

# VUE & TypeScript

## 1. Vue d'Ensemble

Cette application est une **interface de messagerie en temps réel** permettant à un utilisateur connecté :

- De **rejoindre des salons (rooms)** dynamiques,
- D'**envoyer et recevoir des messages** dans différents onglets de discussion,
- De **gérer sa session utilisateur** via un système de login JWT,
- D'**interagir dans une UI moderne et responsive** via TailwindCSS.
- Utilisation de la root pour la page d'accueil donc "/" et pour le profil "/profil"

Les messages sont transmis via **WebSocket** (grâce à Socket.io), permettant un échange instantané.

---

## 2. Structure Principale

### 2.1 Fichier **Home.vue** — Page d'accueil

Ce composant représente la page principale **/home**. Il gère la logique d'entrée dans les rooms et l'affichage conditionnel du chat.

#### Responsabilités :

- Charger les composants Header, ListGame et ChatTextuel.
- Gérer les rooms rejointes.
- Déclencher le login.
- Connecter au serveur WebSocket.

#### Fonctionnement général :

ts

```
// Liste des rooms déjà rejointes
const joinedRooms = ref<{ id: number|string, name: string }[]>([])

// Quand une room est rejointe via ListGame
function handleJoinRoom(roomId: number|string, roomName: string) {
  if (!joinedRooms.value.some(r => r.id === roomId)) {
    joinedRooms.value = [...joinedRooms.value, { id: roomId, name:
roomName }]
  }
  joinRoom(roomId.toString())
  chatTextuelRef.value?.openRoomTab(roomId) // Ouvre un onglet dans
ChatTextuel
}
```

La méthode `chatTextuelRef.value?.openRoomTab(roomId)` permet d'ouvrir dynamiquement un onglet correspondant dans `ChatTextuel.vue`.

---

## 2.2 Fichier `ChatTextuel.vue` — Composant de Chat Persistant

Ce composant est **présent en permanence** dans la page (sauf pour les utilisateurs non connectés). Il s'ouvre via un bouton en bas à droite.

### Fonctionnalités clés :

- Affiche un chat multitar (All, room1, room2, ...).
- Gère la réception des messages via WebSocket (`onMessage`, `onMessageAll`).
- Gère l'envoi de messages dans les bons canaux.
- Maintient un historique par room.
- Scroll automatique en bas à chaque message.

### Architecture des données `messages` :

ts

```
// Messages stockés par room (clé = id room ou "All")
const messages = reactive<{
  [key: string]: {
    text: string
    from: 'user' | 'bot' | 'other'
  }
}>()
```

```

    pseudo?: string
    imageUrl?: string
  }[]
}>({
  'All': []
})

```

### Flux WebSocket (réception) :

ts

```

// Messages dans "All"
handlerAll = (data) => {
  messages['All'].push({
    text: data.content,
    from: data.sender === getSocketId() ? 'user' : 'other',
    pseudo: data.pseudo,
    imageUrl: data.imageUrl
  })
}
onMessageAll(handlerAll)

```

### Envoi de messages :

ts

```

function sendMessageHandler() {
  if (selectedTab.value === 'All') {
    sendMessageAll(text, myPseudo.value, myImageUrl.value)
  } else {
    sendMessage(selectedTab.value, text, myPseudo.value,
myImageUrl.value)
  }
  messages[selectedTab.value].push({ ... })
}

```

## 2.3 Fichier `index.ts` — Router pour définir les différentes routes

```

const routes = [
  { path: '/', name: 'Home', component: Home },
  { path: '/profile', name: 'Profile', component: UserProfileWrapper },
];

```

```
const router = createRouter({
  history: createWebHistory(),
  routes,
});
```

### 3. Authentification JWT

Le token JWT est **généré côté backend** et récupéré via un cookie.

- La fonction `getJwtFromCookie()` est utilisée pour déterminer si l'utilisateur est connecté.
- En cas de non-authentification, le composant `ChatTextuel` n'est **pas monté**.

html


```
<ChatTextuel
  ref="chatTextuelRef"
  v-if="getJwtFromCookie()"
  :joined-rooms="joinedRooms"
  @leave-room="handleLeaveRoom"
/>
```

### 4 . UI & Design - TailwindCSS

L'UI est basée sur Tailwind avec un style responsive et clair.


