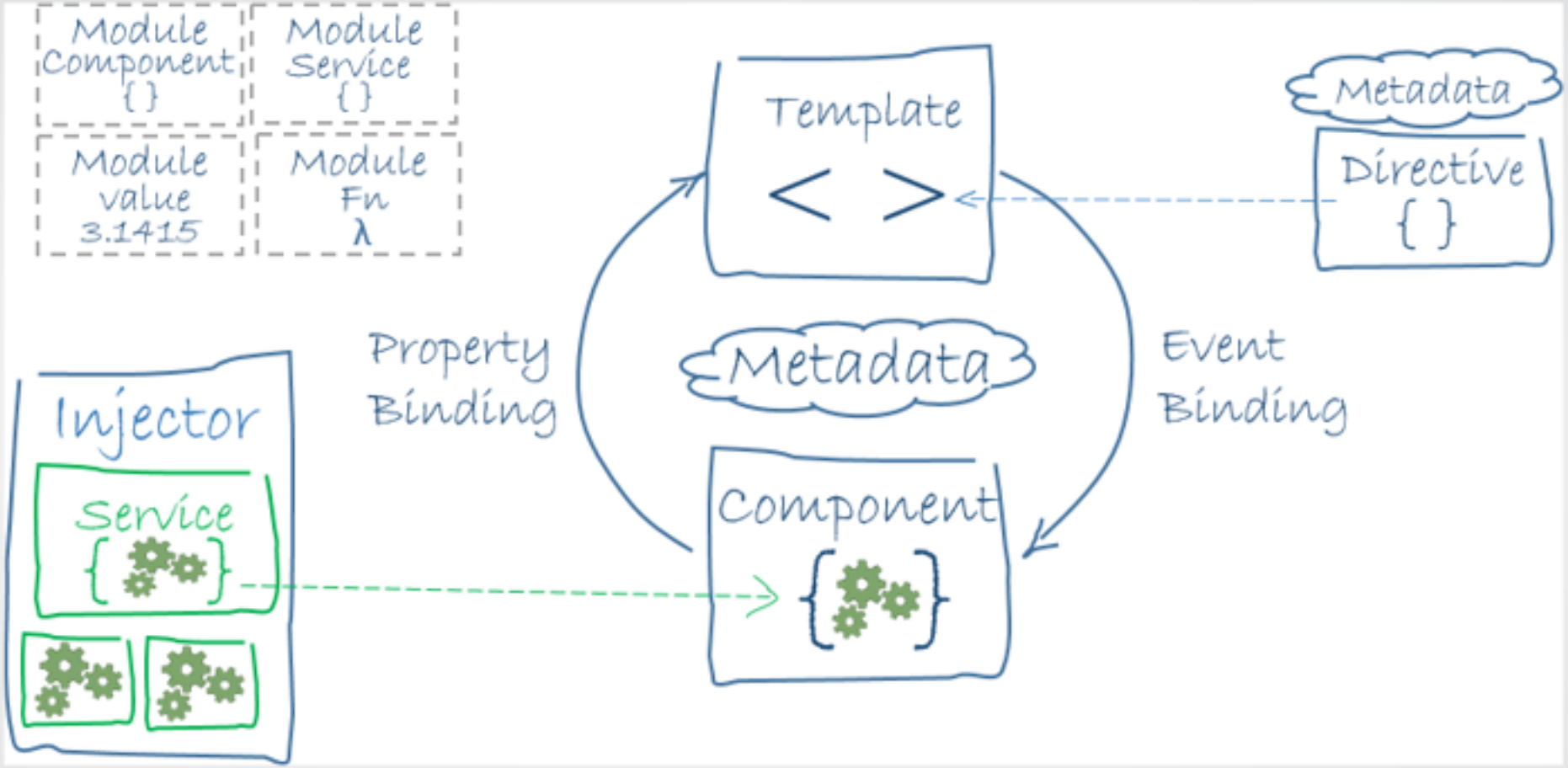
esprit
Se former autrement
HONORIS UNITED UNIVERSITIES

Manipulation de composants dans

- Architecture Angular
- Data-Binding
- Les directives et les pipes
- Cycle de vie d’un composant

