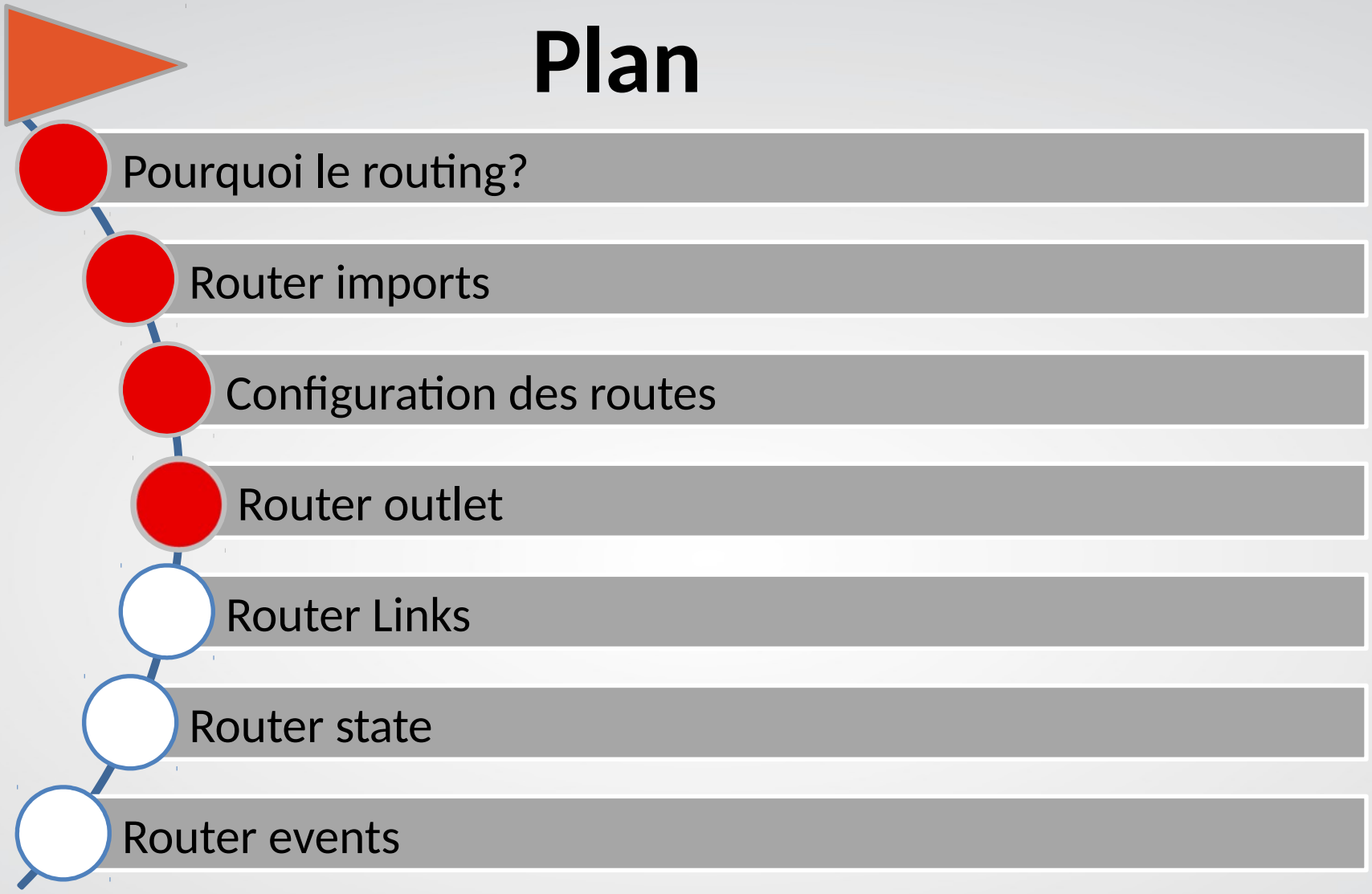
esprit
Se former autrement

Composition d'une application Angular

United Nations Educational, Scientific and Cultural Organization
UNESCO Chair "Project-based learning" ESPRIT School of engineering, Tunisia
CONFÉRENCE DES GRANDES ÉCOLES
EXCELLENCE FOR A COMPLEX WORLD
European Accreditation of Engineering Programmes EUR-ACE®
délivrée par CTI Commission des Titres d'Ingénieur
cdio
CONCEIVE DESIGN IMPLEMENT OPERATE

