Angular 18/19

Une formation complète sur le développement d'applications web modernes avec Angular.

Appuyez sur espace pour la page suivante →

SOMMAIRE 📜

Voici le sommaire de cette formation sur Angular 18/19:

Introduction à Angular

Revenir au sommaire

Qu'est-ce qu'Angular ?

Imaginez Angular comme un kit complet pour construire une maison moderne :

- Les **fondations** (le framework core)
- Les **outils** (CLI, DevTools)
- Les **plans** (architecture)
- Les **matériaux** (composants)

Revenir au sommaire

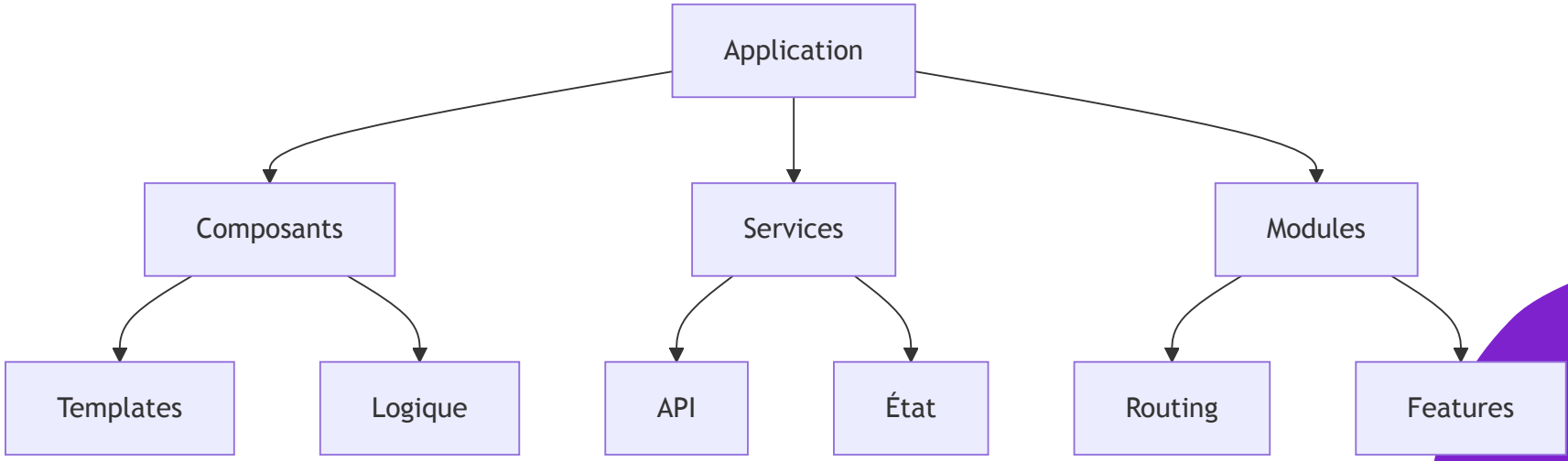
Exemple concret de ces termes :

- **Fondations** :
  - Framework core
  - TypeScript
  - RxJS
  - CLI
  - DevTools
- **Plans** :
  - Architecture orientée composants
  - Routage / Routing
  - State management avec RxJS / NgRx etc
- **Matériaux** :
  - Composants : boutons, formulaires, tables, etc.
  - Services : API, authentification, notifications, etc.

Revenir au sommaire

Architecture d'une application Angular

Structure typique d'un projet :



Revenir au sommaire

Les piliers d'Angular

1. Composants

Comme les LEGO® de votre application :

- Réutilisables
- Autonomes
- Combinables

2. Services

Comme les employés d'une entreprise :

- Spécialisés
- Partagés
- Indépendants

Revenir au sommaire

3. Dependency Injection

Comme un système de livraison automatique :

- Efficace
- Flexible
- Testable

Un exemple concret :

```
 @Injectable({
   providedIn: 'root'
})
export class UserService {
   constructor(private http: HttpClient) {}
}
```

Ce qui veut dire que le service `UserService` est injectable dans n'importe quel composant

Et que c'est un service singleton

C'est à dire que toutes les instances de `UserService` sont la même instance.

Revenir au sommaire

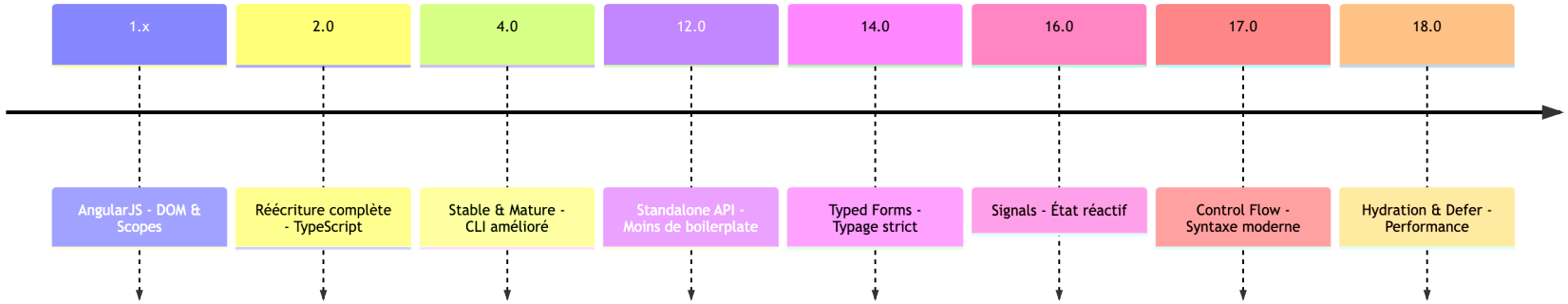
Évolution d'Angular

De AngularJS à Angular Moderne

- **AngularJS (1.x)**
  - Basé sur le DOM et les scopes
  - Directives comme composants
  - JavaScript vanilla
- **Angular 2+ : La révolution**
  - Réécriture complète en TypeScript
  - Architecture orientée composants
  - Performance améliorée
  - Injection de dépendances repensée

Revenir au sommaire

Historique d'évolution de Angular en timeline



Revenir au sommaire

Évolutions majeures

Angular 12-14 : Simplification

- Introduction des Standalone Components
- Suppression progressive des NgModules
- Amélioration du CLI
- Formulaires typés

Angular 15-16 : Réactivité

- Signals pour la gestion d'état
- Meilleure détection des changements
- SSR amélioré
- Hydration intelligente

Angular 17-18 : Modernisation

- Nouveau Control Flow (@if, @for)
- Defer Loading intégré
- Build system avec Vite & ESBuild
- Developer experience améliorée

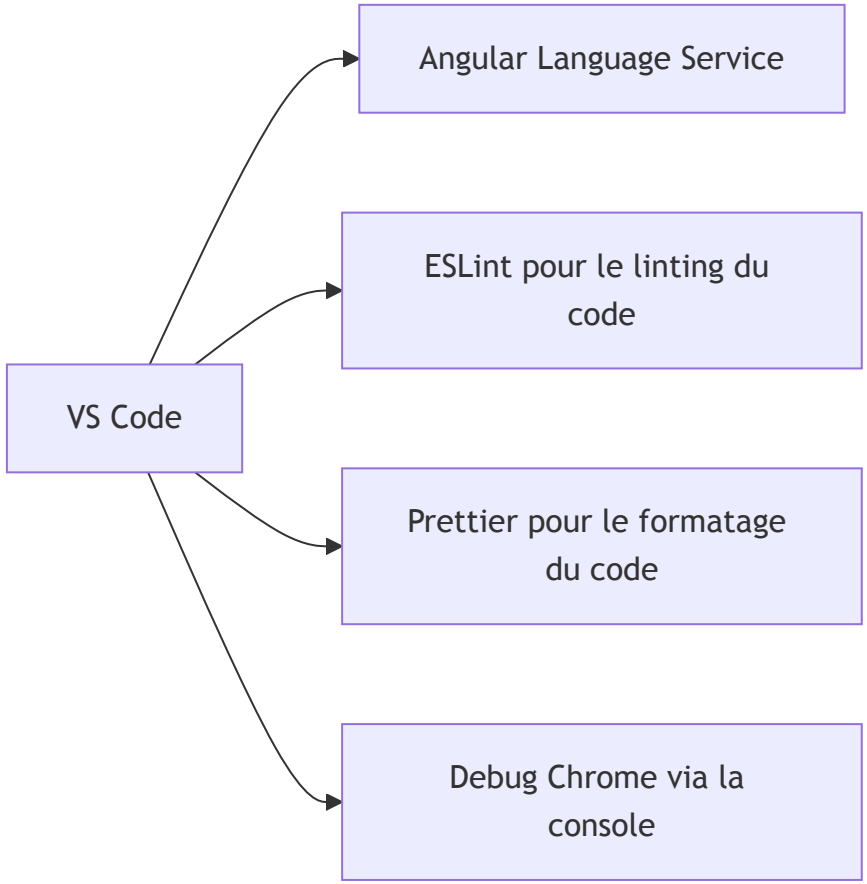
Revenir au sommaire

Comparaison avec d'autres frameworks

Caractéristique	Angular	React	Vue
Architecture	Full-framework	Bibliothèque	Progressive
Courbe d'apprentissage	Plus raide	Modérée	Douce
Tooling	Complet	Flexible	Intermédiaire
TypeScript	Natif	Optionnel	Optionnel

Revenir au sommaire

Outils essentiels



Revenir au sommaire

Extensions indispensables

- Angular Language Service pour la complétion de code
- Angular Snippets pour les snippets de code
- ESLint pour le linting du code
- Prettier pour le formatage du code

Revenir au sommaire

Prérequis techniques

Pour bien démarrer avec Angular, vous devez connaître :

✅ Fondamentaux

- HTML/CSS
- JavaScript moderne
- TypeScript basique
- Programmation orientée objet

❌ Pas nécessaire

- Backend development
- Mobile development
- WebAssembly

Revenir au sommaire

TypeScript Essentiels

Revenir au sommaire

Types de base

```
 // Types primitifs
let name: string = 'John';
let age: number = 25;
let isActive: boolean = true;

// Arrays
let numbers: number[] = [1, 2, 3];
let names: Array<string> = ['John', 'Jane'];

// Tuple
let tuple: [string, number] = ['John', 25];
```

Revenir au sommaire

Interfaces et Types

```typescript
 // Interface
interface User {
  id: number;
  name: string;
  email?: string; // Propriété optionnelle
}

// Type
type UserRole = 'admin' | 'user' | 'guest';

// Utilisation
const user: User = {
  id: 1,
  name: 'John',
  email: 'john@example.com'
```

Revenir au sommaire

Décorateurs TypeScript

```typescript
// Décorateur de classe
function Logger(target: any) {
  console.log('Class decorated:', target);
}

// Décorateur de propriété
function Required(target: any, propertyKey: string) {
  console.log('Property decorated:', propertyKey);
}

@Logger
class Example {
  @Required
  name: string;
```

Revenir au sommaire

Generics

```
 // Fonction générique
function getFirst<T>(array: T[]): T {
  return array[0];
}

// Classe générique
class DataContainer<T> {
  private data: T;

  constructor(data: T) {
    this.data = data;
  }

  getData(): T {
    return this.data;
```

Revenir au sommaire

Utility Types

```typescript
 interface Todo {
  title: string;
  description: string;
  completed: boolean;
}

// Partial - Toutes les propriétés deviennent optionnelles
type PartialTodo = Partial<Todo>;

// Pick - Sélectionne certaines propriétés
type TodoPreview = Pick<Todo, 'title' | 'completed'>;

// Omit - Omet certaines propriétés
type TodoWithoutDescription = Omit<Todo, 'description'>;
```

Revenir au sommaire

Configuration de l'environnement

Revenir au sommaire

Prérequis

```
 # Installation de Node.js via nvm (recommandé)
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.0/install.sh | bash
nvm install --lts

# Installation du CLI Angular
npm install -g @angular/cli@latest
```

Revenir au sommaire

Création d'un projet Angular moderne

```
 # Création d'un nouveau projet
ng new my-app

# Options recommandées
✔ Would you like to add routing? Yes
✔ Which stylesheet format would you like to use? SCSS
✔ Would you like to enable Server-Side Rendering (SSR)? No
```

standalone : pour utiliser les composants sans NgModules par défaut donc pas besoin de préciser

Revenir au sommaire

Structure du projet moderne

```
my-app/
├── src/
│   ├── app/
│   │   ├── components/
│   │   ├── services/
│   │   ├── app.config.ts
│   │   ├── app.routes.ts
│   │   └── app.component.ts
│   ├── assets/
│   └── main.ts
├── package.json
└── angular.json
```

Revenir au sommaire

Analyse des scripts NPM utiles

```
// package.json
{
  "scripts": {
    "start": "ng serve",
    "build": "ng build",
    "build:ssr": "ng build && ng run my-app:server",
    "dev:ssr": "ng run my-app:serve-ssr",
    "lint": "ng lint",
    "test": "ng test",
    "e2e": "ng e2e"
  }
}
```

Revenir au sommaire

Les Bases d'Angular

Revenir au sommaire

Architecture fondamentale

- **Structure d'un projet Angular**
  - Le fichier angular.json
  - Les dossiers src/, app/, assets/
  - Les fichiers de configuration
- **Concepts clés**
  - Modules (NgModule)
  - Composants
  - Services
  - Directives
  - Pipes

Revenir au sommaire

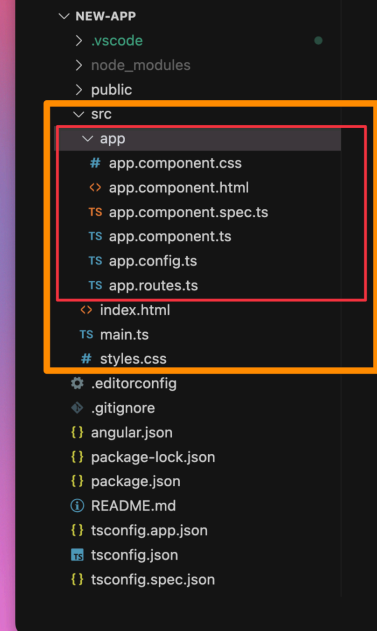
Commandes CLI Angular essentielles

```
 # Création d'un nouveau projet
ng new mon-projet

# Lancer le serveur de développement
ng serve

# Générer un composant
ng generate component mon-composant
# ou version courte
ng g c mon-composant

# Générer un service
ng generate service mon-service

# Générer une directive
```

Revenir au sommaire

SRC : Votre dossier source de votre projet

app : votre app , avec votre composant par défaut la config et les routes



tout les fichiers en dehors du "marquage" sont à la racine du projet

Revenir au sommaire

Comprendre le bootstrapping

```
 // main.ts
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app/app.component';

bootstrapApplication(AppComponent, {
  providers: [
    // Configuration globale
  ]
}).catch(err => console.error(err));
```

Revenir au sommaire

Initialisation de l'application

C'est ici que l'on initialise l'application Angular. Si nous voulons par exemple utiliser un router ou utiliser provideHttpClient, nous pouvons le faire ici.

