20/06/2022

Documentation du projet

MANAO - API DE CHAT

Description du document

Client	Orange Manao
Titre projet	API Chat pour Manao
Titre document	Documentation du projet
Rédacteurs	Adrien CARBONNEL Kim-Céline FRANOT Yannis GUIRONNET Aubin PLUMAIL
Date	20/06/2022

Table des matières

1. Contexte du chat

Ce projet de chat s'intègre dans l'application Manao, le réseau social d'entreprise d'Orange. Il remplace l'ancien chat qui ne répond plus aux besoins actuels et qui accuse des problèmes de performances.

2. Installation

Récupérer le projet en le clonant depuis le GiLab:
https://gitlab.com/DorianSechal/projet-etudiants-chat

a. Installation des dépendances

Une fois le projet cloné :

- Installer Node.js (version 14+) et MongoDB (version 5+)
- Installer les modules du projet avec la commande `npm install`.

b. Lancement du serveur

Se rendre au répertoire racine du code source pour lancer le serveur. Il y a 3 manières principales de démarrer le serveur :

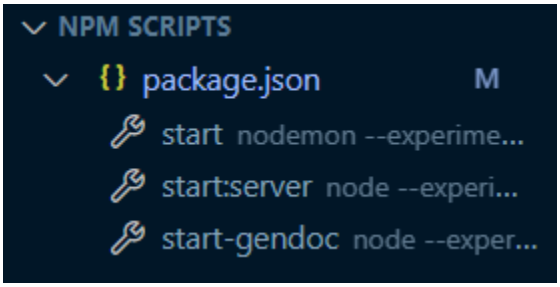

Figure 1 : Capture d'écran des 3 manières de lancer le serveur

- Avec la commande `npm run start`, le serveur est lancé en mode développement. Chaque modification dans le code est directement prise en compte par le serveur. Ce fonctionnement permet de travailler “en direct” sur le code.

- Avec la commande `npm run start:server`, le serveur est lancé normalement. Si une modification des fichiers sources survient après la dernière compilation, cela ne sera pas pris en compte jusqu’à la prochaine compilation du code.

- Avec la commande `npm run start-gendoc`, le serveur est lancé en mode normal. De plus, au démarrage, un script parcourt le code de l’API afin de générer la documentation Swagger.

Par défaut, le serveur se situe à l’adresse http://localhost:8080. Il est également possible de consulter la documentation Swagger de l’API à l’adresse http://localhost:8080/doc.

3. Architecture du code

Le code source est organisé dans plusieurs dossiers pour plus de clarté.

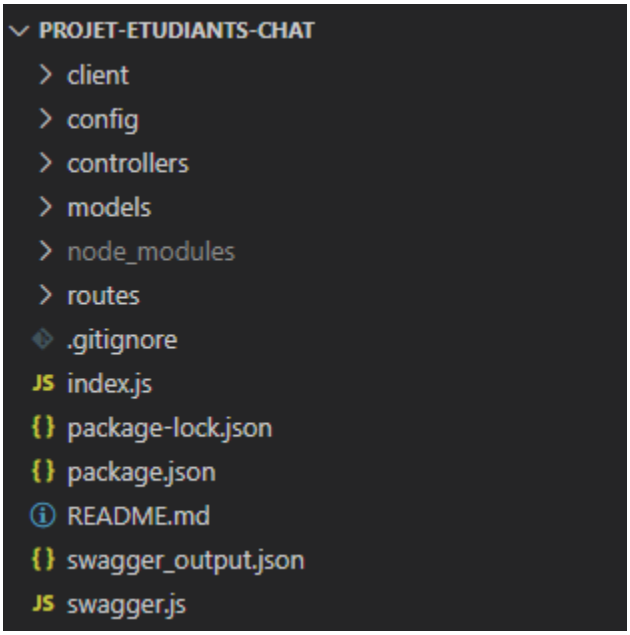

Figure 2 : Capture d'écran des différents dossiers du code source

Dans le répertoire courant, se trouvent les fichiers liés au GitLab (README.md, .gitignore), au Swagger (swagger.js, swagger_output.json), au projet node (package.json, package-lock.json) ainsi que le fichier index.js qui lance le serveur.

