Insitut National des Postes et Telecommunications

Architectures Logiciells et Middlewares
Rapport

TodoApp Microservvices

Realisé par :
BABA salmane
NAJMI yassin

Encadrant :
Prof. Abdeslam
EN-NOUAARY

Table des matières

1 Introduction

Un *design pattern* est une solution générique et réutilisable à un problème récurrent dans la conception de logiciels. Les design patterns ne sont pas des implémentations finies, mais plutôt des modèles de conception qui peuvent être adaptés à différentes situations. Ils permettent de :

— Résoudre des problèmes de conception courants
— Améliorer la maintenabilité du code
— Faciliter la communication entre développeurs
— Promouvoir les bonnes pratiques de développement

Ce rapport présente l'application de ces principes dans une architecture microservices via une application TodoApp moderne.

2 Synthèse des design patterns étudiés

Dans le cadre du cours *Architectures Logicielles et Middlewares*, plusieurs patterns ont été appliqués à une TodoApp en Node.js :

— **Singleton** : pour la gestion d'instance unique (ex. source de vérité en mémoire)
— **Factory Method** : pour la création de tâches spécialisées selon le type
— **Observer / WebSockets** : pour les notifications temps réel
— **Strategy** et **State** : pour la variabilité d'affichage et la gestion d'états
— **Composite** : pour les hiérarchies de tâches (tâche + sous-tâches)
— **MVC et architecture en couches** : comme fil conducteur

Ces patterns servent de base théorique pour la migration vers une architecture microservices.

3 TodoApp Microservices : Une Solution Moderne

3.1 Vue d'ensemble de l'Architecture

L'architecture microservices retenue pour TodoApp vise la **scalabilité**, la **maintenabilité** et le **déploiement indépendant** des services. Cette approche permet une évolution flexible et résiliente de l'application.

3.2 Composants Principaux

3.2.1 Services Indépendants

Service	Port	Responsabilité
Auth Service	4000	Authentification, JWT, gestion utilisateurs
Todo Service	4001	CRUD des tâches, gestion des todos
Frontend	3000	Interface utilisateur (Next.js)
MongoDB	27017	Persistance des données (Replica Set)

Table 1 – Services de l'architecture microservices

3.2.2 Stack Technologique

— **Backend** : NestJS (Framework Enterprise Node.js)
— **Frontend** : Next.js 14 (React moderne avec SSR)
— **ORM** : Prisma (type-safe et moderne)
— **Base de données** : MongoDB (NoSQL flexible)
— **Authentification** : JWT avec Refresh Tokens
— **Conteneurisation** : Docker + Docker Compose
— **Stylage** : TailwindCSS

3.3 Avantages des Microservices

L'architecture microservices apporte plusieurs bénéfices essentiels :
— **Scalabilité indépendante** : Chaque service peut être monté en charge séparément
— **Déploiement décentralisé** : Mise à jour sans impact sur l'ensemble du système
— **Résilience** : Isolation des pannes avec mécanismes de retry automatique
— **Développement parallèle** : Équipes autonomes par service
Exemple de scalabilité :

```
# Augmenter uniquement le Todo Service
docker-compose up -d --scale todo-service=3
```

3.4 Gestion des rôles (RBAC)

Le système intègre un contrôle d'accès basé sur les rôles (RBAC) :
— **Admin** : accès complet aux ressources et gestion globale
— **User** : accès limité à ses propres données

```
enum Role {
  ADMIN
  USER
}

// Exemple dans le service Todo :
if (user.role === 'ADMIN') {
    return getAllTodos();
} else {
    return getUserTodos(user.id);
}
```