Je vous ai mis en évidence les lignes 3 et 4 qui sont les plus importantes.

```
bootstrapApplication(AppComponent, {
  providers: [
    provideRouter(routes),
    provideHttpClient()
  ]
}).catch(err => console.error(err));
```

Revenir au sommaire

Structure d'un composant de base

Nous allons voir un exemple de composant de base.

Ne vous inquiétez pas , nous allons revenir dessus dans les slides suivantes.

Revenir au sommaire

```
 // hello.component.ts
@Component({
  selector: 'app-hello',
  // le selector est le nom de la balise html qui va être utilisée pour utiliser le composant
  standalone: true,
  // standalone: true pour utiliser le composant sans NgModule, le composant vis en autonomie
  template: `
    <h1>Hello {{ name }}</h1>
    <button (click)="sayHello()">
      Click me
    </button>
  `
  // template: ` pour le template du composant , en clair la vue :)
})
export class HelloComponent {
```

Revenir au sommaire

Data Binding fondamental

- **One-way binding (Liaison à sens unique)**

Interpolation: `{{ expression }}`

- Affiche des données du composant dans le template
- Exemple: `{{ user.name }}` affiche le nom de l'utilisateur
- Supporte les expressions simples: `{{ 1 + 1 }}` , `{{ user.firstName + ' ' + user.lastName }}`

Revenir au sommaire

Exemple d'interpolation

```
@Component({
  template: `
    <!-- Interpolation simple -->
    <h1>Bienvenue {{ userName }}</h1>

    <!-- Expressions mathématiques -->
    <p>Total: {{ price * quantity + tax }}</p>

    <!-- Expressions conditionnelles -->
    <span>Status: {{ isActive ? 'Actif' : 'Inactif' }}</span>

    <!-- Méthodes -->
    <div>Message: {{ getMessage() }}</div>
  `
})
```

Revenir au sommaire

Les Composants dans Angular

Revenir au sommaire

Qu'est-ce qu'un composant ?

Un composant est comme une brique LEGO® de votre application :

- Une partie de l'interface utilisateur (UI)
- Autonome et réutilisable
- Avec sa propre logique et son propre template

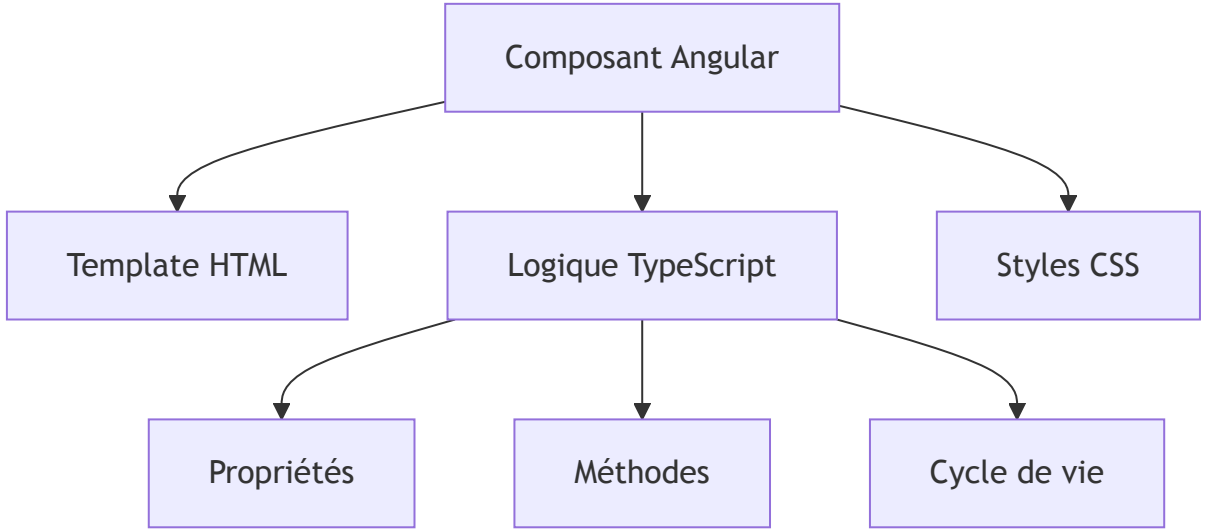
Revenir au sommaire

Création d'un composant avec le CLI

```
 # Générer un composant standalone
ng generate component features/user/user-profile
# ou version courte
ng g c features/user/user-profile

# Générer un composant avec des tests
ng g c features/user/user-profile --spec

# Générer un composant sans fichier de style
ng g c features/user/user-profile --inline-style

# Générer un composant avec template inline
ng g c features/user/user-profile --inline-template

# Générer un composant dans un sous-dossier
```

Revenir au sommaire

Structure d'un composant



Revenir au sommaire

Composants Standalone (Angular 18/19)

```
@Component({
  selector: 'app-user-card',
  standalone: true, // Plus besoin de NgModule !
  imports: [CommonModule],
  template: `...`
})
export class UserCardComponent {}
```

Revenir au sommaire

Le décorateur @Component - Partie 1

```
@Component({
  // Nom de la balise HTML pour utiliser ce composant
  selector: 'app-user-profile',

  // Template HTML du composant
  template: `
    <div class="user-profile">
      <h2>{{ userName }}</h2>
    </div>
  `
})
```

Revenir au sommaire

Le décorateur @Component - Partie 2

```
@Component({
  // Styles spécifiques au composant
  styles: [`
    .user-profile {
      padding: 20px;
      border: 1px solid #ccc;
    }
  `]
})
```

Revenir au sommaire

Styles des composants

```
@Component({
  selector: 'app-styled-button',
  template: `
    <button class="custom-btn">
      <ng-content></ng-content>
    </button>
  `,
  styles: [`
    .custom-btn {
      padding: 10px 20px;
      border-radius: 4px;
      border: none;
      background: #007bff;
      color: white;
      cursor: pointer;
```

Revenir au sommaire

Projection de Contenu

Revenir au sommaire

Qu'est-ce que la projection ?

La projection de contenu permet :

- D'injecter du contenu dans un composant enfant
- De créer des composants réutilisables et flexibles
- De définir des "slots" pour le contenu
- De gérer du contenu dynamique

Revenir au sommaire

ng-content : Projection simple

```
 // card.component.ts
@Component({
  selector: 'app-card',
  template: `
    <div class="card">
      <ng-content></ng-content>
    </div>
  `
})
export class CardComponent {}

// Utilisation
<app-card>
  <h2>Mon titre</h2>
  <p>Mon contenu</p>
```

Revenir au sommaire

ng-content : Projection multiple

```
@Component({
  selector: 'app-layout',
  template: `
    <header>
      <ng-content select="[header]"></ng-content>
    </header>
    <main>
      <ng-content select="[content]"></ng-content>
    </main>
    <footer>
      <ng-content select="[footer]"></ng-content>
    </footer>
  `
})
export class LayoutComponent {}
```

Revenir au sommaire

ng-template : Contenu conditionnel

```
@Component({
 selector: 'app-conditional',
 template: `
   <div>
     <ng-template #loading>
       <p>Chargement en cours...</p>
     </ng-template>

     <ng-template #error>
       <p>Une erreur est survenue</p>
     </ng-template>

     @if (data(); as result) {
       <div>{{ result }}</div>
     } @else if (isLoading()) {
```

Revenir au sommaire

ng-template : Templates réutilisables

```
@Component({
  selector: 'app-list',
  template: `
    <ul>
      @for (item of items(); track item.id) {
        <ng-container
          [ngTemplateOutlet]="itemTemplate"
          [ngTemplateOutletContext]="{ $implicit: item }"
        ></ng-container>
      }
    </ul>
  `
})
export class ListComponent {
  @Input() items = signal<any[]>([]);
```

Revenir au sommaire

ng-container : Groupement logique

```
@Component({
 template: `
   <div class="container">
     <ng-container *ngIf="isAdmin()">
       <button>Éditer</button>
       <button>Supprimer</button>
       <button>Configurer</button>
     </ng-container>

     <ng-container [ngSwitch]="userRole()">
       <div *ngSwitchCase="'admin'">Panel Admin</div>
       <div *ngSwitchCase="'user'">Vue Utilisateur</div>
       <div *ngSwitchDefault>Accès Limité</div>
     </ng-container>
   </div>
```

Revenir au sommaire

Projection de contenu - Structure

```
@Component({
  selector: 'app-card',
  template: `
    <div class="card">
      <div class="header">
        <ng-content select="[header]"></ng-content>
      </div>
      <div class="content">
        <ng-content></ng-content>
      </div>
    </div>
  `
})
```

Revenir au sommaire

Projection de contenu - Utilisation

```
<app-card>
  <h2 header>Mon titre</h2>
  <p>Contenu principal</p>
  <button footer>Action</button>
</app-card>
```

Revenir au sommaire

Bonnes pratiques - À faire ✅

- Un composant = une responsabilité unique
- Garder les composants petits et focalisés
- Utiliser des interfaces pour typer les inputs
- Documenter les inputs/outputs importants

Revenir au sommaire

Bonnes pratiques - À éviter ❌

- Trop de logique dans les templates
- Composants trop complexes
- Duplication de code entre composants
- Couplage fort entre composants

Revenir au sommaire

Communication entre Composants

Revenir au sommaire

Communication Parent → Enfant

Entrées (@Input)

```
@Component({
  selector: 'app-user-card',
  template: `
    <div class="card">
      <h3>{{ userName }}</h3>
      <p>{{ userRole }}</p>
    </div>
  `
})
export class UserCardComponent {
  @Input() userName: string;
  @Input() userRole: string;
}
```

Revenir au sommaire

Utilisation des @Input

```
<app-user-card
  userName="John Doe"
  userRole="Admin"
/>
```

Revenir au sommaire

Communication Enfant → Parent

Sorties (@Output)

```
@Component({
  selector: 'app-counter',
  template: `
    <div>
      <h2>{{ count() }}</h2>
      <button (click)="increment()">+</button>
    </div>
  `
})
export class CounterComponent {
  count = signal(0);
  @Output() countChange = new EventEmitter<number>();

  increment() {
    this.count.update(n => n + 1);
```

Revenir au sommaire

Gestion des événements @Output

Utilisation :

```
<app-counter
  (countChange)="handleCountChange($event)"
/>
```

Revenir au sommaire

Exercice : Composants du Blog

Revenir au sommaire

1. Créez le composant PostList :

```
 // features/posts/post-list.component.ts
@Component({
  selector: 'app-post-list',
  standalone: true,
  template: `
    <div class="posts">
      @for (post of posts; track post.id) {
        <article class="post-card">
          <h2>{{ post.title }}</h2>
          <p>{{ post.excerpt }}</p>
        </article>
      }
    </div>
  `,
  styles: [`
```

Revenir au sommaire

Exercice : Création des composants du Mini-Blog

Structure des composants

```
 // Liste des articles
@Component({
  selector: 'app-post-list',
  standalone: true,
  template: `
    <div class="posts-grid">
      @for (post of posts; track post.id) {
        <app-post-card [post]="post" />
      }
    </div>
  `
})
export class PostListComponent {
  posts = signal<Post[]>([]);
}
```

Revenir au sommaire

Composant de formulaire

```
@Component({
  selector: 'app-post-form',
  standalone: true,
  template: `
    <form class="post-form" (ngSubmit)="onSubmit()">
      <input [(ngModel)]="title" placeholder="Titre" />
      <textarea [(ngModel)]="content" placeholder="Contenu"></textarea>
      <button type="submit">Publier</button>
    </form>
  `
})
export class PostFormComponent {
  @Output() submitted = new EventEmitter<Post>();
}
```

Revenir au sommaire

Dashboard Admin

```
@Component({
  selector: 'app-admin-dashboard',
  standalone: true,
  template: `
    <div class="admin-dashboard">
      <h2>Tableau de bord</h2>
      <div class="stats">
        <div class="stat-card">
          <h3>Articles</h3>
          <p>{{ postCount() }}</p>
        </div>
        <div class="stat-card">
          <h3>Commentaires</h3>
          <p>{{ commentCount() }}</p>
        </div>
```

Revenir au sommaire

Mini-Blog : Création des Composants de Base

Revenir au sommaire

Objectif

Créer la structure de base de notre mini-blog avec les composants essentiels.

Composants à créer

```
 # Génération des composants
ng generate component features/blog/post-list
ng generate component features/blog/post-detail
ng generate component features/blog/post-form
ng generate component features/blog/post-card
ng generate component shared/layout/header
ng generate component shared/layout/footer
```

Revenir au sommaire

Structure des composants

```
 // post-list.component.ts
@Component({
  selector: 'app-post-list',
  template: `
    <div class="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-4">
      @for (post of posts; track post.id) {
        <app-post-card [post]="post" />
      }
    </div>
  `
})
export class PostListComponent {
  posts: Post[] = [
    { id: 1, title: 'Premier article', content: 'Contenu...', author: 'John' },
    { id: 2, title: 'Deuxième article', content: 'Contenu...', author: 'Jane' }
```

Revenir au sommaire

Interface Post

```ts
// models/post.interface.ts
export interface Post {
  id: number
  title: string
  content: string
  author: string
  createdAt?: Date
  imageUrl?: string
}
```

Revenir au sommaire

Composant Card

```
 // post-card.component.ts
@Component({
  selector: 'app-post-card',
  template: `
    <article class="border rounded-lg p-4 hover:shadow-lg transition">
      <h2 class="text-xl font-bold">{{ post.title }}</h2>
      <p class="text-gray-600">Par {{ post.author }}</p>
      <p class="mt-2">{{ post.content | slice:0:100 }}...</p>
      <button class="mt-4 px-4 py-2 bg-blue-500 text-white rounded">
        Lire la suite
      </button>
    </article>
  `
})
export class PostCardComponent {
```

Revenir au sommaire

Cette première étape nous permet de mettre en place la structure de base de notre blog avec des données statiques. Dans les prochains modules, nous ajouterons la navigation, les services et la gestion d'état.

Revenir au sommaire

Tailwind v4 dans Angular

Revenir au sommaire

Installation de Tailwind v4

```
 # Installation des dépendances
npm install -D tailwindcss@latest postcss@latest autoprefixer@latest

# Génération du fichier de configuration Tailwind
npx tailwindcss init
```

Revenir au sommaire

Configuration de Tailwind

Plus obligatoire dans la v4, vous pouvez le supprimer ou continuer à l'utilise avec @config dans votre fichier css/scss.

```
 // tailwind.config.js
/** @type {import('tailwindcss').Config} */
module.exports = {
  content: [
    "./src/**/*.{html,ts}",
    "./src/**/*.{js,jsx,ts,tsx}",
  ],
  theme: {
    extend: {
      // Vos personnalisations ici
    },
  },
  plugins: [],
}
```

Revenir au sommaire

Configuration de PostCSS

```
// .postcssrc.json
{
    "plugins": {
      "@tailwindcss/postcss": {}
    }
  }
```

Revenir au sommaire

Intégration avec Angular

Dans la v4

```
@import "tailwindcss";
```

```
@use "tailwindcss";
```

Avant dans la v3

```
/* src/styles.css */
@tailwind base;
@tailwind components;
@tailwind utilities;
```

Revenir au sommaire

Utilisation dans les composants

```
@Component({
  selector: 'app-header',
  // vous pouvez utiliser les classes tailwind dans le template
  template: `
    <header class="bg-gray-800 text-white p-4">
      <nav class="container mx-auto flex items-center justify-between">
        <h1 class="text-2xl font-bold">Mon App</h1>
        <ul class="flex space-x-4">
          <li><a href="#" class="hover:text-blue-400">Accueil</a></li>
          <li><a href="#" class="hover:text-blue-400">À propos</a></li>
          <li><a href="#" class="hover:text-blue-400">Contact</a></li>
        </ul>
      </nav>
    </header>
  `
```

Revenir au sommaire

Fonctionnalités avancées de Tailwind v4

- Support natif des variables CSS
- Nouveau système de couleurs
- Améliorations des performances
- Meilleure compatibilité avec Angular

Revenir au sommaire

Bonnes pratiques avec Tailwind

- Utiliser les composants pour réutiliser les styles
- Créer des classes utilitaires personnalisées
- Optimiser la taille du bundle final
- Suivre les conventions de nommage

Revenir au sommaire

Syntaxe des Templates

Revenir au sommaire

Interpolation et expressions

Nous avons déjà vu rapidement il y a quelques instants l'interpolation avec la syntaxe `{{ expression }}` mais nous allons revoir des cas concrets différents.

