**FullStack**

**TP1**

Réalisé par

Yaroslav Sakal

**1. Introduction**

Ce projet vise à développer un système permettant la gestion des catégories en utilisant une architecture Full Stack. Le système repose sur un client React pour l’interface utilisateur, un serveur Spring Boot pour la logique métier, et une base de données PostgreSQL. Une configuration Docker est utilisée pour simplifier le déploiement et la gestion des conteneurs.

**2. Exigences Fonctionnelles et Techniques**

**2.1 Gestion des Catégories**

Les fonctionnalités principales incluent :

* **Création de catégories** : Nom obligatoire, avec possibilité d’associer une catégorie parent.
* **Modification de catégories** : Mise à jour des informations de la catégorie.
* **Suppression de catégories** : Une catégorie peut être supprimée si elle n'a pas de dépendances non résolues.
* **Recherche et tri** : Recherche paginée avec filtres (par nom, date de création, catégories racines, etc.).
* **Règles métier** :
  + Une catégorie ne peut pas être enfant d’elle-même.
  + Une catégorie enfant ne peut avoir qu’un seul parent.

**2.2 Architecture**

* **Architecture REST** : Communication entre client et serveur en JSON.
* **SPA** : Le client est une application monopage (Single Page Application).
* **Évolutivité** : Les APIs sont stateless et adaptées à la mise à l’échelle.
* **Frameworks** : React (client) et Spring Boot (serveur).

**2.3 Configuration Technique**

* **Temps de réponse** : Moins de 5 secondes par requête.
* **Port d'écoute** :
  + Client : Port 4200.
  + Serveur : Port 8080.

**2.4 Gestion de Projet**

* **Gestion du code source** : Utilisation de Git avec Gitflow.
* **Déploiement CI/CD** : Optionnel, mais encouragé pour automatiser les tâches répétitives.
* **Conteneurisation Docker** : Simplifie le déploiement des trois services (client, serveur, base de données).

**3. Implémentation**

**3.1 Fichiers Docker**

**docker-compose.yml**

Définit les services suivants :

* **frontend** : Serve l’application React avec Nginx.
* **backend** : Exécute le serveur Spring Boot.
* **db** : Base de données PostgreSQL avec volume persistant.

**Dockerfile.frontend**

Contient deux étapes :

1. **Build** : Compilation du client React.
2. **Serve** : Déploiement avec Nginx.

**Dockerfile.backend**

Contient deux étapes :

1. **Build** : Compilation du projet Spring Boot avec Maven.
2. **Serve** : Exécution du fichier JAR généré.

**3.2 Données de Test**

* Initialisation de la base de données PostgreSQL avec des données de test pour valider les fonctionnalités.

**3.3 API REST**

Les API fournissent toutes les fonctionnalités CRUD nécessaires pour gérer les catégories. Les endpoints suivent des conventions REST (GET, POST, PUT, DELETE).

**4. Déploiement**

**4.1 Ports**

* Client : Accessible sur http://localhost:4200.
* Serveur : Accessible sur http://localhost:8080.

**4.2 Commandes**

* Construction et démarrage : docker-compose up --build.
* Arrêt des conteneurs : docker-compose down.

**Lancement du Docker**

**Exigences fonctionnelles**

* **CRUD des catégories** : Créer, lire, modifier, et supprimer des catégories.
* **Recherche et tri** : Rechercher des catégories avec filtres et critères.
* **Hiérarchie** : Gestion des catégories parent/enfant.
* **Pagination** : Afficher les résultats sous forme paginée.

**Exigences techniques**

* **Architecture client/serveur** :
  + Frontend : React (SPA).
  + Backend : Spring Boot avec API REST.
  + Base de données : PostgreSQL.
* **Docker** :
  + Les conteneurs doivent être définis dans docker-compose.yml.
  + Ports : 4200 (client), 8080 (serveur), 5434 (base de données).