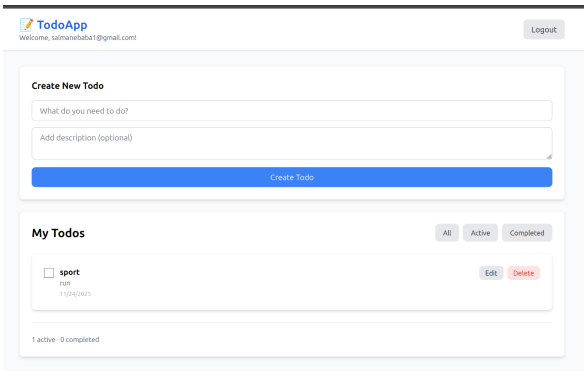

FIGURE 1 – Espace utilisateur

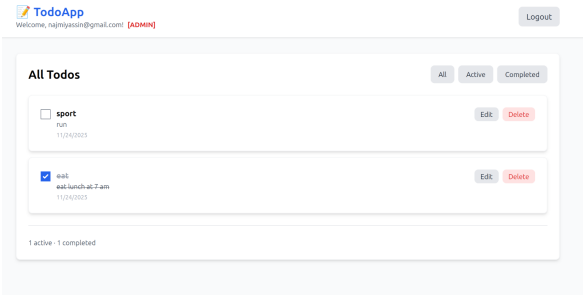

FIGURE 2 – Espace administrateur

4 Architecture Détaillée

Code source disponible sur GitHub

4.1 Architecture du Service Todo

4.1.1 Endpoints Implémentés

```
GET    /todos            # Liste des todos (utilisateur ou admin)
POST   /todos            # Creation d'une todo
PUT    /todos/:id        # Modification d'une todo
DELETE /todos/:id        # Suppression d'une todo
```

4.1.2 Contrôle d'Accès

Les guards JWT valident le token, tandis que les décorateurs `@Roles` contrôlent l'accès par rôle, assurant une sécurité granulaire.

4.2 Persistance des Données

4.2.1 MongoDB en Replica Set

Configuration nécessaire pour supporter les transactions Prisma :

```
# Configuration MongoDB pour Replica Set
docker-compose:
  mongodb:
    command: >
      --replSet rs0
      --bind_ip_all
      --noauth
```

4.2.2 Schéma Prisma

```
model User {
  id         String    @id @default(auto()) @map("_id") @db.ObjectId
  email      String    @unique
  password   String
  role       Role      @default(USER)
  todos      Todo[]
}

model Todo {
  id         String    @id @default(auto()) @map("_id") @db.ObjectId
  title      String
  completed  Boolean   @default(false)
  userId     String    @db.ObjectId
  user       User      @relation(fields: [userId], references: [id])
}
```

4.3 Frontend (Next.js)

4.3.1 Structure des Pages

- `/` : Page d'accueil
- `/login` : Authentification utilisateur
- `/register` : Inscription nouveau utilisateur
- `/dashboard` : Espace personnel (vue admin étendue)