Revenir au sommaire

```
@Component({
  template: `
    <!-- Interpolation basique -->
    <h1>{{ title }}</h1>

    <!-- Expressions -->
    <p>Total: {{ price * quantity }}</p>

    <!-- Méthodes -->
    <div>{{ getMessage() }}</div>

    <!-- Chaînage de propriétés -->
    <span>{{ user?.address?.city }}</span>
  `
})
```

Revenir au sommaire

Bindings de propriétés et d'événements

```
@Component({
  template: `
    <!-- Property binding -->
    <img [src]="imageUrl" [alt]="imageAlt">
    <button [disabled]="isDisabled">Click me</button>

    <!-- Event binding -->
    <button (click)="handleClick($event)">
      Click count: {{ clickCount }}
    </button>

    <!-- Two-way binding -->
    <input [(ngModel)]="userName">
    <p>Hello, {{ userName }}!</p>
  `
```

Revenir au sommaire

Directives structurelles modernes

```
@Component({
  template: `
    <!-- If moderne -->
    @if (isLoggedIn()) {
      <nav>Menu utilisateur</nav>
    } @else {
      <auth-form />
    }

    <!-- For moderne -->
    @for (item of items(); track item.id) {
      <item-card [data]="item" />
    }

    <!-- Switch moderne -->
```

Revenir au sommaire

Pipes et formatage

```
@Component({
  template: `
    <!-- Pipes de base -->
    <p>{{ date | date:'shortDate' }}</p>
    <p>{{ price | currency:'EUR' }}</p>
    <p>{{ text | uppercase }}</p>

    <!-- Chaînage de pipes -->
    <p>{{ data | json | async }}</p>

    <!-- Pipes avec paramètres -->
    <p>{{ number | number:'1.0-2' }}</p>

    <!-- Pipe personnalisé -->
    <p>{{ text | highlight:searchTerm }}</p>
```

Revenir au sommaire

Références de template et variables

```
@Component({
  template: `
    <!-- Référence locale -->
    <input #nameInput type="text">
    <button (click)="greet(nameInput.value)">
      Saluer
    </button>

    <!-- ViewChild -->
    <div #content>
      Contenu dynamique
    </div>

    <!-- Variables de template -->
    @for (item of items; track item.id; let i = $index) {
```

Revenir au sommaire

Exercice : Template du Post

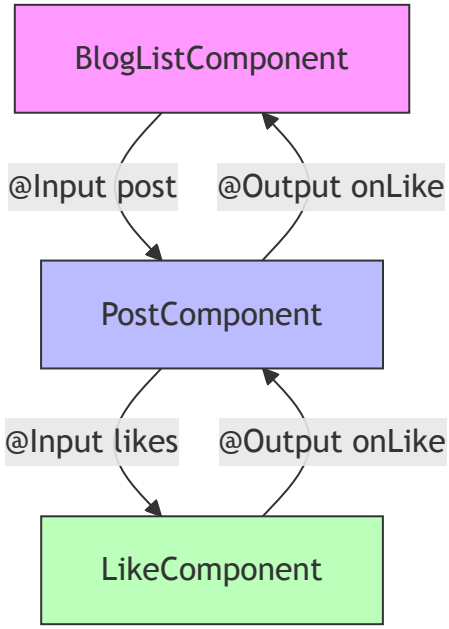
Vous allez devoir modifier le template de votre composant article pour qu'il affiche les informations du post.

Faire un bouton like qui incrémente le nombre de likes.

Créer un composant like qui affiche un bouton like et un compteur de likes. (donc il reçoit le nombre de likes en entrée et affiche un bouton like et un compteur de likes)

Revenir au sommaire

Communication entre composants



Revenir au sommaire

Flux des données

- ⬇️ **@Input** : Les données descendent du parent vers l'enfant
  - `BlogList` → `Post` : passe l'objet post complet
  - `Post` → `Like` : passe le nombre de likes
- ⬆️ **@Output** : Les événements remontent de l'enfant vers le parent
  - `Like` → `Post` : émet quand le bouton est cliqué
  - `Post` → `BlogList` : relaie l'événement avec l'ID du post

Revenir au sommaire

Solution étape par étape

Revenir au sommaire

2. Composant Like (enfant)

```
 // components/like.component.ts
@Component({
  selector: 'app-like',
  standalone: true,
  template: `
    <div class="flex items-center gap-2">
      <button
        (click)="handleLike()"
        class="px-4 py-2 bg-blue-500 text-white rounded-lg hover:bg-blue-600"
      >
        Like
      </button>
      <span>{{ likes }} likes</span>
    </div>
  `
```

Revenir au sommaire

3. Composant Post (parent du Like)

```
 // components/post.component.ts
@Component({
  selector: 'app-post',
  standalone: true,
  imports: [LikeComponent],
  template: `
    <article class="p-4 border rounded-lg shadow-sm">
      <h2 class="text-2xl font-bold">{{ post.title }}</h2>
      <p class="mt-2">{{ post.content }}</p>
      <p class="text-sm text-gray-500">
        Publié le {{ post.publishedAt | date:'dd/MM/yyyy' }}
      </p>
      <app-like
        [likes]="post.likes"
        (onLike)="handleLike()"
```

Revenir au sommaire

4. Composant Blog List (parent du Post)

```
 // components/blog-list.component.ts
@Component({
  selector: 'app-blog-list',
  standalone: true,
  imports: [PostComponent],
  template: `
    <div class="grid gap-4">
      @for (post of posts(); track post.id) {
        <app-post
          [post]="post"
          (onLike)="handlePostLike($event)"
        />
      }
    </div>
  `
```

Revenir au sommaire

Flux de communication entre composants

1. LikeComponent (enfant le plus bas)

- **Entrée** : Reçoit `@Input() likes`
- **Sortie** : Émet `@Output() onLike` vers PostComponent

2. PostComponent (composant intermédiaire)

- **Entrée** : Reçoit `@Input() post`
- **Sortie** : Émet `@Output() onLike` vers BlogListComponent
- **Rôle** : Reçoit l'événement de LikeComponent et le relaie au parent

3. BlogListComponent (parent)

- **État** : Contient le state (signal `posts`)
- **Gestion** : Reçoit l'événement du PostComponent et met à jour le state

Revenir au sommaire

Cette solution montre de manière progressive :

1. La définition des types
2. La création du composant le plus petit (Like)
3. Le composant intermédiaire (Post)
4. Le composant parent qui gère l'état (BlogList)

Points clés :

- Communication parent-enfant avec `@Input()` et `@Output()`
- Gestion d'état avec Signals
- Composants standalone
- Typage fort
- Utilisation des pipes
- Styling avec Tailwind

Control Flow Moderne (Angular 18/19)

Nouvelle Syntaxe @if

Revenir au sommaire

Routing et Navigation dans Angular

Revenir au sommaire

Qu'est-ce que le Routing ?

Le routing dans Angular permet de :

- Gérer la navigation entre les pages
- Créer des applications Single Page (SPA)
- Protéger les routes avec des guards
- Charger les modules de manière paresseuse (lazy loading)

Revenir au sommaire

Configuration de base

```
 // app.routes.ts
export const routes: Routes = [
  {
    path: '',
    component: HomeComponent
  },
  {
    path: 'products',
    component: ProductListComponent
  },
  {
    path: 'products/:id',
    component: ProductDetailComponent
  },
  {
```

Revenir au sommaire

Navigation dans les templates

```
@Component({
  template: `
    <!-- Navigation déclarative -->
    <nav>
      <a routerLink="/">Accueil</a>
      <a routerLink="/products">Produits</a>
      <a [routerLink]="['/products', product.id]">
        Détail Produit
      </a>
    </nav>

    <!-- Outlet pour afficher les composants -->
    <router-outlet></router-outlet>
  `
})
```

Revenir au sommaire

Navigation programmatique

```
@Component({
  template: `
    <button (click)="goToProduct(123)">
      Voir Produit
    </button>
  `
})
export class NavComponent {
  private router = inject(Router);

  goToProduct(id: number) {
    this.router.navigate(['/products', id], {
      queryParams: { source: 'nav' }
    });
  }
```

Revenir au sommaire

Lazy Loading

```
 // Configuration des routes
export const routes: Routes = [
  {
    path: 'admin',
    loadChildren: () => import('./admin/routes')
  },
  {
    path: 'products',
    loadComponent: () => import('./products/product-list.component')
  }
];

// Configuration avec preloading
bootstrapApplication(AppComponent, {
  providers: [
```

Revenir au sommaire

Protection des routes avec Guards

```
 // auth.guard.ts
export const authGuard: CanActivateFn = (route, state) => {
  const authService = inject(AuthService);
  const router = inject(Router);

  if (authService.isAuthenticated()) {
    return true;
  }

  return router.createUrlTree(['/login'], {
    queryParams: { returnUrl: state.url }
  });
};

// Utilisation
```

Revenir au sommaire

Paramètres de route avec Signals

```
@Component({
  template: `
    <h1>Produit {{ productId() }}</h1>
    @if (product()) {
      <div>
        <h2>{{ product().name }}</h2>
        <p>{{ product().description }}</p>
      </div>
    }
  `
})
export class ProductDetailComponent {
  private route = inject(ActivatedRoute);
  private productService = inject(ProductService);
```

Revenir au sommaire

Routes avec Resolvers

```
// routes.ts
export default [{
  path: 'products',
  component: ProductListComponent,
  resolve: {
    products: () => inject(ProductService).getProducts()
  }
}, {
  path: 'products/:id',
  component: ProductDetailComponent,
  resolve: {
    product: (route: ActivatedRoute) => {
      const id = route.paramMap.pipe(map(params => params.get('id')));
      return inject(ProductService).getProduct(id);
    }
```

Revenir au sommaire

Nested Routes avec Outlets multiples

```
 // Configuration
export const routes: Routes = [{
  path: 'dashboard',
  component: DashboardComponent,
  children: [{
    path: '',
    component: DashboardOverviewComponent
  }, {
    path: 'stats',
    component: StatsComponent,
    outlet: 'sidebar'
  }]
}];

// Template
```

Revenir au sommaire

Commandes CLI utiles

```
 # Générer un nouveau module avec routing
ng generate module features/admin --routing

# Générer un guard de route
ng generate guard core/guards/auth --implements CanActivate

# Générer un resolver
ng generate resolver features/product/product-data

# Générer une route lazy-loadée
ng generate module features/dashboard --route dashboard --module app.routes.ts

# Générer un composant avec route
ng generate component features/profile --route profile --type page
```

Revenir au sommaire

Mini-Blog : Configuration du Routing

Revenir au sommaire

Configuration des routes

```
// app.routes.ts
export const routes: Routes = [
  {
    path: '',
    component: HomeComponent,
    title: 'Accueil'
  },
  {
    path: 'blog',
    children: [
      {
        path: '',
        component: PostListComponent,
        title: 'Blog'
      },
```

Revenir au sommaire

Guard d'authentification

```
// guards/auth.guard.ts
export const authGuard: CanActivateFn = (route, state) => {
  const authService = inject(AuthService)

  if (authService.isAuthenticated()) {
    return true
  }

  return inject(Router).createUrlTree(['/login'], {
    queryParams: { returnUrl: state.url }
  })
}
```

Revenir au sommaire

Navigation dans les composants

```
 // post-card.component.ts (composant enfant)
@Component({
  template: `
    <article class="border rounded-lg p-4">
      <h2>{{ post.title }}</h2>
      <button (click)="goToDetail()">Lire la suite</button>
    </article>
  `
})
export class PostCardComponent {
  @Input() post!: Post
  private router = inject(Router)

  goToDetail() {
    this.router.navigate(['/blog', this.post.id])
```

Revenir au sommaire

Menu de navigation

```
 // header.component.ts (composant parent)
@Component({
  template: `
    <nav class="bg-gray-800 text-white p-4">
      <div class="container mx-auto flex justify-between items-center">
        <a routerLink="/" class="text-xl font-bold">Mini-Blog</a>

        <div class="flex gap-4">
          <a routerLink="/blog"
             routerLinkActive="text-blue-400"
             class="hover:text-blue-300">
            Articles
          </a>

          @if (isAuthenticated()) {
```

Revenir au sommaire

Mini-Blog : Protection des Routes

Revenir au sommaire

Configuration des Guards

```
 // guards/auth.guard.ts
export const authGuard: CanActivateFn = (route, state) => {
  const router = inject(Router);
  const authService = inject(AuthService);

  if (authService.isAuthenticated()) {
    return true;
  }

  // Stocker l'URL tentée pour rediriger après login
  return router.createUrlTree(['/login'], {
    queryParams: { returnUrl: state.url }
  });
};
```

Revenir au sommaire

Application des Guards

```
// app.routes.ts
export const routes: Routes = [
  { path: '', component: BlogListComponent },
  { path: 'login', component: LoginComponent },
  {
    path: 'admin',
    canActivate: [authGuard, adminGuard],
    children: [
      { path: 'posts/new', component: PostFormComponent },
      { path: 'posts/:id/edit', component: PostFormComponent }
    ]
  },
  {
    path: 'posts',
    children: [
```

Revenir au sommaire

Gestion de la Redirection

```
 // components/login.component.ts
@Component({
  selector: 'app-login',
  standalone: true,
  template: `
    <form [formGroup]="loginForm" (ngSubmit)="onSubmit()">
      <!-- ... form fields ... -->
    </form>
  `
})
export class LoginComponent {
  private router = inject(Router);
  private route = inject(ActivatedRoute);
  private authService = inject(AuthService);
```

Revenir au sommaire

Cette configuration :

- Protège les routes sensibles avec `authGuard`
- Ajoute une protection supplémentaire pour les admins
- Gère la redirection après login
- Conserve l'URL tentée pour y retourner après authentification

Revenir au sommaire

Cycle de Vie des Composants

Revenir au sommaire

Hooks essentiels

- **ngOnInit**: Après la première initialisation des propriétés
- **ngOnDestroy**: Juste avant que le composant soit détruit
- **ngOnChanges**: Quand une propriété liée aux données change
- **ngAfterViewInit**: Après l'initialisation de la vue

Revenir au sommaire

Exemple d'implémentation

```
@Component({
  selector: 'app-lifecycle',
  template: `
    <h1>{{ title }}</h1>
    <p>{{ message }}</p>
  `
})
export class LifecycleComponent implements OnInit, OnDestroy {
  @Input() title: string;
  message: string;

  ngOnInit() {
    console.log('Component initialized');
    this.message = 'Component is ready!';
  }
```

Revenir au sommaire

Détection des changements d'Input

```
@Component({
  selector: 'app-child',
  template: `
    <div>Data: {{ data }}</div>
  `
})
export class ChildComponent implements OnChanges {
  @Input() data: any;

  ngOnChanges(changes: SimpleChanges) {
    if (changes['data']) {
      console.log(
        'Previous:', changes['data'].previousValue,
        'Current:', changes['data'].currentValue
      );
```

Revenir au sommaire

AfterView et AfterContent

```
 @Component({
  selector: 'app-view-child',
  template: `
    <div #contentDiv>
      <ng-content></ng-content>
    </div>
  `
})
export class ViewChildComponent implements AfterViewInit {
  @ViewChild('contentDiv') contentDiv: ElementRef;

  ngAfterViewInit() {
    console.log('View initialized:', this.contentDiv.nativeElement);
  }
}
```

Revenir au sommaire

Bonnes pratiques

1. **Initialisation**
   - Utiliser `ngOnInit` pour l'initialisation des données
   - Éviter les opérations lourdes dans le constructeur
2. **Nettoyage**
   - Toujours implémenter `ngOnDestroy` pour nettoyer les souscriptions
   - Libérer les ressources (timers, listeners, etc.)
3. **Performance**
   - Éviter les calculs lourds dans `ngOnChanges`
   - Utiliser `ChangeDetectionStrategy.OnPush` quand possible

Revenir au sommaire

Services Angular

Revenir au sommaire

Qu'est-ce qu'un service ?