Architecture Angular

Architecture Angular



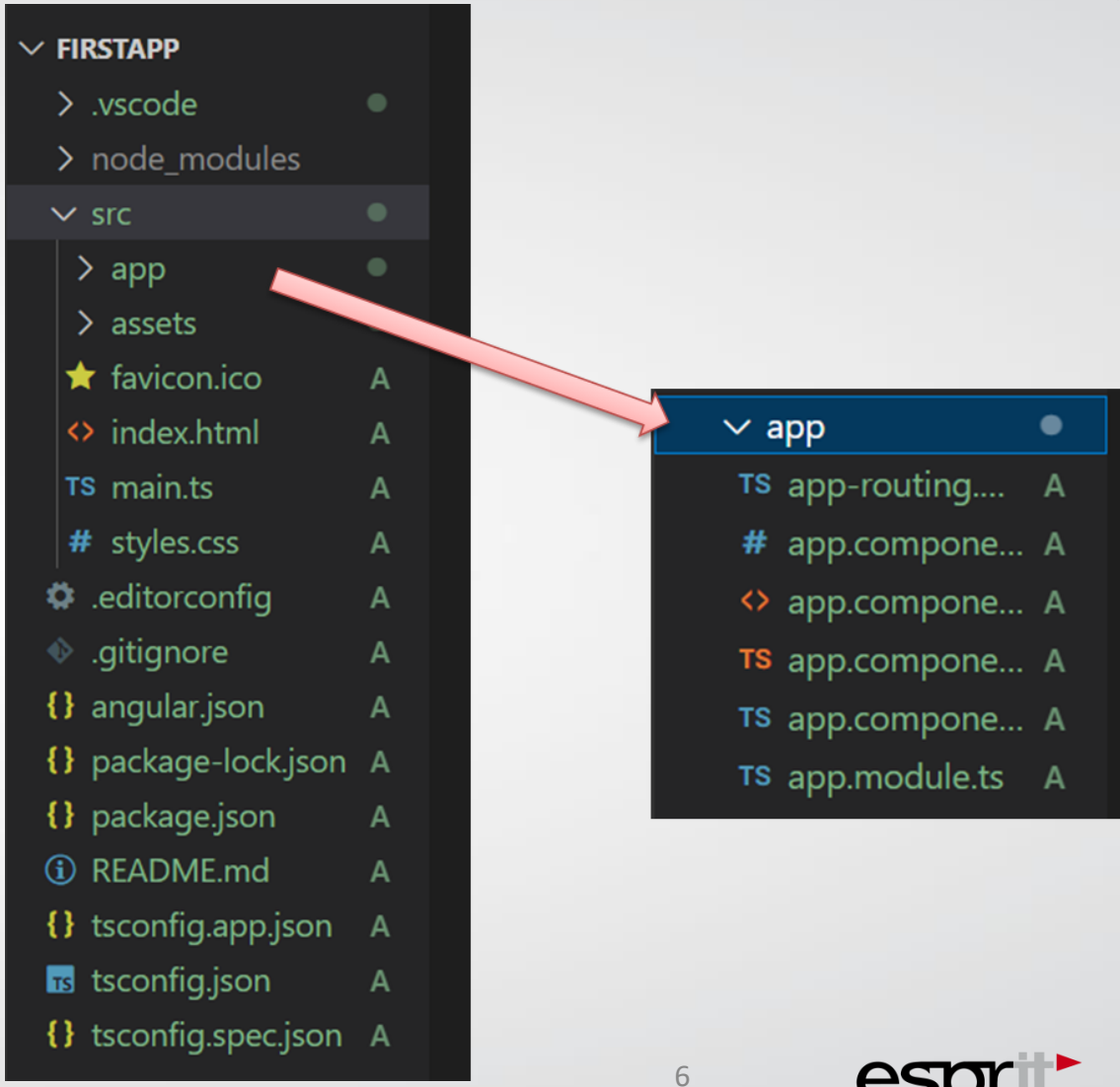
Ref image: https://angular.io/guide/architecture

Architecture Angular

- Modules
- Data-Binding
- Components
- Directives
- Templates
- Services
- MetaData
- Dependency Injection

Angular est modulaire

- 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** »



Modules/NgModules

- 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

Les décorateurs sont des fonctions qui modifient les classes Type Script. Ils sont essentiellement utilisés pour attacher des métadonnées à des classes afin que le système connaisse la configuration de ces classes et leur fonctionnement.

