

Documentation Technique

Application To-Do

Project Spring-boot & angular &
postgres

Realiser Par Yassine Ouali N°25

Table des Matières

1. Aperçu de l'Architecture du Système
2. Analyse Détailée du Backend (Spring Boot)
 - La Couche Contrôleur (TodoController.java)
 - La Couche Service (TodoService.java)
 - Gestion Globale des Erreurs
(GlobalExceptionHandler.java)
3. Analyse Détailée du Frontend (Angular)
 - La Couche Service (todo.service.ts)
 - La Logique du Composant (todo-list.ts)
 - Le Modèle de Vue (todo-list.html)
4. Cycle de Vie d'une Requête de Bout en Bout
5. Conclusion
6. Annexes

Aperçu de l'Architecture du Système

L'application adopte une **architecture N-Tiers standard**, spécialement conçue pour découpler l'interface utilisateur de la gestion des données. Cette approche favorise la modularité, la scalabilité et la maintenabilité, en alignement avec les principes de conception logicielle modernes tels que le **principe de séparation des préoccupations (Separation of Concerns)**.

Description des Tiers

Tier	Technologie	Responsabilités Principales	Communication
Tier 1 : Client (Frontend)	Angular (Composants Standalone)	Rendu de l'UI, capture des entrées utilisateur, gestion de l'état local	HTTP (JSON) avec le serveur
Tier 2 : Serveur d'Application (Backend)	Spring Boot	Logique métier, validation des données, exposition des endpoints API	Accès à la base de données via JPA
Tier 3 : Couche de Données	Base de Données SQL (via Spring Data JPA)	Stockage persistant des tâches	Requêtes SQL générées par Hibernate

Avantages Clés :

- Indépendance** : Chaque tier peut être développé, testé et déployé séparément.
- Sécurité** : Le backend gère l'authentification et l'autorisation, protégeant les données sensibles.
- Performance** : Utilisation de caches (ex. : Redis pour l'état local) et de requêtes asynchrones pour une réactivité optimale.

Analyse Détailée du Backend (Spring Boot)

Le backend est structuré selon le **patron Contrôleur-Service-Repository**, garantissant l'adhésion au **Principe de Responsabilité Unique (Single Responsibility Principle - SRP)**. Ce découplage réduit la complexité et facilite les tests unitaires. Spring Boot 3 intègre nativement Jakarta EE, améliorant la portabilité et la compatibilité avec les conteneurs cloud comme Kubernetes.

2.1 La Couche Contrôleur (TodoController.java)

Cette couche sert d'**interface API RESTful**, gérant le flux de requêtes entrantes depuis le web. Elle agit comme un point d'entrée sécurisé et standardisé.

Conception RESTful

Verbe HTTP	Action	Endpoint	Description
GET	Lecture	/api/todos	Récupère toutes les tâches (findAll)
POST	Création	/api/todos	Crée une nouvelle tâche (create)
PUT	Mise à Jour	/api/todos/{id}	Met à jour une tâche existante (update)
DELETE	Suppression	/api/todos/{id}	Supprime une tâche (delete)

Configuration CORS (@CrossOrigin)

- Problème :** Les navigateurs bloquent les requêtes cross-origin (ex. : de localhost:4200 vers localhost:8080) pour des raisons de sécurité (politique Same-Origin Policy).
- Solution :** L'annotation `@CrossOrigin(origins = "http://localhost:4200")` whitelist l'application Angular, autorisant les accès. En production, cela est configuré via une propriété `application.yml` pour plus de flexibilité.
- Amélioration :** Ajout d'en-têtes personnalisés pour tracer les requêtes (ex. : `X-Request-ID`).

Conversion de Données

- Entrée :** `@RequestBody` désérialise automatiquement le JSON entrant (ex. : `{"title": "Travail", "completed": false}`) en objets Java via Jackson.
- Sortie :** `@ResponseBody` (implicite via `@RestController`) sérialise les objets Java en JSON. Gestion des erreurs de sérialisation via des validateurs Bean Validation (`@Valid`).

2.2 La Couche Service ([TodoService.java](#))

Cette couche encapsule la **logique métier**, appliquant les règles spécifiques à l'application (ex. : validation de la longueur du titre de tâche).

Injection de Dépendances (IoC)

- Utilisation de **Lombok @RequiredArgsConstructor** pour générer un constructeur injectant les dépendances (ex. : TodoRepository).
- Spring Boot scanne les annotations **@Service** et injecte automatiquement via le conteneur IoC, favorisant les tests mockés (ex. : avec Mockito).
- **Exemple de Code (Snippet) :**

```
@Service  
@RequiredArgsConstructor  
public class TodoService {  
    private final TodoRepository todoRepository;  
    public Optional<Todo> findById(Long id) {  
        return todoRepository.findById(id);  
    }  
}
```

Codage Défensif

- **Gestion des Optionals** : findById retourne **Optional<Todo>**. Utilisation de **.orElseThrow(() -> new RuntimeException("Tâche non trouvée"))** pour éviter les **NullPointerException** et propager des erreurs explicites.
- **Transactions** : Annotations **@Transactional** assurent l'atomicité des opérations (ex. : mise à jour en base sans corruption partielle).

Logique de Mise à Jour

Dans update, todo.setId(id) force JPA à exécuter un UPDATE SQL au lieu d'un INSERT. Validation supplémentaire : vérification que l'ID existe avant modification pour éviter les fuites de données.