Un service est une classe qui encapsule la logique métier et les données partagées :

- Réutilisable dans toute l'application
- Suit le principe de responsabilité unique
- Facilite la maintenance et les tests
- Permet la séparation des préoccupations

Revenir au sommaire

Création d'un service avec le CLI

```
 # Générer un service global (root)
ng generate service services/user
# ou version courte
ng g s services/user

# Générer un service dans un feature module
ng g s features/auth/services/auth

# Générer un service avec des tests
ng g s services/user --spec

# Générer un service sans créer un dossier
ng g s services/user --flat

# Générer un service avec une interface
```

Revenir au sommaire

Service classique avec Observables

```
 @Injectable({
   providedIn: 'root'
})
export class UserService {
   private userSubject = new BehaviorSubject<User | null>(null);
   // userSubject est un Observable (new BehaviorSubject donc il peut être observé) de type User | null
   // user$ est un Observable de type User | null
   user$ = this.userSubject.asObservable();

   constructor(private http: HttpClient) {}
   // on injecte le http client

   // on récupère un user par son id dans la fonction getUser
   getUser(id: number): Observable<User> {
     // appel à l'api pour récupérer un user par son id
```

Revenir au sommaire

Utilisation d'un service classique

```
@Component({
  selector: 'app-user-profile',
  template: `
    <div *ngIf="user$ | async as user">
      <h2>{{ user.name }}</h2>
      <button (click)="updateName()">Modifier le nom</button>
    </div>
  `
})
export class UserProfileComponent {
  user$ = this.userService.user$;

  constructor(private userService: UserService) {
    this.userService.getUser(1).subscribe();
  }
```

Revenir au sommaire

Services avec Signals

Revenir au sommaire

Service moderne avec Signals

```
 @Injectable({
   providedIn: 'root'
})
export class UserService {
   private users = signal<User[]>([]);
   private loading = signal(false);
   private error = signal<Error | null>(null);

   // Computed values
   readonly sortedUsers = computed(() =>
     [...this.users()].sort((a, b) => a.name.localeCompare(b.name))
   );

   constructor(private http: HttpClient) {}
```

Revenir au sommaire

Service avec état partagé

```
 @Injectable({
   providedIn: 'root'
})
export class ThemeService {
   private darkMode = signal(false);

   // API publique en lecture seule
   readonly isDarkMode = this.darkMode.asReadonly();

   // Computed values
   readonly theme = computed(() =>
     this.darkMode() ? 'dark' : 'light'
   );

   toggleTheme() {
```

Revenir au sommaire

Service avec gestion d'état avancée

```
 interface AppState {
  user: User | null;
  preferences: UserPreferences;
  notifications: Notification[];
}

@Injectable({
  providedIn: 'root'
})
export class StateService {
  private state = signal<AppState>({
    user: null,
    preferences: defaultPreferences,
    notifications: []
  });
```

Revenir au sommaire

Service avec effets

```
 @Injectable({
  providedIn: 'root'
})
export class AuthService {
  private user = signal<User | null>(null);

  constructor() {
    // Effet pour synchroniser avec le localStorage
    effect(() => {
      const currentUser = this.user();
      if (currentUser) {
        localStorage.setItem('user', JSON.stringify(currentUser));
      } else {
        localStorage.removeItem('user');
      }
```

Revenir au sommaire

Mini-Blog : Service d'Authentification

Revenir au sommaire

Service de Base

```
 // services/auth.service.ts
@Injectable({ providedIn: 'root' })
export class AuthService {
  private readonly storageKey = 'blog_auth';
  private isAuthenticatedSignal = signal(false);
  private userSignal = signal<User | null>(null);

  // Getters publics en lecture seule
  readonly isAuthenticated = this.isAuthenticatedSignal.asReadonly();
  readonly currentUser = this.userSignal.asReadonly();

  constructor() {
    // Restaurer l'état d'authentification au démarrage
    this.checkAuthState();
  }
```

Revenir au sommaire

Utilisation dans un Composant

```
 // components/header.component.ts
@Component({
  selector: 'app-header',
  standalone: true,
  template: `
    <header class="bg-white shadow">
      <nav class="container mx-auto px-4 py-3">
        <div class="flex justify-between items-center">
          <a routerLink="/" class="text-xl font-bold">Mini Blog</a>

          <div class="flex items-center gap-4">
            @if (isAuthenticated()) {
              <span>{{ currentUser()?.name }}</span>
              <button
                (click)="logout()"
```

Revenir au sommaire

Cette implémentation :

- Utilise les Signals pour l'état d'authentification
- Persiste l'état dans le localStorage
- Gère les rôles utilisateur (admin/user)
- Fournit des getters en lecture seule
- S'intègre avec le système de routing et les guards

Revenir au sommaire

Injection de Dépendances

Revenir au sommaire

Qu'est-ce que l'injection de dépendances ?

- Un pattern de conception logicielle fondamental
- Permet de gérer les dépendances entre composants
- Rend le code plus modulaire et testable

Avantages clés

- Découplage du code
- Réutilisabilité accrue
- Facilite les tests unitaires
- Maintenance simplifiée

Revenir au sommaire

Injection de base

```
 // Service injectable basique
@Injectable({
  providedIn: 'root'
})
export class UserService {
  constructor(private http: HttpClient) {}

  getUsers(): Observable<User[]> {
    return this.http.get<User[]>('/api/users');
  }
}

// Utilisation dans un composant
@Component({
  selector: 'app-users'
```

Revenir au sommaire

Commandes CLI pour l'Injection de Dépendances

```
 # Générer un service injectable global
ng generate service core/services/config --providedIn root

# Générer un service avec un scope spécifique
ng generate service features/admin/services/admin --flat

# Générer une interface pour un token
ng generate interface core/interfaces/config-token

# Générer un guard (protection de route)
ng generate guard core/guards/auth

# Générer un resolver
ng generate resolver features/product/product-data
```

Revenir au sommaire

Hiérarchie d'injection

```
 // Niveau composant
@Component({
  selector: 'app-feature',
  providers: [FeatureService] // Scope limité à ce composant
})
export class FeatureComponent {
  constructor(private featureService: FeatureService) {}
}

// Niveau module
@NgModule({
  providers: [
    GlobalService,
    {
      provide: API_URL,
```

Revenir au sommaire

Configuration des providers

```
 // Token d'injection personnalisé
export interface AppConfig {
  apiUrl: string;
  theme: 'light' | 'dark';
}

export const APP_CONFIG = new InjectionToken<AppConfig>('app.config');

// Configuration des providers
export const appConfig: ApplicationConfig = {
  providers: [
    {
      provide: APP_CONFIG,
      useValue: {
        apiUrl: 'https://api.example.com',
```

Revenir au sommaire

Services avec dépendances avancées

```
 @Injectable({
   providedIn: 'root'
})
export class CacheService {
   private cache = new Map<string, any>();

   constructor(
     @Optional() @SkipSelf() parentCache: CacheService,
     @Inject(CACHE_SIZE) private maxSize: number
   ) {
     if (parentCache) {
       this.cache = new Map(parentCache.cache);
     }
   }
```

Revenir au sommaire

Injection Moderne avec inject()

Revenir au sommaire

Introduction à inject()

```
 // Approche classique
@Component({
  template: `...`
})
class OldComponent {
  constructor(
    private userService: UserService,
    private router: Router,
    @Inject(APP_CONFIG) private config: AppConfig
  ) {}
}

// Approche moderne avec inject()
@Component({
  template: `...`
```

Revenir au sommaire

Avantages de inject()

- Plus concis et lisible
- Utilisable en dehors du constructor
- Parfait pour les composants standalone
- Meilleure inférence de type
- Facilite l'utilisation avec les Signals

```
@Component({
  standalone: true,
  template: `
    @if (user()) {
      <h1>Bienvenue {{ user().name }}</h1>
    }
  `
})
class WelcomeComponent {
  private auth = inject(AuthService)
  user = this.auth.currentUser

  logout() {
    inject(Router).navigate(['/login'])
  }
```

Revenir au sommaire

Options avancées avec inject()

```
 @Component({
   template: `...`
})
class FeatureComponent {
   // Injection optionnelle
   private logger = inject(LoggerService, { optional: true })

   // Injection avec fallback
   private analytics = inject(AnalyticsService, {
     optional: true,
     self: true
   }) ?? new NoopAnalyticsService()

   // Injection au niveau parent
   private parentCache = inject(CacheService, {
```

Revenir au sommaire

Injection dans les services modernes

```
 @Injectable({ providedIn: 'root' })
class ModernService {
  // Injection directe sans constructor
  private http = inject(HttpClient)
  private config = inject(APP_CONFIG)

  // Computed basé sur injection
  private apiUrl = computed(() =>
    `${this.config.baseUrl}/api`
  )

  getData() {
    return this.http.get(`${this.apiUrl()}/data`)
  }
}
```

Revenir au sommaire

Exercice : Injection de Dépendances dans le Blog

Revenir au sommaire

Configuration de l'API

```
 // Configuration de l'API
export interface ApiConfig {
  baseUrl: string;
  version: string;
  timeout: number;
}

export const API_CONFIG = new InjectionToken<ApiConfig>('api.config');

// Provider global
export const appConfig: ApplicationConfig = {
  providers: [
    {
      provide: API_CONFIG,
      useValue: {
```

Revenir au sommaire

Service avec injection

```
 @Injectable({ providedIn: 'root' })
export class BlogApiService {
  private http = inject(HttpClient)
  private config = inject(API_CONFIG)
  private auth = inject(AuthService)

  private apiUrl = computed(() =>
    `${this.config.baseUrl}/${this.config.version}`
  )

  getPosts() {
    return this.http.get(`${this.apiUrl()}/posts`, {
      headers: {
        Authorization: `Bearer ${this.auth.getToken()}`
      }
```

Revenir au sommaire

Formulaires Angular

Revenir au sommaire

Commandes CLI pour les Formulaires

```
 # Générer un composant de formulaire réactif
ng generate component features/contact-form --type form

# Générer un validator personnalisé
ng generate validator shared/validators/password-strength

# Générer un formulaire avec validation
ng generate component features/registration --type form --spec

# Générer un formulaire avec états asynchrones
ng generate component features/signup --type form --signals

# Générer un service de validation
ng generate service shared/services/form-validation
```

Revenir au sommaire

Formulaires réactifs modernes avec Signals

```
@Component({
  selector: 'app-signup',
  template: `
    <form [formGroup]="form" (ngSubmit)="onSubmit()">
      <div>
        <label for="email">Email</label>
        <input id="email" type="email" formControlName="email">
        @if (emailErrors()) {
          <span class="error">{{ emailErrors() }}</span>
        }
      </div>

      <div>
        <label for="password">Mot de passe</label>
        <input id="password" type="password" formControlName="password">
```

Revenir au sommaire

Validation personnalisée

```
 // Validateur personnalisé
function passwordStrength(): ValidatorFn {
  return (control: AbstractControl): ValidationErrors | null => {
    const value = control.value;

    if (!value) return null;

    const hasNumber = /\d/.test(value);
    const hasUpper = /[A-Z]/.test(value);
    const hasLower = /[a-z]/.test(value);
    const hasSpecial = /[!@#$%^&*]/.test(value);

    const valid = hasNumber && hasUpper && hasLower && hasSpecial;

    return valid ? null : {
```

Revenir au sommaire

Formulaires dynamiques

```
 interface DynamicField {
  name: string;
  label: string;
  type: 'text' | 'email' | 'number';
  validators: ValidatorFn[];
}

@Component({
  template: `
    <form [formGroup]="form" (ngSubmit)="onSubmit()">
      @for (field of fields; track field.name) {
        <div>
          <label [for]="field.name">{{ field.label }}</label>
          <input
            [id]="field.name"
```

Revenir au sommaire

Formulaires imbriqués

```
@Component({
  template: `
    <form [formGroup]="form" (ngSubmit)="onSubmit()">
      <div formGroupName="personal">
        <input formControlName="firstName">
        <input formControlName="lastName">
      </div>

      <div formGroupName="address">
        <input formControlName="street">
        <input formControlName="city">
        <input formControlName="zipCode">
      </div>

      <div formArrayName="phones">
```

Revenir au sommaire

Exercice : Formulaire d'inscription complet

Créez un formulaire d'inscription avec validation avancée :

```
interface RegistrationForm {
  personal: {
    firstName: string;
    lastName: string;
    email: string;
  };
  credentials: {
    password: string;
    confirmPassword: string;
  };
  preferences: {
    newsletter: boolean;
    notifications: string[];
  };
}
```

Revenir au sommaire

Cet exercice vous à permis de pratiquer :

- Les formulaires imbriqués
- La validation personnalisée
- La validation croisée
- La gestion des FormArray
- Les états de chargement avec Signals

Revenir au sommaire

Exercice : Formulaire de Post

Revenir au sommaire

1. Structure du Template

```
 // features/posts/post-form.component.ts
@Component({
  selector: 'app-post-form',
  standalone: true,
  template: `
    <form [formGroup]="form" (ngSubmit)="onSubmit()">
      <div class="form-field">
        <label for="title">Titre</label>
        <input id="title" type="text" formControlName="title">
        @if (titleErrors()) {
          <span class="error">{{ titleErrors() }}</span>
        }
      </div>

      <div class="form-field">
```

