TP4 : Connexion Frontend/Backend

Vue.js 3 + Spring Boot + H2

Ibrahim ALAME

Année 2024-2025

Objectif

À l'issue de ce TP, vous serez capable de :

- Créer une API REST avec Spring Boot
- Configurer une base de données H2 (base embarquée)
- Connecter votre application Vue.js à l'API backend
- Effectuer les opérations CRUD (Create, Read, Update, Delete)
- Gérer les problèmes de CORS

Table des matières

1 Introduction et Prérequis

1.1 Point de départ

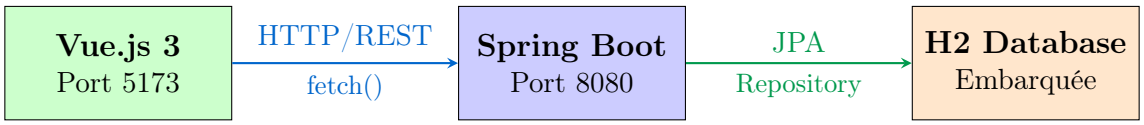
Vous disposez du projet frontend Vue.js développé lors du TP3, disponible sur :

`https://github.com/ialame/vuejs-boutique-tp4`

Attention

Actuellement, les données des produits sont stockées dans le fichier `src/data/product.ts`. L'objectif de ce TP est de remplacer ces données statiques par une vraie base de données accessible via une API REST.

1.2 Architecture cible



1.3 Prérequis logiciels

- Node.js 18+ et npm
- Java 17+ (JDK)
- Maven ou utilisation de Spring Initializr
- Un IDE : VS Code, IntelliJ IDEA, ou Eclipse

2 Partie 1 : Création du projet Spring Boot (30 min)

▶ Rappel de cours

Spring Boot est un framework Java qui simplifie la création d'applications web. Il fournit :

- Une configuration automatique (auto-configuration)
- Un serveur web embarqué (Tomcat)
- Des starters pour ajouter facilement des dépendances

2.1 Exercice 1.1 : Générer le projet

1. Rendez-vous sur `https://start.spring.io/`
2. Configurez le projet avec les paramètres suivants :
   - **Project** : Maven
   - **Language** : Java
   - **Spring Boot** : 3.2.x (dernière version stable)
   - **Group** : `com.pokeshop`

- **Artifact** : `backend`
- **Name** : `backend`
- **Packaging** : Jar
- **Java** : 17

3. Ajoutez les dépendances suivantes :
   - Spring Web — Pour créer l'API REST
   - Spring Data JPA — Pour la persistance des données
   - H2 Database — Base de données embarquée
   - Spring Boot DevTools — Rechargement automatique
   - Lombok — Réduction du code boilerplate (optionnel)
4. Cliquez sur **Generate** et décompressez l'archive

Question 1

Quelle est la différence entre une base de données embarquée (H2) et une base de données externe (MySQL, PostgreSQL) ? Dans quel contexte utilise-t-on chacune ?

2.2 Exercice 1.2 : Configuration de H2

Ouvrez le fichier `src/main/resources/application.properties` et ajoutez :

```
# Configuration du serveur
server.port=8080

# Configuration H2
spring.datasource.url=jdbc:h2:mem:pokeshop
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=

# Console H2 (acces web a la BDD)
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console

# JPA/Hibernate
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.jpa.hibernate.ddl-auto=create-drop
spring.jpa.show-sql=true
```

▶ Rappel de cours

`spring.jpa.hibernate.ddl-auto` contrôle la gestion du schéma :

- `create-drop` : Crée les tables au démarrage, les supprime à l'arrêt
- `create` : Recrée les tables à chaque démarrage
- `update` : Met à jour le schéma sans perdre les données
- `validate` : Vérifie le schéma sans le modifier

2.3 Exercice 1.3 : Vérification

1. Lancez l'application : `mvn spring-boot:run`
2. Ouvrez `http://localhost:8080/h2-console`
3. Connectez-vous avec :
   - JDBC URL : `jdbc:h2:mem:pokeshop`
   - User : `sa`
   - Password : (vide)