**Exemple de bouton flottant (ouvrir le chat) :**

html

```
<button class="fixed bottom-4 right-4 bg-blue-600 text-white px-4 py-4
rounded-full shadow-lg hover:bg-blue-700 transition">
  
</button>
```

**Modal avec tabs dynamiques :**

html

```
<button class="fixed bottom-4 right-4 bg-blue-600 text-white px-4 py-4 rounded-full shadow-lg hover:bg-blue-700 transition">  
    
</button>
```

Le design est **minimal mais efficace**, facilement extensible à du Dark Mode ou à des avatars custom.

## 5 . Gestion du Profil Utilisateur

Cette vue permet à un utilisateur connecté de :

- Modifier son prénom, nom, pseudo
- Changer ou ajouter une photo de profil
- Modifier son mot de passe
- Supprimer son compte

### Principales fonctionnalités :

- Les champs sont pré-remplis à partir du `props.user`.
- Une image peut être uploadée via un champ `<input type="file" />`, et envoyée à l'API avec `FormData`.
- Si un nouveau mot de passe est saisi, il doit être confirmé.
- Le token JWT est extrait des cookies pour authentifier chaque requête.

### Exemple de code :

ts

```
const form = reactive<UserForm>({  
  prenom: props.user.prenom ?? '',  
  nom: props.user.nom ?? '',  
  pseudo: props.user.pseudo ?? '',  
})
```

```

    image: props.user.image ?? null,
    password: '',
    passwordConfirm: ''
  })

```

### Fonction `submit()` :

ts

```

const submit = async () => {
  if (form.password && form.password !== form.passwordConfirm) {
    message.value = "Les mots de passe ne correspondent pas."
    return
  }

  if (imageFile.value) {
    form.image = await uploadImage(imageFile.value)
  }

  const res = await fetch('http://localhost:8000/api/me/edit', {
    method: 'PATCH',
    headers: {
      'Content-Type': 'application/merge-patch+json',
      Authorization: `Bearer ${token}`
    },
    body: JSON.stringify(body)
  })
}

```

### Suppression de compte :

ts

```

const deleted = async () => {
  await fetch('http://localhost:8000/api/me/deleted', {
    method: 'DELETE',
    headers: {
      Authorization: `Bearer ${token}`
    }
  })
}

```

```
}
```

## 6 . Services Utilitaires

### 6.1 IsConnected.ts

Utilitaire pour récupérer le token JWT depuis les cookies :

ts

```
export function getJwtFromCookie(): string | null {  
  const match = document.cookie.match(new RegExp('(^\s*)jwt=([^\s;]+)'))  
  return match ? match[2] : null  
}
```

### 6.2 TakeSpeudoImage.ts

Utilisé pour récupérer dynamiquement le pseudo et l'image d'un utilisateur (ex : à l'ouverture de `Home.vue`).

ts

```
export async function getSpeudoImage() {  
  const res = await fetch('http://localhost:8000/api/me', {  
    headers: { Authorization: `Bearer ${getJwtFromCookie()}` }  
  })  
  const user = await res.json()  
  myPseudo.value = user.pseudo  
  myImageUrl.value = user.image  
}
```

# Websocket

## Partie 1 – Architecture WebSocket & Intégration dans le serveur

### Objectif

Gérer la communication en temps réel pour un système de rooms multijoueurs (jeu d'échecs, chat, spectateurs), tout en s'appuyant sur MongoDB pour persister les données (messages, rooms).

## Mise en place de Socket.IO et MongoDB

js

```
const { Server } = require("socket.io");
const mongoose = require("mongoose");
const express = require("express");
const app = express();

const MONGO_URL = "mongodb://mongodb:27017/all";

mongoose.connect(MONGO_URL, {
  useNewUrlParser: true,
  useUnifiedTopology: true,
})
.then(() => {
  const httpServer = app.listen(3002);
  const io = new Server(3001, {
    cors: { origin: "*", methods: ["GET", "POST"] },
  });
  // ...
});
```

Deux serveurs sont lancés :

- **HTTP (port 3002)** → pour accéder à l'API REST ([/history](#), [/rooms](#))
- **WebSocket (port 3001)** → pour la communication en temps réel via [Socket.IO](#)

## Middleware CORS

js

```
app.use((req, res, next) => {
  res.header("Access-Control-Allow-Origin", "*");
  res.header("Access-Control-Allow-Methods", "GET, POST");
  next();
});
```

💡 Permet aux clients de domaines différents (front-end) d'accéder aux endpoints.

