Comparaison Angular : NgModule vs Standalone

Architecture Classique vs Moderne

Rapport Technique

Année Universitaire 2025

Table des matières

1 Introduction

Angular est un framework frontend développé par Google, basé sur TypeScript, permettant de construire des applications web robustes et maintenables.

Historiquement, Angular reposait sur le concept de **NgModule** pour organiser les composants et les dépendances. Depuis Angular 15, une approche moderne appelée **Standalone Components** a été introduite afin de simplifier l'architecture.

Ce document présente :

— Une implémentation complète avec NgModule

— Une implémentation complète avec Standalone

— Une comparaison claire et structurée

2 Angular avec NgModule (Version Classique)

2.1 Structure du projet

```
src/app/
  app.module.ts
  app.component.ts
  app.component.html
  product/
      product.component.ts
      product.component.html
      product.service.ts
      product.model.ts
```

2.2 Modèle Product

Listing 1 – product.model.ts

```ts
export interface Product {
  id: number;
  name: string;
  price: number;
}
```

2.3 Service Product

Listing 2 – product.service.ts

```ts
@Injectable({ providedIn: 'root' })
```

```
export class ProductService {
  private products: Product[] = [];

  getAll(): Product[] {
    return this.products;
  }

  add(product: Product): void {
    this.products.push(product);
  }

  delete(id: number): void {
    this.products = this.products.filter(p => p.id !== id);
  }
}
```

2.4 Composant Product

Listing 3 – product.component.ts

```
@Component({
  selector: 'app-product',
  templateUrl: './product.component.html'
})
export class ProductComponent {
  name = '';
  price = 0;
  products: Product[];

  constructor(private service: ProductService) {
    this.products = this.service.getAll();
  }
}
```

2.5 Module Principal

Listing 4 – app.module.ts

```
@NgModule({
  declarations: [
    AppComponent,
```

```
    ProductComponent
  ],
  imports: [
    BrowserModule,
    FormsModule
  ],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

2.6 Fonctionnement

— Tous les composants sont déclarés dans un module
— Les dépendances sont importées globalement
— Architecture centralisée

3 Angular Standalone (Version Moderne)

3.1 Structure du projet

```
src/app/
  main.ts
  app.component.ts
  product/
      product.component.ts
      product.service.ts
      product.model.ts
```

3.2 ProductComponent Standalone

Listing 5 – product.component.ts

```
@Component({
  selector: 'app-product',
  standalone: true,
  imports: [CommonModule, FormsModule],
  templateUrl: './product.component.html'
})
export class ProductComponent {}
```

3.3 AppComponent Standalone

Listing 6 – app.component.ts

```
@Component({
  selector: 'app-root',
  standalone: true,
  imports: [ProductComponent],
  template: '<app-product></app-product>'
})
export class AppComponent {}
```

3.4 Bootstrap

Listing 7 – main.ts

```
bootstrapApplication(AppComponent);
```

4 Comparaison détaillée entre NgModule et Standalone

Cette section présente une comparaison claire et détaillée entre l'architecture Angular basée sur **NgModule** et l'architecture moderne **Standalone**. La comparaison porte sur les aspects structurels, techniques et maintenabilité.

Critère	**Angular avec NgModule (Classique)**	**Angular Standalone (Moderne)**
Gestion des modules	Présence obligatoire d'un module racine (`AppModule`) qui centralise les composants, directives et dépendances	Aucun module global requis ; chaque composant gère localement ses dépendances
Déclaration des composants	Les composants doivent être déclarés dans la section `declarations[]` du module	Chaque composant est autonome grâce à l'attribut `standalone: true`
Gestion des dépendances	Importation centralisée des modules (`FormsModule`, `HttpClientModule`, etc.)	Importation locale et explicite au niveau du composant concerné
Lisibilité du code	Lisibilité moyenne due à la séparation logique entre modules et composants	Lisibilité élevée, car toutes les dépendances sont visibles directement dans le composant
Complexité de l'architecture	Architecture plus complexe, surtout pour les grands projets ou les débutants	Architecture simplifiée et plus intuitive
Scalabilité	Adaptée aux applications existantes et aux architectures historiques	Meilleure scalabilité grâce à des composants indépendants et réutilisables
Temps de configuration	Temps de configuration plus long (modules + déclarations)	Démarrage rapide et configuration minimale
Approche moderne	Approche traditionnelle, aujourd'hui considérée comme legacy	Approche moderne recommandée par l'équipe Angular
Type de projets recommandés	Maintenance et évolution d'anciens projets Angular	Nouveaux projets Angular (Angular 15+)

Analyse générale

L'architecture basée sur **NgModule** reste parfaitement fonctionnelle et stable, mais elle introduit une complexité supplémentaire dans l'organisation du code.

À l'inverse, l'approche **Standalone** offre :

— une meilleure lisibilité du code,

— une réduction significative du couplage,

— une configuration simplifiée,

— une meilleure compréhension pour les développeurs débutants.

Ainsi, pour tout **nouveau projet Angular**, l'approche **Standalone** est aujourd'hui la solution la plus recommandée.

5 Conclusion

Angular Standalone représente une évolution majeure vers une architecture plus simple, plus claire et plus maintenable.

Pour les nouveaux projets Angular, l'approche Standalone est fortement recommandée