2.3 Gestion Globale des Erreurs ([GlobalExceptionHandler.java](#))

Cette classe applique le **patron Aspect-Oriented Programming (AOP)** pour centraliser la gestion des exceptions, évitant la duplication de code.

- **Logique Centralisée** : @ControllerAdvice intercepte les exceptions globalement (ex. : RuntimeException, DataIntegrityViolationException).
- **Conformité RFC 7807** : Retourne des objets ProblemDetail en JSON standardisé :

```
{  
    "type": "https://example.com/problems/not-found",  
    "title": "Tâche non trouvée",  
    "status": 404,  
    "detail": "Aucune tâche avec l'ID 5 n'existe."  
}
```

- **Avantages** : Facilite l'affichage d'erreurs user-friendly sur le frontend et l'intégration avec des outils de monitoring comme Sentry.

Analyse Détailée du Frontend (Angular)

Le frontend utilise des **Composants Standalone** d'Angular 17+, éliminant les NgModules pour une structure plus légère et modulaire. Cela réduit le boilerplate et accélère le développement.

3.1 La Couche Service (todo.service.ts)

Ce service agit comme **passerelle d'accès aux données**, abstraillant les appels HTTP.

Patron Singleton

@Injectable({ providedIn: 'root' }) garantit une instance unique, partageant l'état global (ex. : cache des tâches).

Programmation Réactive (RxJS)

Les méthodes retournent Observable<Todo>, permettant des flux événementiels : annulation, retry, ou transformation (ex. : pipe(map(todo => todo.title.toUpperCase()))).

Exemple :

```
deleteTodo(id: number): Observable<void> {
    return this.http.delete<void>(`${this.apiUrl}/${id}`).pipe(
        catchError(this.handleError('deleteTodo'))
    );
}
```

Sécurité des Types

Interface stricte Todo (dans models/) : interface Todo { id: number; title: string; completed: boolean; }. Évite les erreurs runtime via TypeScript.

3.2 La Logique du Composant (todo-list.ts)

Ce composant gère la **contrôle de vue**, synchronisant l'état UI avec les données.

Gestion du Cycle de Vie

Implémente OnInit : ngOnInit() déclenche loadTodos() une fois le DOM prêt, évitant les fuites mémoire. Utilisation de OnDestroy pour unsubscribe des Observables.

Mises à Jour Optimistes vs. Pessimistes

- **Suppression (Pessimiste)** : Attend la réponse serveur avant this.todos = this.todos.filter(t => t.id !== id).
- **Toggle (Optimiste)** : Met à jour l'UI immédiatement (todo.completed = !todo.completed), puis synchronise en arrière-plan. Rollback en cas d'échec via catchError.
- **Gestion d'État** : Utilisation de ChangeDetectionStrategy.OnPush pour des performances optimales.

3.3 Le Modèle de Vue (todo-list.html)

Cette couche présente les données de manière réactive et accessible.

Flux de Contrôle Moderne

- **@for** : Syntaxe Angular 17+ : @for (todo of todos; track todo.id) { {{ todo.title }} }. Plus performant que *ngFor grâce au tracking.
- **@if** : @if (loading) { <div>Skeleton...</div> } @else if (todos.length === 0) { <p>Aucune tâche</p> }.

Variables de Référence de Template

#input sur <input> permet (input)="addTodo(input.value)", simplifiant l'accès sans propriétés supplémentaires.

Cycle de Vie d'une Requête de Bout en Bout

Pour illustrer l'intégration, voici le trace technique d'une opération "Supprimer une Tâche" (avec diagrammes textuels pour visualisation).

Déclenchement d'Événement	Traitement Frontend	Transmission Réseau
Clic sur <button (click)="delete(todo.id)">  </button> dans todo-list.html.	Composant → TodoService.deleteTodo(id) → HTTP DELETE vers http://localhost:8080/api/todos/5. Subscription : subscribe(() => this.todos = this.todos.filter(...)).	Navigateur → Preflight OPTIONS (CORS validé) → Requête DELETE.
Traitement Backend	Réponse & Mise à Jour	
TodoController.delete(id) → TodoService.delete(id) → TodoRepository.deleteById(id) → SQL : DELETE FROM todo WHERE id = ?.	Backend → HTTP 204 No Content. Frontend → Mise à jour du DOM via Angular Change Detection.	

Diagramme Simplifié

Utilisateur → Frontend (Angular) → HTTP DELETE → Backend (Spring) → DB (SQL) → Response 204 → UI Update

Conclusion

Ce codebase démontre une maturité élevée adaptée au développement web moderne.

Modularité	Scalabilité	Standards
Séparation frontend/backend pour un développement agile et des déploiements CI/CD indépendants.	Architecture 3-tiers extensible (ex. : ajout de microservices ou de WebSockets pour temps réel).	Respect des principes REST, HTTP/2, et fonctionnalités phares des frameworks pour une robustesse accrue.

Recommandations Futures

1. Authentification Utilisateur (JWT)

- Inscription
- Connexion
- Sécurité basée sur JWT
- Todos propres à chaque utilisateur

2. Catégories / Tags

Permettre d'assigner les tâches à :

- des catégories (Travail, Études, Santé...)
- ou plusieurs tags

3. Niveaux de Priorité

Basse / Moyenne / Haute avec couleurs dans l'interface.

5. Dates Limite & Rappels

- dates d'échéance
- petites notifications de rappel (mail ou notification navigateur)