Revenir au sommaire

2. Configuration du Formulaire

```
export class PostFormComponent {
  private postService = inject(PostService)
  private router = inject(Router)

  form = new FormGroup({
    title: new FormControl('', [
      Validators.required,
      Validators.minLength(3)
    ]),
    content: new FormControl('', [
      Validators.required,
      Validators.minLength(50)
    ])
  })
```

Revenir au sommaire

3. Gestion des Erreurs

```
 titleErrors = computed(() => {
   const control = this.form.get('title')
   if (control?.errors && control.touched) {
     if (control.errors['required']) return 'Le titre est requis'
     if (control.errors['minlength']) return 'Le titre doit faire au moins 3 caractères'
   }
   return null
 })

 contentErrors = computed(() => {
   const control = this.form.get('content')
   if (control?.errors && control.touched) {
     if (control.errors['required']) return 'Le contenu est requis'
     if (control.errors['minlength']) return 'Le contenu doit faire au moins 50 caractères'
   }
   return null
 })
```

Revenir au sommaire

4. Soumission du Formulaire

```
  async onSubmit() {
    if (this.form.valid) {
      this.saving.set(true)
      try {
        await firstValueFrom(this.postService.createPost({
          ...this.form.value,
          author: 'Utilisateur actuel',
          date: new Date(),
          excerpt: this.form.value.content?.slice(0, 100) + '...'
        }))
        this.router.navigate(['/posts'])
      } finally {
        this.saving.set(false)
      }
    }
  }
}
```

Revenir au sommaire

Exercice : Formulaires du Mini-Blog

Formulaire de connexion (Template-driven)

```
@Component({
  selector: 'app-login-form',
  template: `
    <form #loginForm="ngForm" (ngSubmit)="onSubmit(loginForm)">
      <div class="form-group">
        <label for="email">Email</label>
        <input
          type="email"
          id="email"
          name="email"
          [(ngModel)]="loginData.email"
          required
          email
          #email="ngModel"
        >
```

Revenir au sommaire

Formulaire d'article (Reactive Form)

```
@Component({
  selector: 'app-post-form',
  template: `
    <form [formGroup]="postForm" (ngSubmit)="onSubmit()">
      <div class="form-group">
        <label for="title">Titre</label>
        <input
          type="text"
          id="title"
          formControlName="title"
        >
        @if (titleErrors()) {
          <span class="error">{{ titleErrors() }}</span>
        }
      </div>
```

Revenir au sommaire

Mini-Blog : Formulaires Réactifs

Revenir au sommaire

Formulaire de connexion

```
 // login.component.ts
@Component({
  template: `
    <div class="max-w-md mx-auto mt-8 p-6 bg-white rounded-lg shadow-lg">
      <h2 class="text-2xl font-bold mb-6">Connexion</h2>

      <form [formGroup]="loginForm" (ngSubmit)="onSubmit()">
        <div class="mb-4">
          <label class="block text-gray-700 mb-2" for="email">
            Email
          </label>
          <input
            id="email"
            type="email"
            formControlName="email"
```

Revenir au sommaire

Formulaire de création d'article

```
// post-form.component.ts
@Component({
  template: `
    <div class="max-w-2xl mx-auto mt-8 p-6 bg-white rounded-lg shadow-lg">
      <h2 class="text-2xl font-bold mb-6">
        {{ isEditing ? 'Modifier l\'article' : 'Nouvel article' }}
      </h2>

      <form [formGroup]="postForm" (ngSubmit)="onSubmit()">
        <div class="mb-4">
          <label class="block text-gray-700 mb-2" for="title">
            Titre
          </label>
          <input
            id="title"
            type="text"
            formControlName="title"
            class="w-full px-3 py-2 border rounded"
            [class.border-red-500]="title.invalid && title.touched"
          >
          @if (title.invalid && title.touched) {
            <p class="text-red-500 text-sm mt-1">
              Le titre est requis et doit faire au moins 3 caractères
            </p>
```

Revenir au sommaire

Signals

Revenir au sommaire

Commandes CLI pour les Signals

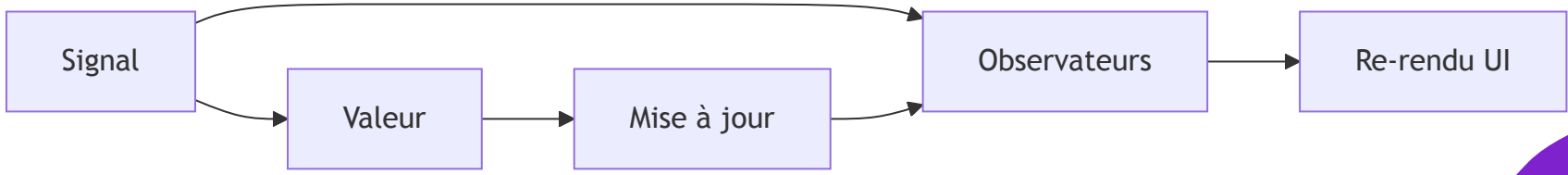
```
 # Générer un composant avec support des signals
ng generate component features/counter --signals

# Générer un service avec signals
ng generate service shared/services/state --signals

# Générer un store avec signals
ng generate store features/todo/todo-store --signals

# Générer un composant avec état signal
ng generate component features/user-profile --signals --state

# Générer un service d'état global
ng generate service core/services/app-state --signals --global
```

Revenir au sommaire

Introduction aux Signals

Un Signal est comme une "boîte réactive" qui :

- Contient une valeur
- Notifie automatiquement quand cette valeur change
- Permet un suivi précis des dépendances



Revenir au sommaire

Comparaison avec les variables classiques

Imaginons un thermomètre :

Sans Signal :

```
 // La température change mais personne n'est notifié
class Thermometer {
  temperature = 20;

  update() {
    this.temperature++; // L'affichage ne sait pas qu'il doit se mettre à jour
  }
}
```

Avec Signal :

```
 class Thermometer {
  temperature = signal(20);

  update() {
    this.temperature.update(t => t + 1); // Notification automatique !
  }
}
```

Revenir au sommaire

Les différents types de Signals

1. Signal Writeable

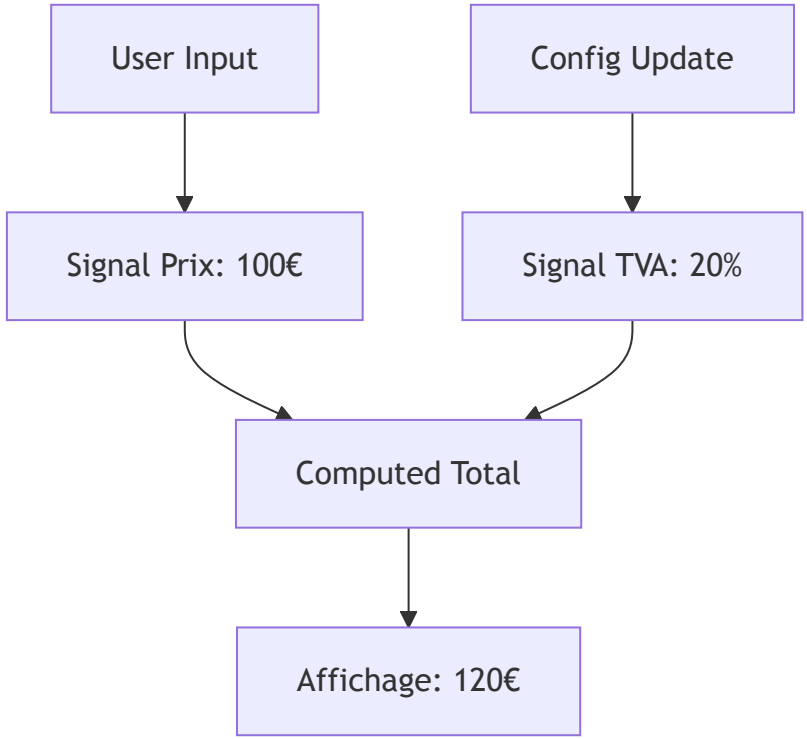
```
 // Comme une variable qu'on peut lire et modifier
const counter = signal(0);

// Lecture
console.log(counter()); // 0

// Écriture
counter.set(5);         // Remplacement direct
counter.update(n => n + 1); // Mise à jour basée sur valeur précédente
```

2. Signal Computed

```
 // Comme une formule Excel qui se recalcule automatiquement
const price = signal(100);
const taxRate = signal(0.2);

const total = computed(() => {
  const basePrice = price();
  const tax = basePrice * taxRate();
  return basePrice + tax;
});
```

Revenir au sommaire

Visualisation du flux de données



Revenir au sommaire

Exemple concret : Panier d'achat

```
@Component({
 template: `
   <div class="cart-summary">
     <h2>Votre panier</h2>

     <!-- Affichage réactif des totaux -->
     <div class="totals">
       <p>Sous-total: {{ subtotal() }}€</p>
       <p>TVA ({{ vatRate() * 100 }}%): {{ vatAmount() }}€</p>
       <p class="grand-total">Total: {{ total() }}€</p>
     </div>

     <!-- La mise à jour d'un produit recalcule automatiquement tous les totaux -->
     @for (item of cartItems(); track item.id) {
       <cart-item
```

Revenir au sommaire

Bonnes pratiques avec les Signals

✅ À faire

- Utiliser computed() pour les valeurs dérivées
- Préférer .update() à .set() pour les modifications basées sur l'état précédent
- Garder les signals privés quand possible

❌ À éviter

- Créer des signals dans des boucles
- Appeler des signals dans des setters
- Modifier plusieurs signals de manière non atomique

Computed Signals

```
 const count = signal(0);
const doubled = computed(() => count() * 2);
const isEven = computed(() => count() % 2 === 0);
```

Revenir au sommaire

Effects

```
@Component({
  template: `
    <button (click)="increment()">
      Compteur: {{ count() }}
    </button>
  `
})
export class CounterComponent {
  count = signal(0);

  constructor() {
    // L'effet se déclenche à chaque changement de count
    effect(() => {
      console.log(`Nouvelle valeur: ${this.count()}`);
      localStorage.setItem('count', this.count().toString());
```

Revenir au sommaire

Signals avec objets et tableaux

```
 interface User {
  id: number;
  name: string;
  email: string;
}

@Component({
  template: `
    <div>
      <h2>{{ user().name }}</h2>
      @for (friend of friends(); track friend.id) {
        <div>{{ friend.name }}</div>
      }
    </div>
  `
```

Revenir au sommaire

Signals avec async data

```
@Component({
  template: `
    @if (loading()) {
      <spinner />
    } @else if (error()) {
      <error-message [message]="error()" />
    } @else if (data(); as users) {
      @for (user of users; track user.id) {
        <user-card [user]="user" />
      }
    }
  `
})
export class UsersComponent {
  private userService = inject(UserService);
```

Revenir au sommaire

Signal Inputs

```
 @Component({
  selector: 'app-user-profile',
  template: `
    <div>
      <h2>{{ name() }}</h2>
      <p>Age: {{ age() }}</p>
      <p>Score: {{ score() }}</p>
    </div>
  `
})
export class UserProfileComponent {
  name = input.required<string>();
  age = input<number>(18); // Valeur par défaut
  score = input<number>(); // Optionnel
}
```

Revenir au sommaire

Signals avec formulaires

```
@Component({
  template: `
    <form [formGroup]="form" (ngSubmit)="onSubmit()">
      <input formControlName="name">
      <input formControlName="email">

      <div>Form Value: {{ formValue() | json }}</div>
      <div>Valid: {{ isValid() }}</div>

      <button type="submit" [disabled]="!isValid()">
        Envoyer
      </button>
    </form>
  `
})
```

Revenir au sommaire

Gestion de l'État avec Signals

Avant les Signals

```
@Component({
  template: `
    <h1>{{ count }}</h1>
    <button (click)="increment()">+1</button>
  `
})
class CounterComponent {
  count = 0

  increment() {
    this.count++ // Déclenche détection globale
  }
}
```

Revenir au sommaire

Avec les Signals

```
@Component({
  template: `
    <h1>{{ count() }}</h1>
    <button (click)="increment()">+1</button>
  `
})
class CounterComponent {
  count = signal(0)

  increment() {
    this.count.update(n => n + 1) // Mise à jour granulaire
  }
}
```

Revenir au sommaire

Exercice : État Global du Blog

Revenir au sommaire

1. Créez le service d'état :

```
 // core/state/blog.state.ts
interface BlogState {
  posts: Post[]
  selectedPost: Post | null
  filters: {
    search: string
    category: string
  }
  loading: boolean
  error: string | null
}

@Injectable({ providedIn: 'root' })
export class BlogStateService {
  // État privé
```

Revenir au sommaire

2. Utilisez l'état dans un composant :

```
@Component({
  selector: 'app-post-list',
  template: `
    <div class="filters">
      <input
        type="text"
        [ngModel]="searchTerm()"
        (ngModelChange)="onSearch($event)"
        placeholder="Rechercher..."
      >
      <select
        [ngModel]="category()"
        (ngModelChange)="onCategoryChange($event)"
      >
        <option value="all">Toutes catégories</option>
```

Revenir au sommaire

Exercice : Gestion d'État du Mini-Blog avec Signals

État des Posts

```
 // stores/post.store.ts
interface PostState {
  posts: Post[];
  selectedPost: Post | null;
  loading: boolean;
  error: Error | null;
}

@Injectable({ providedIn: 'root' })
export class PostStore {
  // État privé
  private state = signal<PostState>({
    posts: [],
    selectedPost: null,
    loading: false,
```

Revenir au sommaire

État de l'Authentification

```
 // stores/auth.store.ts
interface AuthState {
  user: User | null;
  token: string | null;
  loading: boolean;
  error: Error | null;
}

@Injectable({ providedIn: 'root' })
export class AuthStore {
  // État privé
  private state = signal<AuthState>({
    user: null,
    token: null,
    loading: false,
```

Revenir au sommaire

Mini-Blog : Gestion d'état avec Signals

Revenir au sommaire

Store des articles

```
 // stores/post.store.ts
@Injectable({ providedIn: 'root' })
export class PostStore {
  // État privé
  private _posts = signal<Post[]>([])
  private _selectedPost = signal<Post | null>(null)
  private _loading = signal(false)
  private _error = signal<string | null>(null)

  // Sélecteurs publics
  readonly posts = this._posts.asReadonly()
  readonly selectedPost = this._selectedPost.asReadonly()
  readonly loading = this._loading.asReadonly()
  readonly error = this._error.asReadonly()
```

Revenir au sommaire

Utilisation dans les composants

```
 // post-list.component.ts
@Component({
  template: `
    @if (loading()) {
      <div class="flex justify-center p-8">
        <span class="loading">Chargement...</span>
      </div>
    } @else if (error()) {
      <div class="text-red-500 p-4">
        {{ error() }}
      </div>
    } @else {
      <div class="mb-4">
        <h3 class="text-lg font-semibold">
          {{ postCount() }} articles publiés
```

Revenir au sommaire

Composant de détail avec effet

```
// post-detail.component.ts
@Component({
  template: `
    @if (loading()) {
      <div class="flex justify-center p-8">
        <span class="loading">Chargement...</span>
      </div>
    } @else if (post(); as post) {
      <article class="prose lg:prose-xl mx-auto">
        <h1>{{ post.title }}</h1>
        <p class="text-gray-600">
          Par {{ post.author }} le {{ post.createdAt | date }}
        </p>
        <div>{{ post.content }}</div>
      </article>
    } @else {
      <p>Article non trouvé</p>
    }
  `
})
export class PostDetailComponent {
  private store = inject(PostStore)
  private route = inject(ActivatedRoute)
```

Revenir au sommaire

RxJS et Observables

Revenir au sommaire

Concepts de base

Bon déjà c'est quoi un Observable ?