**Étapes pour démarrer**

1. **Prérequis** :
   * Docker et Docker Compose installés.
   * Maven pour la compilation manuelle du backend (optionnel).
2. **Commandes** :
   * Lancer tous les services : docker-compose up --build.
   * Accéder au frontend : http://localhost:4200.
   * Accéder à l'API backend : [http://localhost:8080](http://localhost:8080" \t "_new).

**Fichiers fournis**

1. docker-compose.yml : Configuration des conteneurs.

version: "3.8"

services:

  frontend:

    build:

      context: ./shop-client  # Adjusted context to shop-client

      dockerfile: Dockerfile.frontend

    ports:

      - "4200:80"  # Map port 4200 on your host to port 80 in the container

    volumes:

      - ./shop-client:/app

    environment:

      - NODE\_ENV=production

  backend:

    build:

      context: ./shop-server  # Adjusted context to shop-server

      dockerfile: Dockerfile.backend

    ports:

      - "8080:8080"

    environment:

      - SPRING\_PROFILES\_ACTIVE=prod

      - SPRING\_DATASOURCE\_URL=jdbc:postgresql://db:5432/shopdb

      - SPRING\_DATASOURCE\_USERNAME=postgres

      - SPRING\_DATASOURCE\_PASSWORD=password

    volumes:

      - ./shop-server:/app

    depends\_on:

      - db

  db:

    image: postgres:15

    ports:

      - "5434:5432"

    environment:

      - POSTGRES\_DB=shopdb

      - POSTGRES\_USER=postgres

      - POSTGRES\_PASSWORD=password

    volumes:

      - postgres\_data:/var/lib/postgresql/data

volumes:

  postgres\_data:

    driver: local

1. Dockerfile.frontend : Build et serveur pour le frontend.

# Stage 1: Build the React app

FROM node:16 AS client-build

WORKDIR /app

# Copy package.json and install dependencies

COPY package\*.json ./

RUN npm install

# Copy the rest of the app source

COPY ./ ./

# Build the React app

RUN npm run build

# Stage 2: Serve the React app with Nginx

FROM nginx:1.23 AS client-serve

WORKDIR /usr/share/nginx/html

# Copy the React build files from the previous stage

COPY --from=client-build /app/build/ ./

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]

1. Dockerfile.backend : Build et exécution du backend.

# Stage 1: Build the Spring Boot app

FROM maven:3.8.7-openjdk-17-slim AS build

WORKDIR /app

# Copy the pom.xml and install dependencies

COPY pom.xml ./

RUN mvn dependency:resolve

# Copy the source code

COPY src ./src

# Build the Spring Boot app

RUN mvn package -DskipTests

# Stage 2: Create a minimal runtime environment for the Spring Boot app

FROM openjdk:17-jdk-slim AS backend-serve

WORKDIR /app

# Copy the JAR file from the build stage

COPY --from=build /app/target/\*.jar app.jar

EXPOSE 8080

CMD ["java", "-jar", "app.jar"]

A computer screen shot of text

Description automatically generated

**5 Traitement du Git Repository**

**5.1 Initialisation du Git Repository**

A screenshot of a computer

Description automatically generated

* 1. **Ajout des fichiers au dépôt** :

A screen shot of a computer

Description automatically generated

**6. Inculstion des imports suivants, création de la base de données** **:**A screen shot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