Dans le dossier **client**, se trouve l'ensemble des fichiers qui constituent le front de notre interface de démonstration ainsi que les ressources.

Dans le dossier **config**, se trouve les fichiers de configurations pour la connexion à la base de données MongoDB. Nous retrouvons les fichiers :

- index.js
- mongo.js

Dans le dossier **controllers**, se trouvent les méthodes qui sont utilisées lors de l'appel des routes API définies. Nous retrouvons les fichiers :

- member.js
- room.js
- user.js

Dans le dossier **models**, se trouvent les fichiers qui définissent les schémas pour chaque table de la base de données ainsi que les méthodes associées.

Nous retrouvons les fichiers :

- member.js
- message.js
- middleware.js
- room.js
- user.js

Dans le dossier **routes**, se trouvent les fichiers qui définissent les routes API disponibles. Nous retrouvons les fichiers :

- login.js
- member.js
- room.js
- user.js.

4. Schéma de la base de données

Pour le développement de l'API de chat, une base de données NoSQL MongoDB a été utilisée pour permettre le stockage des différentes données relatives aux utilisateurs de chat, les rooms de discussions ainsi que les messages envoyés.

Cette base de données est constituée des 4 tables suivantes : users, rooms, members et messages.

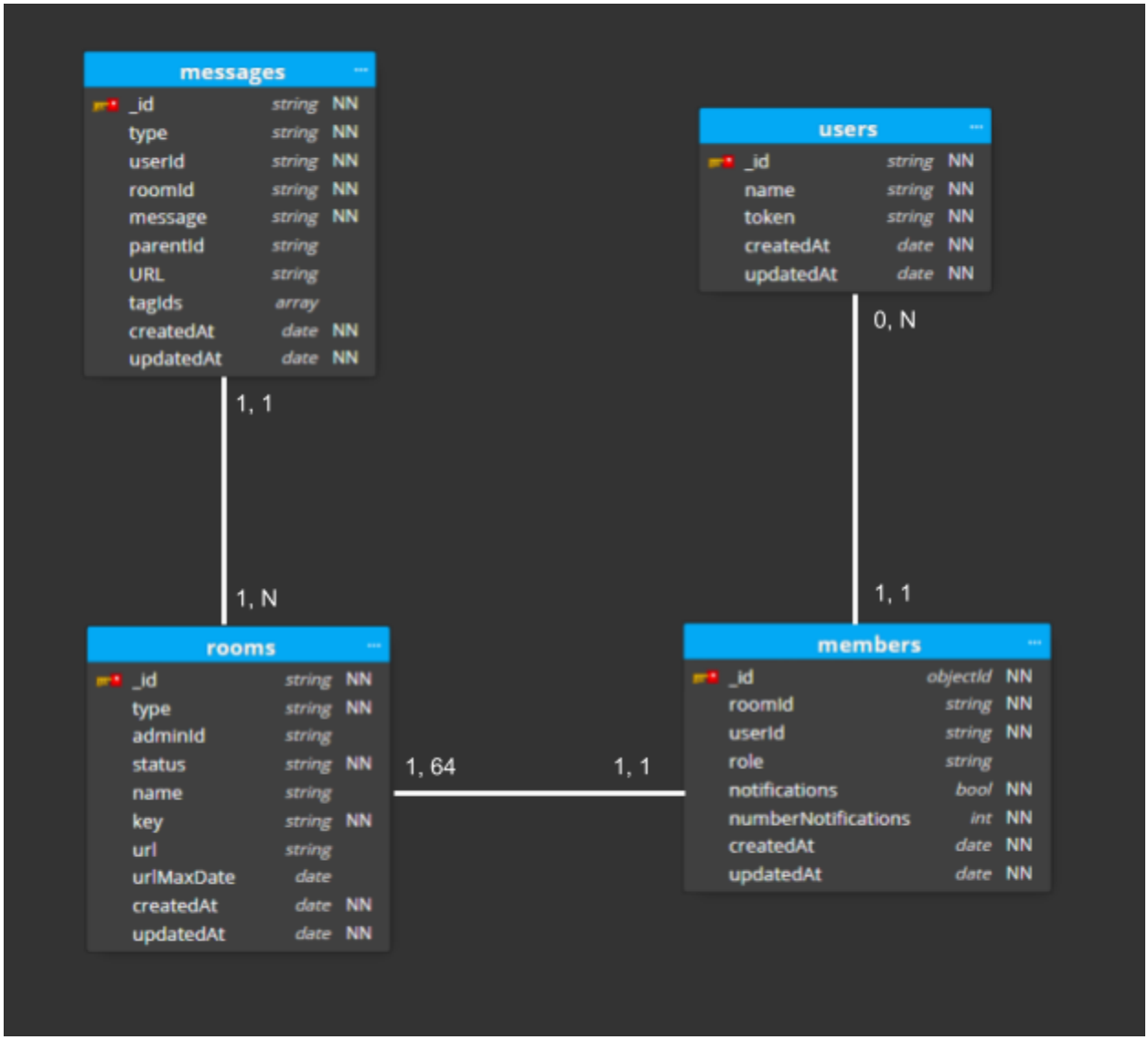

Figure 3 : Schéma de la base de données adoptée

Description des tables :

users	
Nom du champ	**Description**
_id	Id de l'objet user automatiquement généré à partir d'un UUID version 4.
name	Nom de l'utilisateur. Champs créé pour plus de facilité de lecture pendant le projet.
token	Token associé à l'utilisateur au moment de sa création. Généré automatiquement à partir d'un UUID version 4. Permet l'authentification de l'utilisateur avec le couple **_id-token** lors de la connexion sur l'interface de démonstration.
createdAt	Champ généré automatiquement lors de la création de l'objet. Stocke la date de création et la date de modification de l'objet.
updatedAt	

members	
Nom du champ	**Description**
_id	Id de l'objet généré automatiquement par la base de données à la création.
roomId	Id de la room associé à ce membre.
userId	Id de l'utilisateur associé à ce membre.
role	Rôle du membre dans la room. Ce champ est fixé à **null** dans le cas d'une room en one-to-one. Dans le cas d'une room à plusieurs, ce champ peut avoir la valeur **admin**.
notifications	Booléen permettant de fixer la valeur liée à la réception de notifications. Par défaut, cette valeur est fixée à **true**, signifiant que le membre reçoit des notifications de la room.