Un Observable est un objet qui représente une séquence de valeurs, émises à des moments différents.

Imaginons je fais un appel api , regardons du marble "testing" pour mieux comprendre :

```
const apiCall$ = new Observable(observer => {
  fetch('https://api.example.com/data')
    .then(response => response.json())
    .then(data => observer.next(data))
    .catch(error => observer.error(error))
    .finally(() => observer.complete())
})
```

Revenir au sommaire

Marble Testing

Donc ce flux : --1–2–3–4–5–

ce qui veut dire :

- 1 premiere étape : j'aurais donc une réponse de l'api
- 2 deuxième étape : j'aurais donc une autre réponse de l'api
- 3 troisième étape : j'aurais donc une autre réponse de l'api
- 4 quatrième étape : j'aurais donc une autre réponse de l'api
- 5 cinquième étape : j'aurais donc une autre réponse de l'api

Revenir au sommaire

Observer - Partie 1

Un observer est un objet qui écoute les événements d'un Observable.

```
 // Observable simple
const numbers$ = of(1, 2, 3, 4, 5)
```

Revenir au sommaire

Observer - Partie 2

```
 // Observer
numbers$.subscribe({
  next: value => console.log(value),
  error: err => console.error(err),
  complete: () => console.log('Terminé')
})
```

Revenir au sommaire

Types de Subjects - Partie 1

```
 // Subject basique
const subject = new Subject<string>()
subject.subscribe(value => console.log('A:', value))
subject.subscribe(value => console.log('B:', value))
subject.next('Hello') // A: Hello, B: Hello
```

Revenir au sommaire

Types de Subjects - Partie 2

```
 // BehaviorSubject - Garde la dernière valeur
const behavior = new BehaviorSubject<number>(0)
behavior.subscribe(value => console.log('Valeur actuelle:', value))
console.log('Valeur stockée:', behavior.value)
```

Revenir au sommaire

Types de Subjects - Partie 3

```
 // ReplaySubject - Rejoue X valeurs
const replay = new ReplaySubject<string>(2)
replay.next('Un')
replay.next('Deux')
replay.next('Trois')
replay.subscribe(value => console.log(value)) // Deux, Trois
```

Revenir au sommaire

Opérateurs essentiels - Filtrage - Partie 1

```
 const source$ = interval(1000)

// filter - Garde uniquement les valeurs qui passent le prédicat
source$.pipe(
  filter(n => n % 2 === 0)
) // 0, 2, 4, 6...
```

Revenir au sommaire

Opérateurs essentiels - Filtrage - Partie 2

```
 // take - Prend X valeurs puis complète
source$.pipe(
  take(3)
) // 0, 1, 2, complete

// takeUntil - Émet jusqu'à ce qu'un autre Observable émette
const stop$ = timer(5000)
source$.pipe(
  takeUntil(stop$)
) // Émet pendant 5 secondes
```

Revenir au sommaire

Opérateurs essentiels - Filtrage - Partie 3

```
 // distinctUntilChanged - Ignore les doublons consécutifs
of(1, 1, 2, 2, 3, 3).pipe(
  distinctUntilChanged()
) // 1, 2, 3
```

Revenir au sommaire

Opérateurs essentiels - Transformation - Partie 1

```
 // map - Transforme chaque valeur
source$.pipe(
  map(n => n * 2)
) // 0, 2, 4, 6...
```

Revenir au sommaire

Opérateurs essentiels - Transformation - Partie 2

```
 // switchMap - Annule l'Observable précédent
searchTerm$.pipe(
  switchMap(term =>
    this.http.get(`/api/search?q=${term}`)
  )
)
```

Revenir au sommaire

Opérateurs essentiels - Transformation - Partie 3

```
 // mergeMap - Fusionne tous les Observables
userIds$.pipe(
  mergeMap(id =>
    this.http.get(`/api/user/${id}`)
  )
)
```

Revenir au sommaire

Opérateurs essentiels - Transformation - Partie 4

```
 // concatMap - Attend que chaque Observable soit complété
uploads$.pipe(
  concatMap(file =>
    this.uploadFile(file)
  )
)
```

Revenir au sommaire

Opérateurs essentiels - Combinaison - Partie 1

```
 // combineLatest - Combine les dernières valeurs
combineLatest({
  user: userProfile$,
  preferences: userPrefs$,
  theme: themeSettings$
}).subscribe(({ user, preferences, theme }) => {
  console.log('État complet:', { user, preferences, theme })
})
```

Revenir au sommaire

Opérateurs essentiels - Combinaison - Partie 2

```
 // merge - Fusionne plusieurs Observables
merge(
  clicks$,
  keypresses$,
  touches$
).subscribe(event => {
  console.log('Interaction utilisateur:', event)
})
```

Revenir au sommaire

Opérateurs essentiels - Combinaison - Partie 3

```
 // forkJoin - Attend que tous les Observables soient complétés
forkJoin({
  posts: this.http.get('/api/posts'),
  comments: this.http.get('/api/comments'),
  users: this.http.get('/api/users')
}).subscribe(data => {
  console.log('Toutes les données chargées:', data)
})
```

Revenir au sommaire

Cas d'utilisation concrets - Partie 1

```
@Component({
  template: `
    <input [ngModel]="searchTerm()"
           (ngModelChange)="searchTerm$.next($event)">
    <div *ngFor="let result of results()">
      {{ result.name }}
    </div>
  `
})
```

Revenir au sommaire

Cas d'utilisation concrets - Partie 2

```
class SearchComponent {
 private searchTerm$ = new BehaviorSubject<string>('')

 results = toSignal(
   this.searchTerm$.pipe(
     debounceTime(300),
     distinctUntilChanged(),
     filter(term => term.length >= 2),
     switchMap(term => this.searchService.search(term)),
     catchError(err => {
       console.error('Erreur de recherche:', err)
       return of([])
     })
   ),
   { initialValue: [] }
```

Revenir au sommaire

Gestion des websockets - Partie 1

```
class WebSocketService {
  private wsSubject = new BehaviorSubject<WebSocket | null>(null)
  private messagesSubject = new Subject<any>()
  private reconnectAttempts = 0

  connect() {
    const ws = new WebSocket('ws://api.example.com')

    ws.addEventListener('message', event => {
      this.messagesSubject.next(JSON.parse(event.data))
    })

    ws.addEventListener('close', () => {
      this.reconnect()
    })
```

Revenir au sommaire

Gestion des websockets - Partie 2

```
class WebSocketService {
  messages$ = this.messagesSubject.asObservable().pipe(
    retry(3),
    share()
  )

  private reconnect() {
    if (this.reconnectAttempts < 3) {
      this.reconnectAttempts++
      timer(1000 * this.reconnectAttempts).pipe(
        take(1)
      ).subscribe(() => this.connect())
    }
  }
}
```

Revenir au sommaire

Intégration avec Signals - Partie 1

```
@Component({
  template: `
    <div>Données: {{ data() }}</div>
    <div>Statut: {{ status() }}</div>
  `
})
```

Revenir au sommaire

Intégration avec Signals - Partie 2

```
export class DataComponent {
  private dataService = inject(DataService)

  // Conversion d'Observable en Signal
  data = toSignal(
    this.dataService.getData().pipe(
      catchError(error => {
        console.error('Erreur:', error)
        return of(null)
      })
    ),
    { initialValue: null }
  )

  // Signal computed basé sur Observable
```

Revenir au sommaire

Opérateurs RxJS modernes - Partie 1

```
 @Injectable({
   providedIn: 'root'
})
export class SearchService {
   private http = inject(HttpClient)

   search(term: string): Observable<Result[]> {
     return of(term).pipe(
       // Attendre que l'utilisateur arrête de taper
       debounceTime(300),

       // Ignorer si le terme est le même
       distinctUntilChanged()
     )
   }
```

Revenir au sommaire

Opérateurs RxJS modernes - Partie 2

```
export class SearchService {
  search(term: string): Observable<Result[]> {
    return of(term).pipe(
      // Annuler la requête précédente
      switchMap(term =>
        this.http.get<Result[]>(`/api/search?q=${term}`).pipe(
          // Gérer les erreurs par requête
          catchError(error => {
            console.error('Erreur de recherche:', error)
            return of([])
          })
        )
      )
    )
  }
```

Revenir au sommaire

Gestion des souscriptions - Partie 1

```
@Component({
  template: `
    <input [ngModel]="searchTerm()"
           (ngModelChange)="searchTerm.set($event)">

    @if (results(); as items) {
      @for (item of items; track item.id) {
        <div>{{ item.name }}</div>
      }
    }
  `
})
```

Revenir au sommaire

Gestion des souscriptions - Partie 2

```
export class SearchComponent implements OnDestroy {
  private searchService = inject(SearchService)
  private destroy$ = new Subject<void>()

  searchTerm = signal('')

  // Conversion du signal en Observable
  private searchTerm$ = toObservable(this.searchTerm)

  results = toSignal(
    this.searchTerm$.pipe(
      debounceTime(300),
      distinctUntilChanged(),
      switchMap(term => this.searchService.search(term)),
      takeUntil(this.destroy$)
```

Revenir au sommaire

Combinaison d'Observables - Partie 1

```
@Component({
  template: `
    <div>
      Utilisateur: {{ userData()?.name }}
      Préférences: {{ userData()?.preferences }}
      Notifications: {{ userData()?.notifications }}
    </div>
  `
})
```

Revenir au sommaire

Combinaison d'Observables - Partie 2

```
export class UserDashboardComponent {
  private userService = inject(UserService)
  private prefService = inject(PreferencesService)
  private notifService = inject(NotificationService)

  userData = toSignal(
    combineLatest({
      user: this.userService.getCurrentUser(),
      preferences: this.prefService.getPreferences(),
      notifications: this.notifService.getNotifications()
    }).pipe(
      map(({ user, preferences, notifications }) => ({
        name: user.name,
        preferences,
        notifications: notifications.length
```

Revenir au sommaire

Gestion des erreurs avancée - Partie 1

```
 @Injectable({
   providedIn: 'root'
})
export class ErrorHandlingService {
   private errorSubject = new Subject<Error>()

   error$ = this.errorSubject.asObservable().pipe(
     // Grouper les erreurs similaires
     groupBy(error => error.message),
     // Limiter les notifications
     mergeMap(group => group.pipe(
       debounceTime(1000),
       take(3)
     ))
   )
```

Revenir au sommaire

Gestion des erreurs avancée - Partie 2

```
@Component({
  template: `
    @if (error()) {
      <div class="error-banner">
        {{ error() }}
      </div>
    }
  `
})
export class ErrorComponent {
  private errorService = inject(ErrorHandlingService)

  error = toSignal(
    this.errorService.error$.pipe(
      map(error => error.message)
```

Revenir au sommaire

WebSocket avec RxJS - Partie 1

```
 @Injectable({
   providedIn: 'root'
})
export class WebSocketService {
   private socket$ = new WebSocket('ws://example.com')
   private messagesSubject = new Subject<any>()

   messages$ = this.messagesSubject.asObservable().pipe(
     // Reconnexion automatique
     retryWhen(errors => errors.pipe(
       tap(error => console.error('WebSocket error:', error)),
       delay(1000)
     ))
   )
}
```

Revenir au sommaire

WebSocket avec RxJS - Partie 2

```
export class WebSocketService {
  messages$ = this.messagesSubject.asObservable().pipe(
    // Filtrage et transformation des messages
    filter(msg => msg.type === 'data'),
    map(msg => msg.payload),
    share()
  )

  constructor() {
    this.socket$.addEventListener('message', (event) => {
      this.messagesSubject.next(JSON.parse(event.data))
    })
  }

  send(message: any) {
```

Revenir au sommaire

Exercice : Recherche en temps réel pour le Mini-Blog

Revenir au sommaire

Composant de recherche - Partie 1

```
@Component({
  selector: 'app-post-search',
  template: `
    <div class="search-container">
      <input
        type="text"
        [ngModel]="searchTerm()"
        (ngModelChange)="searchTerm$.next($event)"
        placeholder="Rechercher des articles..."
        class="search-input"
      >

      @if (loading()) {
        <div class="loading-spinner">Chargement...</div>
      }
```

Revenir au sommaire

Composant de recherche - Partie 2

```
@Component({
  template: `
    <div class="search-results">
      @for (post of searchResults(); track post.id) {
        <div class="post-card">
          <h3>{{ post.title }}</h3>
          <p>{{ post.excerpt }}</p>
          <a [routerLink]="['/posts', post.id]">Lire la suite</a>
        </div>
      }
    </div>
  `
})
```

Revenir au sommaire

Composant de recherche - Partie 3

```
export class PostSearchComponent {
  private searchTerm$ = new BehaviorSubject<string>('')
  private loading = signal(false)
  private error = signal<string | null>(null)

  // État de la recherche avec Signals
  searchTerm = toSignal(this.searchTerm$)

  // Résultats de recherche
  searchResults = toSignal(
    this.searchTerm$.pipe(
      // Attendre que l'utilisateur arrête de taper
      debounceTime(300),
      // Éviter les recherches identiques
      distinctUntilChanged(),
```

Revenir au sommaire

Service de recherche - Partie 1

```
 @Injectable({ providedIn: 'root' })
export class PostService {
  searchPosts(term: string): Observable<Post[]> {
    return this.http.get<Post[]>('/api/posts/search', {
      params: { q: term }
    }).pipe(
      // Transformer les résultats
      map(posts => posts.map(post => ({
        ...post,
        excerpt: this.createExcerpt(post.content)
      })))
    )
  }
}
```

Revenir au sommaire

Service de recherche - Partie 2

```
export class PostService {
  private createExcerpt(content: string): string {
    return content.substring(0, 150) + '...'
  }

  private sortByRelevance(posts: Post[], term: string): Post[] {
    return [...posts].sort((a, b) => {
      const aScore = this.calculateRelevance(a, term)
      const bScore = this.calculateRelevance(b, term)
      return bScore - aScore
    })
  }

  private calculateRelevance(post: Post, term: string): number {
    const titleMatch = post.title.toLowerCase().includes(term.toLowerCase())
```

Revenir au sommaire

Filtres combinés - Partie 1

```
@Component({
  selector: 'app-post-filters',
  template: `
    <div class="filters">
      <select [ngModel]="category()" (ngModelChange)="category$.next($event)">
        <option value="">Toutes les catégories</option>
        @for (cat of categories(); track cat) {
          <option [value]="cat">{{ cat }}</option>
        }
      </select>

      <select [ngModel]="sortBy()" (ngModelChange)="sortBy$.next($event)">
        <option value="date">Plus récents</option>
        <option value="title">Alphabétique</option>
        <option value="popularity">Popularité</option>
```

