École Nationale d'Ingénieurs de Carthage

DOCUMENT D'ARCHITECTURE

Structure du Framework & Stack Technique

Projet de Validation Logicielle (ISTQB)

Todo App (Spring Boot + React)

Équipe QA

Mohamed Dhia Eddine Thabet - Test Manager
Mohamed Aziz Dridi - Tester

Décembre 2025
Version 1.1

Table des matières

1 Vue d'ensemble de l'Architecture

L'application **Todo App** repose sur une architecture moderne de type *FullStack*, séparant distinctement la logique de présentation (Frontend) de la logique métier (Backend). Cette séparation garantit une meilleure maintenabilité et facilite les tests isolés.

- **Frontend :** Single Page Application (SPA) développée avec **React**.
- **Backend :** API RESTful robuste développée avec **Spring Boot (Java)**.
- **Base de Données :** H2 (In-Memory) pour le développement, compatible PostgreSQL.
- **CI/CD :** Pipeline automatisé via **Jenkins**.

2 Structure du Projet

L'organisation des fichiers a été structurée pour séparer clairement les responsabilités techniques et les livrables qualité.

2.1 1. Organisation Globale

À la racine du dépôt, nous distinguons quatre zones principales :

```
.├──
 react/          # Application Frontend (Client)├──
 todo-app/       # Application Backend (Serveur)├──
 docs/           # Documentation du projet (Livrables)└──
 photos/         # Preuves d'exécution et rapports visuels
```

2.2 2. Détail du Module Frontend (React)

Le dossier `react/` contient la logique de l'interface utilisateur. L'architecture sépare les composants de la logique de test.

```
react/├──
 public/                 # Ressources statiques (index.html)├──
 src/│├──
    components/         # Composants UI réutilisables│├──
       Footer/         # Pied de page│└──
       todoitem.jsx    # Composant unitaire de tâche│├──
    App.js              # Contrôleur principal et logique API│└──
    index.js            # Point d'entrée de l'application└──
 cypress/└──
     e2e/                # Tests automatisés End-to-End└──
         todo_app.cy.js  # Scénarios de tests fonctionnels
```

2.3 3. Détail du Module Backend (Spring Boot)

Le dossier `todo-app/` encapsule l'API REST et respecte l'architecture en couches standard de Spring Boot.

```
todo-app/├──
 Jenkinsfile             # Configuration du Pipeline CI/CD├──
 pom.xml                 # Gestionnaire de dépendances Maven└──
 src/├──
     main/java/.../todoapp/│├──
        web/            # Couche Exposition (Controllers REST)│├──
        service/        # Couche Métier (Business Logic)│├──
        repository/     # Couche Données (JPA Repository)│└──
        domain/         # Entités (Modèle de données)└──
     test/java/.../todoapp/├──
         integration/    # Tests d'Intégration├──
         performance/    # Scripts de charge└──
         web/            # Tests Unitaires des Contrôleurs
```

3 Détails du Framework Backend (Spring Boot)

Le backend suit une architecture en couches stricte (Layered Architecture), implémentée dans le dossier `todo-app`.

3.1 Couche Web (Controller)

Package : `com.dhia.todoapp.web`
Le `TodoController` expose les endpoints REST (GET, POST, PUT, DELETE). Il ne contient aucune logique métier, mais délègue les traitements à la couche Service.

3.2 Couche Service (Business Logic)

Package : `com.dhia.todoapp.service`
Le `TodoService` contient les règles de gestion. C'est ici que les transactions sont gérées. Il est testé unitairement avec JUnit et Mockito.

3.3 Couche Persistance (Repository)

Package : `com.dhia.todoapp.repository`
Le `TodoRepository` étend l'interface `JpaRepository` de Spring Data, permettant une abstraction complète des requêtes SQL.

4 Détails du Framework Frontend (React)

Le frontend est construit autour de la bibliothèque React, située dans le dossier `react`.

4.1 Composants et État

L'application utilise les **Hooks** (`useState`, `useEffect`) pour gérer le cycle de vie des composants sans utiliser de classes.

— **App.js :** Composant racine qui gère la liste des tâches (`todoItems`) et la synchronisation avec l'API.

— **todoitem.jsx :** Composant pur (stateless) responsable uniquement de l'affichage d'une ligne de tâche.

4.2 Tests End-to-End (Cypress)

Les tests d'interface sont situés dans `react/cypress`. Ils lancent un navigateur réel pour simuler les actions utilisateur (clic, saisie, validation).

5 Outils de Qualité et Automatisation

L'infrastructure de test repose sur les outils suivants, visibles dans le dossier `photos/` du projet :

1. **Jenkins :** Orchestrateur du déploiement continu.
2. **SonarQube :** Analyse statique du code (Détection de Code Smells et Bugs).
3. **JaCoCo :** Rapport de couverture de code (Code Coverage) pour le Java.
4. **JMeter :** Tests de performance et de charge.