Les bases Angular 19

Un petit aide-mémoire pour t'aider à te rappeler des bases d'Angular 19.

Installation et Configuration

Prérequis

Assure-toi d'avoir Node.js (version 18 ou ultérieure) et npm installés.

Installation de l'interface en ligne de commande Angular (CLI)

```
npm install -g @angular/cli
```

Création d'un nouveau projet

```
ng new nom-du-projet
cd nom-du-projet
```

Lancement du serveur de développement

```
ng serve
```

Structure du Projet

Un projet Angular typique comprend les dossiers et fichiers suivants :

- `src/` : Contient le code source de l'application.
  - `app/` : Contient les composants, services, modules, etc.
  - `assets/` : Contient les ressources statiques (images, fichiers, etc.).
  - `environments/` : Contient les configurations pour différents environnements (développement, production).
- `angular.json` : Fichier de configuration principal du projet Angular.

Composants

Création d'un composant

```
ng generate component nom-du-composant
```

Structure d'un composant

- `nom-du-composant.component.ts` : Contient la logique du composant.

- `nom-du-composant.component.html` : Contient le template HTML.
- `nom-du-composant.component.css` : Contient les styles CSS.
- `nom-du-composant.component.spec.ts` : Contient les tests unitaires.

Services

Création d'un service

```
ng generate service nom-du-service
```

Utilisation d'un service

```
import { NomDuService } from './nom-du-service.service';

constructor(private nomDuService: NomDuService) {}
```

Modules

Module principal

`AppModule` (défini dans `app.module.ts`).

Création d'un module

```
ng generate module nom-du-module
```

Importation d'un module

```
import { NomDuModule } from './nom-du-module/nom-du-
        module.module';

@NgModule({
  imports: [NomDuModule],
  // ...
})
export class AppModule {}
```

Routing

Définis les routes dans un tableau de type `Routes` :

```
import { Routes, RouterModule } from '@angular/router';
import { NomDuComposant } from './nom-du-composant/nom-du-
        composant.component';
```

```
const routes: Routes = [
  { path: 'chemin', component: NomDuComposant },
  { path: '', redirectTo: '/chemin', pathMatch: 'full' },
  { path: '**', component: PageNotFoundComponent },
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule],
})
export class AppRoutingModule {}
```

Data Binding

Définitions

- L'**interpolation** permet d'afficher des valeurs de composant dans le template HTML en utilisant la syntaxe `{{ expression }}`. Par exemple, `{{ nom }}` affichera la valeur de la propriété `nom` du composant.
- Le **property binding** permet de lier une propriété d'un élément HTML à une propriété de composant en utilisant la syntaxe `[propriete]="expression"`. Par exemple, `<img [src]="urlImage">` lie la propriété `src` de l'élément `img` à la propriété `urlImage` du composant.
- L'**event binding** permet de lier un événement d'un élément HTML à une méthode de composant en utilisant la syntaxe `(evenement)="expression"`. Par exemple, `<button (click)="onClick()">` appelle la méthode `onClick` du composant lorsque le bouton est cliqué.
- Le **two-way** binding permet de lier une propriété de composant à une propriété d'un élément HTML de manière bidirectionnelle, en utilisant la directive `ngModel`. Par exemple, `<input [(ngModel)]="nom">` lie la propriété `nom` du composant à la valeur de l'élément `input`, permettant des mises à jour dans les deux sens.

Résumé

- **Interpolation** : `{{ expression }}`
- **Property Binding** : `[property]="expression"`
- **Event Binding** : `(event)="expression"`
- **Two-way Binding** : `[(ngModel)]="expression"` (nécessite le module `FormsModule`)