numberNotifications	Integer qui stocke le nombre de notifications reçues par le membre dans une room. Cette valeur est remise à zéro à

members	
Nom du champ	**Description**
	l'ouverture de la conversation.
createdAt	Champs généré automatiquement lors de la création de l'objet. Stocke la date de création et la date de modification de l'objet.
updatedAt	

rooms	
Nom du champ	**Description**
_id	Id de l'objet room automatiquement généré à partir d'un UUID version 4.
type	Type de room créé. Les valeurs peuvent être **1on1** dans le cas d'une discussion entre 2 personnes et **group** dans le cas d'une conversation à plusieurs.
adminId	Id de l'utilisateur qui a créé la room de groupe. Dans le cas d'une conversation simple, ce champ est **null**.
status	Statut de la room de groupe. Par défaut, la valeur est fixée à **active**. Lorsque la room n'a plus de membres, la valeur devient **inactive**.
name	Nom de la room de la groupe. Cette valeur n'est pas obligatoire.
key	Clé AES de la room pour le chiffrement/déchiffrement des messages. Généré automatiquement avec nanoid.
url	Url d'invitation pour une room de groupe générée à la demande. Par défaut, ce champ est vide.
urlMaxDate	Date maximale de validité de l'url d'invitation. Par défaut, ce champ est vide.
createdAt	Champs généré automatiquement lors de la création de l'objet. Stocke la date de création et la date de modification de l'objet.
updatedAt	

messages	
Nom du champ	**Description**
_id	Id de l'objet message automatiquement généré à partir d'un UUID version 4.
type	Type de message stocké en base de données. Il peut s'agir de : - **-1** : un ancien message envoyé qui a été supprimé. - **0** : un message de type texte envoyé. - **1** : un message bot indiquant qu'un utilisateur a rejoint ou quitté une discussion. - **2** : un message avec une photo envoyée. - **3** : un message avec une vidéo envoyée. - **4** : un message avec une note vocale envoyée.
userId	Id de l'utilisateur qui a envoyé le message.
roomId	Id de la room dans laquelle le message a été envoyé.
message	Le texte du message envoyé.
parentId	Id du message auquel on répond. Cette valeur est vide par défaut.
URL	L'URL vers le S3 AWS pour le média envoyé.
tagIds	Un tableau contenant les id des membres tagués dans le message.
createdAt	Champs généré automatiquement lors de la création de l'objet. Stocke la date de création et la date de modification de l'objet.
updatedAt	

