RAPPORT DU TRAVAIL

PRÉSENTÉ PAR:

ABDELHAFIS ISSA MAHAMAT
DJASRANE TOLBA SUCCÈS
HISSEIN MAHAMAT DRYA

Formateur: HASSAN MOUSTAPHA OUSMANE

Table des Matières

Introduction

Angular est un framework frontend développé par Google, basé sur TypeScript. Il permet de créer des applications web dynamiques et robustes en utilisant une architecture basée sur des composants. Ce rapport présente les concepts fondamentaux d'Angular dans le contexte du développement d'une application de gestion des produits pour une entreprise e-commerce.

L'objectif de ce projet est de développer une interface utilisateur permettant de gérer un inventaire de produits avec les opérations CRUD (Create, Read, Update, Delete).

1. Les Concepts Fondamentaux d'Angular

1.1 Les Composants

Les composants sont les éléments de base d'Angular. Ils encapsulent la logique métier, les données et l'interface utilisateur.

Structure d'un composant :

- Classe TypeScript : contient la logique et les propriétés
- Template HTML : définit l'interface utilisateur
- Styles CSS : gestion de l'apparence
- Métadonnées : configuration via le décorateur @Component

1. Les Concepts Fondamentaux d'Angular

1.1 Les Composants

Application dans le projet **:** Dans notre application de gestion des produits, nous utiliserons plusieurs composants :

App Module et configuration

- `app.module.ts`: permet d'importer tous les composants
- `app.rout.ts` : permet de gérer les routes
- `config.ts` : permet la configuration générale du projet
- `composant.ts` : est la racine de l'application

1.2 Les Services

Les services permettent de centraliser la logique et le partage de données entre composants. Ils respectent le principe de séparation des responsabilités.

Caractéristiques des services :

- Singleton par défaut (une seule instance)
- Injection de dépendances avec @Injectable
- Séparation de la logique métier des composants
- Communication avec les APIs externes

Application dans le projet :

- `ProductService` : gestion des opérations CRUD sur les produits
- `ApiService` : communication avec le backend

1.3 Le Routing

Le système de routing d'Angular permet de naviguer entre différentes vues de l'application sans recharger la page (Page Application).

Concepts clés :

- Route : association entre une URL et un composant
- RouterModule : module de configuration des routes
- Router : service de navigation programmatique
- RouterOutlet : directive d'affichage des composants routés

Application dans le projet :

```
dashboard        => DashboardComponent
produits         => ProductListComponent
produits/add     =>ProductFormComponent
produits/:id     => ProductDetailComponent
produits/:id/edit => ProductFormComponent
```

1.4 Les Formulaires

Angular propose deux approches pour gérer les formulaires :

Template-driven Forms :

- Configuration dans le template HTML
- Utilisation de ngModel pour la liaison bidirectionnelle
- Validation déclarative

Reactive Forms (recommandé) :

- Configuration dans le composant TypeScript
- Contrôle programmatique des données
- Validation plus flexible et testable

Application dans le projet **:** Les Reactive Forms seront utilisées pour les formulaires de gestion des produits, permettant une évaluation robuste des champs (nom, description, prix, catégorie, stock).

1.5 Les Directives

Les directives étendent les fonctionnalités HTML en ajoutant des comportements dynamiques.

Types de directives **:**

- Directives structurelles : modifient la structure du DOM (@If, @For, @Switch)
- Directives d'attribut : modifient l'apparence ou le comportement (ngClass, ngStyle)
- Directives personnalisées : fonctionnalités spécifiques

Application dans le projet :

- `@For` : itération sur la liste des produits
- `@If` : affichage conditionnel des éléments

1.6 Les Pipes

Les pipes transforment les données pour l'affichage sans modifier les données sources.

Pipes intégrés :

- DatePipe : formatage des dates
- CurrencyPipe : formatage monétaire
- UpperCasePipe/LowerCasePipe : transformation de casse
- DecimalPipe : formatage des nombres

Application dans le projet :

- Formatage des prix avec CurrencyPipe
- Formatage des dates de création/modification
- Pipes personnalisés pour le formatage spécifique

3. Implémentation du CRUD

L'implémentation complète des opérations CRUD (Create, Read, Update, Delete) constitue le cœur fonctionnel de l'application.