Revenir au sommaire

Filtres combinés - Partie 2

```
@Component({
  template: `
    <select [ngModel]="author()" (ngModelChange)="author$.next($event)">
      <option value="">Tous les auteurs</option>
      @for (auth of authors(); track auth.id) {
        <option [value]="auth.id">{{ auth.name }}</option>
      }
    </select>
  `
})
export class PostFiltersComponent {
  // Sources des filtres
  private category$ = new BehaviorSubject<string>('')
  private sortBy$ = new BehaviorSubject<string>('date')
  private author$ = new BehaviorSubject<string>('')
```

Revenir au sommaire

Filtres combinés - Partie 3

```
export class PostFiltersComponent {
  // Données des filtres
  categories = computed(() =>
    [...new Set(this.postStore.posts().map(p => p.category))]
  )

  authors = computed(() =>
    [...new Set(this.postStore.posts().map(p => p.author))]
  )

  // Combinaison des filtres
  filteredPosts = toSignal(
    combineLatest({
      posts: this.postStore.posts$,
      category: this.category$,
```

Revenir au sommaire

Filtres combinés - Partie 4

```
export class PostFiltersComponent {
  filteredPosts = toSignal(
    combineLatest({
      posts: this.postStore.posts$,
      sortBy: this.sortBy$
    }).pipe(
      map(({ posts, sortBy }) => {
        // Tri
        return [...posts].sort((a, b) => {
          switch (sortBy) {
            case 'date':
              return new Date(b.date).getTime() - new Date(a.date).getTime()
            case 'title':
              return a.title.localeCompare(b.title)
            case 'popularity':
              return b.views - a.views
            default:
              return 0
          }
        })
      })
    ),
    { initialValue: [] }
  )
```

Revenir au sommaire

HTTP dans Angular

Revenir au sommaire

Commandes CLI pour HTTP - Partie 1

```
 # Générer un service HTTP
ng generate service core/services/api

# Générer un intercepteur HTTP
ng generate interceptor core/interceptors/auth

# Générer un service avec mock
ng generate service core/services/api --mock
```

Revenir au sommaire

Commandes CLI pour HTTP - Partie 2

```
 # Générer un intercepteur avec transformation
ng generate interceptor core/interceptors/transform --transform

# Générer un service HTTP avec cache
ng generate service core/services/cached-api --cache
```

Revenir au sommaire

Commandes CLI pour HTTP - Partie 3

```
 # Générer un intercepteur de gestion d'erreurs
ng generate interceptor core/interceptors/error-handler

# Générer un service avec retry policy
ng generate service core/services/resilient-api --retry

# Générer un service avec pagination
ng generate service features/products/product-api --paginated
```

Revenir au sommaire

HTTP et Communication

Revenir au sommaire

Configuration moderne du HttpClient

```
 // app.config.ts
export const appConfig: ApplicationConfig = {
  providers: [
    provideHttpClient(
      withInterceptors([authInterceptor]), // Intercepteurs pour les requêtes (jwt/token)
      withFetch(), // Nouvelle API Fetch
      withJsonpSupport() // Support JSONP
    )
  ]
}
```

Revenir au sommaire

Service HTTP avec Signals - Partie 1

```
 @Injectable({
   providedIn: 'root'
})
export class ApiService {
   private http = inject(HttpClient)
   private baseUrl = 'https://api.example.com'

   // États avec Signals
   private loading = signal(false)
   private error = signal<string | null>(null)

   // Getters publics en lecture seule
   readonly isLoading = this.loading.asReadonly()
   readonly currentError = this.error.asReadonly()
}
```

Revenir au sommaire

Service HTTP avec Signals - Partie 2

```
export class ApiService {
  getData<T>(endpoint: string) {
    this.loading.set(true)
    this.error.set(null)

    return this.http.get<T>(`${this.baseUrl}/${endpoint}`).pipe(
      catchError(err => {
        this.error.set(err.message)
        return EMPTY
      }),
      finalize(() => this.loading.set(false))
    )
  }
}
```

Revenir au sommaire

Intercepteurs modernes - Partie 1

```ts
// auth.interceptor.ts
export const authInterceptor: HttpInterceptorFn = (req, next) => {
  const authToken = localStorage.getItem('token')

  if (authToken) {
    const authReq = req.clone({
      headers: req.headers.set('Authorization', `Bearer ${authToken}`)
    })

    return next(authReq)
  }

  return next(req)
}
```

Revenir au sommaire

Intercepteurs modernes - Partie 2

```
export const authInterceptor: HttpInterceptorFn = (req, next) => {
  return next(req).pipe(
    catchError(error => {
      if (error.status === 401) {
        // Redirection vers login
        inject(Router).navigate(['/login'])
      }
      return throwError(() => error)
    })
  )
}
```

Revenir au sommaire

Gestion des erreurs HTTP - Partie 1

```
 @Injectable({
   providedIn: 'root'
})
export class ErrorHandlingService {
   handleHttpError(error: HttpErrorResponse): Observable<never> {
     let errorMessage = 'Une erreur est survenue'

     if (error.error instanceof ErrorEvent) {
       // Erreur côté client
       errorMessage = error.error.message
     }

     return throwError(() => new Error(errorMessage))
   }
}
```

Revenir au sommaire

Gestion des erreurs HTTP - Partie 2

```
export class ErrorHandlingService {
 handleHttpError(error: HttpErrorResponse): Observable<never> {
   let errorMessage = 'Une erreur est survenue'

   // Erreur côté serveur
   switch (error.status) {
     case 404:
       errorMessage = 'Ressource non trouvée'
       break
     case 403:
       errorMessage = 'Accès non autorisé'
       break
     case 500:
       errorMessage = 'Erreur serveur'
       break
```

Revenir au sommaire

Requêtes parallèles - Partie 1

```
@Component({
  template: `
    @if (data(); as result) {
      <div>
        <h2>Utilisateur: {{ result.user.name }}</h2>
        <p>Posts: {{ result.posts.length }}</p>
        <p>Commentaires: {{ result.comments.length }}</p>
      </div>
    }
  `
})
```

Revenir au sommaire

Requêtes parallèles - Partie 2

```
export class UserDataComponent {
 private http = inject(HttpClient)

 data = toSignal(
   forkJoin({
     user: this.http.get<User>('/api/user'),
     posts: this.http.get<Post[]>('/api/posts'),
     comments: this.http.get<Comment[]>('/api/comments')
   }).pipe(
     catchError(error => {
       console.error('Erreur de chargement:', error)
       return of(null)
     })
   ),
   { initialValue: null }
```

Revenir au sommaire

Upload de fichiers - Partie 1

```
 @Injectable({
   providedIn: 'root'
})
export class FileUploadService {
   private http = inject(HttpClient)

   uploadFile(file: File) {
     const formData = new FormData()
     formData.append('file', file)

     return this.http.post('/api/upload', formData, {
       reportProgress: true,
       observe: 'events'
     })
   }
```

Revenir au sommaire

Upload de fichiers - Partie 2

```
export class FileUploadService {
 uploadFile(file: File) {
   return this.http.post('/api/upload', formData, {
     reportProgress: true,
     observe: 'events'
   }).pipe(
     map(event => {
       switch (event.type) {
         case HttpEventType.UploadProgress:
           const progress = event.total
             ? Math.round(100 * event.loaded / event.total)
             : 0
           return { type: 'progress', progress }
         case HttpEventType.Response:
           return { type: 'complete', data: event.body }
```

Revenir au sommaire

Upload de fichiers - Partie 3

```
@Component({
  template: `
    <input type="file" (change)="onFileSelected($event)">
    @if (uploadProgress() > 0) {
      <progress [value]="uploadProgress()" max="100">
        {{ uploadProgress() }}%
      </progress>
    }
  `
})
export class UploadComponent {
  private uploadService = inject(FileUploadService)
  uploadProgress = signal(0)

  onFileSelected(event: Event) {
```

Revenir au sommaire

Cache HTTP avec Signals - Partie 1

```
 @Injectable({
  providedIn: 'root'
})
export class CacheService {
  private cache = signal<Map<string, any>>(new Map())
  private http = inject(HttpClient)

  getData<T>(url: string, options: { ttl?: number } = {}) {
    const cached = this.cache().get(url)

    if (cached && (!options.ttl || Date.now() - cached.timestamp < options.ttl)) {
      return of(cached.data)
    }

    return this.http.get<T>(url)
```

Revenir au sommaire

Cache HTTP avec Signals - Partie 2

```
export class CacheService {
 getData<T>(url: string, options: { ttl?: number } = {}) {
   return this.http.get<T>(url).pipe(
     tap(data => {
       this.cache.update(cache => {
         cache.set(url, {
           data,
           timestamp: Date.now()
         })
         return cache
       })
     })
   )
 }
```

Revenir au sommaire

Mini-Blog : Simulation d'API REST

Revenir au sommaire

Service API

```
 // services/api.service.ts
@Injectable({ providedIn: 'root' })
export class ApiService {
  private http = inject(HttpClient)
  private baseUrl = 'https://api.example.com' // À remplacer par votre API

  // Simulation de délai réseau
  private delay(ms: number) {
    return new Promise(resolve => setTimeout(resolve, ms))
  }

  // GET /api/posts
  getPosts(): Observable<Post[]> {
    return of(MOCK_POSTS).pipe(
      delay(500) // Simulation de latence réseau
```

Revenir au sommaire

Intercepteur d'authentification

```
// interceptors/auth.interceptor.ts
export const authInterceptor: HttpInterceptorFn = (req, next) => {
  const token = localStorage.getItem('token')

  if (token) {
    req = req.clone({
      setHeaders: {
        Authorization: `Bearer ${token}`
      }
    })
  }

  return next(req).pipe(
    catchError(error => {
      if (error.status === 401) {
```

Revenir au sommaire

Service d'authentification avec HTTP

```
 // services/auth.service.ts
@Injectable({ providedIn: 'root' })
export class AuthService {
  private api = inject(ApiService)
  private isAuthenticatedSubject = signal(false)

  readonly isAuthenticated = this.isAuthenticatedSubject.asReadonly()

  constructor() {
    // Vérifier le token au démarrage
    this.isAuthenticatedSubject.set(!!localStorage.getItem('token'))
  }

  login(email: string, password: string): Observable<boolean> {
    return this.api.login({ email, password }).pipe(
```

Revenir au sommaire

Store avec appels HTTP

```
 // stores/post.store.ts
@Injectable({ providedIn: 'root' })
export class PostStore {
  private api = inject(ApiService)

  // État
  private _posts = signal<Post[]>([])
  private _loading = signal(false)
  private _error = signal<string | null>(null)

  // Sélecteurs
  readonly posts = this._posts.asReadonly()
  readonly loading = this._loading.asReadonly()
  readonly error = this._error.asReadonly()
```

Revenir au sommaire

Cette implémentation simule une API REST complète pour notre mini-blog, avec :

- Gestion des requêtes HTTP
- Simulation de latence réseau
- Gestion des erreurs
- Intercepteur d’authentification
- Store centralisé avec état de chargement et erreurs

Revenir au sommaire

Directives

Revenir au sommaire

Qu'est ce qu'une directive ?

Une directive est un élément qui modifie le comportement ou la structure d'un élément HTML.

A vrai dire, depuis le début de la formation, vous en avez utilisé plusieurs :

- `*ngIf` maintenant @if
- `*ngFor` maintenant @for
- `*ngSwitch` maintenant @switch
- `*ngStyle` maintenant [ngStyle]
- `*ngClass` maintenant [ngClass]

Revenir au sommaire

Pipes

Mais aussi des pipes sont des directives :

- `date`
- `uppercase`
- `lowercase`
- `currency`
- `slice`
- `json` !!

Revenir au sommaire

Création de directives avec le CLI

```
 # Générer une directive standalone
ng generate directive shared/directives/highlight
# ou version courte
ng g d shared/directives/highlight

# Générer une directive avec des tests
ng g d shared/directives/highlight --spec

# Générer une directive sans créer un dossier
ng g d shared/directives/highlight --flat

# Générer une directive avec un préfixe personnalisé
ng g d shared/directives/highlight --prefix app

# Générer un pipe standalone
```

Revenir au sommaire

Types de directives

Il existe plusieurs types de directives :

- Directives d'attributs personnalisées
- Directives de structure
- Pipes

Mais vous en avez d'autres, des directives personnalisées, des pipes personnalisés, des directives avec injection de dépendances, etc. Ce que je sous entend par là, c'est aussi que vous pouvez en créer vous même.

Revenir au sommaire

Types de directives

```
 // Nouvelle syntaxe de control flow (Angular 18+)
@Component({
  template: `
    @if (isLoggedIn()) {
      <nav>Menu utilisateur</nav>
    } @else {
      <auth-form />
    }

    @for (item of items(); track item.id) {
      <item-card [data]="item" />
    }

    @switch (status()) {
      @case ('loading') {
```

Revenir au sommaire

Directives d'attributs personnalisées

```
 // highlight.directive.ts
@Directive({
  selector: '[appHighlight]',
  standalone: true
})
export class HighlightDirective {
  @Input('appHighlight') highlightColor = '';

  constructor(private el: ElementRef) {}

  @HostListener('mouseenter')
  onMouseEnter() {
    this.highlight(this.highlightColor || 'yellow');
  }
```

Revenir au sommaire

Utilisation des directives

```
@Component({
  selector: 'app-root',
  template: `
    <div appHighlight="lightblue">
      Survolez-moi !
    </div>

    <p [appHighlight]="color">
      Couleur dynamique
    </p>
  `,
  imports: [HighlightDirective]
})
export class AppComponent {
  color = 'lightgreen';
```

Revenir au sommaire

Pipes personnalisés

```
 // time-ago.pipe.ts
@Pipe({
  name: 'timeAgo',
  standalone: true
})
export class TimeAgoPipe implements PipeTransform {
  transform(value: Date | string): string {
    const date = new Date(value);
    const now = new Date();
    const seconds = Math.floor((now.getTime() - date.getTime()) / 1000);

    if (seconds < 60) {
      return 'Il y a quelques secondes';
    }
    if (seconds < 3600) {
```

Revenir au sommaire

Utilisation des pipes

```
@Component({
  selector: 'app-post',
  template: `
    <article>
      <h2>{{ title | uppercase }}</h2>
      <p>Publié {{ date | timeAgo }}</p>
      <p>Prix: {{ price | currency:'EUR' }}</p>
      <p>{{ content | slice:0:100 }}...</p>
    </article>
  `,
  imports: [TimeAgoPipe]
})
export class PostComponent {
  title = 'Mon article';
  date = new Date();
```

Revenir au sommaire

Pipes avec Signals

```
 @Pipe({
  name: 'filter',
  standalone: true
})
export class FilterPipe implements PipeTransform {
  transform(items: Signal<any[]>, searchTerm: Signal<string>): Signal<any[]> {
    return computed(() => {
      const term = searchTerm().toLowerCase();
      return items().filter(item =>
        item.name.toLowerCase().includes(term)
      );
    });
  }
}
```

Revenir au sommaire

Directive avec Injection de dépendances

```
 @Directive({
   selector: '[appTooltip]',
   standalone: true
})
export class TooltipDirective implements OnDestroy {
   @Input('appTooltip') text = '';

   private tooltip: HTMLElement | null = null;

   constructor(
     private el: ElementRef,
     private renderer: Renderer2,
     private zone: NgZone
   ) {}
```

Revenir au sommaire

Exercice : Création d'une directive de validation

Créez une directive de validation personnalisée pour les formulaires :

```
 @Directive({
  selector: '[appPasswordStrength]',
  standalone: true,
  providers: [{
    provide: NG_VALIDATORS,
    useExisting: PasswordStrengthDirective,
    multi: true
  }]
})
export class PasswordStrengthDirective implements Validator {
  validate(control: AbstractControl): ValidationErrors | null {
    const value = control.value;

    if (!value) {
      return null;
```

Cet exercice vous à permis de comprendre comment créer des directives de validation personnalisées et les intégrer avec les formulaires Angular.