5. Description des sockets utilisés

Pour tous les échanges entre client et serveur, Socket.io est utilisé, et c'est le serveur qui fait les requêtes vers la base de données.

Avant de pouvoir faire des requêtes au serveur, le client doit d'abord se connecter, et doit donc envoyer son _id et son token, qui correspondent alors au chatId, et au token hashé de Manao. Ces deux paramètres sont placés dans le socket.auth, puis socket.connect est

appelé. Côté serveur, nous vérifions si ces deux paramètres sont dans la BDD, mais il faut changer ceci pour vérifier dans la BDD de Manao.

Après connection, les échanges via socket fonctionnent comme ceci :

Socket	**Emetteur**	**Cas d'utilisation**	**Paramètre(s)**
set keys	Serveur	Une fois le client connecté, le serveur lui envoie les clés des rooms auxquelles il appartient	Keys : Object , avec keys[roomId] = key
get all conversations	Client	Le client veut la liste des conversations	
set all conversations	Serveur	→ 'get all conversations' Le serveur envoie les conversations (le dernier message de chaque room, le nom de la room, et le nom de l'émetteur).	listConversations[Object] : contient des conv{ id, type, tagIds, message, roomId, createdAt, userId, nbNotifications, senderName, roomName }
send message	Client	Le client envoie un message	message : String messageURL : String tagIds[] parentId : String type : String
receive message	Serveur	→ 'send message' → 'join group room' → 'leave room' → 'kick user' Le serveur envoie un message à tous les membres de la room	message : Message user : User
create group room	Client	Le client veut créer une room de groupe	name : String (nom de la room)
join group room	Client	Le client veut entrer dans une room de groupe existante via URL	roomURL : String
enter group room	Serveur	→ 'create group room' → 'enter group room' → 'open room' Le serveur indique au client d'entrer dans la room	name : String (nom de la room) roomId : String

Socket	Emetteur	Cas d'utilisation	Paramètre(s)
create 1to1 room	Client	Le client veut créer une room 1on1	userId : String (id de l'autre utilisateur)
enter 1to1 room	Serveur	→ 'create 1to1 room' à 'open room' Le serveur indique au client d'entrer dans la room	userId : String roomId : String
add key	Serveur	→ 'create group room' → 'enter group room' → 'create 1to1 room' Le serveur envoie une nouvelle clé au client	key : String id : String (roomId)
add conversation	Serveur	→ 'create group room' → 'enter group room' → 'create 1to1 room' Le serveur envoie une nouvelle conversation au client	Conv : Object { id, type, tagIds, message, roomId, createdAt, userId, nbNotifications, senderName, roomName }
leave room	Client	Le client souhaite quitter une room	roomId : String
has left room	Serveur	→ 'leave room' → 'kick user' Le serveur confirme que le client ne fait plus partie de la room	
open room	Client	Le client cique sur une conversation	roomId : String
display notifications	Serveur	→ 'open room' Le serveur affiche si les notifications du client dans la room sont activées ou non	notifications : Bool
read conv	Client	Le client a cliqué sur une conversation	roomId : String
has read conv	Serveur	→ 'read conv' Le nombre de notifications sur la room est bien à 0 pour le client	
get message	Client	Le client souhaite récupérer les messages de la room dans laquelle il se truve	limit : Integer
print message	Serveur	→ 'get message' (limit) Le serveur renvoie les 'limit' derniers messages de la room	

