
    <div class="avatars">
      <!-- options d'avatars avec images -->
      
      
      
      
    </div>
  </div>
  <!-- bouton pour soumettre le formulaire -->
  <button type="submit">Se connecter</button>
</form>
```

Le code plus précis qui nous permet d'intégrer les avatars choisis ici :

```
<!-- options d'avatars avec images -->




```



Par la suite on a voulu modifier la police d'écriture pour tout de suite donner un aspect rétro à cette interface , ce qu'on cherchait depuis le début :

```
<!-- lien vers la police "Press Start 2P" utilisée pour l'aspect rétro -->
<link href="https://fonts.googleapis.com/css2?family=Press+Start+2P&display=swap" rel="stylesheet">
```





A noter que les codes ci-dessus étaient principalement des codes en HTML/CSS destinés à la création d'une première interface et d'une personnalisation notable.

Maintenant on poursuit avec une partie en javascript , qui va notamment servir à créer une connexion WebSocket avec un serveur local (exécutant sur localhost à partir du port 9999) :

```
const socket = new WebSocket('ws://localhost:9999');
```

Le prochain code est un gestionnaire d'événements pour traiter les messages reçus par la connexion WebSocket.

Voici ce qu'il fait, étape par étape :

Réception du message : Lorsqu'un message est reçu par le WebSocket, la fonction onmessage est déclenchée, et le message est accessible via event.data.

Traitement du message : Le message est d'abord divisé en un tableau avec split(':'). Cela suppose que le message est sous un format spécifique (séparé par des :).

Cas 1 — Liste des utilisateurs (userList) : Si le premier élément du tableau est 'userList', la fonction updateUserList est appelée avec la seconde partie des données (index 1).

Cas 2 — Liste des amis (friendList) : Si le premier élément est 'friendList', la seconde partie des données (index 1) est divisée en plusieurs amis. Chaque ami est décomposé en avatar et username et stocké sous forme d'objet dans un tableau. Ensuite, updateFriendsList(friendsData) est appelée pour mettre à jour la liste des amis.

Cas 3 — Message utilisateur : Si le message contient au moins quatre éléments, il est traité comme un message d'utilisateur. Il est divisé en plusieurs parties : Le nom de l'utilisateur (index 0). Le message (index 1). Le nom d'utilisateur est affiché avec son avatar (index 3). Un nouvel élément <div> est créé pour afficher le message, avec un avatar et le texte du message. L'avatar est affiché sous forme d'image et sa taille est de 30px par 30px, avec un bord arrondi pour donner un aspect circulaire. Le message est ajouté à un conteneur DOM avec l'id 'messages', et la fonction scrollToBottom() permet de faire défiler la fenêtre pour afficher le dernier message.

Cas 4 — Format incorrect : Si le message ne correspond à aucun des formats attendus, une erreur est loguée avec `console.error`.

```
// quand un message est reçu du serveur
socket.onmessage = function(event) {
  const data = event.data.split(':');

  if (data[0] === 'userList') {
    updateUserList(data[1]);
  } else if (data[0] === 'friendList') {
    const friendsData = data[1].split(';').map(friendData => {
      const [avatar, username] = friendData.split(',');
      return { avatar, username };
    });
    updateFriendsList(friendsData); // mise à jour de la liste des amis
  } else if (data.length >= 4) {
    const user = data[0];
    const message = data[1];
    const userAvatar = data[3];

    const messageDiv = document.createElement('div');
    const avatarImg = document.createElement('img');
    avatarImg.src = userAvatar && userAvatar.trim() !== '' ? userAvatar : 'assets/avatar-1.png';
    avatarImg.classList.add('avatar');

    avatarImg.style.width = '30px';
    avatarImg.style.height = '30px';
    avatarImg.style.borderRadius = '50%';

    messageDiv.textContent = `${user}: ${message}`;
    messageDiv.prepend(avatarImg);

    document.getElementById('messages').appendChild(messageDiv);
    scrollToBottom();
  } else {
    console.error('Format de message incorrect:', event.data);
  }
};
```

La prochaine fonction met à jour dynamiquement la liste des utilisateurs affichée sur l'interface. Elle prend une chaîne de caractères représentant les utilisateurs (avec avatar et nom séparés par des virgules), les divise, puis crée un élément HTML pour chaque utilisateur, en affichant leur avatar et nom dans une liste.