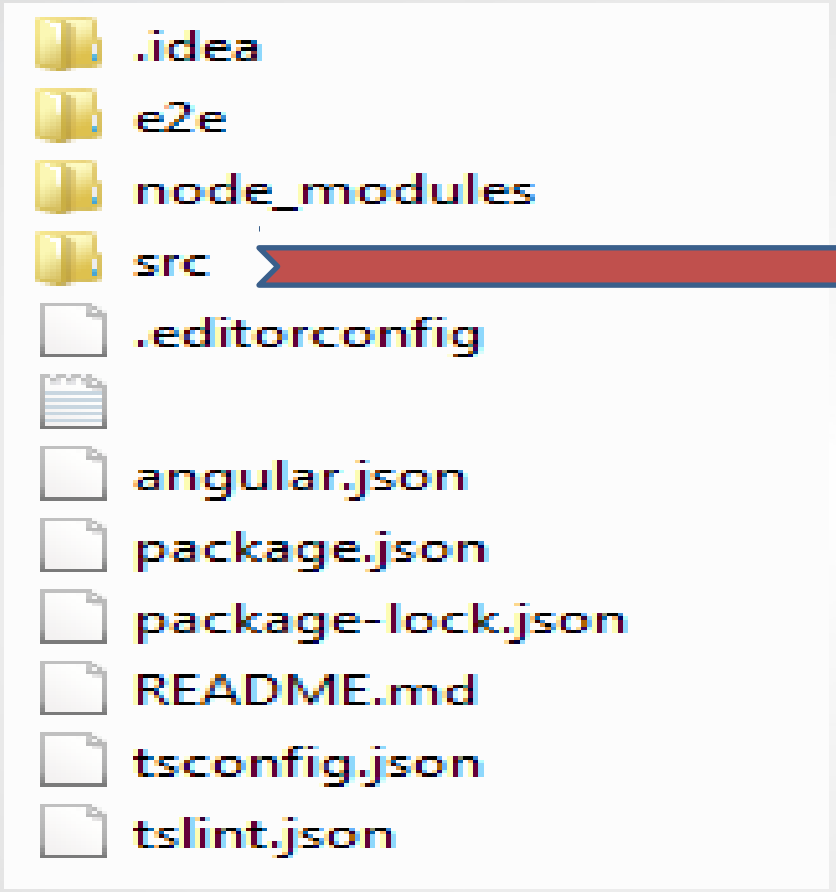
Plan


Pourquoi le routing?
Router imports
Configuration des routes
Router outlet
Router Links
Router state
Router events

Application **Angular - Présentation**

- Une application Angular est modulaire.
- Elle possède au moins un module appelé « module racine » ou « root module »
- Elle peut contenir d'autres modules à part le module racine.
- Par convention, le module racine est appelé « **AppModule** » et se trouve dans un fichier appelé « **app.module.ts** »

Application Angular - Structure



Application **Angular – Structure**

- **e2e** : Tout ce qui va concerner les tests end to end (tests d'intégration)
- **node_modules** : contient les dépendances installés avec npm
- **src**: les différents éléments de l'application (composants, services, …)
- **src/app** : le code de l'application
- **package.json** : fichier déclarant les dépendances NPM tirées lors de l'installation du projet et nécessaire à la compilation et les tests

Application **Angular – Structure**

- **tslint.json** : fichier définissant les règles de codage TypeScript.
- **tsconfig.json:** un fichier de définition TypeScrit
- **src/assets** : où placer tous les *assets* tels que les images.
- **src/environments** : on retrouve les différents fichiers de configuration spécifiques aux environnements d'exécution.
- ……

Modules/NgModules

- Un module est un bloc de code qui sert à encapsuler des fonctionnalités similaires une application.
- Le système de modularité dans Angular est appelé **NgModules**.
- Un module peut être exporté sous forme de classe.
- La classe qui décrit le module Angular est une classe décorée par @NgModule.