---



## Partie 2 – Gestion des Rooms et Attribution des Rôles

### Objectif

Créer un système où plusieurs clients peuvent rejoindre une "room" de jeu. Chaque room peut avoir :

- 1 joueur 1
- 1 joueur 2
- Des spectateurs

### Implémentation

js

```
const roomPlayers = {}; // stocke les joueurs par room

socket.on("join-room", (roomId) => {
  roomPlayers[roomId] = roomPlayers[roomId] || [];

  if (!roomPlayers[roomId].includes(socket.id)) {
    roomPlayers[roomId].push(socket.id);
  }

  const room = roomPlayers[roomId];

  if (!room.player1) {
    room.player1 = socket.id;
  } else if (!room.player2) {
    room.player2 = socket.id;
  } else {
    room.spectators = socket.id;
  }

  socket.join(roomId);
  io.to(roomId).emit("room-players", {
    roomId,
    count: room.length,
    players: {
      player1: room.player1,
      player2: room.player2,
      spectators: room.spectators,
    },
  });
});
```

```
});
```

## Recherche de rooms

js

```
socket.on('search_rooms', async ({ name }) => {  
  const rooms = await Room.find({  
    name: { $regex: name, $options: 'i' }  
  }).limit(20);  
  socket.emit('search_results', { rooms });  
});
```

Utilisation de regex MongoDB pour une recherche partielle insensible à la casse (\$options: 'i').

---

## Partie 3 – Communication & Échanges en Temps Réel

### Chat global et messages persistés

js

```
socket.on("messageAll", async (messageAll) => {  
  io.emit("messageAll", {  
    sender: socket.id,  
    content: messageAll.content,  
    pseudo: messageAll.pseudo,  
    imageUrl: messageAll.imageUrl,  
  });  
  
  await Message.create({  
    roomId: "all",  
    sender: socket.id,  
    content: messageAll.content,  
    pseudo: messageAll.pseudo,  
    imageUrl: messageAll.imageUrl,  
    timestamp: new Date(),  
  });  
});
```

```
});
```

Tous les messages globaux sont :

- diffusés à tous les clients connectés
  - sauvegardés dans MongoDB dans la collection `messages`
- 

## Message de room

js

```
socket.on("message", ({ roomId, message, pseudo, imageUrl }) => {
  io.to(roomId).emit("message", {
    roomId,
    content: message,
    sender: socket.id,
    pseudo,
    imageUrl
  });
});
```

Ces messages sont **diffusés uniquement aux clients connectés à une room spécifique**, sans persistance.

---

## Mouvement de jeu (échecs)

js

```
socket.on("chess-move", ({ roomId, move }) => {
  io.to(roomId).emit("chess-move", { roomId, move });
});
```

Simplicité du système : les mouvements ne sont **pas sauvegardés** ici, uniquement transmis.

## Partie 4 – Modèles de données Mongoose (MongoDB)

## Objectif

Utiliser MongoDB via Mongoose pour stocker **les messages échangés** et **les rooms créées**. Cela permet de :

- consulter l'historique ([/history](#))
- retrouver les rooms disponibles ([/rooms](#))
- stocker les actions importantes (création, suppression, messages globaux)

---

## Modèle **Message**

js

```
const mongoose = require('mongoose');

const MessageSchema = new mongoose.Schema({
  roomId: { type: String, required: true },
  sender: { type: String, required: true },
  content: { type: String, required: true },
  pseudo: { type: String },
  imageUrl: { type: String },
  timestamp: { type: Date, default: Date.now }
});

module.exports = mongoose.model('Message', MessageSchema);
```

# Symfony

## Introduction

Ce backend est développé avec le framework **Symfony** et s'appuie sur **API Platform** pour l'exposition automatique des entités sous forme d'API REST. Le système est enrichi par :

- des **DTO** pour dissocier les entités des données exposées ou attendues,
- des **State Processors** pour implémenter la logique métier spécifique,
- et un système d'**authentification JWT** (via [lexik/jwt-authentication-bundle](#)) permettant la sécurisation des endpoints.