```
// Fonction pour mettre à jour la liste des utilisateurs
function updateUserList(userDataString) {
  const userListDiv = document.getElementById('user-list-container');
  userListDiv.innerHTML = '';

  const users = userDataString.split(';');
  users.forEach(userData => {
    const [avatar, user] = userData.split(',');
    const userDiv = document.createElement('div');

    const avatarImg = document.createElement('img');
    avatarImg.src = avatar;
    avatarImg.classList.add('avatar');

    avatarImg.style.width = '30px';
    avatarImg.style.height = '30px';
    avatarImg.style.borderRadius = '50%';

    userDiv.textContent = user;
    userDiv.prepend(avatarImg);

    userListDiv.appendChild(userDiv);
  });
}
```

Ensuite on va rapidement s'occuper de notre serveur local WebSocket connecté au javascript et plus précisément au port 9999

```
const server = new WebSocket.Server({ port: 9999 });
```

Ce code est un serveur WebSocket qui gère les connexions des utilisateurs, la gestion des messages et des amis

Une explication plus détaillée , étape par étape :

### 1) Connexion d'un utilisateur

Lorsque qu'un nouvel utilisateur se connecte au serveur WebSocket, l'événement 'connection' est déclenché. Un objet socket est créé pour représenter la connexion de cet utilisateur. Un message est affiché dans la console pour signaler qu'un utilisateur est connecté.

```
server.on('connection', (socket) => {  
  console.log('un utilisateur s'est connecté.');// affiche dans la console qu'un utilisateur s'est connecté
```

### 2) Gestion des messages envoyés par l'utilisateur

```
socket.on('message', (message) => {  
  const [type, ...data] = message.toString().split(':'); // on divise le message en parties séparées par ":"
```

Lorsqu'un utilisateur envoie un message via le WebSocket, l'événement 'message' est déclenché. Le message est divisé en plusieurs parties en fonction du séparateur ":". La première partie (type) détermine le type du message, et le reste est stocké dans data

### 3) Déconnexion d'un utilisateur

```
socket.on('close', () => {
  console.log('un utilisateur s\'est déconnecté.');// on affiche que l'utilisateur s'est déconnecté
  const clientInfo = clients.get(socket);
  if (clientInfo) {
    // Supprimer l'utilisateur de la liste des clients et des amis
    const { username } = clientInfo;
    friendsList.delete(username); // retirer l'utilisateur des amis
  }
  clients.delete(socket); // on retire l'utilisateur de la liste des clients
  broadcastUserList(); // on met à jour la liste des utilisateurs pour tous les clients
});
});
```

## AJOUT D AMIS

1. Lors de notre tentative de mise en place d'une fonctionnalité de liste d'amis, nous avons intégré les premières étapes du processus, comme l'ajout d'amis et la gestion des interactions entre utilisateurs. Cependant, pour que cette fonctionnalité soit complète et fonctionnelle sur le long terme, il est nécessaire d'intégrer une base de données pour stocker les relations d'amitié de manière persistante. Cela nécessite un travail supplémentaire pour concevoir et implémenter la gestion de la base de données. En conséquence, bien que les premiers morceaux de code pour l'ajout d'amis soient déjà en place, cette fonctionnalité n'est pas encore entièrement fonctionnelle, car elle dépend d'une étape suivante importante, celle de l'intégration de la base de données. Ce processus demande plus de temps pour être pleinement opérationnel.