**7. Réalisation des exigeances :**

***E\_CAT\_10 Le système permet aux utilisateurs de créer une nouvelle catégorie.***

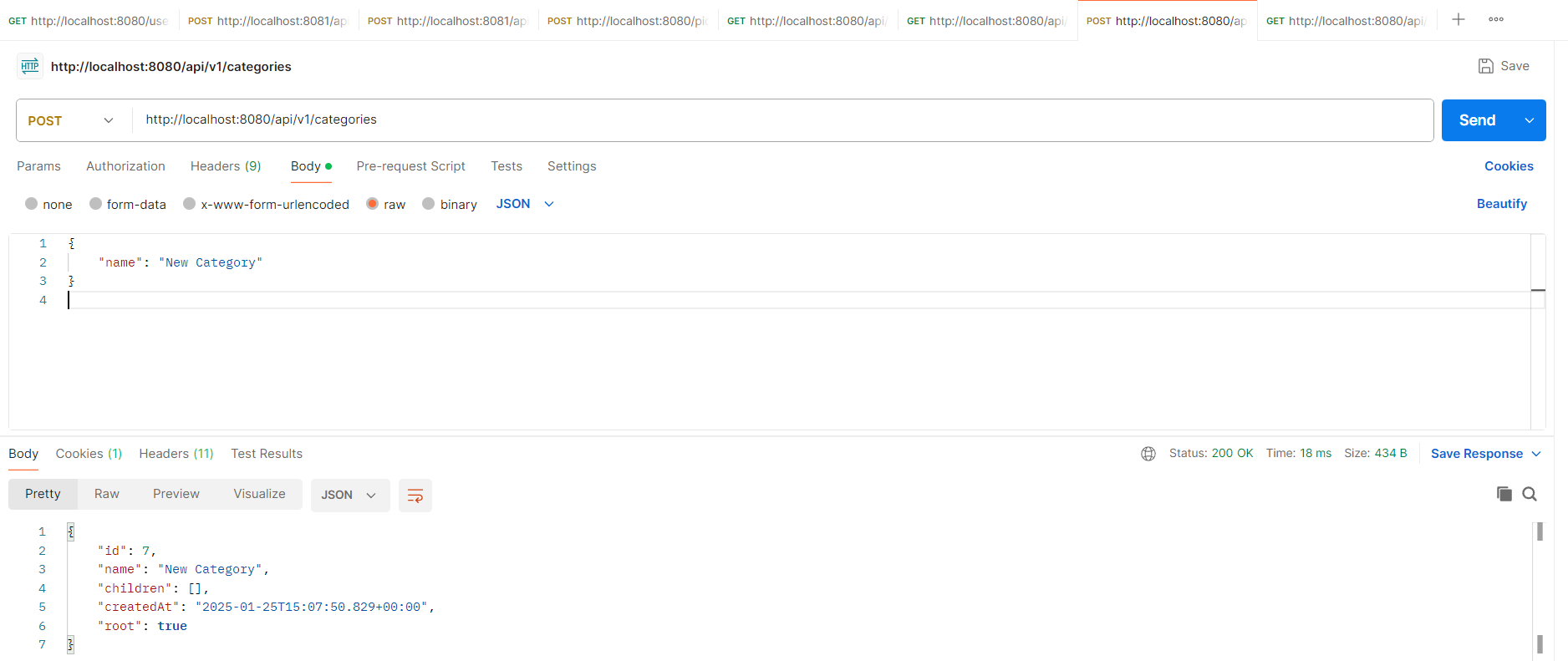
@PostMapping

public ResponseEntity<Category> createCategory(@Valid @RequestBody Category category) {

    return ResponseEntity.ok(service.createCategory(category));

}

Ce point correspond à la création d’une nouvelle catégorie. Lorsqu'un utilisateur soumet une nouvelle catégorie via une requête POST, les informations minimales demandées sont le nom de la catégorie. La méthode createCategory dans le service valide et enregistre la catégorie dans la base de données.



***E\_CAT\_20: Le système permet aux utilisateurs d’associer une catégorie parent à une catégorie enfant***

@JsonBackReference // Prévenir la récursion infinie pour le parent

@ManyToOne

@JoinColumn(name = "parent\_id")

private Category parent;

Cette partie du code permet d'associer une catégorie à une catégorie parent. Dans le modèle Category, le champ parent définit une relation ManyToOne, indiquant que chaque catégorie peut avoir un parent. Le décorateur @JsonBackReference est utilisé pour éviter une récursion infinie lors de la sérialisation JSON.

