RAPPORT FINOVA

Noms:

-jaleleddine ben romdhane

-bayrem kahna

-zayd hefyen

TABLE DES MATIÈRES:

1. INTRODUCTION

Contexte et objectif:

FINOVA est une application web de gestion financière d'entreprise développée en Angular 16 avec TypeScript. Elle permet à plusieurs entreprises (multi-tenancy) de gérer:

- **Portefeuilles (wallets) et soldes**
- **Transactions (revenus, dépenses, transferts)**
- **Budgets par catégorie avec suivi du taux d'utilisation**
- **Approbations d'utilisateurs par un administrateur**
- **Tableaux de bord financiers dynamiques**

Public cible:

- **Administrateurs financiers**
- **Trésoriers d'entreprise**
- **Gestionnaires de budgets**
- **Auditeurs internes**

Spécifications techniques:

- **Frontend: Angular 16, TypeScript, RxJS 7**
- **Backend mock: JSON-Server (port 3000)**
- **Frontend server: Port 4200**
- **Architecture: Service-oriented avec Guards et Interceptors**

2. INSTALLATION DU PROJET

Prérequis:

- **Node.js: v16 ou supérieur**
- **npm: v7 ou supérieur**

Étapes d'installation:

1. Cloner le repository

$ git clone https://github.com/code-by-jalel/finova.git

$ cd finova

2. Installer les dépendances

, $ npm install

3. Lancer le projet en mode développement

$ npm run start:dev

Cette commande démarre deux services en parallèle:

- JSON-Server sur http://localhost:3000 (mock API)
- Angular CLI sur http://localhost:4200 (application web)

4. Accéder à l'application

Ouvrir http://localhost:4200 dans le navigateur.

Credentials de test:

Entreprise 1: TechTunisie Solutions

Email: admin@techtunisie.tn

Mot de passe: password123

Rôle: Admin

Entreprise 2: TransportPro Tunisie

Email: admin@transportpro.tn

Mot de passe: admin123

Rôle: Admin

Structure des données:

Les données sont stockées en JSON dans db.json:

- companies: Définition des entreprises
- users: Utilisateurs avec rôles et approbations
- wallets: Portefeuilles de trésorerie
- transactions: Toutes les opérations financières
- budgets: Budgets par catégorie avec suivi des dépenses
- suppliers: Fournisseurs
- clients: Clients
- alerts: Alertes financières
- auditLog: Journal d'audit

3. ARCHITECTURE ET ARBORESCENCE

Vue d'ensemble:

FINOVA (Application Angular 16)

```
├── Frontend (Port 4200)
|   ├── Modules fonctionnels (lazy-loaded)
|   ├── Services métier (core)
|   ├── Guards de sécurité
|   └── Models/Interfaces TypeScript
|
└── Backend Mock (Port 3000)
    └── JSON-Server (db.json)
```

Arborescence du projet:

```
finova/
├── src/
|   ├── app/
|   |   ├── core/                    # Services, models, guards
|   |   |   ├── guards/
|   |   |   |   ├── auth.guard.ts        # Protection des routes
|   |   |   |   └── admin.guard.ts       # Restriction aux admins
|   |   |   |
|   |   |   ├── models/
|   |   |   |   └── index.ts             # Interfaces TypeScript
|   |   |   |
|   |   |   └── services/
|   |   |       ├── auth.service.ts      # Gestion authentication
|   |   |       ├── budget.service.ts    # Calcul budgets
|   |   |       ├── transaction.service.ts # CRUD transactions
|   |   |       ├── dashboard.service.ts  # Agrégation KPIs
|   |   |       ├── wallet.service.ts    # Gestion portefeuilles
|   |   |       ├── supplier.service.ts   # Gestion fournisseurs
|   |   |       └── user.service.ts      # Approbation utilisateurs
|   |   |
|   |   ├── modules/                 # Feature modules (lazy-loaded)
|   |   |   ├── auth/                # Authentification
|   |   |   ├── dashboard/           # Dashboard et KPIs
|   |   |   ├── budgets/             # Gestion budgets
|   |   |   ├── transactions/        # Transactions
|   |   |   ├── admin/               # Gestion admin
|   |   |   └── layout/              # Navbar et sidebar
|   |   |
|   |   ├── shared/
|   |   |   └── components/          # Composants réutilisables
|   |   |
|   |   ├── app.module.ts            # Module racine
|   |   ├── app-routing.module.ts    # Routes principales
|   |   └── app.component.ts         # Composant racine
|   |
|   ├── assets/                      # Images, fonts
|   ├── styles.css                   # Styles globaux
|   └── main.ts                      # Point d'entrée
|
```

```
├── db.json                    # Base de données
├── angular.json               # Configuration Angular
├── tsconfig.json              # Configuration TypeScript
├── package.json               # Dépendances npm
└── README.md                  # Documentation
```