- des entités créées par les commandes de symfony.

---

## Entity: Exemple de la structure correcte de l'entity

```
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column]
    private ?int $id = null;

    #[ORM\Column(length: 255)]
    private ?string $pseudo = null;

    #[ORM\Column(length: 255)]
    private ?string $email = null;

    #[ORM\Column(type: Types::TEXT)]
    private ?string $password = null;

    #[ORM\Column(length: 255, nullable: true)]
    private ?string $nom = null;

    #[ORM\Column(length: 255, nullable: true)]
    private ?string $prenom = null;

    /**
     * @var Collection<int, Role>
     */
    #[ORM\ManyToMany(targetEntity: Role::class, mappedBy: 'user_id')]
    private Collection $roles;

    #[ORM\Column(type: Types::TEXT, nullable: true)]
    private ?string $image = null;

    public function __construct()
    {
        $this->roles = new ArrayCollection();
    }

    public function getId(): ?int
    {
        return $this->id;
    }
}
```

```

public function getPseudo(): ?string
{
    return $this->pseudo;
}

public function setPseudo(string $pseudo): static
{
    $this->pseudo = $pseudo;

    return $this;
}

```

suite de l'entity ici :

<https://github.com/melwin-duquenne/roomis/blob/main/back/src/Entity/User.php>

Les entités me permettent de créer ma base de données mais aussi d'implémenter simplement et correctement avec api platform.

## Fonctionnalité : Authentification avec JWT

### But

Permettre à un utilisateur de se connecter avec son email et mot de passe, puis recevoir un **token JWT** à inclure dans les appels protégés.

### Endpoint concerné

Une route personnalisée d'API Platform traite la connexion via un **State Processor**, indépendamment du système `security.yaml`.

### DTO d'entrée : `LoginInput`

Représente les données reçues pour une tentative de connexion.

php

```

namespace App\ApiResource\login;

class LoginInput
{
    public string $email;
    public string $password;
}

```

## DTO de sortie : **LoginOutput**

Structure standardisée de réponse après traitement de la connexion.

php

```
namespace App\Dto\login;

class LoginOutput
{
    public function __construct(
        public bool $success,
        public ?string $token = null,
        public ?string $pseudo = null,
        public ?string $image = null,
        public ?string $message = null,
    ) {}
}
```

## State : **LoginProcessor**

Traitement du login personnalisé via un **ProcessorInterface**.

php

```
class LoginProcessor implements ProcessorInterface
```

Ce processor :

1. Vérifie le type de l'input reçu.
2. Cherche l'utilisateur en base par email.
3. Valide le mot de passe.
4. Génère un token JWT s'il est valide.
5. Retourne une instance de **LoginOutput**.

**Extrait de code :**

php

```
$user = $this->userRepository->findOneBy(['email' => $data->email]);
if (!$user || !$this->passwordHasher->isPasswordValid($user,
$data->password)) {
```

```
return new LoginOutput(false, message: 'Identifiants invalides');
}
$token = $this->jwtManager->create($user);
return new LoginOutput(true, $token, $user->getPseudo(),
$user->getImage(), 'Connexion réussie');
```

## Authentification sécurisée avec LexikJWTAuthenticationBundle

L'authentification repose sur un **JWT** généré à la connexion. Le token encode l'identité de l'utilisateur et peut être utilisé comme **Bearer Token** dans les requêtes HTTP sécurisées.

### Génération du token

Utilise le service :

php

```
$jwtManager->create($user);
```

Le token est ensuite renvoyé dans la réponse **LoginOutput**.

---

## Exemple de flux de connexion

### Requête **POST /api/login**

json

```
{
  "email": "jean@example.com",
  "password": "motdepasse"
}
```

### Réponse attendue (succès)

json

```
{
  "success": true,
  "token": "eyJhbGciOiJIUzI1...",
  "pseudo": "JeanDupont",
  "image": "avatar.png",
}
```



```
"message": "Connexion réussie"
}
```

## Réponse (échec)

json

```
{
  "success": false,
  "token": null,
  "pseudo": null,
  "image": null,
  "message": "Identifiants invalides"
}
```

## Récupération du profil (GET /api/me)

### Description

Permet à un utilisateur connecté de récupérer **ses propres informations**.