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';


@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Modules/NgModules

- **Declarations**: les composants – directives – pipes utilisés par ce module

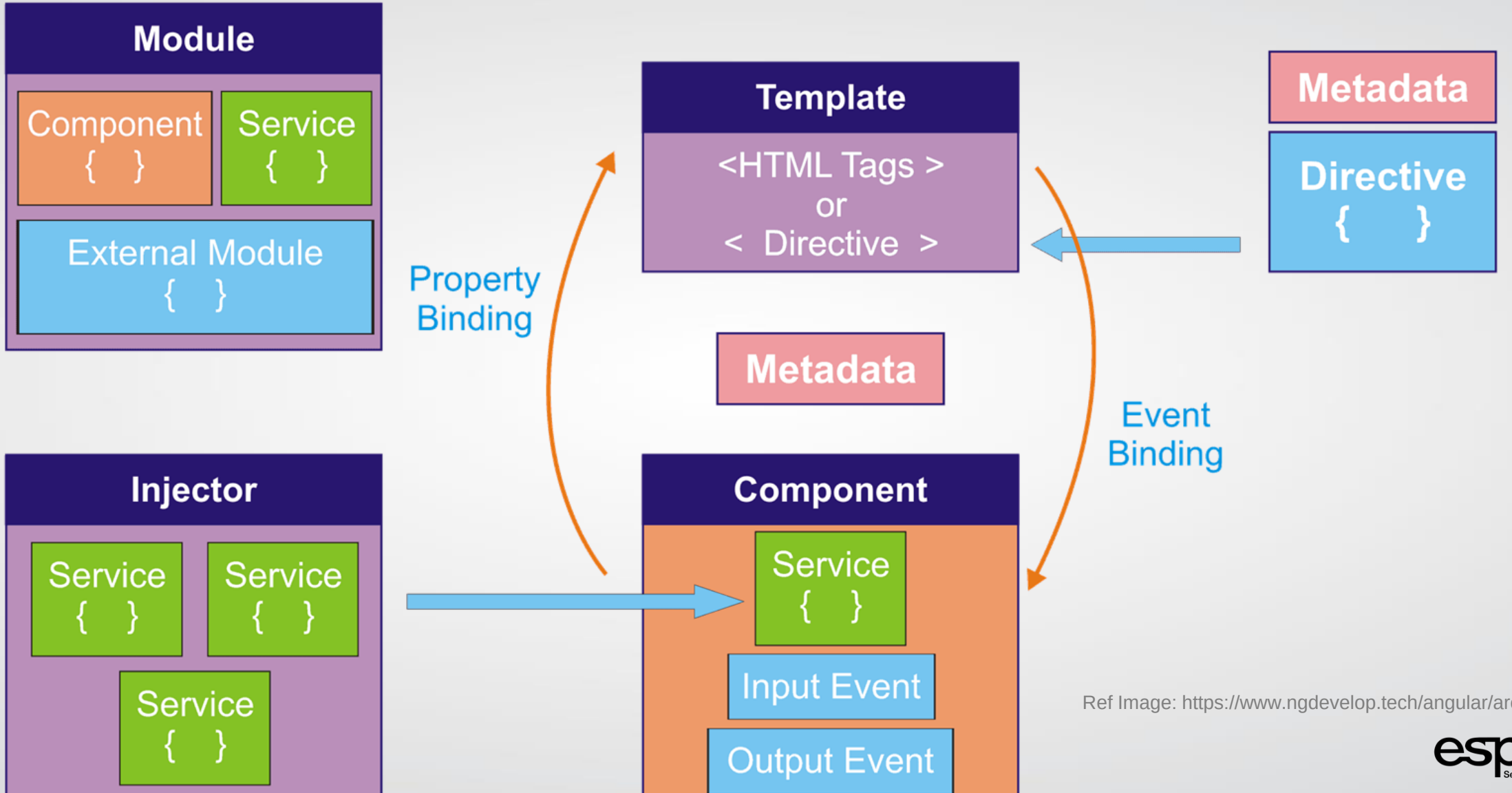
- **Imports**: les modules internes ou externes utilisés dans ce module

- **Providers**: Les services utilisés

- **Bootstrap**: déclare la vue principale de l'application (celle du composant racine).