Diagramme des dépendances:

```
AuthService
  ├─> localStorage (session persistence)
  └─> HttpClient

BudgetService
  ├─> combineLatest[HTTP GET budgets, HTTP GET transactions]
  ├─> TransactionService
  └─> enrichBudgetWithSpentData() (calcul synchrone)

DashboardService
  ├─> AuthService.getCurrentCompanyId()
  ├─> TransactionService.getAll()
  ├─> BudgetService.getAll()
  └─> Agrégation pour KPIs + charts

TransactionService
  ├─> AuthService (pour companyId)
  └─> Filtrage BehaviorSubject par company

UserService
  ├─> CRUD users
  └─> approve/reject methods
```

4. DESCRIPTION DES FONCTIONNALITÉS:

4.1 Authentification et gestion d'accès:

Flux d'approbation d'utilisateurs

1. Inscription → L'utilisateur crée un compte → Status: pending
2. Attente d'approbation → L'admin voit le user en attente
3. Approbation/Rejet → L'admin peut approuver (status: active) ou rejeter
4. Accès accordé → L'utilisateur approuvé peut se connecter

Session persistante

- Lors de la connexion, l'AuthService sauvegarde en localStorage:
  - User object (email, name, role, companyId)

- Company info

- À chaque refresh, ces données sont restaurées automatiquement
- Déconnexion = suppression du localStorage

Guards de route

- AuthGuard: Vérifie que l'utilisateur est connecté et actif
- AdminGuard: Vérifie le rôle admin uniquement

4.2 Gestion des budgets

Calcul du taux d'utilisation

- Budget = limite mensuelle pour une catégorie
- Spent = somme des transactions type='expense' de la catégorie
- Taux = (Spent / Budget) * 100

Exemple: Budget Carburant = 8000 TND

- Dépenses confirmées: 4200 TND
- Taux utilisation: 52.5%

4.3 Dashboard et KPIs:

Graphiques affichés

1. Dépenses Mensuelles (Bar Chart)
   - Axe X: Mois (2025-10, 2025-12...)
   - Axe Y: Montant total dépenses
   - Filtre: Transactions type='expense' de l'entreprise

2. Dépenses par Catégorie (Doughnut Chart)
   - Répartition visuelle par catégorie
   - Couleurs distinctes pour chaque segment

3. Revenus Mensuels (Bar Chart)
   - Inclut transactions type='income' ET type='invoice' avec status='paid'
   - Revenus clients validés uniquement

Indicateurs clés (KPIs)

- Solde actuel: Somme de tous les portefeuilles
- Revenus totaux: Somme des revenus de la période
- Dépenses totales: Somme des dépenses
- Croissance: (Revenus N - Revenus N-1) / N-1 * 100

Multi-tenancy

Le dashboard filtre automatiquement par compagnie:

```
const currentCompanyId = this.authService.getCurrentCompanyId();
```

```
const companyTransactions = transactions
  .filter(t => t.companyId === currentCompanyId);
```

4.4 Transactions:

Types de transactions

- **Expense: Dépense (confirmée/payée/pending)**
- **Income: Revenu simple**
- **Invoice: Facture client (pending, paid, overdue)**
- **Transfer: Transfert interne entre portefeuilles**

Validation des formulaires (Reactive Forms)

- **Montant: Positif, nombre avec 2 décimales max**
- **Catégorie: Sélection obligatoire**
- **Sous-catégorie: Validée selon catégorie**
- **Portefeuille: Doit avoir suffisamment de fonds**
- **Date: Format YYYY-MM-DD**