Socket	Emetteur	Cas d'utilisation	Paramètre(s)
is user admin	Client	Le client demande s'il est admin de la room	
is admin	Serveur	→ 'is user admin' Le serveur réponds, et ce seulement si l'utilisateur est bien admin	
create URL	Client	Le client admin souhaite générer l'URL d'invitation de sa room	date : Date
get url	Serveur	→ 'create URL' Le serveur renvoie l'URL généré	url : String
change notifications	Client	Le client souhaite activer ou désactiver les notifications	notifications : Bool
kick user	Client	Le client admin souhaite virer un utilisateur d'une room	userId : String
change group name	Client	Le client admin souhaite renommer la room	roomId : String name : String
update room name	Serveur	→ 'change group name' Le serveur renvoie le nouveau nom à tous les utilisateurs de la room	room : Room
delete message	Client	Le client souhaite supprimer un message	messageId : String
hide message	Serveur	Le serveur indique à tous les utilisateurs que le message a été supprimé	messageId : String roomId : String
get members in room	Client	Le client souhaite avoir accès à la liste des membres de sa room	
display members in room	Serveur	→ 'get members in room' Le serveur renvoie la liste des membres de la room	memberList [Member]

6. Fonctionnement de l'interface de démonstration

Au lancement du serveur, on arrive sur la page de login sur laquelle se trouve un formulaire de connexion. Il faut alors fournir l'id et le token de l'utilisateur pour être redirigé à la page suivante. (cf. Figure 4)

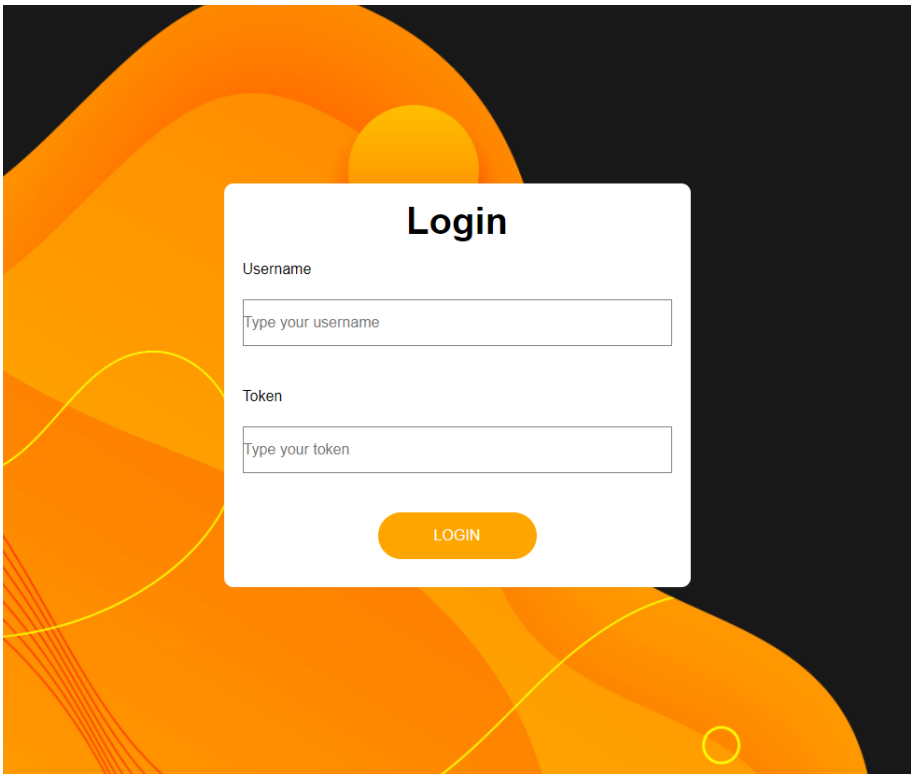

Figure 4 : Capture d'écran de la page de login

Une fois l'authentification réussie, on est redirigé sur une page de dashboard sur lequel on retrouve plusieurs éléments :

- une boîte de création de room en 1-to-1 dans laquelle se trouve une liste déroulante avec l'ensemble des utilisateurs du chat. **1**
- une boîte de création de room de groupe dans laquelle l'utilisateur peut renseigner un nom de groupe. **2**

- une boîte pour rejoindre une room de groupe grâce à un url à indiquer dans la barre de saisie. **3**
- une boîte où on visualise l'ensemble des conversations de l'utilisateur avec le dernier message reçu. **4**