3 Partie 2 : Modèle de données et Repository (45 min)

▶ Rappel de cours

En Spring Data JPA, l'architecture suit le pattern suivant :

- **Entity** : Classe Java mappée sur une table de la BDD
- **Repository** : Interface pour les opérations CRUD
- **Service** (optionnel) : Logique métier
- **Controller** : Expose les endpoints REST

3.1 Exercice 2.1 : Création de l'entité Product

Analysons d'abord les interfaces TypeScript du frontend :

```typescript
// src/interfaces/Product.interface.ts (Frontend Vue.js)
import type { Extension } from '@/interfaces/type'

export interface ProductInterface {
  id: number;
  title: string;
  image: string;
  price: number;
  description: string;
  quantity?: number | undefined;
  extension: Extension;
}
```

```typescript
// src/interfaces/type.ts
export type Extension = 'jungle' | 'fossile' | 'expedition' | 'aquapolis' | 'all';
```

Question 2

En vous basant sur ces interfaces TypeScript, créez la classe Java `Product` correspondante dans le package `com.pokeshop.backend.model`.
Indices :

- Utilisez l'annotation `@Entity` pour marquer la classe comme entité JPA
- Utilisez `@Id` et `@GeneratedValue` pour l'identifiant

- Pensez à créer un enum pour l'extension

Créez d'abord l'énumération pour les extensions :

```java
// src/main/java/com/pokeshop/backend/model/Extension.java
package com.pokeshop.backend.model;

public enum Extension {
    JUNGLE,
    FOSSILE,
    EXPEDITION,
    AQUAPOLIS,
    ALL
}
```

Puis l'entité Product :

```java
// src/main/java/com/pokeshop/backend/model/Product.java
package com.pokeshop.backend.model;

import jakarta.persistence.*;

@Entity
@Table(name = "products")
public class Product {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private String title;

    private String image;

    @Column(nullable = false)
    private Double price;

    @Column(length = 1000)
    private String description;

    @Enumerated(EnumType.STRING)
    private Extension extension;

    // Constructeur par défaut (requis par JPA)
    public Product() {}

    // Constructeur avec paramètres
    public Product(String title, String image, Double price,
                   String description, Extension extension) {
        this.title = title;
        this.image = image;
        this.price = price;
        this.description = description;
        this.extension = extension;
    }
```

```
    // TODO: Générer les getters et setters
    // (ou utiliser @Data de Lombok)
}
```

Attention

N'oubliez pas de générer les **getters** et **setters** pour tous les attributs ! Si vous utilisez Lombok, ajoutez simplement `@Data` au-dessus de la classe.

3.2 Exercice 2.2 : Création du Repository

▶ Rappel de cours

Spring Data JPA fournit des interfaces de base qui génèrent automatiquement les méthodes CRUD :

- `CrudRepository` : Opérations CRUD de base
- `JpaRepository` : Étend CrudRepository avec pagination et tri

Question 3

Créez l'interface `ProductRepository` dans le package `com.pokeshop.backend.repository`. Cette interface doit :

- Étendre `JpaRepository<Product, Long>`
- Ajouter une méthode pour rechercher par extension

```
// src/main/java/com/pokeshop/backend/repository/ProductRepository.java
package com.pokeshop.backend.repository;

import com.pokeshop.backend.model.Product;
import com.pokeshop.backend.model.Extension;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import java.util.List;

@Repository
public interface ProductRepository extends JpaRepository<Product, Long> {

    // Spring Data génère automatiquement l'implémentation !
    List<Product> findByExtension(Extension extension);

    // Recherche par titre (contient)
    List<Product> findByTitleContainingIgnoreCase(String title);
}
```