Pipes

Pipes intégrés

- `{{ valeur | date }}` : Formate une date.
- `{{ valeur | uppercase }}` : Convertit en majuscules.
- `{{ valeur | lowercase }}` : Convertit en minuscules.
- `{{ valeur | titlecase }}` : Met en majuscules la première lettre de chaque mot.
- `{{ valeur | currency:'EUR':'symbol':'1.2-2' }}` : Formate une valeur en monnaie (exemple en euros, avec symbole € et 2 décimales).
- `{{ valeur | percent }}` : Formate un nombre en pourcentage.
- `{{ valeur | decimal:'1.2-2' }}` : Formate un nombre en décimal (min 1 chiffre, 2 après la virgule).
- `{{ valeur | number:'1.0-3' }}` : Formate un nombre avec un nombre de décimales spécifié.
- `{{ valeur | json }}` : Convertit un objet en JSON formaté.
- `{{ valeur | slice:1:3 }}` : Extrait une sous-chaîne (de l'index 1 à 3).
- `{{ array | async }}` : Attend la résolution d'une `Promise` ou d'un `Observable`.
- `{{ valeur | i18nSelect:mapping }}` : Sélectionne une valeur en fonction d'une clé locale.
- `{{ valeur | i18nPlural:mapping }}` : Gère les pluriels en fonction de la valeur.

Création d'un pipe personnalisé

```
ng generate pipe nom-du-pipe
```

Formulaires

Template-driven Forms

Les **Template-driven Forms** utilisent des directives Angular dans le template HTML pour gérer les formulaires.

```
<form #f="ngForm" (ngSubmit)="onSubmit(f)">
  <input ngModel name="username" required>
  <button type="submit">Submit</button>
</form>
```

Reactive Forms

Les **Reactive forms** utilisent des classes TypeScript pour créer et gérer les formulaires de manière plus programmatique et réactive.

```
import { FormBuilder, FormGroup, Validators } from '@angular/
        forms';

export class NomDuComposant {
  myForm: FormGroup;

  constructor(private fb: FormBuilder) {
    this.myForm = this.fb.group({
      username: ['', Validators.required]
    });
  }

  onSubmit() {
    if (this.myForm.valid) {
      console.log(this.myForm.value);
    }
  }
}
```

HTTP Client

Utilisation du HttpClient

```
import { HttpClientModule } from '@angular/common/http';

@NgModule({
  imports: [HttpClientModule],
  // ...
})
export class AppModule {}
```

Injectez `HttpClient` dans votre service ou composant

```
import { HttpClient } from '@angular/common/http';

constructor(private http: HttpClient) {}

getData() {
  return this.http.get('url-de-l-api');
}
```

Observables et RxJS

Définition

Un **Observable** est un flux de données asynchrones utilisé pour :
- Requêtes HTTP (`HttpClient`)
- Événements utilisateur
- Communication entre composants

Création et souscription

```
import { Observable } from 'rxjs';

const monObservable = new Observable(observer => {
  observer.next('Valeur');
  observer.complete();
});

monObservable.subscribe(val => console.log(val));
```

Opérateurs essentiels

- `map(val => val * 2)` : Transformation
- `filter(val => val > 10)` : Filtrage
- `debounceTime(500)` : Limite les appels rapides (ex. recherche)
- `merge(obs1, obs2)` : Fusionne plusieurs Observables

Observables vs Promises

Observables	**Promises**
Émettent **plusieurs valeurs**	Une seule valeur
Annulables (`unsubscribe()`)	Non annulables
Fonctionnent **par paresse**	Exécutées immédiatement

Utilisation avec HttpClient

```
this.http.get('https://api.exemple.com/data').subscribe(data =>
        console.log(data));
```

Annulation d’un Observable

```
const sub = this.http.get('url').subscribe();
sub.unsubscribe(); // Stoppe l’écoute
```

Tests

Tests unitaires

- Utilisez Jasmine et Karma (configurés par défaut).
- Les fichiers de test se terminent par `.spec.ts`.

Tests de bout en bout (E2E)

- Utilisez Protractor ou Cypress pour les tests E2E.

Déploiement

Build pour production

```
ng build --configuration=production
```

Cela génère les fichiers optimisés dans le dossier `dist/`.

Ressources

Documentation Officielle Angular

- https://angular.io/docs
- Angular CLI
- RxJS

Tutoriels et Cours en Ligne

- Angular University
- FreeCodeCamp Angular Guide
- Egghead.io

Communauté et Forums

- Stack Overflow - Angular
- Reddit - r/Angular
- Discord - Angular