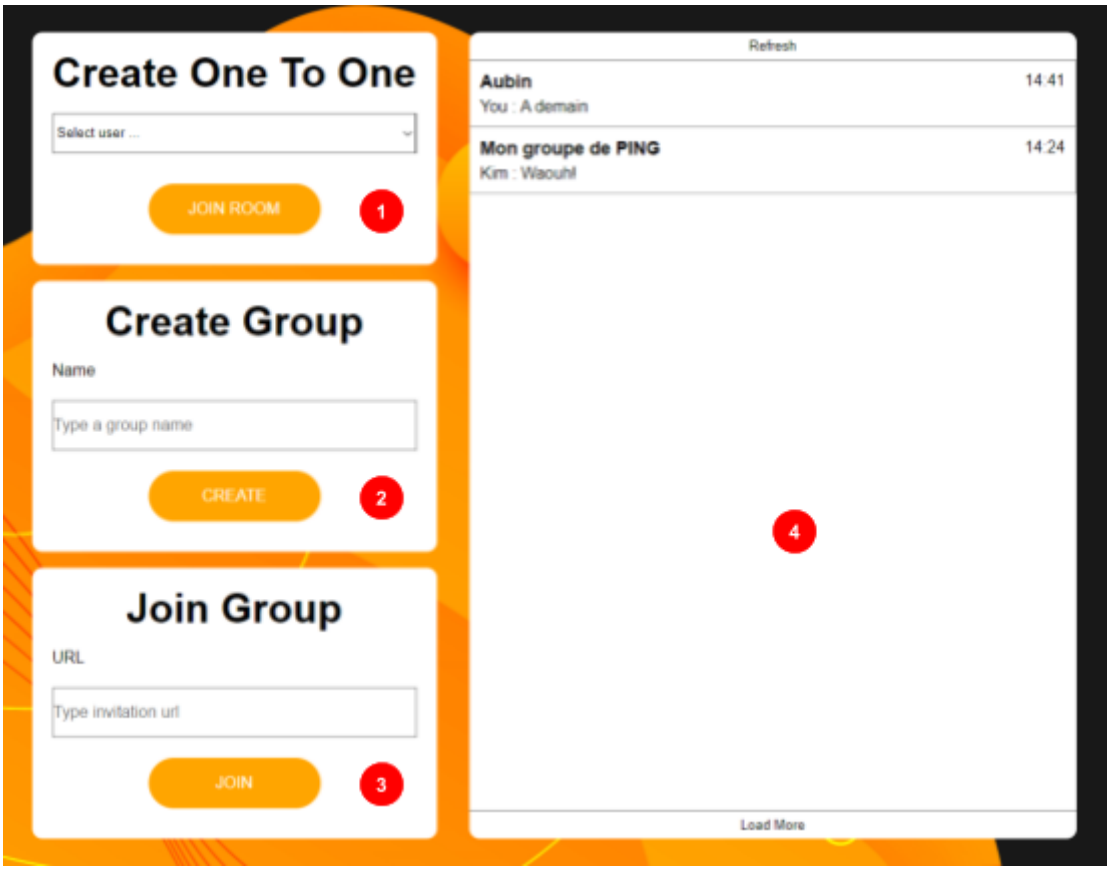

Figure 5 : Capture d'écran de la page de dashboard

Enfin, lorsqu'on décide de rejoindre une conversation, on est redirigé sur la page suivante :