3.3 Exercice 2.3 : Données initiales

Pour tester notre API, ajoutons des données au démarrage. Créez un composant d'initialisation :

```
// src/main/java/com/pokeshop/backend/DataInitializer.java
package com.pokeshop.backend;
```

```
import com.pokeshop.backend.model.*;
import com.pokeshop.backend.repository.ProductRepository;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

@Component
public class DataInitializer implements CommandLineRunner {

    private final ProductRepository productRepository;

    public DataInitializer(ProductRepository productRepository) {
        this.productRepository = productRepository;
    }

    @Override
    public void run(String... args) {
        // Produits initiaux
        productRepository.save(new Product(
                "Carte Pikachu Rare",
                "/images/103.jpg",
                29.99,
                "Carte Pikachu édition limitée, état mint",
                Extension.JUNGLE
        ));

        productRepository.save(new Product(
                "Booster Pack Fossile",
                "/images/104.jpg",
                4.99,
                "Pack de 10 cartes aléatoires",
                Extension.FOSSILE
        ));

        productRepository.save(new Product(
                "Carte Dracaufeu",
                "/images/105.jpg",
                199.99,
                "Carte holographique rare",
                Extension.EXPEDITION
        ));

        productRepository.save(new Product(
                "Pack Aquapolis",
                "/images/111.jpg",
                12.99,
                "Extension Aquapolis complète",
                Extension.AQUAPOLIS
        ));

        System.out.println("=== Données initialisées : "
                + productRepository.count() + " produits ===");
    }
}
```

Attention

Évitez les caractères accentués dans les fichiers Java et properties pour éviter les erreurs d'encodage Maven. Si vous souhaitez utiliser des accents, ajoutez dans votre `pom.xml` :

```
<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>
```

4 Partie 3 : Création de l'API REST (45 min)

▶ Rappel de cours

Une API REST utilise les méthodes HTTP pour les opérations CRUD :

Méthode HTTP	Opération	Exemple
GET	Lire	`GET /api/products`
POST	Créer	`POST /api/products`
PUT	Mettre à jour (complet)	`PUT /api/products/1`
PATCH	Mettre à jour (partiel)	`PATCH /api/products/1`
DELETE	Supprimer	`DELETE /api/products/1`

4.1 Exercice 3.1 : Le contrôleur REST

Question 4

Créez le contrôleur `ProductController` qui expose les endpoints suivants :

- `GET /api/products` — Liste tous les produits
- `GET /api/products/{id}` — Récupère un produit par son ID
- `POST /api/products` — Crée un nouveau produit
- `PUT /api/products/{id}` — Met à jour un produit
- `DELETE /api/products/{id}` — Supprime un produit

```java
// src/main/java/com/pokeshop/backend/controller/ProductController.java
package com.pokeshop.backend.controller;

import com.pokeshop.backend.model.Product;
import com.pokeshop.backend.repository.ProductRepository;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import java.util.List;
import java.util.Optional;

@RestController
@RequestMapping("/api/products")
public class ProductController {
```

```
    private final ProductRepository productRepository;

    public ProductController(ProductRepository productRepository) {
        this.productRepository = productRepository;
    }

    // GET /api/products
    @GetMapping
    public List<Product> getAllProducts() {
        return productRepository.findAll();
    }

    // GET /api/products/{id}
    @GetMapping("/{id}")
    public ResponseEntity<Product> getProductById(@PathVariable Long id) {
        Optional<Product> product = productRepository.findById(id);
        return product.map(ResponseEntity::ok)
                      .orElse(ResponseEntity.notFound().build());
    }

    // POST /api/products
    @PostMapping
    @ResponseStatus(HttpStatus.CREATED)
    public Product createProduct(@RequestBody Product product) {
        return productRepository.save(product);
    }

    // PUT /api/products/{id}
    @PutMapping("/{id}")
    public ResponseEntity<Product> updateProduct(
            @PathVariable Long id,
            @RequestBody Product productDetails) {

        return productRepository.findById(id)
            .map(product -> {
                product.setTitle(productDetails.getTitle());
                product.setImage(productDetails.getImage());
                product.setPrice(productDetails.getPrice());
                product.setDescription(productDetails.getDescription());
                product.setExtension(productDetails.getExtension());
                return ResponseEntity.ok(productRepository.save(product));
            })
            .orElse(ResponseEntity.notFound().build());
    }

    // DELETE /api/products/{id}
    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteProduct(@PathVariable Long id) {
        if (productRepository.existsById(id)) {
            productRepository.deleteById(id);
            return ResponseEntity.noContent().build();
        }
        return ResponseEntity.notFound().build();
    }
}
```