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';


@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Composition d'un module Angular



Ref Image: https://www.ngdevelop.tech/angular/architecture/

Résumé

- Une application Angular possède au moins un module, le root module, nommé par convention **AppModule.**
- Un module Angular est défini simplement avec une classe *(généralement vide)* et le décorateur NgModule.
- Un module Angular est un mécanisme permettant de :
  - regrouper des composants (mais aussi des services, directives, pipes etc...),
  - définir leurs dépendances
  - définir leur visibilité.

Architecture Angular

- Modules
- Data-Binding
- Components
- Directives
- Templates
- Services
- MetaData
- Dependency Injection

Composant Angular: Définition

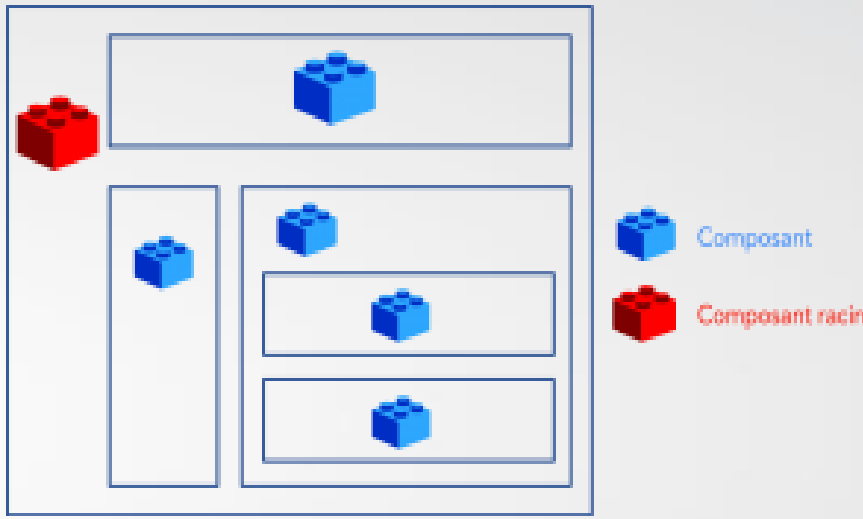
- Composant web est un élément personnalisé et réutilisable.

- Un composant peut correspondre à une page entière ou un élément de l'interface (menu, header, modal, datepicker, etc…).

- Un module peut contenir un ou plusieurs composants.

- Une application a au moins un composant racine: Root Component.

- Une classe décoré par le décorateur @Component (angular/core) qui le définit comme étant un composant et spécifie sa métadata qui indique comment il sera utilisé.



Composant Angular: Composition

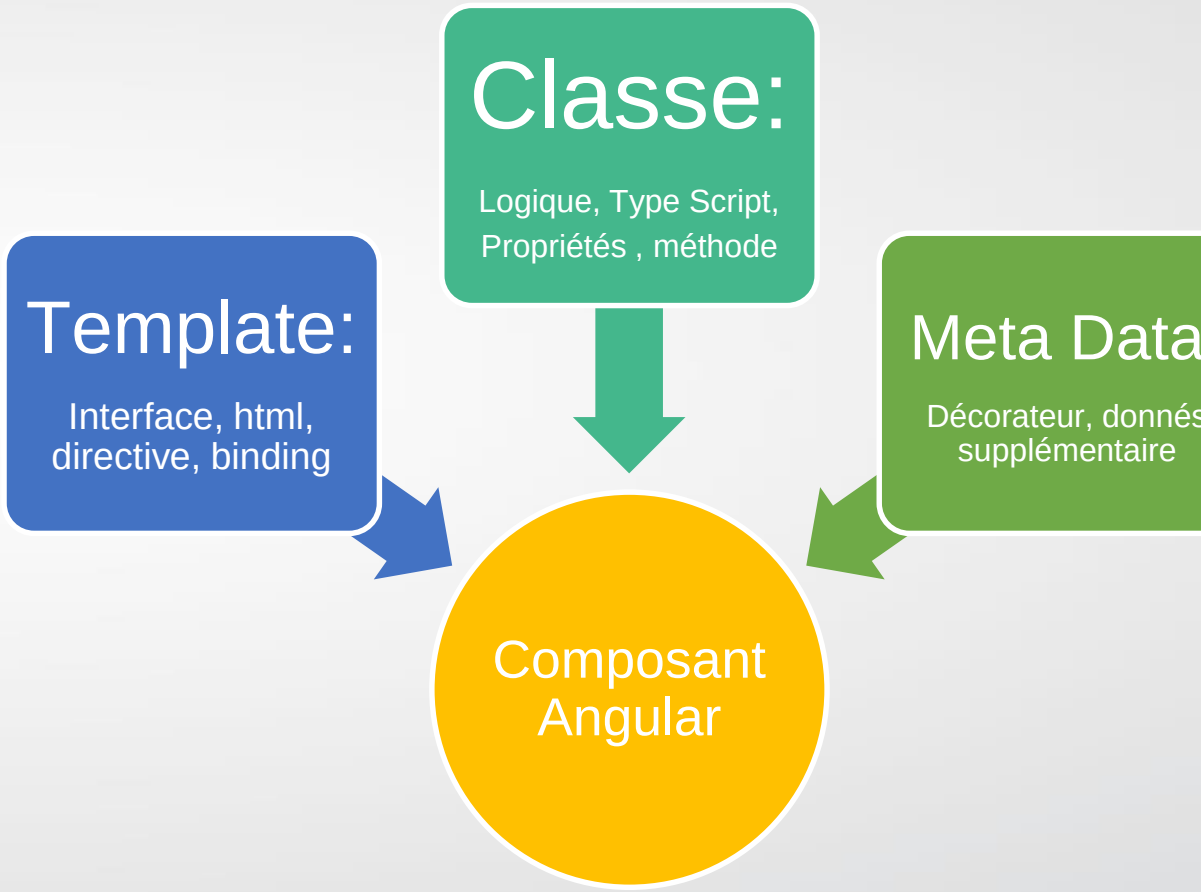
Un composant Angular contient:

.

- Un **Template** contenant l'interface utilisateur en HTML. Nous utiliserons le databinding afin de rendre la vue dynamique,

- **Classe** (class) contenant le code associé à la vue , des propriétés et méthodes logiques qui seront utilisées dans la vue,

- Des **Metadata** nous permettant de définir la classe comme étant un composant Angular (component).



Composant Angular: Création

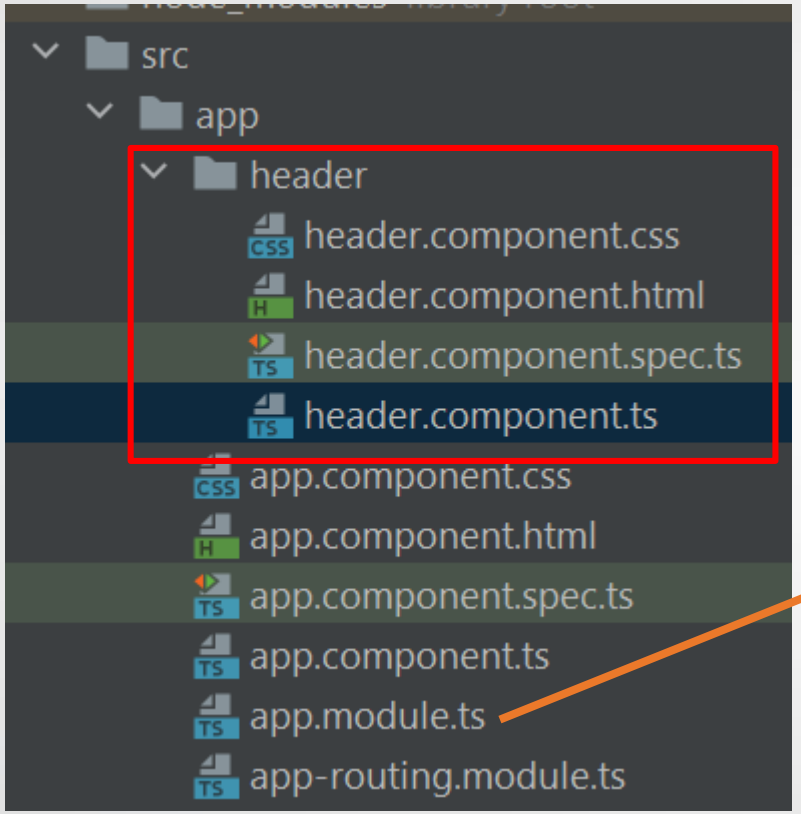
- Générer un nouveau composant dans votre Application angular:

ng generate component header ou ng g c header

```
C:\Users\hp\Desktop\Cours\Angular\Projet\Correction Examen\NomProjet>ng g c header
CREATE src/app/header/header.component.html (21 bytes)
CREATE src/app/header/header.component.spec.ts (628 bytes)
CREATE src/app/header/header.component.ts (275 bytes)
CREATE src/app/header/header.component.css (0 bytes)
UPDATE src/app/app.module.ts (475 bytes)
```