Exemple: FormsModule, HttpClientModule, RouterModule

Modules - Exemples

Exemple1: Module racine

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';
import { HelloWorldComponent } from './hello-world/hello-world.component';
@NgModule({
  declarations: [
    AppComponent,
    HelloWorldComponent
  ],
  imports: [
    BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Exemple2: autre Module

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';

import { ProjetRoutingModule } from './projet-routing.module';
import { CreateProjectComponent } from './create-project/create-project.component';

@NgModule({
  imports: [
    CommonModule,
    ProjetRoutingModule
  ],
  declarations: [CreateProjectComponent]
})
export class ProjetModule { }
```

NgModule

- NgModule = classe importé de angular/core et marké apr le décorateur @NgModule et sa métadata
- @NgModule : décorateur qui permet de définir la classe en tant que NgModule
- Métadata ou métadonnée est un objet qui décrit le comportement d'un élément qui est dans notre cas le module

NgModule - métadata

- **Declarations**: les composants – directives – pipes utilisés par ce module
- **Imports**: les modules internes ou externes utilisés dans ce module
- **Providers**: Les services utilisés
- **Bootstrap**: déclare la vue principale de l'application (celle du composant racine). Seul le *root NgModule* modifie cette propriétés.

Composant-Définition

- Un composant est un élément réutilisable, indépendant et responsable d'une seule action métier.
- Un module peut contenir un ou plusieurs composants.
- Une application a au moins un composant racine

Composant-Définition

Composant-Définition

- Un composant crée est décrit par défaut par quatre fichiers :

Fichier	Rôle
nomComp.component.ts	Classe décrit le métier du composant
nomComp.component.html	Fichier template du composant
nomComp.component.css	Fichier de style du composant
nomComp.component.spec.ts	Fichier de test du composant

- Le composant racine est décrit par les fichiers app.component.css|html|ts|spec.ts

Composant-Définition

- Un composant définit de(s) vue(s) et utilise de(s) service(s) pour réaliser un traitement.
- Le métier du composant est décrit dans la

classe du composant qui peut être exportée,

définit par le décorateur @component et décrite par un objet « métadata »

Composant-Métadata

- **selector** : définit une instance du composant dans le fichier HTML. Exemple: <app-root></app-root>
- **templateUrl**: l'url du fichier HTML associé au composant
- **styleUrls**: l'url du fichier CSS associé au template du composant
- **providers**: les services dont le composant a besoin pour son fonctionnement

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css'],
  providers:  [ TestService ]})
```