4.2 Exercice 3.2 : Test de l'API

Redémarrez l'application et testez avec `curl` ou un outil comme Postman :

```
# Lister tous les produits
curl http://localhost:8080/api/products

# Récupérer un produit
curl http://localhost:8080/api/products/1

# Créer un produit
curl -X POST http://localhost:8080/api/products \
  -H "Content-Type: application/json" \
  -d '{"title":"Nouveau","price":9.99,"category":"POKEMON"}'

# Supprimer un produit
curl -X DELETE http://localhost:8080/api/products/1
```

Question 5

Testez chaque endpoint et notez les codes de réponse HTTP. Que se passe-t-il quand vous demandez un produit qui n'existe pas ?

5 Partie 4 : Configuration CORS (15 min)

▶ Rappel de cours

Le CORS (Cross-Origin Resource Sharing) est un mécanisme de sécurité des navigateurs. Par défaut, une page web ne peut pas faire de requêtes vers un domaine différent. Notre frontend tourne sur `localhost:5173` et le backend sur `localhost:8080`. Sans configuration CORS, les requêtes seront bloquées !

5.1 Exercice 4.1 : Configuration globale CORS

Créez une classe de configuration :

```
// src/main/java/com/pokeshop/backend/config/CorsConfig.java
package com.pokeshop.backend.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.cors.CorsConfiguration;
import org.springframework.web.cors.UrlBasedCorsConfigurationSource;
import org.springframework.web.filter.CorsFilter;

@Configuration
public class CorsConfig {

    @Bean
    public CorsFilter corsFilter() {
        CorsConfiguration config = new CorsConfiguration();

        // Autoriser le frontend Vue.js
        config.addAllowedOrigin("http://localhost:5173");
```

```
        // Méthodes HTTP autorisées
        config.addAllowedMethod("*");

        // Headers autorisés
        config.addAllowedHeader("*");

        UrlBasedCorsConfigurationSource source =
                new UrlBasedCorsConfigurationSource();
        source.registerCorsConfiguration("/**", config);

        return new CorsFilter(source);
    }
```

Attention

En production, remplacez `http://localhost:5173` par l'URL réelle de votre frontend. N'utilisez jamais `*` avec `allowCredentials=true` !

6 Partie 5 : Connexion du Frontend Vue.js (1h)

Objectif

Dans cette partie, nous allons modifier le frontend Vue.js pour qu'il récupère les données depuis notre API Spring Boot au lieu du fichier statique.

6.1 Exercice 5.1 : Création du service API

▶ Rappel de cours

En Vue.js, il est recommandé d'encapsuler les appels API dans un service dédié. Cela permet :

- Une meilleure organisation du code
- La réutilisation des fonctions
- Une gestion centralisée des erreurs

Clonez le projet frontend et créez le fichier de service :

```
git clone https://github.com/ialame/vuejs-boutique-tp4
cd vuejs-boutique-tp4
npm install
```