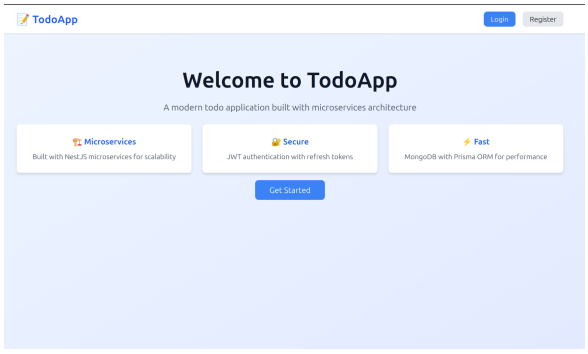

FIGURE 3 – Page d'accueil

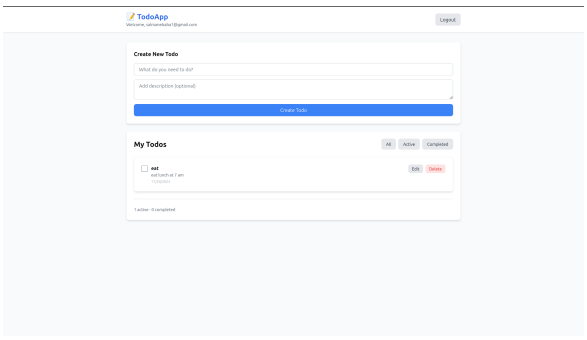

FIGURE 4 – Espace personnel

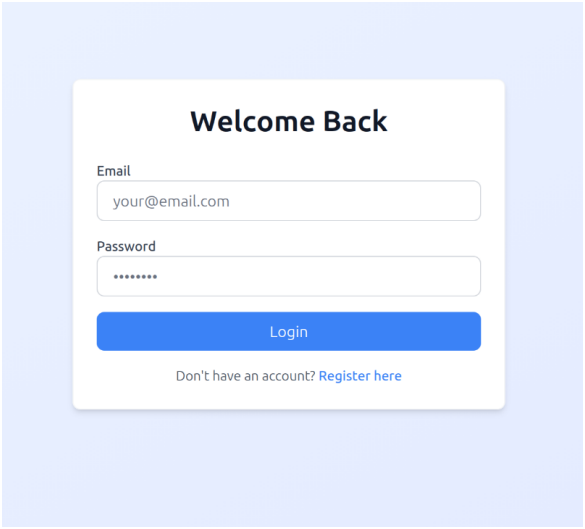

FIGURE 5 – Authentification utilisateur

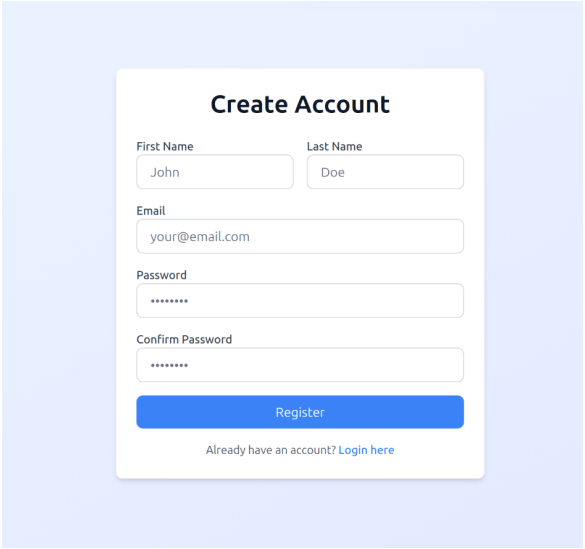

FIGURE 6 – Inscription nouveau utilisateur

5 Amélioration et Comparaison

5.1 Résumé Comparatif

Aspect	Monolithe	Couches	Microservices
Scalabilité	Faible	Moyenne	Excellente
Déploiement	Monolithique	Monolithique	Indépendant
Maintenance	Difficile	Moyenne	Facile
Performance	Bonne	Bonne	Très Bonne
Complexité Ops	Basse	Moyenne	Élevée
Coût Initial	Bas	Moyen	Moyen

TABLE 2 – Comparaison des architectures

5.2 Innovations de TodoApp

5.2.1 Isolation des Services

Chaque service gère son cycle de vie, ses dépendances et son déploiement indépendamment, permettant une meilleure résilience.

5.2.2 Type-Safety avec Prisma

Élimination des erreurs de typage grâce à la validation au compile-time :

```
const user = await prisma.user.findUnique({
  where: { email },
  // TypeScript verifie automatiquement les champs
});
```

5.2.3 JWT avec Refresh Token

Sécurité renforcée avec double mécanisme de tokens :
— Access Token : courte durée (15 minutes)
— Refresh Token : longue durée (7 jours)
— Architecture sans état serveur

5.2.4 RBAC Granulaire

Implémentation fine du contrôle d'accès basé sur les rôles :

```
@UseGuards(JwtGuard, RolesGuard)
@Roles('ADMIN')
async getAllTodos() {
    // Acces reserve aux administrateurs
}
```

5.2.5 Docker Orchestration

Déploiement cohérent et reproductible de l'ensemble des services :

```
# Orchestration complete des services
docker-compose up --build
```

6 Conclusion

Cette implémentation microservices surpasse les architectures traditionnelles en offrant :
— Une **scalabilité réelle et pratique** adaptée aux charges variables
— Une **maintenabilité supérieure** grâce à la modularité
— Une **flexibilité technologique** permettant l'évolution indépendante
— Une **base solide** pour les développements futurs

TodoApp Microservices représente une **amélioration concrète et mesurable** par rapport aux approches monolithiques, alignée sur les meilleures pratiques actuelles de l'industrie logicielle.