Architecture CRUD

Create (Création) :

- Formulaire de saisie avec validation
- Service pour l'envoi des données
- Redirection après succès

Read (Lecture) :

- Affichage de la liste des produits
- Vue détaillée d'un produit
- Pagination et recherche

Update (Modification) :

- Pré-remplissage du formulaire avec les données existantes
- Validation et mise à jour
- Confirmation des modifications

3. Implémentation du CRUD

Delete (Suppression) :

- Confirmation avant suppression
- Gestion des erreurs
- Mise à jour de l'interface

Intégration avec les Services

Les opérations CRUD sont centralisées dans le `ProductService` qui expose les méthodes :

- `getProducts()` : récupération de tous les produits
- `getProduct(id)` : récupération d'un produit spécifique
- `createProduct(product)` : création d'un nouveau produit
- `updateProduct(id, product)` : mise à jour d'un produit
- `deleteProduct(id)` : suppression d'un produit

4. Les Reactive Forms

Les Reactive Forms offrent une approche programmatique pour la gestion des formulaires, particulièrement adaptée aux applications complexes.

Avantages des Reactive Forms

- Contrôle programmatique : gestion complète depuis le composant
- Validation flexible : validateurs synchrones et asynchrones
- Testabilité : logique de formulaire facilement testable
- Performance : évite les cycles de détection de changements

Structure des Reactive Forms

FormBuilder : service pour construire les formulaires FormGroup : groupe de contrôles de formulaire FormControl : contrôle individuel d'un champ Validators : fonctions de validation

1. Les Reactive Forms

Application dans le Projet

Pour le formulaire de produit :

typescript

```
productForm = this.fb.group({
  name: ['', [Validators.required, Validators.minLength(2)]],
  description: ['', Validators.required],
  price: ['', [Validators.required, Validators.min(0)]],
  category: ['', Validators.required],
  stock: ['', [Validators.required, Validators.min(0)]]
  });
```

Validation des Données

- Validation côté client : validation immédiate pour l'expérience utilisateur
- Messages d'erreur dynamiques : affichage des erreurs en temps réel
- Validation personnalisée : règles métier spécifiques

2. Structure et Bonnes Pratiques

Architecture de l'Application

Structure recommandée :

```
src/app/
├── core/              # Services singleton, guards
├── shared/            # Composants, directives, pipes partagés
├── features/          # Modules fonctionnels
│   └── products/
│       ├── components/   # Composants spécifiques aux produits
│       ├── services/     # Services liés aux produits
│       └── models/       # Interfaces et modèles
└── layout/            # Composants de mise en page
```

Bonnes Pratiques Angular

Organisation du Code :

- Un composant par fichier
- Nommage cohérent et descriptif
- Séparation des responsabilités

3. Structure et Bonnes Pratiques

Performance :

- OnPush change detection strategy
- Lazy loading des modules
- TrackBy functions pour @For

Maintenabilité :

- Typage fort avec TypeScript
- Interfaces pour les modèles de données
- Documentation du code

Sécurité :

- Sanitization automatique d'Angular
- Validation côté serveur
- Gestion des erreurs

4. Connexion Backend (Optionnel)

Architecture Client-Serveur

L'application frontend communique avec un backend Node.js/Express/MongoDB via des APIs RES.

HttpClient Angular

Configuration :

- Import du HttpClientModule
- Injection du service HttpClient
- Gestion des headers et interceptors

Opérations HTTP :

- GET : récupération des données
- POST : création de nouvelles ressources
- PUT/PATCH : mise à jour des données
- DELETE : suppression de ressources

5. Connexion Backend (Optionnel)

Gestion des Erreurs

- Try/catch pour les opérations asynchrones
- Messages d'erreur utilisateur-friendly
- Retry automatique en cas d'échec réseau

Alternative : API Simulée

Pour le développement, utilisation d'un fichier JSON local avec json-server pour simuler une API backend.

6. Conclusion

Ce rapport a présente les concepts fondamentaux d'Angular nécessaires au développement d'une application de gestion des produits. Les éléments étudiés (composants, services, routing, formulaires, directives, pipes) constituent les piliers du framework et permettent de construire des applications web robustes et maintenables.

L'implémentation d'un CRUD complet avec les Reactive Forms garantit une expérience utilisateur optimale tout en respectant les bonnes pratiques de développement. La structure modulaire d'Angular facilite la maintenance et l'évolution future de l'application.

Le respect des bonnes pratiques et l'utilisation appropriée des concepts Angular assurent la qualité, la performance et la scalabilité de l'application développée.

Mots-clés : Angular, TypeScript, CRUD, Reactive Forms, SPA, Composants, Services, Routing