Question 6

Créez un fichier `src/services/api.ts` qui contient les fonctions pour interagir avec l'API. Utilisez l'API `fetch` native de JavaScript.

```
// src/services/api.ts
import type { ProductInterface } from '@/interfaces/Product.interface';
import type { Extension } from '@/interfaces/type';
```

```typescript
const API_BASE_URL = 'http://localhost:8080/api';

// Mapping des extensions (frontend -> backend)
const extensionMapping: Record<Extension, string> = {
  'jungle': 'JUNGLE',
  'fossile': 'FOSSILE',
  'expedition': 'EXPEDITION',
  'aquapolis': 'AQUAPOLIS',
  'all': 'ALL'
};

const reverseExtensionMapping: Record<string, Extension> = {
  'JUNGLE': 'jungle',
  'FOSSILE': 'fossile',
  'EXPEDITION': 'expedition',
  'AQUAPOLIS': 'aquapolis',
  'ALL': 'all'
};

// Transformer un produit backend vers frontend
function transformProduct(backendProduct: any): ProductInterface {
  return {
    ...backendProduct,
    extension: reverseExtensionMapping[backendProduct.extension]
               || backendProduct.extension
  };
}

// Transformer un produit frontend vers backend
function transformToBackend(product: Partial<ProductInterface>): any {
  return {
    ...product,
    extension: product.extension
      ? extensionMapping[product.extension] || product.extension.toUpperCase()
      : undefined
  };
}

// GET - Récupérer tous les produits
export async function fetchProducts(): Promise<ProductInterface[]> {
  const response = await fetch(`${API_BASE_URL}/products`);
  if (!response.ok) {
    throw new Error('Erreur lors de la récupération des produits');
  }
  const products = await response.json();
  return products.map(transformProduct);
}

// GET - Récupérer un produit par ID
export async function fetchProductById(id: number): Promise<ProductInterface> {
  const response = await fetch(`${API_BASE_URL}/products/${id}`);
  if (!response.ok) {
    throw new Error(`Produit ${id} non trouvé`);
  }
  return transformProduct(await response.json());
}

// POST - Créer un nouveau produit
```

```
export async function createProduct(
  product: Omit<ProductInterface, 'id'>
): Promise<ProductInterface> {
  const response = await fetch(`${API_BASE_URL}/products`, {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify(transformToBackend(product)),
  });
  if (!response.ok) {
    throw new Error('Erreur lors de la création du produit');
  }
  return transformProduct(await response.json());
}

// PUT - Mettre à jour un produit
export async function updateProduct(
  id: number,
  product: Partial<ProductInterface>
): Promise<ProductInterface> {
  const response = await fetch(`${API_BASE_URL}/products/${id}`, {
    method: 'PUT',
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify(transformToBackend(product)),
  });
  if (!response.ok) {
    throw new Error(`Erreur lors de la mise à jour du produit ${id}`);
  }
  return transformProduct(await response.json());
}

// DELETE - Supprimer un produit
export async function deleteProduct(id: number): Promise<void> {
  const response = await fetch(`${API_BASE_URL}/products/${id}`, {
    method: 'DELETE',
  });
  if (!response.ok) {
    throw new Error(`Erreur lors de la suppression du produit ${id}`);
  }
}
```

6.2 Exercice 5.2 : Modification de App.vue

Question 7

Modifiez `App.vue` pour charger les produits depuis l'API au lieu du fichier statique. Utilisez le hook `onMounted` pour charger les données au démarrage.

Modifiez la section `<script setup>` de `App.vue` :

```
<script setup lang="ts">
import { reactive, onMounted, ref } from 'vue';
import Header from '@/components/Header.vue';
import Footer from '@/components/Footer.vue';
```

```
import Boutique from '@/features/boutique/Boutique.vue';
import Admin from '@/features/admin/Admin.vue';
import type { ProductInterface } from '@/interfaces';
import {
  fetchProducts,
  createProduct,
  deleteProduct as apiDeleteProduct
} from '@/services/api';

// État réactif
const state = reactive<{
  products: ProductInterface[];
  page: 'boutique' | 'admin';
}>({
  products: [],  // Initialement vide
  page: 'boutique'
});

// État de chargement et erreurs
const loading = ref(true);
const error = ref<string | null>(null);

// Charger les produits au montage
onMounted(async () => {
  try {
    loading.value = true;
    state.products = await fetchProducts();
  } catch (e) {
    error.value = e instanceof Error ? e.message : 'Erreur inconnue';
    console.error('Erreur:', e);
  } finally {
    loading.value = false;
  }
});

// Navigation
function navigate(page: 'boutique' | 'admin') {
  state.page = page;
}

// Ajouter un produit via l'API
async function addProduct(product: Omit<ProductInterface, 'id'>) {
  try {
    const newProduct = await createProduct(product);
    state.products.push(newProduct);
  } catch (e) {
    console.error('Erreur création:', e);
  }
}

// Supprimer un produit via l'API
async function deleteProductHandler(productId: number) {
  try {
    await apiDeleteProduct(productId);
    state.products = state.products.filter(p => p.id !== productId);
  } catch (e) {
    console.error('Erreur suppression:', e);
  }
```

```
}
</script>
```