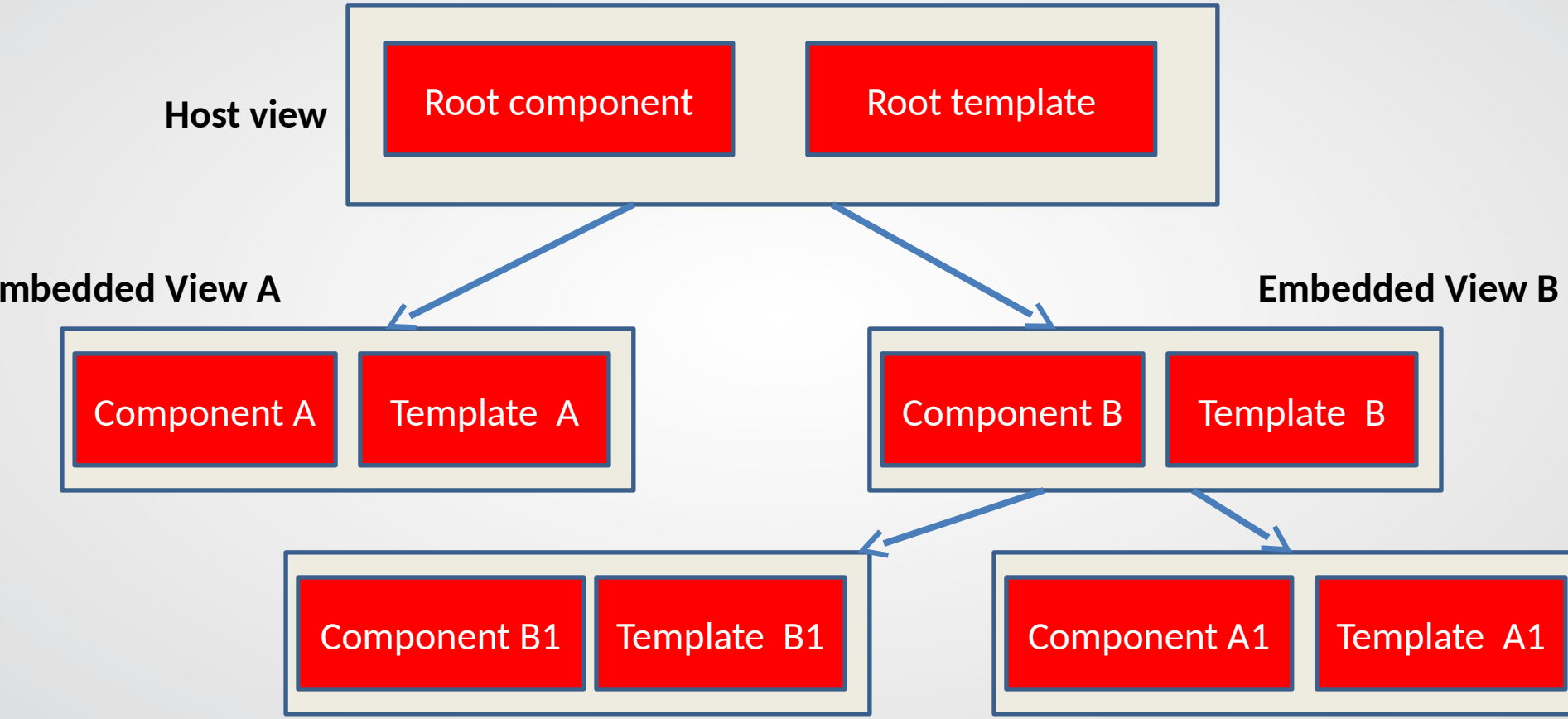
métadata

Vue / Template

- Une vue est définit par le composant et son template.
- Les vues sont oragnisées d'une façon hiérarchique afin de pouvoir modifier ou cacher des sections de pages ou des pages entières.
- Le template attaché directement au composant est la vue hôte de ce composant « host view »

Vue / Template

Un composant peut définir une hiérarchie de vues.



Template

Un template est un fichier HTML contenant:

1- La syntaxe régulière d'HTML (,<h1>,<p>,…)

2- La syntaxe template de Angular où les éléments peuvent être :

- composant
- directives
- pipes
- databinding

Angular Syntax template

- **Les directives**: appliquent une logique aux éléments du DOM.
- Les directives intégrées dans Angular sont groupées en deux types:

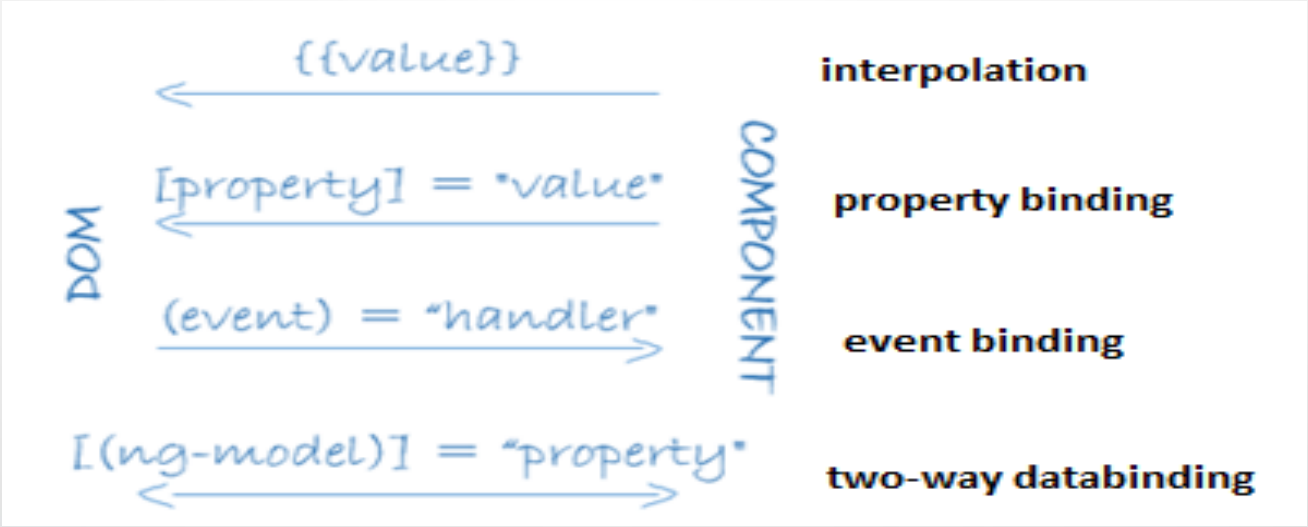
1. Directives structurelles: change le DOM en ajoutant et retirant des éléments (NgIF, NgForOf, NgSwitch)
2. Directives attributs: change l'apparence et l'attitude d'un élément DOM, composant ou autre directive (NgStyle, NgClass, NgModel)

