Anthony RIMET: IUT Nancy-Charlemagne 2024

TD 4 - Vue - Tic Tac Toe

Objectif

L'objectif de ce TD est de créer une application web de Tic Tac Toe en utilisant Vue.js. Il est possible de faire ce TD en binôme.

Exercice 1 : Initialiser le Projet

En tant que développeur, je veux initialiser un nouveau projet Vue.js pour créer une application web de Tic Ta Toe.

- Initialiser un nouveau projet Vue.
- Ajouter vue-router au projet.
- Créer une view `Login`. Ajouter un template simple "Page login".
- Créer une vue `Register`. Ajouter un template simple "Page d'inscription".
- Créer une vue `Dashboard`. Ajouter un template simple "Tableau de bord".

Client HTTP

Vous allez créer un client HTTP pour interagir avec une API backend. Suivez les étapes ci-dessous pour compléter l'exercice. Certains détails sont fournis, mais vous devrez implémenter certaines parties du code vous-même.

Le backend de l'API vous est fourni dans le dossier `backend`. Vous pouvez l'exécuter en utilisant la commande `npm run dev` après avoir fait un `npm install`. L'API tourne sur le port 3001.

Étape 1 : Définir l'URL de base

1. Créez un nouveau fichier nommé `services/httpClient.js`
2. Définissez l'URL de base pour vos endpoints API en haut du fichier.

```
const BASE_URL = 'http://localhost:3001';
```

Étape 2 : Créer la fonction de requête

1. Écrivez une fonction asynchrone nommée request qui prend les paramètres suivants :
    - endpoint (string) : L'adresse de l'API.
    - method (string) : La méthode HTTP (par défaut "GET").
    - body (object) : Le corps de la requête (par défaut `null`).
    - isAuthRequest (boolean) : Indicateur pour indiquer si la requête nécessite une authentification (par défaut `false`).

```
```

```
const request = async (
    endpoint,
    method = 'GET',
    body = null,
    isAuthRequest = false,
) => {
    // Récupérer le token
    const token = localStorage.getItem('token');
    // Définir les en-têtes
    const headers = {
        'Content-Type': 'application/json',
        ...(!isAuthRequest && token && { Authorization: `Bearer ${token}`
}),
    };

    // Configurer la requête
    const config = {
        method,
        headers,
        ...(body && { body: JSON.stringify(body) }),
    };

    try {
        // Envoyer la requête
        const response = await fetch(`${BASE_URL}${endpoint}`, config);

        if (response.status === 403 || response.status === 401) {
            // Token expired or invalid, remove token and redirect to login
            removeToken();
            window.location.href = '/login';
            return;
        }

        // Gérer la réponse
        if (!response.ok) {
            const errorBody = await response.json();
            throw new Error(errorBody.message || 'Something went wrong');
        }

        // Vérifier si la réponse a du contenu
        const contentType = response.headers.get('content-type');
        return contentType && contentType.includes('application/json')
            ? await response.json()
            : null;
    } catch (error) {
        // Gestion des erreurs
        console.error('API Error:', error);
        throw error;
    }
};
```

Étape 3 : Créer les méthodes d'authentification

1. Créez des fonctions pour la connexion et l'inscription qui utilisent la fonction request.

```
const login = (username, password) => {
    return request('/login', 'POST', { username, password }, true);
};

const register = (username, password) => {
    return request('/register', 'POST', { username, password }, true);
};
```

Étape 4 : Exporter les fonctions

1. Exportez les fonctions pour qu'elles puissent être utilisées dans toute votre application.

```
export { request, login, register };
```

AuthProvider

Dans le fichier `services/AuthProvider.js`, créez un contexte d'authentification avec les fonctionnalités suivantes :

- isAuthenticated() : Un état pour indiquer si l'utilisateur est connecté. Il va chercher le token dans le local storage.
- removeToken() : Une fonction pour supprimer le token du local storage.
- setToken(token) : Une fonction pour enregistrer le token dans le local storage.
- getUserIdentity() : Une fonction pour récupérer l'utilisateur à partir du token. Récupère le token du local storage et le décode pour obtenir les informations de l'utilisateur.

Pensez à les exporter pour les utiliser dans votre application.

Exercice 2: Login et Register

En tant qu'utilisateur, je veux pouvoir me connecter et m'inscrire pour accéder à l'application.

Register

Étape 1 : Importer les Services

Importer la fonction nécessaire depuis les services : `register` depuis `../services/httpClient`

Étape 2 : Définir les Données du Composant

Définir les données du composant :

- `username` : chaîne de caractères vide pour stocker le nom d'utilisateur.
- `password` : chaîne de caractères vide pour stocker le mot de passe.
- `confirmPassword` : chaîne de caractères vide pour confirmer le mot de passe.
- `error` : null pour stocker les messages d'erreur.