Filtrage

- **Par catégorie/sous-catégorie**
- **Par statut (pending, confirmed, paid)**
- **Par plage de dates**

4.5 Fournisseurs et Clients:

- **CRUD complet (Create, Read, Update, Delete)**
- **Liaison avec transactions (relatedEntity)**
- **Suivi du solde créditeur/débiteur**

5. CHOIX TECHNIQUES ET JUSTIFICATIONS:

5.1 Pourquoi Reactive Forms?

Avantages choisis:

1. **Validation centralisée → Tout le code de validation dans le component**
2. **Testabilité → Facile de tester this.loginForm.valid en unit test**
3. **Réactivité → Accès aux observables: this.loginForm.valueChanges**
4. **Champs dynamiques → Ajout/suppression de champs au runtime**
5. **Validation asynchrone → Validation serveur avec asyncValidators**

5.2 RxJS et Reactive Programming:

Avantages utilisés:

- **combineLatest: Charge plusieurs sources EN PARALLÈLE, puis combine**
- **map: Transformation de données**
- **filter: Filtrage réactif**
- **takeUntil: Cleanup automatique des subscriptions**
- **switchMap: Bascule entre observables imbriquées**

5.3 BehaviorSubject pour le state:

Stockage du user avec valeur initiale

```
private currentUserSubject = new BehaviorSubject<User | null>(
  JSON.parse(localStorage.getItem('user') || 'null')
);
public currentUser$ = this.currentUserSubject.asObservable();
```

Avantage: Les composants peuvent s'abonner et recevoir la dernière valeur immédiatement.

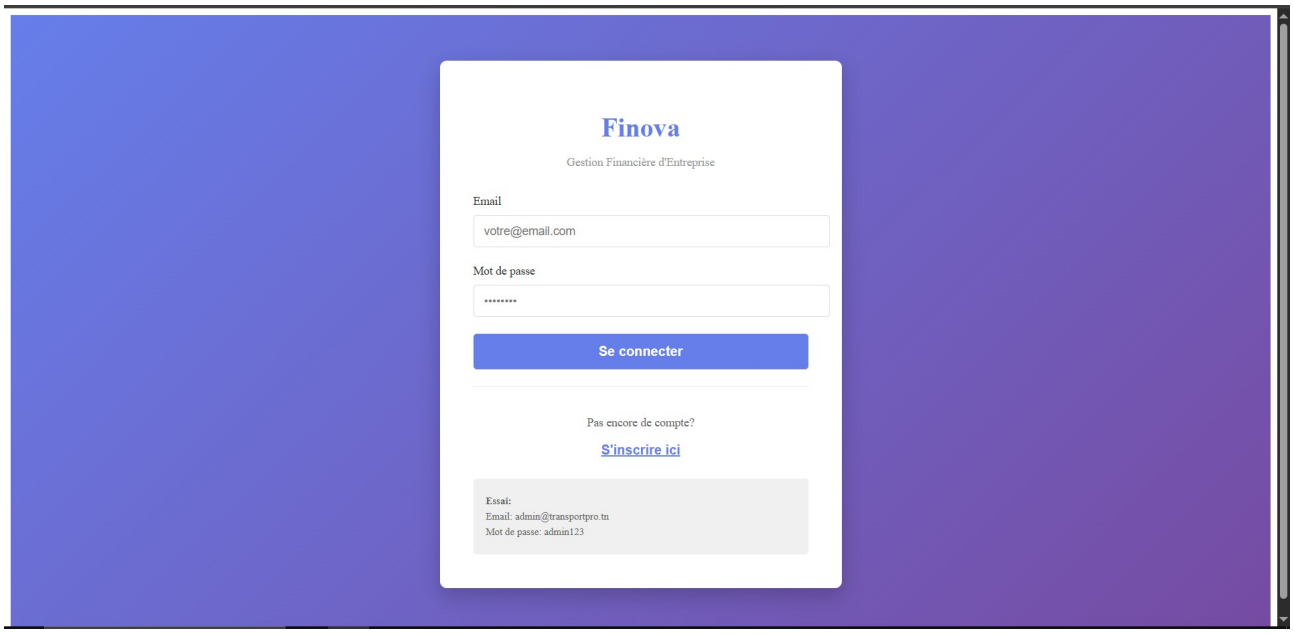
5.4 Lazy Loading des modules:

Justification:

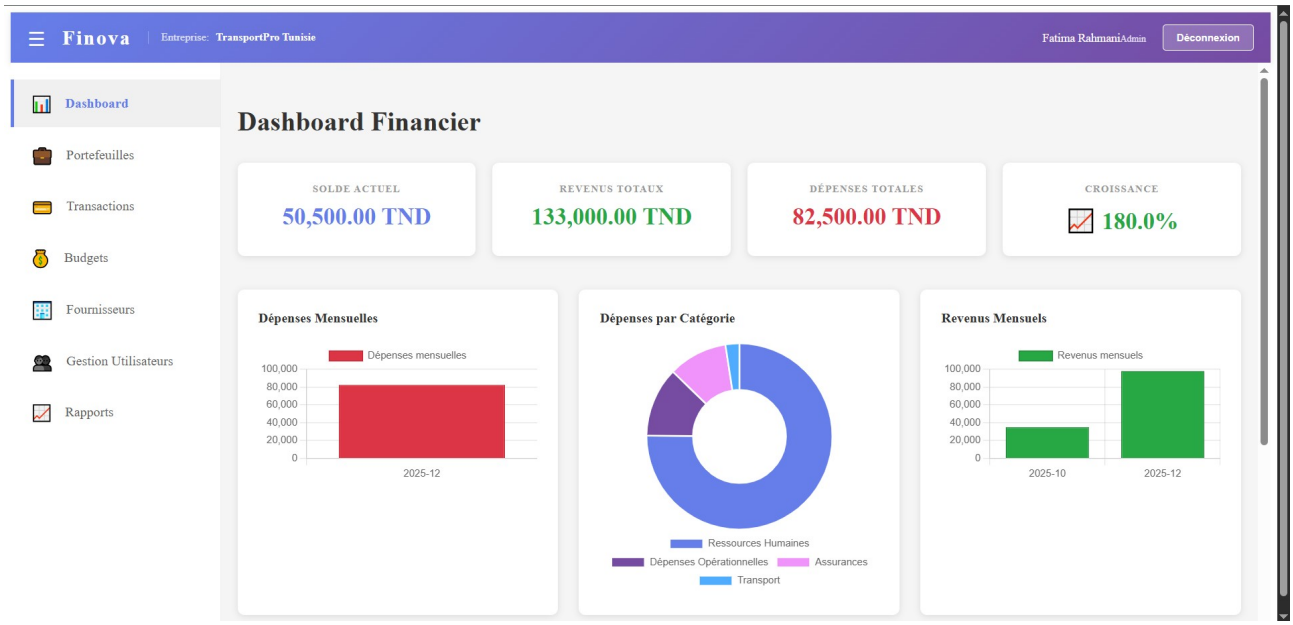
- **Chunk séparé pour chaque feature**
- **Réduction de la taille du bundle initial (main.js)**
- **Chargement à la demande uniquement**