Revenir au sommaire

Nouveau Control Flow (Angular 18/19)

```
@Component({
  template: `
    @if (isLoading()) {
      Loading...
    } @else {
      @for (item of items(); track item.id) {
        {{ item }}
      }
    }
  `
})
```

Revenir au sommaire

Performance et Optimisation

Revenir au sommaire

Stratégies d'optimisation

```
 // Avant (avec Zone.js)
@Component({
  template: `
    <div>{{ value }}</div>
    <button (click)="increment()">+</button>
  `
})
class OldComponent {
  value = 0;

  increment() {
    this.value++;  // Déclenche la détection des changements globale
  }
}
```

Revenir au sommaire

Optimisation des templates

```
@Component({
  template: `
    <!-- Utilisation de @if au lieu de *ngIf -->
    @if (showContent()) {
      <heavy-component />
    }

    <!-- Utilisation de @for au lieu de *ngFor -->
    @for (item of items(); track item.id) {
      <item-component [data]="item" />
    }

    <!-- Defer loading -->
    @defer {
      <expensive-component />
```

Revenir au sommaire

Lazy Loading moderne

```
 // app.routes.ts
export const routes: Routes = [{
  path: 'admin',
  loadChildren: () => import('./admin/routes'),
  canActivate: [authGuard],
  providers: [
    importProvidersFrom(AdminModule)
  ]
}];

// Composant lazy-loadé
@Component({
  standalone: true,
  imports: [CommonModule],
  template: `
```

Revenir au sommaire

Optimisation des images

```
@Component({
  template: `
    <img
      ngSrc="{{ imageUrl }}"
      width="300"
      height="200"
      priority
      loading="eager"
    />

    @for (image of images(); track image) {
      <img
        ngSrc="{{ image.url }}"
        [width]="image.width"
        [height]="image.height"
```

Revenir au sommaire

Memoization avec Signals

```
@Component({
  template: `
    <div>Résultat filtré: {{ filteredData() }}</div>
  `
})
export class MemoizedComponent {
  data = signal<number[]>([]);
  threshold = signal(10);

  // Computed value avec memoization intégrée
  filteredData = computed(() => {
    console.log('Calcul coûteux effectué');
    return this.data().filter(n => n > this.threshold());
  });
```

Revenir au sommaire

Optimisation des requêtes HTTP

```
 @Injectable({
  providedIn: 'root'
})
export class OptimizedApiService {
  private cache = new Map<string, any>();
  private http = inject(HttpClient);

  getData<T>(url: string, options: { ttl?: number } = {}) {
    const cached = this.cache.get(url);
    if (cached && (!options.ttl || Date.now() - cached.timestamp < options.ttl)) {
      return of(cached.data);
    }

    return this.http.get<T>(url).pipe(
      tap(data => {
```

Revenir au sommaire

Defer Loading (Nouveauté Angular 18/19)

```
@Component({
  template: `
    @defer {
      <heavy-component />
    } @loading {
      <spinner />
    } @error {
      <error-message />
    } @placeholder {
      <div>Chargement...</div>
    }
  `
})
```

Revenir au sommaire

Tests dans Angular

Revenir au sommaire

Commandes CLI pour les tests

```
 # Lancer tous les tests
ng test

# Lancer les tests avec couverture de code
ng test --code-coverage

# Lancer les tests en mode watch
ng test --watch

# Lancer les tests d'un fichier spécifique
ng test --include src/app/features/user/user.component.spec.ts

# Générer un composant avec des tests
ng g c features/user --spec
```

Revenir au sommaire

Introduction aux tests
Tests unitaires avec Jasmine et Karma

```
 // user.service.spec.ts
describe('UserService', () => {
  let service: UserService;
  let httpMock: HttpTestingController;

  beforeEach(() => {
    TestBed.configureTestingModule({
      imports: [HttpClientTestingModule],
      providers: [UserService]
    });

    service = TestBed.inject(UserService);
    httpMock = TestBed.inject(HttpTestingController);
  });
```

Revenir au sommaire

Tests de composants avec Signals

```
// counter.component.spec.ts
describe('CounterComponent', () => {
  let component: CounterComponent;
  let fixture: ComponentFixture<CounterComponent>;

  beforeEach(async () => {
    await TestBed.configureTestingModule({
      imports: [CounterComponent]
    }).compileComponents();

    fixture = TestBed.createComponent(CounterComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });
```

Revenir au sommaire

Tests d'intégration

```
 // app.component.integration.spec.ts
describe('AppComponent (integration)', () => {
  let component: AppComponent;
  let fixture: ComponentFixture<AppComponent>;
  let userService: UserService;

  beforeEach(async () => {
    await TestBed.configureTestingModule({
      imports: [
        AppComponent,
        HttpClientTestingModule,
        RouterTestingModule
      ],
      providers: [UserService]
    }).compileComponents();
```

Revenir au sommaire

Tests E2E avec Cypress

```
// cypress/e2e/login.cy.ts
describe('Login Flow', () => {
  beforeEach(() => {
    cy.visit('/login');
  });

  it('should login successfully', () => {
    cy.intercept('POST', '/api/login', {
      statusCode: 200,
      body: { token: 'fake-token' }
    }).as('loginRequest');

    cy.get('[data-testid=email-input]')
      .type('user@example.com');
```

Revenir au sommaire

Tests de services avec Signals

```
 // data.service.spec.ts
describe('DataService', () => {
  let service: DataService;

  beforeEach(() => {
    TestBed.configureTestingModule({
      providers: [DataService]
    });
    service = TestBed.inject(DataService);
  });

  it('should update data and notify subscribers', () => {
    // Test du signal
    expect(service.data()).toEqual([]);
```

Revenir au sommaire

Tests de formulaires réactifs

```
 // registration.component.spec.ts
describe('RegistrationComponent', () => {
  let component: RegistrationComponent;
  let fixture: ComponentFixture<RegistrationComponent>;

  beforeEach(async () => {
    await TestBed.configureTestingModule({
      imports: [
        ReactiveFormsModule,
        RegistrationComponent
      ]
    }).compileComponents();

    fixture = TestBed.createComponent(RegistrationComponent);
    component = fixture.componentInstance;
```

Revenir au sommaire

Exercice : Tests complets d'une fonctionnalité

Créez une suite de tests complète pour une fonctionnalité de panier d'achat :

```
// cart.service.spec.ts
describe('CartService', () => {
  let service: CartService;

  beforeEach(() => {
    TestBed.configureTestingModule({
      providers: [CartService]
    });
    service = TestBed.inject(CartService);
  });

  it('should add items to cart', () => {
    const item = { id: 1, name: 'Test', price: 10 };

    service.addItem(item);
```

Revenir au sommaire

Cet exercice vous à permis de pratiquer :

- Les tests unitaires de services
- Les tests de composants
- L’utilisation des Signals dans les tests
- Les tests d’intégration
- Les mocks et les spies

Revenir au sommaire

Déploiement d'Applications Angular

Revenir au sommaire

Commandes CLI de Build et Déploiement

```
 # Build de production
ng build --configuration production

# Build avec stats
ng build --stats-json

# Analyser la taille du bundle
ng build --configuration production --source-map

# Servir une version de production localement
ng serve --configuration production

# Build avec optimisation SSR
ng build --configuration production --ssr
```

Revenir au sommaire

Build de production

```
 # Build standard
ng build --configuration production

# Build avec SSR
ng build && ng run my-app:server
```

Revenir au sommaire

Configuration des environnements

```
 // environment.prod.ts
export const environment = {
  production: true,
  apiUrl: 'https://api.production.com',
  features: {
    analytics: true,
    monitoring: true
  }
};

// app.config.ts
import { environment } from './environments/environment';

export const appConfig: ApplicationConfig = {
  providers: [
```

Revenir au sommaire

Optimisation du bundle

```
// angular.json
{
  "projects": {
    "my-app": {
      "architect": {
        "build": {
          "builder": "@angular-devkit/build-angular:browser",
          "options": {
            "outputPath": "dist/my-app",
            "index": "src/index.html",
            "main": "src/main.ts",
            "polyfills": ["zone.js"],
            "tsConfig": "tsconfig.app.json",
            "assets": [
              "src/favicon.ico",
```

Revenir au sommaire

Server-Side Rendering (SSR)

```
 // server.ts
import { APP_BASE_HREF } from '@angular/common';
import { CommonEngine } from '@angular/ssr';
import express from 'express';
import { fileURLToPath } from 'url';
import { dirname, join, resolve } from 'path';
import bootstrap from './src/main.server';

const app = express();
const port = process.env.PORT || 4000;
const __dirname = dirname(fileURLToPath(import.meta.url));
const distFolder = join(__dirname, '../dist/my-app/browser');

// Servir les fichiers statiques
app.use(express.static(distFolder));
```

Revenir au sommaire

Docker

```dockerfile
 # Dockerfile
# Stage 1: Build
FROM node:20-alpine as builder

WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build

# Stage 2: Run
FROM node:20-alpine

WORKDIR /app
COPY --from=builder /app/dist ./dist
```

Revenir au sommaire

CI/CD avec GitHub Actions

```yaml
 # .github/workflows/deploy.yml
name: Deploy

on:
  push:
    branches: [ main ]

jobs:
  deploy:
    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v4

    - name: Setup Node.js
```

Revenir au sommaire

Nginx Configuration

```
 # nginx.conf
server {
    listen 80;
    server_name example.com;
    root /usr/share/nginx/html;
    index index.html;

    # Gzip compression
    gzip on;
    gzip_types text/plain text/css application/json application/javascript;
    gzip_min_length 1000;

    # Cache control
    location ~* \.(js|css|png|jpg|jpeg|gif|ico)$ {
        expires 1y;
```

Revenir au sommaire

Monitoring et Analytics

```
 // app.config.ts
import { ApplicationConfig } from '@angular/core';
import { provideClientHydration } from '@angular/platform-browser';
import { provideRouter } from '@angular/router';
import { routes } from './app.routes';

export const appConfig: ApplicationConfig = {
  providers: [
    provideRouter(routes),
    provideClientHydration(),
    // Monitoring
    {
      provide: ErrorHandler,
      useClass: GlobalErrorHandler
    },
```

Revenir au sommaire

Exercice : Déploiement complet

Configurez un déploiement complet avec :

- Build optimisé
- SSR
- Docker
- CI/CD
- Monitoring

```
 # 1. Préparation du build
npm run lint
npm test
npm run build:ssr

# 2. Construction de l'image Docker
docker build -t my-angular-app .
docker run -p 4000:4000 my-angular-app

# 3. Déploiement avec monitoring
```

Revenir au sommaire

Monitoring

```
 // monitoring.service.ts
@Injectable({
  providedIn: 'root'
})
export class MonitoringService {
  private errorCount = signal(0);
  private performanceMetrics = signal<PerformanceMetrics>({
    fcp: 0,
    lcp: 0,
    cls: 0
  });

  trackError(error: Error) {
    this.errorCount.update(count => count + 1);
    // Envoi à un service de monitoring
```

Revenir au sommaire

Déploiement avec Vercel

Revenir au sommaire

Configuration Vercel

1. Installez Vercel CLI :

```
npm i -g vercel
```

2. Connectez-vous à votre compte :

```
vercel login
```

Revenir au sommaire

Déploiement automatique

1. Créez un fichier `vercel.json` à la racine :

```
{
  "version": 2,
  "builds": [
    {
      "src": "dist/*",
      "use": "@vercel/static"
    }
  ],
  "routes": [
    {
      "src": "/(.*)",
      "dest": "/index.html"
    }
  ]
}
```

Revenir au sommaire

2. Configurez le build :

```
{
  "scripts": {
    "build": "ng build --configuration production"
  }
}
```

Revenir au sommaire

Intégration Continue

1. Connectez votre repo GitHub à Vercel

2. Activez les déploiements automatiques :

- Sur chaque push sur main
- Preview sur les Pull Requests
- Rollback automatique en cas d'erreur

3. Variables d'environnement :

```
vercel env add PRODUCTION_API_URL
```

Revenir au sommaire

Optimisations Vercel

- Edge Functions pour l'API
- Image Optimization
- Analytics intégrées
- Monitoring temps réel
- Certificats SSL automatiques

```
// next.config.js
module.exports = {
  images: {
    domains: ['assets.example.com'],
  },
}
```

Revenir au sommaire

Commandes utiles

```
 # Déploiement manuel
vercel

# Déploiement en production
vercel --prod

# Voir les logs
vercel logs

# Lister les déploiements
vercel list

# Supprimer un déploiement
vercel remove <deployment-id>
```

Revenir au sommaire

Bonnes Pratiques

Revenir au sommaire

Architecture et organisation

```
 // ❌ À éviter
@Component({
  template: `
    <div>
      <h1>{{ title }}</h1>
      <div *ngFor="let item of items">
        <!-- Logique complexe -->
      </div>
    </div>
  `
})
class BigComponent {
  // Trop de responsabilités
}
```

Revenir au sommaire

Gestion de l'état

```
 // ❌ État global mutable
class StateService {
  public data = [];

  updateData(newData) {
    this.data = newData;  // Mutation directe
  }
}

// ✅ État immutable avec Signals
@Injectable({ providedIn: 'root' })
class StateService {
  private _data = signal<Data[]>([]);
  readonly data = this._data.asReadonly();
```

Revenir au sommaire

Performance

```
 // ❌ Calculs inutiles
@Component({
  template: `
    <div>{{ heavyComputation() }}</div>
  `
})
class SlowComponent {
  heavyComputation() {
    // Recalculé à chaque cycle
    return this.data.reduce((acc, val) => acc + val, 0);
  }
}

// ✅ Optimisé avec computed
@Component({
```

Revenir au sommaire

Gestion des formulaires

```
 // ❌ Validation non structurée
@Component({
  template: `
    <form (ngSubmit)="onSubmit()">
      <input [(ngModel)]="email">
      <span *ngIf="!isValidEmail">Email invalide</span>
    </form>
  `
})
class FormComponent {
  isValidEmail = true;

  validateEmail() {
    this.isValidEmail = /^[^@]+@[^@]+\.[^@]+$/.test(this.email);
  }
```

Revenir au sommaire

Gestion des erreurs

```
 // ❌ Gestion d'erreur basique
@Injectable()
class ApiService {
  getData() {
    return this.http.get('/api/data').pipe(
      catchError(error => {
        console.error(error);
        return of(null);
      })
    );
  }
}

// ✅ Gestion d'erreur robuste
@Injectable()
```

Revenir au sommaire

Organisation du code

```
 // ❌ Fichier surchargé
// huge-file.ts
@Component({
  // 500+ lignes de template
})
class HugeComponent {
  // 1000+ lignes de logique
}
```