Étape 3 : Créer la Méthode d'Inscription

- Définir une méthode handleRegister :
- Vérifier si les mots de passe correspondent.
- Appeler la fonction register avec le nom d'utilisateur et le mot de passe.
- En cas de succès, rediriger vers la page de connexion en utilisant `this.$router.push("/login")`.
- Gérer les erreurs en les assignant à this.error.

Étape 4 : Créer le Template

- Créer un formulaire avec des champs pour le nom d'utilisateur, le mot de passe et la confirmation du mot de passe.
- Ajouter un bouton de submit.
- Afficher les messages d'erreur si nécessaire.
- Ajouter un lien pour rediriger vers la page de connexion.
- Faites en sorte que cela soit joli. Vous pouvez utiliser des classes CSS ou Tailwind (ou autre).

Login

Étape 1 : Importer les Services

Importer les fonctions nécessaires depuis les services :

- `setToken depuis @/services/authService`
- `login depuis @/services/httpClient`

Étape 2 : Définir les Données du Composant

Définir les données du composant :

- `username` : chaîne de caractères vide pour stocker le nom d'utilisateur.
- `password` : chaîne de caractères vide pour stocker le mot de passe.
- `error` : null pour stocker les messages d'erreur.

Étape 3 : Créer la Méthode de Connexion

Définir une méthode `handleLogin` :

- Appeler la fonction `login` avec le nom d'utilisateur et le mot de passe. Si tout se passe bien, la fonction retournera un token. (voir API Documentation)
- Stocker le token en utilisant `setToken`.
- Rediriger vers le Dashboard en utilisant `this.$router.push("/")`.
- Gérer les erreurs en les assignant à this.error.

Étape 4 : Créer le Template

Créer le template HTML :

- Créer un formulaire avec des champs pour le nom d'utilisateur et le mot de passe.
- Ajouter un bouton submit.
- Afficher les messages d'erreur si nécessaire.
- Ajouter un lien pour rediriger vers la page d'inscription.

Router

On va protéger les routes de l'application pour que l'utilisateur ne puisse pas accéder au tableau de bord sans être connecté. Pour cela, on va utiliser la propriété `meta` des routes de vue-router.

Étape 1 : Ajouter les Meta Données

Ajouter des métadonnées aux routes de vue-router pour indiquer si la route nécessite une authentification.

Quelque chose comme ça :

```
{
    path: "/register",
    name: "Register",
    component: Register,
    meta: { requiresGuest: true },
  },
  {
    path: "/",
    name: "Dashboard",
    component: Dashboard,
    meta: { requiresAuth: true },
  },
```

Étape 2 : Protéger les Routes

Ajouter un guard à la navigation pour protéger les routes qui nécessitent une authentification.

```
router.beforeEach((to, from, next) => {
  const isLogged = isAuthenticated();
  if (to.meta.requiresAuth && !isLogged) {
    ....
  }
});
```

Étape 3: Afficher l'utilisateur connecté

Dans le composant `Dashboard`, affichez le nom de l'utilisateur connecté. Vous pouvez utiliser la fonction `getUserIdentity` du `AuthProvider` pour obtenir les informations de l'utilisateur. Cette function devra

décoder le token et retourner les informations de l'utilisateur. Pour ça utilisez la librairie `jwt-decode`.

Ajoutez un bouton de déconnexion qui appelle la fonction `removeToken` du `AuthProvider` pour déconnecter l'utilisateur.

Avec ça vous devriez avoir une application fonctionnelle avec un système de connexion et d'inscription.

Pensez à vérifier que le Dashboard n'est pas accessible sans être connecté.

Exercice 3: Profile

En tant qu'utilisateur, je veux pouvoir voir et modifier mon profil.

A partir de maintenant, vous serez moins guidé dans la réalisation de l'exercice. Vous devrez réfléchir à la structure de vos composants et à la manière de les organiser.

Nous voulons que l'utilisateur puisse voir son profil et modifier ses informations. Il pourra aussi se logout depuis ici.

Une fois modifié, vous devriez voir sur la page Dashboard le nom de l'utilisateur mis à jour. Pensez à mettre des liens pour naviguer entre les pages.

Tips: Pensez à créer des fonctions `getUser` et `updateUser` dans le `httpClient` pour récupérer et mettre à jour les informations de l'utilisateur. L'API a une route pour modifier les informations de l'utilisateur et qui retourne un nouvel token, qu'en faire ?

Tips 2: N'hésitez pas à appeler votre professeur si vous avez des questions ou des problèmes. C'est normal de ne pas tout savoir et de demander de l'aide.