### Implémentation

Le provider `UserMeProvider` est utilisé pour exposer uniquement les champs utiles via un **DTO spécifique** `UserMeResource`.

### DTO : `UserMeResource`

php

```
class UserMeResource
{
    public ?int $id;
    public ?string $pseudo;
    public ?string $email;
    public ?string $prenom;
    public ?string $nom;
    public ?string $image;
}
```

### Traitement

- Récupération de l'utilisateur courant avec `$security->getUser()`

- Transfert des données dans le DTO
- Retour de l'objet DTO si authentifié, sinon exception `AccessDeniedException`

### Exemple de réponse

json

```
{
  "id": 12,
  "pseudo": "JeanDupont",
  "email": "jean@example.com",
  "prenom": "Jean",
  "nom": "Dupont",
  "image": "avatar.png"
}
```

---

## Mise à jour du profil (PATCH /api/me/update)

### Description

Permet à l'utilisateur de modifier ses **informations personnelles** (pseudo, image, prénom, nom, mot de passe...).

### DTO d'entrée : `UserUpdateInput`

php

```
class UserUpdateInput
{
    public ?string $pseudo = null;
    public ?string $prenom = null;
    public ?string $nom = null;
    public ?string $password = null;
    public ?string $image = null;
}
```

### Traitement via `UserUpdateProcessor`

1. Récupération de l'utilisateur connecté.
2. Mise à jour conditionnelle des champs non nuls.
3. Validation de l'unicité du `pseudo`.

4. Hash du mot de passe si modifié.
5. Persistance via Doctrine ([EntityManager](#)).

### Cas particulier : pseudo déjà pris

Une exception est levée si un autre utilisateur utilise déjà ce pseudo :

php

```
if ($existingUser && $existingUser !== $user) {  
    throw new \RuntimeException('Pseudo déjà utilisé.');
```

### Exemple de requête

json

```
{  
  "pseudo": "JeanD",  
  "prenom": "Jean",  
  "nom": "Dumont",  
  "password": "nouveauMotDePasse",  
  "image": "avatar-v2.png"  
}
```

### Réponse

Retourne l'utilisateur à jour (formaté selon la configuration [ApiResponse](#) de l'entité [User](#)).

---

### Sécurité

Ces deux endpoints nécessitent que l'utilisateur soit **authentifié avec un JWT** valide. Le token doit être passé dans les en-têtes HTTP :

makefile

```
Authorization: Bearer <votre_token>
```

# Test back

## Outil utilisé

Les tests sont réalisés avec [PHPUnit](#), un framework de test unitaire pour PHP.

bash

```
composer require --dev phpunit/phpunit
```

## Structure des tests

Les tests sont organisés dans le dossier `tests/` :

mathematica

```
tests/
├── Entity/
│   ├── RoleTest.php
│   └── UserTest.php
├── State/
│   ├── RegisterProcessorTest.php
│   ├── UserUpdateProcessorTest.php
│   ├── LoginProcessorTest.php
│   ├── UserMeProviderTest.php
│   └── UserUpdateProcessorTest.php
```

## 1. `UserUpdateProcessorTest` – Mise à jour du profil utilisateur

exemple de Tests couverts :

Test	Description
<code>testSuccessfulUpdate()</code>	Vérifie la mise à jour complète d'un utilisateur avec nom, prénom, pseudo, password

`testPseudoAlreadyTaken()` Vérifie qu'une exception est levée si le pseudo est déjà utilisé par un autre utilisateur

### Mocks utilisés :

- `Security` : simule l'utilisateur connecté (`getUser`)
  - `EntityManagerInterface` : vérifie que les appels à `persist()` et `flush()` sont effectués
  - `UserPasswordHasherInterface` : simule le hash du mot de passe
  - `UserRepository` : contrôle le comportement de recherche du pseudo
- 