A screenshot of a computer

Description automatically generated

***E\_CAT\_30: Le système permet aux utilisateurs de sélectionner une catégorie existante et de modifier les champs renseignés lors de la création de la catégorie***

@PutMapping

public ResponseEntity<Category> updateCategory(@Valid @RequestBody Category category) {

    return ResponseEntity.ok(service.updateCategory(category));

}

Ce point permet à un utilisateur de modifier une catégorie existante. La méthode updateCategory permet de mettre à jour les informations d'une catégorie en utilisant une requête PUT. Elle vérifie que la catégorie existe et que ses informations sont valides avant de procéder à la mise à jour dans la base de données.

A screenshot of a computer

Description automatically generated

***E\_CAT\_40 Le système permet aux utilisateurs de supprimer une catégorie***

@PutMapping

public ResponseEntity<Category> updateCategory(@Valid @RequestBody Category category) {

    return ResponseEntity.ok(service.updateCategory(category));

}

Ce point permet de supprimer une catégorie. La méthode deleteCategoryById dans le service supprime la catégorie de la base de données via un appel à la méthode deleteById. Elle utilise une requête DELETE avec un identifiant de catégorie passé dans l'URL.

A screenshot of a computer

Description automatically generated

***E\_CAT\_50 Le système permet d’effectuer une recherche paginée sur toutes les catégories qui sont présentes en affichant :***

***● Leur nom***

***● Leur date de création***

***● Leur catégorie enfant***

***● Si elles sont une catégorie racine 1***

@GetMapping

public ResponseEntity<Page<Category>> getCategories(Pageable pageable) {

    return ResponseEntity.ok(service.getCategoriesByFilter(isRoot, afterDate, beforeDate, pageable));

}

Ce point permet une recherche paginée sur toutes les catégories. Il renvoie une liste de catégories avec des informations telles que le nom, la date de création, les catégories enfants, et s'il s'agit d'une catégorie racine. La pagination est gérée grâce au paramètre Pageable dans la méthode getCategoriesByFilter. A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

***E\_CAT\_60 Le système permet d’effectuer une recherche sur les catégories avec un ou une combinaison de filtres qui sont :***

***● Si elles sont une catégorie racine***

***● La date de création***

***○ Après une date précise***

***○ Avant une date précise***

***○ Entre deux dates précises***

public Page<Category> getCategoriesByFilter(Boolean isRoot, Date afterDate, Date beforeDate, Pageable pageable) {

    if (isRoot != null) {

        return isRoot ? repository.findByParentIsNull(pageable) : repository.findByParentIsNotNull(pageable);

    } else if (afterDate != null && beforeDate != null) {

        return repository.findByCreatedAtBetween(afterDate, beforeDate, pageable);

    } else if (afterDate != null) {

        return repository.findByCreatedAtAfter(afterDate, pageable);

    } else if (beforeDate != null) {

        return repository.findByCreatedAtBefore(beforeDate, pageable);

    }

    return repository.findAll(pageable);

}

A screenshot of a computer

Description automatically generated

Ce point permet de filtrer les catégories par plusieurs critères : si elles sont racines, ou en fonction des dates de création (après, avant ou entre deux dates). La méthode getCategoriesByFilter applique ces filtres en fonction des paramètres fournis par l'utilisateur.

***E\_CAT\_70 Le système permet d’effectuer une recherche sur les catégories en triant par :***

***● Le nom***

***● La date de création***

***● Le nombre de catégorie enfant***

@GetMapping("/sorted-by-children")

public ResponseEntity<Page<Category>> getCategoriesSortedByChildCount(Pageable pageable) {

    return ResponseEntity.ok(service.getCategoriesSortedByChildCount(pageable));

}

Ce point permet de trier les catégories par plusieurs critères : nom, date de création ou nombre de catégories enfants. La méthode getCategoriesSortedByChildCount récupère les catégories triées en fonction du nombre d'enfants via une requête spécifique dans le repository.