Exercice 4: Dashboard

En tant qu'utilisateur, je veux pouvoir voir la liste des parties, créer une partie et rejoindre celle non finie.

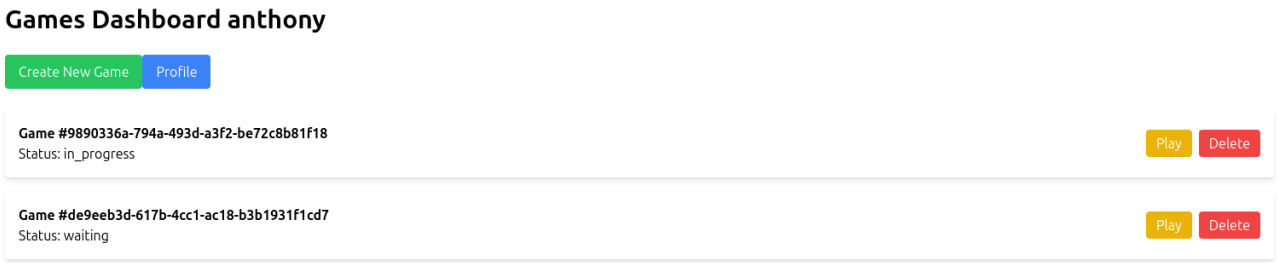
Le dashboard devra afficher la liste des parties en cours. L'utilisateur pourra créer une nouvelle partie ou rejoindre une partie non finie. Pour cela, vous allez devoir créer plusieurs méthodes dans le `httpClient` pour récupérer la liste des parties, créer une partie et rejoindre une partie ou la supprimer si vous êtes le créateur.

Regarder la documentation de l'API pour savoir comment interagir avec les endpoints.

Affichez les informations, pensez bien à mettre des indicateurs pour votre utilisateur pour savoir si la partie est en cours ou terminée.

Au clic sur une partie, vous devrez être redirigé vers la page de jeu. **Pour cela, vous devrez passer l'id de la partie dans l'url de votre application. Regardez la documentation de vue-router pour savoir comment passer et récupérer les paramètres de l'url.**

Tips: Un exemple d'interface pour vous aider à visualiser ce que vous devez faire. Il manque surement des informations, à vous de les ajouter.



Exercice 5: Game

En tant qu'utilisateur, je veux pouvoir jouer à un jeu de Tic Tac Toe.

Créez une page pour jouer au jeu de Tic Tac Toe. Pensez à découper votre page en sous-composants pour plus de lisibilité.

En arrivant sur cette page vous allez devoir récupérer les informations de la partie en cours. **Pour ce faire vous devrez passer l'id de la partie dans l'url de votre application. Regardez la documentation de vue-router pour savoir comment passer et récupérer les paramètres de l'url.**

Une partie est stocké sous la forme suivante:

```
 {
"id": "9890336a-794a-493d-a3f2-be72c8b81f18",
"creator": "55c0657e-2080-465d-8fdd-9e5fd7934a76",
"player1": "55c0657e-2080-465d-8fdd-9e5fd7934a76",
"player2": "55c0657e-2080-465d-8fdd-2sads33aasds",
"board": [
  null,
  null,
  "55c0657e-2080-465d-8fdd-9e5fd7934a76",
  null,
  "55c0657e-2080-465d-8fdd-2sads33aasds",
  null,
  null,
  null,
  null
],
"status": "in_progress",
"currentPlayer": "55c0657e-2080-465d-8fdd-9e5fd7934a76",
},
```

Il faut afficher le nom des joueurs, le tour du joueur et le plateau de jeu. Si il n'y a pas de joueur 2, vous devrez afficher un message d'attente.

Pour avoir un effet temps réel, il existe plusieurs solutions. Vous pouvez utiliser les WebSockets ou les Server-Sent Events. Ces méthodes demandent de la configuration côté serveur. Par simplicité, nous allons

utiliser une méthode plus simple: le polling.

Le polling consiste à envoyer une requête à intervalle régulier pour récupérer les nouvelles informations. Pour cela, vous pouvez utiliser la fonction `setInterval` de JavaScript.