6.3 Exercice 5.3 : Gestion des états de chargement

Ajoutez l'affichage conditionnel dans le template :

```
<template>
  <div class="music-shop">
    <Header @navigate="navigate" :page="state.page" />

    <!-- État de chargement -->
    <div v-if="loading" class="loading">
      <p>Chargement des produits...</p>
    </div>

    <!-- Erreur -->
    <div v-else-if="error" class="error">
      <p>{{ error }}</p>
      <button @click="retryLoad">Réessayer</button>
    </div>

    <!-- Contenu principal -->
    <template v-else>
      <Boutique
        v-if="state.page === 'boutique'"
        :products="state.products"
      />
      <Admin
        v-else
        @add-product="addProduct"
        @delete-product="deleteProductHandler"
        :products="state.products"
      />
    </template>

    <Footer />
  </div>
</template>
```

6.4 Exercice 5.4 : Test complet

1. Assurez-vous que le backend Spring Boot tourne sur le port 8080
2. Lancez le frontend : `npm run dev`
3. Ouvrez `http://localhost:5173`
4. Vérifiez que les produits s'affichent
5. Testez l'ajout d'un produit via l'interface Admin
6. Vérifiez dans la console H2 que le produit est bien en base

Question 8

Que se passe-t-il si vous arrêtez le serveur Spring Boot et rafraîchissez la page ? Comment améliorer l'expérience utilisateur dans ce cas ?

7 Partie 6 : Authentification (1h)

Objectif

Dans cette partie, nous allons ajouter un système d'authentification simple permettant aux utilisateurs de s'inscrire et de se connecter.

Attention

L'authentification présentée ici est **simplifiée** à des fins pédagogiques (mot de passe en clair). En production, utilisez Spring Security avec des mots de passe hashés (BCrypt) et des tokens JWT.

7.1 Exercice 6.1 : Entité User

Créez la classe `User` dans le package `model` :

```
// src/main/java/com/pokeshop/backend/model/User.java
package com.pokeshop.backend.model;

import jakarta.persistence.*;

@Entity
@Table(name = "users")
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(unique = true, nullable = false)
    private String username;

    @Column(nullable = false)
    private String password;

    private String role = "USER";

    // Constructeur par defaut (requis par JPA)
    public User() {}

    // Constructeur avec parametres
    public User(String username, String password) {
        this.username = username;
        this.password = password;
        this.role = "USER";
    }

    // Getters
    public Long getId() { return id; }
```

```
    public String getUsername() { return username; }
    public String getPassword() { return password; }
    public String getRole() { return role; }

    // Setters
    public void setId(Long id) { this.id = id; }
    public void setUsername(String username) { this.username = username; }
    public void setPassword(String password) { this.password = password; }
    public void setRole(String role) { this.role = role; }
}
```

7.2 Exercice 6.2 : UserRepository

Créez l'interface `UserRepository` :

```
// src/main/java/com/pokeshop/backend/repository/UserRepository.java
package com.pokeshop.backend.repository;

import com.pokeshop.backend.model.User;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import java.util.Optional;

@Repository
public interface UserRepository extends JpaRepository<User, Long> {

    Optional<User> findByUsername(String username);
}
```