Angular Syntax template

- **Pipes** : transforment les données avant de les afficher. Exemple: DatePipe, UpperCasePipe, LowerCasePipe, ….
- **Data binding**: liaison de données qui permet de coordonner entre les éléments du composants et ceux du template. (Interpolation, event binding, property binding et two-way data binding)

Angular Syntax template-Data binding

- Le databinding est un mécanisme de coordination entre le composant et le template dans un seul sens ou dans les deux sens.
- Il existe 4 formes de databinding



Angular Syntax template-Data binding

Soit deux composants A et B comme suit:

Composant A{
Propriété A1
Propriété A2
Méthode A
}

Template A
<h1></h1>
<button>Envoyer</button>
<composantB></composantB>

Composant B{
Propriété B
Méthode B
}

Template B

Angular Syntax template-Data binding

L'interpolation permet de passer une valeur du composant vers le template afin de l'afficher.

```
Template A
<h1> {{Propiété A1}} </h1>
```

Property binding permet de modifier la valeur d'une propriété d'un composant ou d'un élément DOM.

```
Template A
<composantB [Propriété B]=
''PropriétéA2 ''></composantB>
```

Angular Syntax template-Data binding

Event binding permet d'appeler une méthode du composant suite à une action faite par l'utilisateur au niveau du template

Template A

```
<button (click)="MéthodeA()"> envoyer
</button>
```

Angular Syntax template-Data binding

Two-way data binding permet de récupérer une valeur à partir du template et l'envoyer vers une propriété et vis versa.

Template A

```
<input [(ngModel)]= ''PropriétéA1''/>
```

Le contenu de la propriétéA1 est affiché au niveau de l'input et si on modifie la valeur de l'input, la valeur de la propriété est modifiée.

Interaction entre composants

Deux composants peuvent avoir plusieurs interactions. Nous allons voir ici deux interactions possibles entre parent et fils:

1- Le parent passe des informations au fils: input binding « @Input() »

2- Le parent intercepte les événements du fils « @Output() »

Interaction entre composants

- Un composant peut avoir des propriétés d'entrée ou de sortie, appelés respectivement « input property » et « output property ».
- La propriété d'entrée permet de récupérer une valeur depuis le composant parent et décorée par « @Input() »
- La propriété de sortie expose les producteurs d'événements, tels que les objets EventEmitter et décorée par « @Output() »

Interaction entre composants

Pour déclarer les propriétés input et output deux façons possibles (utiliser l'une des deux):

1- Dans la classe du composant fils

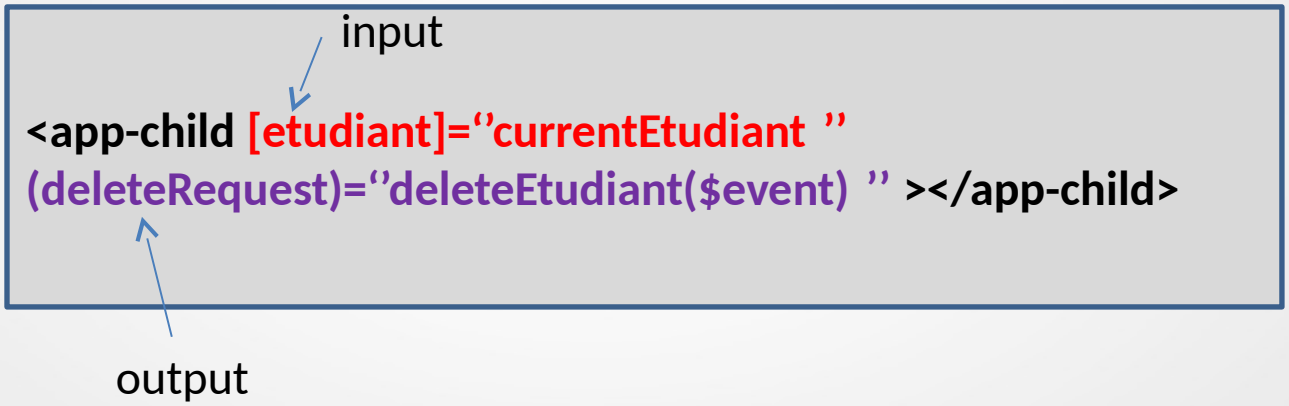
```
@Input() etudiant: Etudiant;
@Output() deleteRequest = new EventEmitter<Etudiant>();
```