A screenshot of a computer

Description automatically generated

***E\_CAT\_80 Le système permet d’afficher le détail d’une catégorie en affichant les champs et les entités tels que définis par E\_CAT\_10, E\_CAT\_20 et E\_CAT\_50.***

Ce point permet de récupérer les détails d’une catégorie spécifique, y compris ses champs et ses entités associées. La méthode getCategoryById récupère la catégorie via son identifiant.

***E\_CAT\_90 Le système ne permet pas à une catégorie d’être enfant d’elle-même.***

@GetMapping("/{id}")

public ResponseEntity<Category> getCategoryById(@PathVariable Long id) {

    return ResponseEntity.ok(service.getCategoryById(id));

}

Ce point empêche une catégorie d’être enfant d’elle-même. Lors de la création ou de la mise à jour d’une catégorie, un contrôle est effectué pour vérifier que l'identifiant de la catégorie parente ne correspond pas à l'identifiant de la catégorie elle-même

***E\_CAT\_100 Le système ne permet pas à une catégorie enfant d’avoir deux parents***

Ce contrôle est géré implicitement par la relation ManyToOne entre une catégorie et son parent, où chaque catégorie ne peut avoir qu’un seul parent.

Le système empêche une catégorie enfant d’avoir plusieurs parents, en garantissant que la relation entre la catégorie et son parent est bien définie comme une relation ManyToOne. Une catégorie peut avoir plusieurs enfants, mais un seul parent.

**Execution :**

A screenshot of a computer program

Description automatically generated

**Partie client :**

**Objectifs**

* Créer une interface utilisateur simple et efficace pour gérer les catégories (CRUD).
* Implémenter les fonctionnalités demandées : création, modification, suppression, recherche, tri, et affichage des détails des catégories.
* Garantir une expérience utilisateur fluide via une Single Page Application (SPA) développée avec React.

**Technologies Utilisées**

* **Framework Frontend** : React.js
* **Gestion d'état** : React Context API
* **Requêtes API** : Axios
* **Styles** : CSS Modules
* **Test** : Utilisation des données de test (mock API)

**Fonctionnalités Implémentées**

1. **Création de catégories** : Formulaire permettant d'ajouter une nouvelle catégorie avec validation des champs requis.
2. **Modification de catégories** : Option d'édition des champs existants.
3. **Suppression de catégories** : Suppression d'une catégorie via une action utilisateur.
4. **Recherche et tri** : Recherche de catégories avec des filtres et tri selon différents critères (nom, date de création, etc.).
5. **Détails des catégories** : Affichage des informations détaillées, y compris la catégorie parent/enfant.

**Résumé des Contraintes**

* Toutes les interactions avec l'API doivent être effectuées en JSON.
* Les performances doivent garantir des réponses en moins de 5 secondes pour toutes les requêtes.
* L'interface client fonctionne sur le port 4200.

**CategoryContext.js** : Gestion de l'état des catégories

* Ce fichier contient le **contexte** de l'application pour gérer les catégories (ajout, modification, suppression).
* Il crée et exporte un **Contexte** qui est utilisé dans tous les composants pour accéder et manipuler les catégories.

import React, { createContext, useState } from 'react';

// Création du contexte

export const CategoryContext = createContext();

// Provider qui enveloppe l'application et fournit les données aux composants

export const CategoryProvider = ({ children }) => {

    const [categories, setCategories] = useState([]); // État initial des catégories

    // Fonction pour ajouter une catégorie

    const addCategory = (category) => {

        setCategories([...categories, category]);

    };

    // Fonction pour mettre à jour une catégorie

    const updateCategory = (id, updatedCategory) => {

        setCategories(categories.map(cat => cat.id === id ? updatedCategory : cat));

    };

    // Fonction pour supprimer une catégorie

    const deleteCategory = (id) => {

        setCategories(categories.filter(cat => cat.id !== id));