Composant Angular: Création

- Le composant header est dans votre projet:



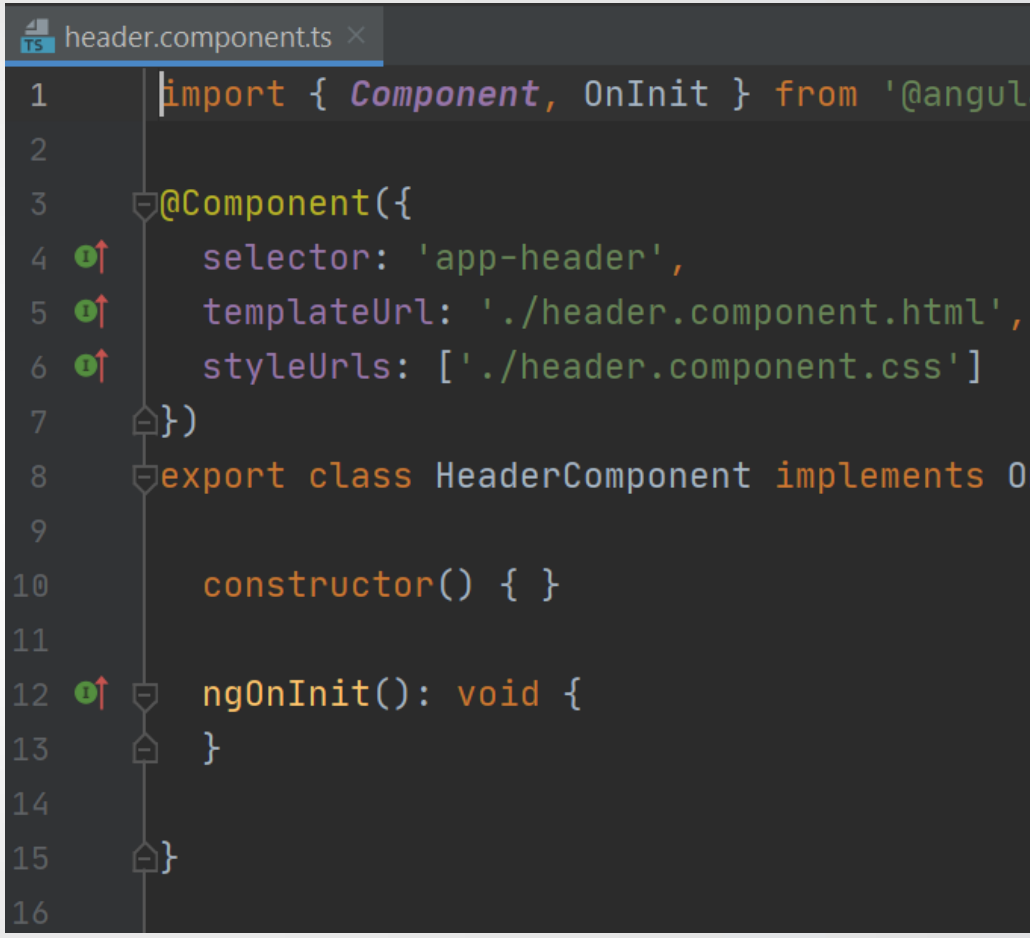

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';
import { HeaderComponent } from './header/header.component';


@NgModule({
  declarations: [
    AppComponent,
    HeaderComponent
  ],
  imports: [
    BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Composant Angular: Declaration

Le décorateur @Component (angular/core) doit contenir au moins les 3 premiers tags:

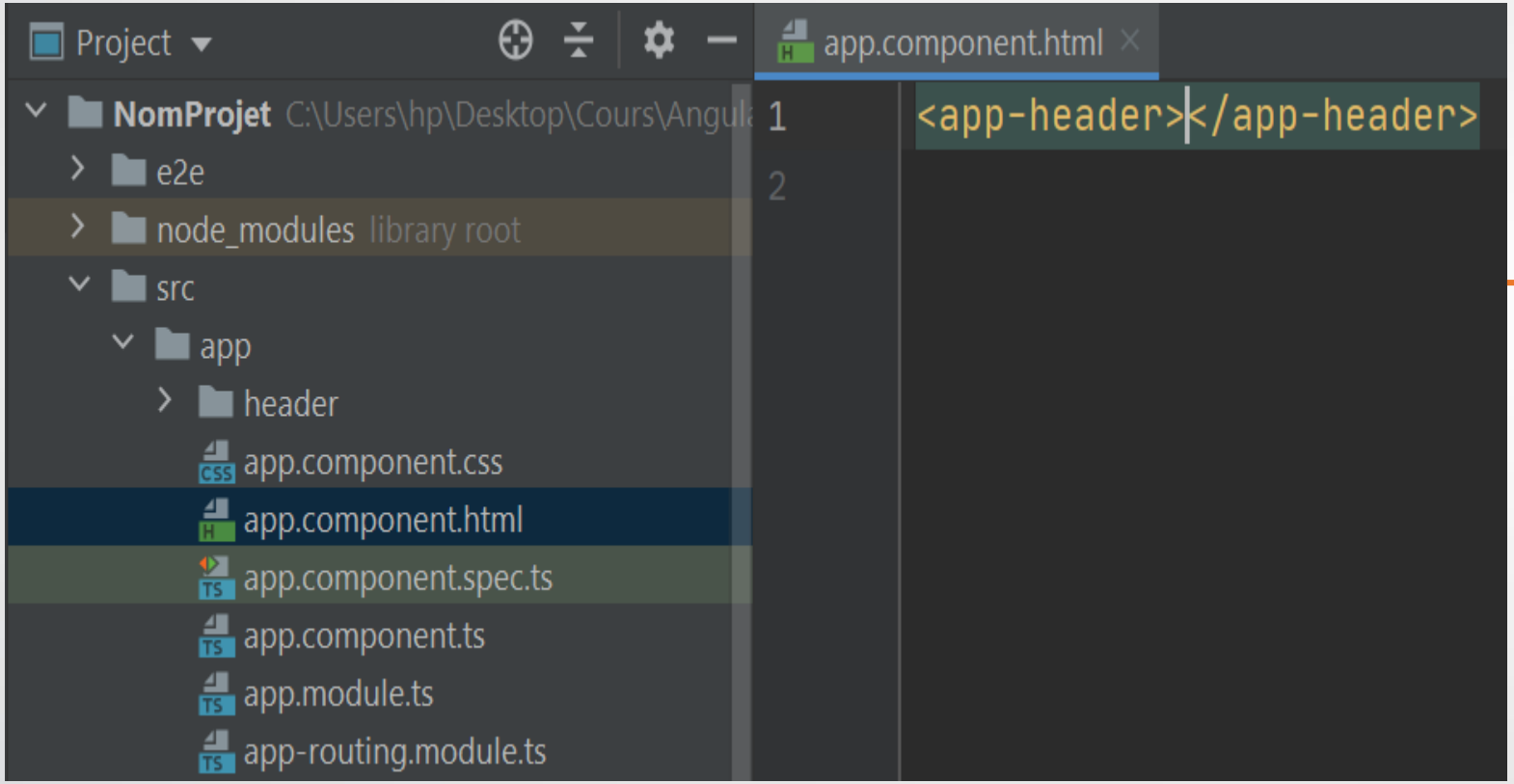
- **Le sélecteur**: Il indique la déclaration qui permet d'insérer le composant dans le document HTML.
- **TemplateUrl**: permet d'associer un fichier externe HTML contenant la structure de la vue du composant
- **StyleUrls**: spécifier les feuilles de styles CSS associées à ce composant.