2- Au niveau du métadata du composant fils

```
@Componentt({
 inputs: ['etudiant'],
 outputs: ['deleteRequest'], })
```

Interaction entre composants

Le composant parent peut alors envoyer des données au composant fils et entend les événements déclenchés au niveau de son fils.



RQ:
<fils></fils> : selector du composant fils
currentEtudiant : propriété du composant parent
deleteEtudiant(): méthode du composant parent

Interaction entre composants

Le composant fils peut modifier la valeur de l'input :

- Le setter de la propriété

```
@Input()
set etudiant(et: Etudiant) {
  this.e = traitement;
   this.changed.emit(this.e);
}
```

- La méthode hook **ngOnChanges**

```
ngOnChanges() {
  this.e = traitement;
   this.changed.emit(this.e);
}
```

Interaction entre composants

Le composant parent accède aux propriétés/méthodes du composant fils en utilisant:

- Variable référence de template (accès limité au niveau du template)
- @ViewChild() (accès à partir de la classe du composant)

Interaction entre composants

Pour utiliser ViewChild(), il faut:

- Importer {ViewChild} from @angular/core
- Définir le composant child dans le ts:

```
@ViewChild(ChildComponent)
Private exempleComponent : ChildComponent;
```

NB: Le composant fils est déjà appelé au niveau du template du parent

- Accéder aux méthodes/propriétés du child à partir de la classe du composant

Interaction entre composants

- Le composant fils devient disponible quand la vue du parent est initialisée.
- Si un traitement est à faire quand la vue du composant est initialisée, la classe du composant parent doit implémenter l'interface **AfterViewInit** et la méthode **ngAfterViewInit.**

Cycle de vie d'un composant

Un composant passe par plusieurs phases depuis sa création jusqu'à sa destruction : cycle de vie

Angular maintient et suit ces différentes phases en utilisant des méthodes appelées « hooks ».

On peut alors à chaque phase implémenter une logique.

Ces méthodes se trouvent dans des interfaces dans la librairie « @angular/core »

Cycle de vie d'un composant

Cycle de vie d’un composant

RQ: le constructeur d’un composant n’est pas un hook mais il fait partie du cycle de vie d’un composant : sa création

Il est logiquement appelé en premier, et c'est à ce moment que les dépendances (services) sont injectées dans le composant par Angular.

Cycle de vie d'un composant

Méthode/hook	Rôle
ngOnChanges	Appelé lorsqu'une propriété input est définie ou modifiée de l'extérieur. L'état des modifications sur ces propriétés est fourni en paramètre
ngOnInit	Appelé **une seule fois** après le 1er appel du hook ngOnChanges(). Permet de réaliser l'initialisation du composant, qu'elle soit lourde ou asynchrone (on ne touche pas au constructeur pour ça)
ngDoCheck	Appelé après chaque détection de changements
ngAfterContentInit	Appelé une fois que le contenu externe est projeté dans le composant (transclusion)

Cycle de vie d'un composant

Méthode	Rôle
ngAfterContentChecked	Appelé chaque fois qu'une vérification du contenu externe (transclusion) est faite
ngAfterViewInit	Appelé dès lors que la vue du composant ainsi que celle de ses enfants sont initialisés
ngAfterViewChecked	Appelé après chaque vérification des vues du composant et des vues des composants enfants.
ngOnDestroy	Appelé juste avant que le composant soit détruit par Angular. Il permet alors de réaliser le nettoyage adéquat de son composant. C'est ici qu'on veut se désabonner des Observables ainsi que des events handlers sur lesquels le composant s'est abonné.

Cycle de vie d'un composant

Les méthodes **ngAfterContentInit**, **ngAfterContentChecked**,**ngAfterViewInit** et **ngAfterViewChecked** sont exclusives aux composants, tandis que toutes les autres le sont aussi pour les directives.

Références

https://angular.io/