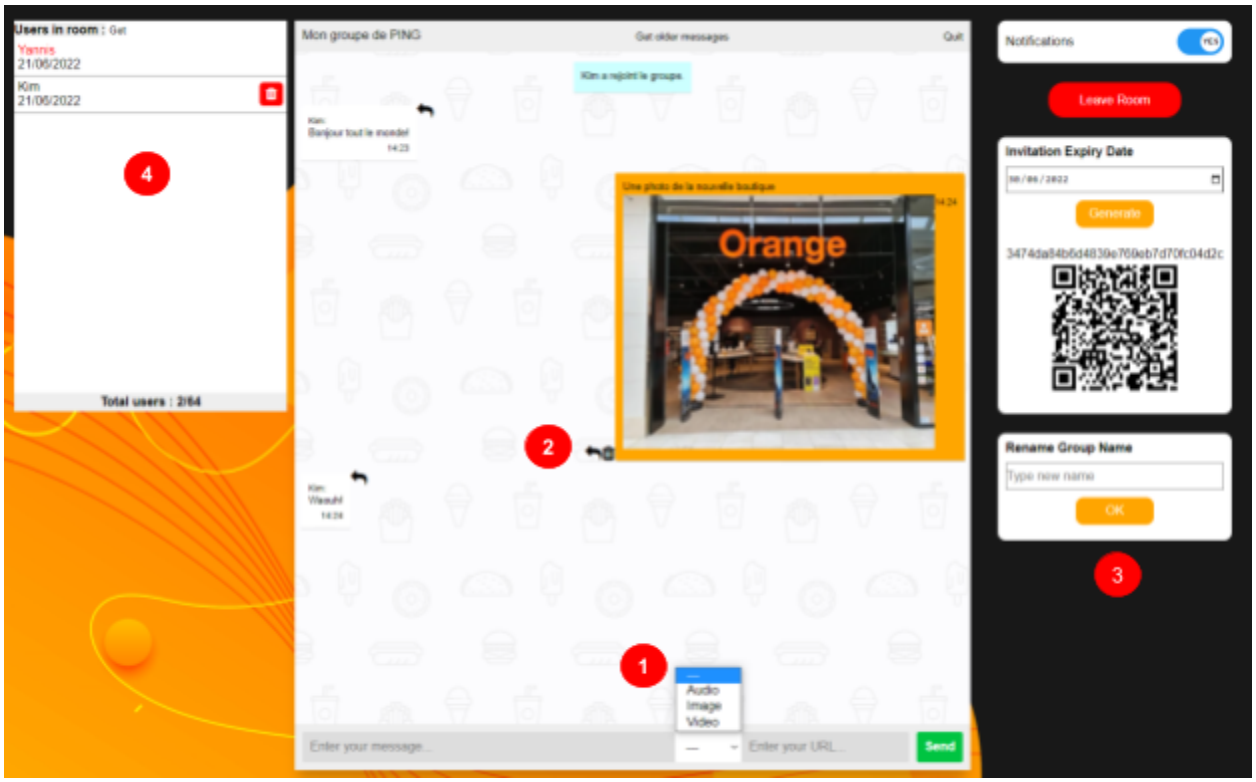

Figure 6 : Capture d'écran d'une page de chat de groupe

Sur cette page, nous retrouvons les éléments suivants :

- La boîte de chat où il est possible de visualiser les messages de la conversation, mais aussi d'en envoyer en remplissant la barre de saisie avec un message. Il est également possible d'envoyer une image, une vidéo ou une note vocale en sélectionnant le type du média dans la liste déroulante et en fournissant l'URL du média. **1**

  Pour supprimer un message, il suffit de cliquer sur l'icône **poubelle** située à côté du message en question. **2**

  Pour répondre à un message, il suffit de rédiger son message et de cliquer sur l'icône de **réponse** située à côté du message en question. **2**

- Les différentes options concernant la room de discussion.

  Le bouton de commande **Notifications** permet à l'utilisateur d'activer ou de désactiver les notifications pour cette conversation.

  Le bouton **Leave room** permet à un utilisateur de quitter définitivement une conversation de groupe. Lorsque l'administrateur décide de quitter une conversation, son rôle est automatiquement ré-attribué au membre le plus ancien de la conversation.

  La boîte de **génération d'invitation** permet à l'administrateur de générer un url d'invitation valable jusqu'à une certaine date. Ceci affiche également le qr code associé.

  La boîte de **changement de nom** permet à l'administrateur de renommer la room.

  Certaines options sont disponibles pour tous (bouton de commande **Notifications** et le bouton **Leave room**) et d'autres sont réservées à l'administrateur de la room (la boîte de génération d'invitation et la boîte pour changer le nom de la conversation). **3**

- La liste des membres de la room avec leur date d'ajout. L'administrateur est indiqué en rouge. En face de chaque membre (hormis l'administrateur) se trouve un bouton **poubelle** qui permet à l'administrateur d'expulser un membre. **4**