## 2. RegisterProcessorTest – Inscription utilisateur

Emplacement : `tests/State/RegisterProcessorTest.php`

### Tests couverts :

Test	Description
<code>testInvalidData()</code>	Teste l'appel avec un objet non conforme (mauvais type)
<code>testInvalidEmail()</code>	Rejette un email non valide
<code>testPseudoAlreadyUsed()</code>	Vérifie que l'inscription échoue si le pseudo existe déjà
<code>testEmailAlreadyUsed()</code>	Vérifie que l'inscription échoue si l'email est déjà utilisé
<code>testSuccessfulRegistration()</code>	Teste l'inscription complète d'un nouvel utilisateur valide

### Mocks utilisés :

- `UserRepository` : contrôle la vérification de doublons `pseudo` / `email`
- `RoleRepository` : renvoie un rôle fictif (`ROLE_USER`)

- `UserPasswordHasherInterface` : simule le hash du mot de passe
  - `EntityManagerInterface` : vérifie les appels à `persist()` et `flush()`
- 

### 3. UserTest – Entité User

Emplacement : `tests/Entity/UserTest.php`

Tests couverts :

Test	Description
<code>testUserProperties()</code>	Vérifie les getters/setters standards
<code>testDefaultRoles()</code>	Vérifie que l'utilisateur a au moins <code>ROLE_USER</code>
<code>testAddRole()</code>	Vérifie l'ajout d'un rôle <code>ROLE_ADMIN</code>
<code>testRemoveRole()</code>	Vérifie la suppression d'un rôle
<code>testEraseCredentialsDoesNothing()</code>	Vérifie que <code>eraseCredentials()</code> est neutre (conformité interface <code>UserInterface</code> )

## Test Front

### 1. Objectif des tests E2E

Les tests End-to-End (E2E) visent à valider les **principaux parcours utilisateur** dans l'application via le navigateur, de manière automatisée, en simulant un comportement réel.

Les tests couvrent des cas comme :

- L'inscription et la connexion d'un utilisateur
- La gestion du profil utilisateur
- La création/suppression de rooms
- L'envoi de messages dans le chat

---

## 2. Stack technique

Outil	Rôle
Cypress	Framework de test E2E
Vite	Serveur dev frontend (localhost:5173)
Cookie JWT	Vérification de l'authentification

---

## 3. Liste des tests

Fichier	Cas de test couverts
<code>auth.cy.js</code>	<ul style="list-style-type: none"><li>- Ouverture du modal d'authentification</li><li>- Inscription utilisateur</li><li>- Connexion utilisateur</li></ul>
<code>profile.cy.js</code>	<ul style="list-style-type: none"><li>- Navigation au profil</li><li>- Mise à jour du profil</li><li>- Suppression du compte</li></ul>
<code>room.cy.js</code>	<ul style="list-style-type: none"><li>- Création de room</li><li>- Suppression de room</li></ul>
<code>message.cy.js</code>	<ul style="list-style-type: none"><li>- Envoi d'un message dans la messagerie</li></ul>

---

## 4. Structure d'un test Cypress

Chaque fichier de test suit cette structure :

js

```
describe('Nom du test', () => {
  beforeEach(() => {
    cy.visit('http://localhost:5173/');
  });

  it('effectue une action', () => {
    // Étapes du test
  });
});
```

```
});  
});
```

Les **valeurs dynamiques** comme l'email ou le pseudo sont générées avec `Date.now()` pour éviter les conflits entre runs :

js

```
const testEmail = `user${Date.now()}@test.com`;
```

---

## 5. Bonnes pratiques

- **Sélecteurs robustes** : usage de `data-testid` pour cibler les éléments critiques.
- **Timeouts gérés** : usage de `{ timeout: 20000 }` pour les opérations asynchrones.
- **Nettoyage implicite** : le `beforeEach` remet l'état à zéro avec `cy.visit()`.
- **Tests indépendants** : chaque `it()` est autonome (pas de dépendance entre tests).
- **Vérification de l'état d'authentification** : `cy.getCookie('jwt')`.

---

## 6. Exécution des tests

**En mode graphique (pour le debug) :**

bash

```
npx cypress open
```

**En mode headless (CI) :**

bash

```
npx cypress run
```