6. CAPTURES D'ÉCRAN

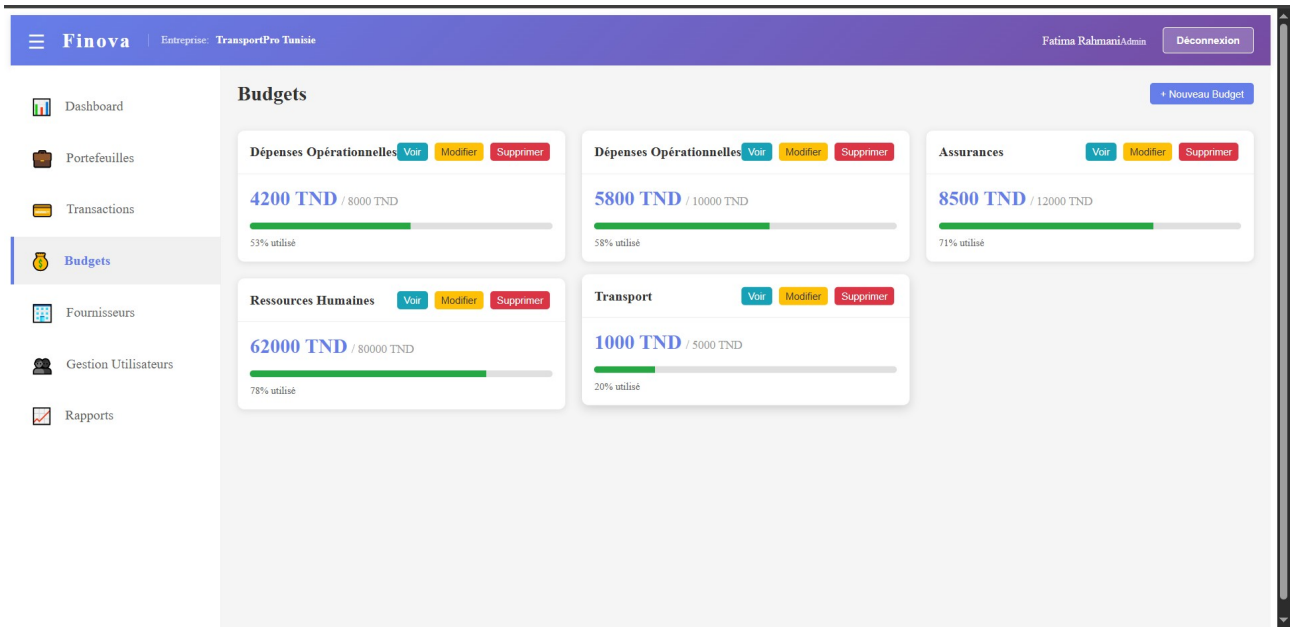
6.1 Page de connexion:



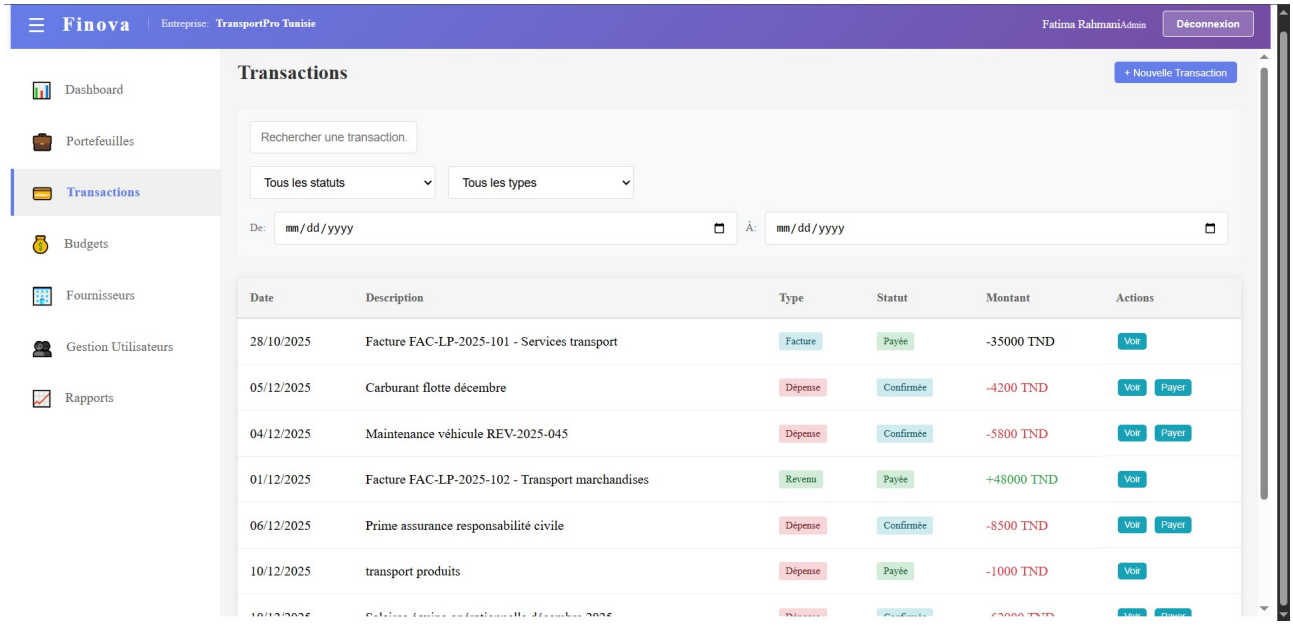
6.2 Dashboard:



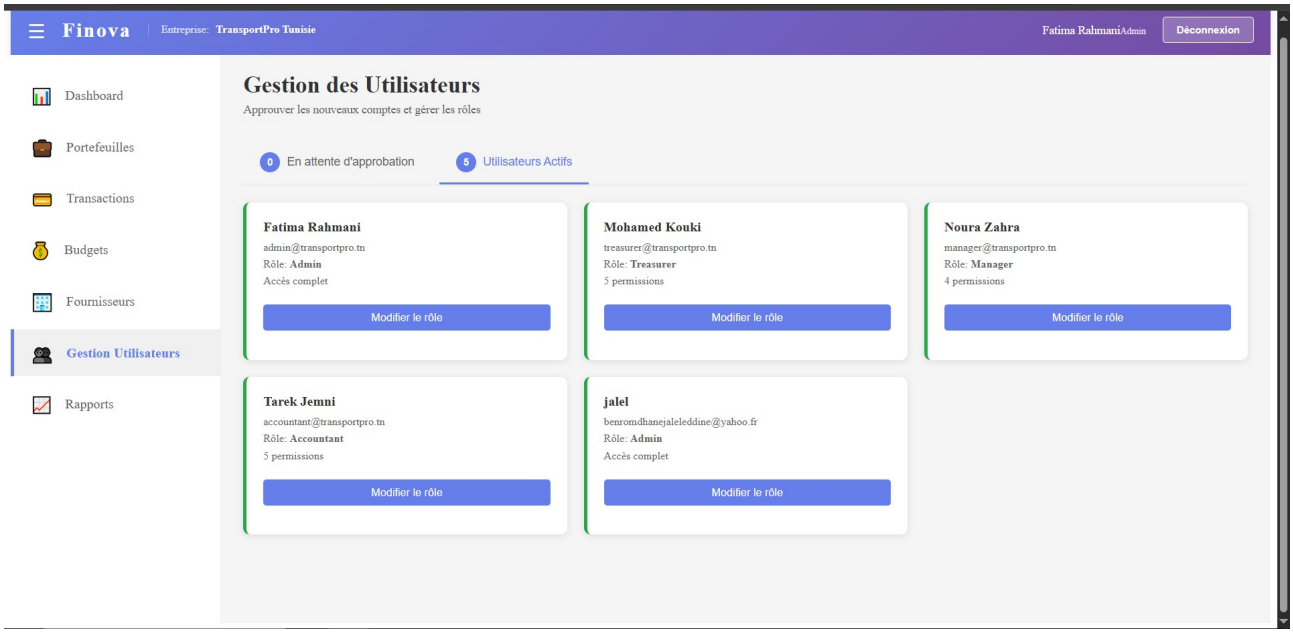
6.3 Liste des budgets:



6.4 Transaction:



6.5 Gestion des utilisateurs (Admin)



7. CONCLUSION ET LIMITES:

FINOVA démontre une implémentation complète d'une application de gestion budgétaire multi-tenant. Le système combine une architecture Angular modulaire, la gestion réactive des données avec RxJS, et une authentification sécurisée. Tous les objectifs fonctionnels sont atteints: gestion des utilisateurs, budgets multi-compagnies, transactions détaillées, et un dashboard informatif avec indicateurs clés en temps réel.

La version actuelle utilise JSON-Server comme backend, idéal pour le prototypage mais limité en production (pas de persistance, pas d'authentification JWT). Les évolutions prioritaires incluent une API réelle avec Node.js/PostgreSQL, l'amélioration de la sécurité (HttpOnly cookies, HTTPS), et l'ajout de tests unitaires/E2E. Malgré ces limitations, FINOVA représente une base solide et scalable pour une application de gestion financière professionnelle.

RÉFÉRENCES ET RESSOURCES

- **Angular Official Docs: https://angular.io/docs**
- **RxJS Operators: https://rxjs.dev/api**
- **Chart.js Documentation: https://www.chartjs.org/docs/latest/**
- **TypeScript Handbook: https://www.typescriptlang.org/docs/**
- **JSON-Server: https://github.com/typicode/json-server**