Compte Rendu : TP Applications Réparties - Todolist API

Mohamed Aziz Hadadi & Malek tababi

Introduction

Ce TP porte sur le développement d'une application répartie de gestion de tâches (Todo List) en utilisant Java et Spring Boot. L'application adopte une architecture client-serveur où le backend est développé avec Spring Boot et expose une API REST, tandis que le frontend est une application web utilisant HTML, CSS et JavaScript.

Architecture Globale

L'application suit le modèle MVC (Modèle-Vue-Contrôleur) et repose sur l'architecture REST pour les communications entre le frontend et le backend :

- **Backend** : API REST développée avec Spring Boot
- **Persistance** : Base de données H2 en mémoire avec JPA
- **Frontend** : Interface web avec HTML, Bootstrap et JavaScript vanilla

Technologies Utilisées

Backend

- **Java 17** : Langage de programmation
- **Spring Boot 3.4.5** : Framework pour le développement d'applications
- **Spring Data JPA** : Pour la gestion de la persistance
- **H2 Database** : Base de données en mémoire pour le stockage des données
- **Maven** : Outil de gestion de dépendances et de build

Frontend

- HTML5
- Bootstrap 5.3.0
- JavaScript (ES6)

Structure du Backend

Modèle de Données

Le modèle principal de l'application est la classe `Todo` :

```
@Entity
public class Todo {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String title;
    private String description;
    private boolean completed = false;

    // Constructeurs, getters et setters
}
```

Cette classe représente une tâche avec un identifiant unique, un titre, une description et un statut d'achèvement.

Couche Repository

L'interface `TodoRepository` étend `JpaRepository` pour bénéficier des opérations CRUD standards :

```
public interface TodoRepository extends JpaRepository<Todo, Long> {}
```

Cette approche utilise le pattern Repository de Spring Data JPA, ce qui permet d'obtenir les opérations CRUD de base sans écrire d'implémentation.

Contrôleur REST

Le contrôleur `TodoController` expose les endpoints REST pour manipuler les ressources Todo :

```java
@RestController
@RequestMapping("/api/todos")
public class TodoController {
    @Autowired
    private TodoRepository repository;

    // Méthodes du contrôleur...
}
```

Points d'API et Fonctionnalités

Récupération des tâches

1. **Obtenir toutes les tâches** :

```java
@GetMapping
public List<Todo> getAll() {
    return repository.findAll();
}
```

2. **Obtenir une tâche par ID** :

```java
@GetMapping("/{id}")
public ResponseEntity<Todo> getById(@PathVariable Long id) {
    return repository.findById(id)
            .map(ResponseEntity::ok)
            .orElse(ResponseEntity.notFound().build());
}
```

3. **Recherche de tâches par mot-clé** :

```java
@GetMapping("/search")
public List<Todo> search(@RequestParam String query) {
    return repository.findAll().stream()
            .filter(todo ->
todo.getTitle().toLowerCase().contains(query.toLowerCase()) ||

 todo.getDescription().toLowerCase().contains(query.toLowerCase()))
            .collect(Collectors.toList());
}
```

4. **Filtrer par statut d'achèvement** :

```java
@GetMapping("/status")
public List<Todo> filterByStatus(@RequestParam boolean completed) {
    return repository.findAll().stream()
            .filter(todo -> todo.isCompleted() == completed)
            .collect(Collectors.toList());
}
```

Création, Modification et Suppression

5. **Créer une nouvelle tâche** :

```
@PostMapping
public ResponseEntity<Todo> create(@RequestBody Todo todo) {
    if (todo.getTitle() == null || todo.getDescription() == null) {
        return ResponseEntity.badRequest().build();
    }
    return ResponseEntity.ok(repository.save(todo));
}
```

6. **Mettre à jour une tâche existante** :

```
@PutMapping("/{id}")
public ResponseEntity<Todo> update(@PathVariable Long id, @RequestBody Todo
updated) {
    return repository.findById(id).map(todo -> {
        todo.setTitle(updated.getTitle());
        todo.setDescription(updated.getDescription());
        return ResponseEntity.ok(repository.save(todo));
    }).orElse(ResponseEntity.notFound().build());
}
```

7. **Basculer l'état d'achèvement d'une tâche** :

```
@PatchMapping("/{id}/toggle")
public ResponseEntity<Todo> toggleCompletion(@PathVariable Long id) {
    return repository.findById(id).map(todo -> {
        todo.setCompleted(!todo.isCompleted());
        return ResponseEntity.ok(repository.save(todo));
    }).orElse(ResponseEntity.notFound().build());
}
```

8. **Supprimer une tâche** :

```
@DeleteMapping("/{id}")
public ResponseEntity<Void> delete(@PathVariable Long id) {
    if (!repository.existsById(id)) return ResponseEntity.notFound().build();
    repository.deleteById(id);
    return ResponseEntity.noContent().build();
}
```

Analyse Technique

Points Forts du Backend

1. **Utilisation des annotations Spring** : Le code utilise efficacement les annotations de Spring Boot et Spring Web pour définir clairement les rôles des classes et les mappings REST.
2. **Gestion des erreurs** : Utilisation appropriée des codes de statut HTTP (200, 400, 404) via `ResponseEntity` pour les différentes réponses possibles.

3. **Programmation fonctionnelle** : Utilisation des streams Java pour le filtrage et la transformation des données.
4. **Structure modulaire** : Séparation claire des responsabilités entre les couches modèle, repository et contrôleur.

Conclusion

Ce TP d'applications réparties en Java présente une implémentation fonctionnelle d'une API REST pour une application de gestion de tâches. L'application démontre efficacement l'utilisation de Spring Boot pour développer un service web RESTful avec une persistance des données via JPA.

L'architecture adoptée est simple mais puissante, suivant les bonnes pratiques de développement d'applications web modernes. La séparation des préoccupations est bien respectée, avec des couches distinctes pour le modèle de données, l'accès aux données et l'exposition de l'API.

Le frontend, bien que non détaillé dans ce rapport, s'intègre parfaitement avec le backend grâce aux endpoints REST clairement définis et à une structure de réponse cohérente.