Voici un exemple de code pour récupérer les informations de la partie toutes les 5 secondes (a placer au bon endroit dans votre composant):

```
async created() {
  await this.fetchGameState();
  this.startPolling();
}
beforeUnmount() {
  this.stopPolling();
}
methods: {
  startPolling() {
    if (this.polling) return;
    this.polling = setInterval(this.fetchGameState, 10000); // Poll every
10 seconds
  },
  stopPolling() {
    if (this.polling) {
      clearInterval(this.polling);
      this.polling = null;
    }
  },
}
```

Tips 1: Vous allez devoir définir vos data réactives pour stocker les informations de la partie. Pensez à bien les initialiser. Voici un example que j'ai utilisé pour mon POC rapide

```
 data() {
    return {
      gameId: null,
      gameState: null,
      board: Array(3)
        .fill(null)
        .map(() => Array(3).fill(null)), // 3x3 board pour affichage
      currentPlayer: null,
      winner: null,
      error: null,
      userId: null,
      polling: null,
      playerNames: {
        player1: null,
        player2: null,
      },
    };
  },
```

Tips 2: L'API sauvegarde les coups joués dans un tableau de 9 cases. Vous devrez convertir ce tableau en une matrice 3x3 pour l'affichage. Voici un exemple de code pour le faire:

```
formatBoard(flatBoard) {
    // Convertit le tableau 1D en un tableau 2D (3x3) pour l'affichage
    return [
      [flatBoard[0], flatBoard[1], flatBoard[2]],
      [flatBoard[3], flatBoard[4], flatBoard[5]],
      [flatBoard[6], flatBoard[7], flatBoard[8]],
    ];
  },
```

Tips 3: Donnez envie à l'utilisateur de jouer. Faite en sorte que l'utilisateur voit quand il peut jouer et quand il ne peut pas. Vous pouvez utiliser des classes CSS pour cela. Faite un design simple et efficace.

Encouragements

Ce TD est un peu plus complexe que les précédents. Vous allez devoir réfléchir à la structure de vos composants et à la manière de les organiser. Vous allez voir des notions que vous n'avez pas encore appliquées comme les guards de vue-router, les contextes d'authentification, le httpClient, etc. Si vous avez des difficultés, n'hésitez pas à demander de l'aide. C'est normal de ne pas tout savoir et de demander de l'aide. Vous pouvez aussi regarder la documentation de Vue.js pour trouver des réponses à vos questions.

Bon courage ! Vous êtes sur la bonne voie pour devenir un développeur web complet !

API Documentation

Method	Route	Body	Description	Response Example
POST	/register	`{ "username": "string", "password": "string" }`	Register a new user	`{ "message": "User created", "userId": "uuid" }`
POST	/login	`{ "username": "string", "password": "string" }`	Login a user and return a JWT token	`{ "token": "jwt_token" }`
GET	/user/:id	N/A	Get user information by ID	`{ "id": "uuid", "username": "string" }`

Method	Route	Body	Description	Response Example
PUT	/user	`{ "username": "string", "password": "string" }`	Update username and password	`{ "message": "User updated", "user": { "id": "uuid", "token": "new_jwt_token", "username": "string" } }`
POST	/games	N/A	Create a new game	`{ "id": "uuid", "creator": "uuid", "player1": "uuid", "player2": null, "board": [null, null, null, null, null, null, null, null, null], "status": "waiting", "currentPlayer": "uuid" }`
GET	/games	N/A	List all available games	`[ { "id": "uuid", "creator": "uuid", "player1": "uuid", "player2": null, "board": [null, null, null, null, null, null, null, null, null], "status": "waiting", "currentPlayer": "uuid" } ]`
POST	/games/:gameId/join	N/A	Join a game by ID	`{ "id": "uuid", "creator": "uuid", "player1": "uuid", "player2": "uuid", "board": [null, null, null, null, null, null, null, null, null], "status": "in_progress", "currentPlayer": "uuid" }`

Method	Route	Body	Description	Response Example
POST	/games/:gameId/move/:row/:col	N/A	Make a move in the game	`{ "id": "uuid", "creator": "uuid", "player1": "uuid", "player2": "uuid", "board": ["uuid", null, null, null, null, null, null, null, null], "status": "in_progress", "currentPlayer": "uuid" }`
DELETE	/games/:gameId	N/A	Delete a game by ID	`{ "message": "Game deleted" }`
GET	/games/:gameId	N/A	Get game state by ID	`{ "id": "uuid", "creator": "uuid", "player1": "uuid", "player2": "uuid", "board": [null, null, null, null, null, null, null, null, null], "status": "waiting", "currentPlayer": "uuid" }`

Game Status

- `waiting` : Ce statut est utilisé lorsque la partie vient d'être créée et qu'elle attend qu'un deuxième joueur la rejoigne.
- `in_progress` : Ce statut est utilisé lorsque les deux joueurs sont présents et que la partie est en cours.
- `finished` : Ce statut est utilisé lorsque la partie est terminée et qu'un joueur a gagné.
- `draw` : Ce statut est utilisé lorsque la partie est terminée sans gagnant (toutes les cases sont remplies sans combinaison gagnante).