    };

    return (

        <CategoryContext.Provider value={{ categories, addCategory, updateCategory, deleteCategory }}>

            {children}

        </CategoryContext.Provider>

    );

};

**CategoryForm.jsx** : Formulaire pour ajouter une nouvelle catégorie

* Ce composant permet à l'utilisateur de saisir les informations pour créer une nouvelle catégorie.
* Lorsqu'un utilisateur soumet le formulaire, la fonction addCategory est appelée pour ajouter la catégorie à l'état.
* import React, { useState, useContext } from 'react';
* import { CategoryContext } from '../context/CategoryContext';
* const CategoryForm = () => {
* const { addCategory } = useContext(CategoryContext); // Accès à la fonction de gestion des catégories
* const [name, setName] = useState('');
* const [parent, setParent] = useState('');
* // Gestion de l'envoi du formulaire
* const handleSubmit = (e) => {
* e.preventDefault();
* addCategory({ id: Date.now(), name, parent, createdAt: new Date().toISOString() });
* setName('');
* setParent('');
* };
* return (
* <form onSubmit={handleSubmit}>
* <input
* type="text"
* placeholder="Nom de la catégorie"
* value={name}
* onChange={(e) => setName(e.target.value)}
* required
* />
* <input
* type="text"
* placeholder="Catégorie parent"
* value={parent}
* onChange={(e) => setParent(e.target.value)}
* />
* <button type="submit">Ajouter</button>
* </form>
* );
* };
* export default CategoryForm;

**CategoryList.jsx** : Liste des catégories avec option de suppression

* Ce composant affiche une liste des catégories.
* Chaque catégorie a un bouton pour être supprimée via la fonction deleteCategory.

import React, { useContext } from 'react';

import { CategoryContext } from '../context/CategoryContext';

const CategoryList = () => {

    const { categories, deleteCategory } = useContext(CategoryContext);

    return (

        <ul>

            {categories.map((category) => (

                <li key={category.id}>

                    {category.name} - {category.createdAt}

                    <button onClick={() => deleteCategory(category.id)}>Supprimer</button>

                </li>

            ))}

        </ul>

    );

};

export default CategoryList;

**CategoryDetails.jsx** : Détails d'une catégorie sélectionnée

* Ce composant affiche les détails d'une catégorie spécifique (nom et parent).
* Il pourrait également inclure une fonctionnalité pour modifier la catégorie.

import React, { useContext } from 'react';

import { CategoryContext } from '../context/CategoryContext';

const CategoryDetails = ({ id }) => {

    const { categories } = useContext(CategoryContext);

    const category = categories.find(cat => cat.id === id);

    if (!category) return <div>Catégorie non trouvée</div>;

    return (

        <div>

            <h3>Détails de la catégorie</h3>

            <p><strong>Nom : </strong>{category.name}</p>

            <p><strong>Catégorie parent : </strong>{category.parent}</p>

            <p><strong>Date de création : </strong>{category.createdAt}</p>

        </div>

    );

};

export default CategoryDetails;

**App.jsx** : Composant principal

* Ce fichier organise l'application en affichant le formulaire d'ajout de catégorie, la liste des catégories, et éventuellement les détails d'une catégorie sélectionnée.
* Il utilise le **CategoryProvider** pour envelopper l'application et fournir l'état des catégories à tous les composants enfants.

import React, { useState } from 'react';

import { CategoryProvider } from './context/CategoryContext';

import CategoryForm from './components/CategoryForm';

import CategoryList from './components/CategoryList';

import CategoryDetails from './components/CategoryDetails';

const App = () => {

    const [selectedCategoryId, setSelectedCategoryId] = useState(null);

    return (

        <CategoryProvider>

            <h1>Gestion des Catégories</h1>

            <CategoryForm />

            <CategoryList />

            {selectedCategoryId && <CategoryDetails id={selectedCategoryId} />}

        </CategoryProvider>

    );

};

export default App;