7.3 Exercice 6.3 : AuthController

Créez le contrôleur d'authentification :

```
// src/main/java/com/pokeshop/backend/controller/AuthController.java
package com.pokeshop.backend.controller;

import com.pokeshop.backend.model.User;
import com.pokeshop.backend.repository.UserRepository;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import java.util.Map;
import java.util.HashMap;

@RestController
@RequestMapping("/api/auth")
public class AuthController {

    private final UserRepository userRepository;

    public AuthController(UserRepository userRepository) {
        this.userRepository = userRepository;
    }

    // POST /api/auth/login
    @PostMapping("/login")
    public ResponseEntity<?> login(@RequestBody Map<String, String> credentials) {
        String username = credentials.get("username");
```

```
        String password = credentials.get("password");

        return userRepository.findByUsername(username)
            .filter(user -> user.getPassword().equals(password))
            .map(user -> {
                Map<String, String> response = new HashMap<>();
                response.put("message", "Connexion reussie");
                response.put("username", user.getUsername());
                response.put("role", user.getRole());
                return ResponseEntity.ok(response);
            })
            .orElse(ResponseEntity.status(401)
                .body(Map.of("error", "Identifiants invalides")));
    }

    // POST /api/auth/register
    @PostMapping("/register")
    public ResponseEntity<?> register(@RequestBody Map<String, String> credentials) {
        String username = credentials.get("username");
        String password = credentials.get("password");

        if (userRepository.findByUsername(username).isPresent()) {
            return ResponseEntity.badRequest()
                .body(Map.of("error", "Nom d'utilisateur deja pris"));
        }

        User user = new User(username, password);
        User saved = userRepository.save(user);

        Map<String, String> response = new HashMap<>();
        response.put("message", "Inscription reussie");
        response.put("username", saved.getUsername());
        response.put("role", saved.getRole());

        return ResponseEntity.ok(response);
    }
}
```

7.4 Exercice 6.4 : Test de l'API d'authentification

Redémarrez le backend et testez :

```
# Inscription
curl -X POST http://localhost:8080/api/auth/register \
  -H "Content-Type: application/json" \
  -d '{"username":"alice","password":"secret123"}'

# Connexion
curl -X POST http://localhost:8080/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{"username":"alice","password":"secret123"}'
```

7.5 Exercice 6.5 : Composant Login.vue (Frontend)

Créez le composant de connexion/inscription :

```
<!-- src/components/Login.vue -->
```

```
<script setup lang="ts">
import { ref } from 'vue';

const props = defineProps<{
  mode: 'login' | 'register';
}>();

const emit = defineEmits<{
  (e: 'success', user: { username: string; role: string }): void;
  (e: 'close'): void;
}>();

const username = ref('');
const password = ref('');
const error = ref('');
const loading = ref(false);

async function handleSubmit() {
  loading.value = true;
  error.value = '';

  const endpoint = props.mode === 'login'
    ? 'http://localhost:8080/api/auth/login'
    : 'http://localhost:8080/api/auth/register';

  try {
    const response = await fetch(endpoint, {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({
        username: username.value,
        password: password.value
      })
    });

    const data = await response.json();

    if (response.ok) {
      emit('success', {
        username: data.username,
        role: data.role || 'USER'
      });
    } else {
      error.value = data.error || 'Erreur';
    }
  } catch (e) {
    error.value = 'Impossible de contacter le serveur';
  } finally {
    loading.value = false;
  }
}
</script>

<template>
  <div class="modal-overlay" @click.self="emit('close')">
    <div class="modal">
      <button class="modal__close" @click="emit('close')">
        &times;
```

```
      </button>
      <h2>{{ mode === 'login' ? 'Connexion' : 'Inscription' }}</h2>
      <form @submit.prevent="handleSubmit">
        <div class="form-group">
          <label>Nom d'utilisateur</label>
          <input v-model="username" type="text" required />
        </div>
        <div class="form-group">
          <label>Mot de passe</label>
          <input v-model="password" type="password" required />
        </div>
        <p v-if="error" class="error">{{ error }}</p>
        <button type="submit" :disabled="loading">
          {{ loading ? 'Chargement...' :
             (mode === 'login' ? 'Se connecter' : "S'inscrire") }}
        </button>
      </form>
    </div>
  </div>
</template>

<style scoped>
.modal-overlay {
  position: fixed;
  inset: 0;
  background: rgba(0,0,0,0.5);
  display: flex;
  justify-content: center;
  align-items: center;
  z-index: 1000;
}
.modal {
  background: white;
  padding: 2rem;
  border-radius: 8px;
  width: 100%;
  max-width: 400px;
  position: relative;
}
.modal__close {
  position: absolute;
  top: 10px;
  right: 15px;
  font-size: 1.5rem;
  background: none;
  border: none;
  cursor: pointer;
}
.form-group {
  margin-bottom: 1rem;
}
.form-group label {
  display: block;
  margin-bottom: 0.5rem;
}
.form-group input {
  width: 100%;
  padding: 0.75rem;
```

```
  border: 1px solid #ccc;
  border-radius: 4px;
}
.error {
  color: red;
  margin-bottom: 1rem;
}
button[type="submit"] {
  width: 100%;
  padding: 0.75rem;
  background: #4CAF50;
  color: white;
  border: none;
  border-radius: 4px;
  cursor: pointer;
}
</style>
```

7.6 Exercice 6.6 : Modification du Header.vue

Ajoutez les boutons de connexion/inscription dans le header :

```
<!-- Ajouter dans src/components/Header.vue -->
<script setup lang="ts">
defineProps<{
  page: 'boutique' | 'admin';
  isLoggedIn?: boolean;
  username?: string;
}>();

const emit = defineEmits<{
  (e: 'navigate', page: 'boutique' | 'admin'): void;
  (e: 'show-login'): void;
  (e: 'show-register'): void;
  (e: 'logout'): void;
}>();
</script>

<!-- Ajouter dans le template du Header -->
<div class="header__auth">
  <template v-if="isLoggedIn">
    <span>{{ username }}</span>
    <button @click="emit('logout')">Deconnexion</button>
  </template>
  <template v-else>
    <button @click="emit('show-login')">Connexion</button>
    <button @click="emit('show-register')">Inscription</button>
  </template>
</div>
```

7.7 Exercice 6.7 : Intégration dans App.vue

Modifiez `App.vue` pour gérer l'état d'authentification :

```
<script setup lang="ts">
import { reactive, onMounted, ref } from 'vue';
import Header from '@/components/Header.vue';
```

```
import Footer from '@/components/Footer.vue';
import Login from '@/components/Login.vue';
// ... autres imports

// Etat d'authentification
const isLoggedIn = ref(false);
const currentUser = ref<{ username: string; role: string } | null>(null);
const showAuthModal = ref(false);
const authMode = ref<'login' | 'register'>('login');

function showLogin() {
  authMode.value = 'login';
  showAuthModal.value = true;
}

function showRegister() {
  authMode.value = 'register';
  showAuthModal.value = true;
}

function handleAuthSuccess(user: { username: string; role: string }) {
  isLoggedIn.value = true;
  currentUser.value = user;
  showAuthModal.value = false;
}

function logout() {
  isLoggedIn.value = false;
  currentUser.value = null;
}
</script>

<template>
  <Header
    :page="state.page"
    :is-logged-in="isLoggedIn"
    :username="currentUser?.username"
    @navigate="navigate"
    @show-login="showLogin"
    @show-register="showRegister"
    @logout="logout"
  />

  <!-- Modal d'authentification -->
  <Login
    v-if="showAuthModal"
    :mode="authMode"
    @success="handleAuthSuccess"
    @close="showAuthModal = false"
  />

  <!-- ... reste du template -->
</template>
```

Question 9

Comment pourriez-vous protéger la page Admin pour qu'elle ne soit accessible qu'aux utilisateurs connectés ? Implémentez cette fonctionnalité.

8 Conclusion et Pour aller plus loin

8.1 Récapitulatif

Dans ce TP, vous avez appris à :

1. Créer un projet Spring Boot avec les dépendances nécessaires
2. Configurer une base de données H2 embarquée
3. Définir des entités JPA et des repositories
4. Créer une API REST complète (CRUD)
5. Gérer la problématique CORS
6. Connecter le frontend Vue.js au backend
7. Implémenter un système d'authentification (inscription/connexion)

8.2 Pour aller plus loin

- **Spring Security** : Authentification robuste avec JWT
- **Validation** : Annotations `@Valid` pour valider les entrées
- **Tests** : Tests unitaires avec JUnit et Mockito
- **Documentation API** : Swagger/OpenAPI
- **Base de données** : Migration vers PostgreSQL ou MySQL
- **Déploiement** : Docker, Heroku, ou cloud

8.3 Ressources

- Documentation Spring Boot : `https://spring.io/projects/spring-boot`
- Documentation Vue.js : `https://vuejs.org/`
- H2 Database : `https://www.h2database.com/`
- Spring Data JPA : `https://spring.io/projects/spring-data-jpa`