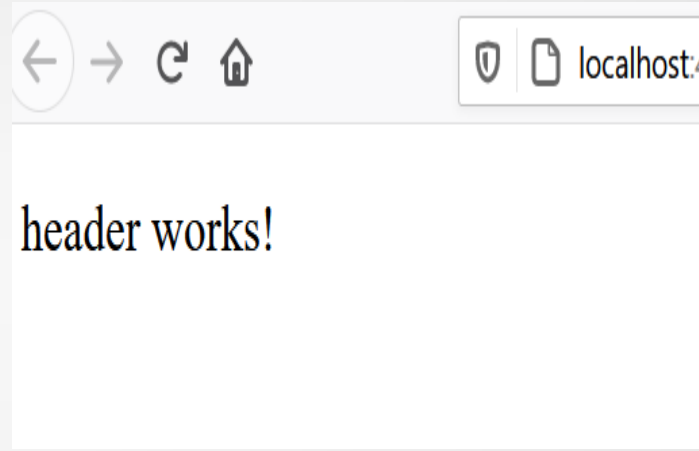
.



Composant Angular: Affichage

- Le sélecteur permet de faire appel à votre composant:



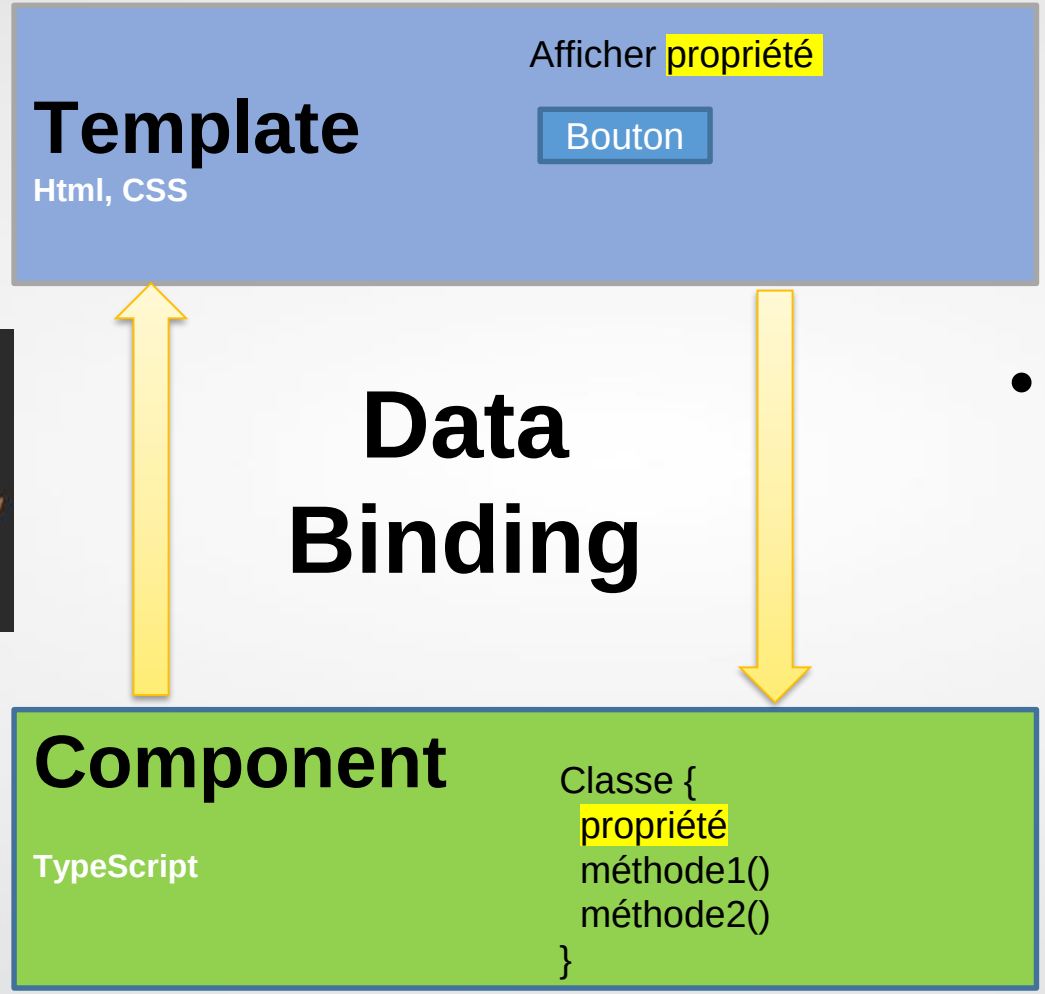


Data-Binding

Liaison entre Template et Component

MetaData

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



- Le databinding est un mécanisme de coordination entre le composant et le Template dans un seul sens ou dans les deux sens.

esprit
Se former autrement

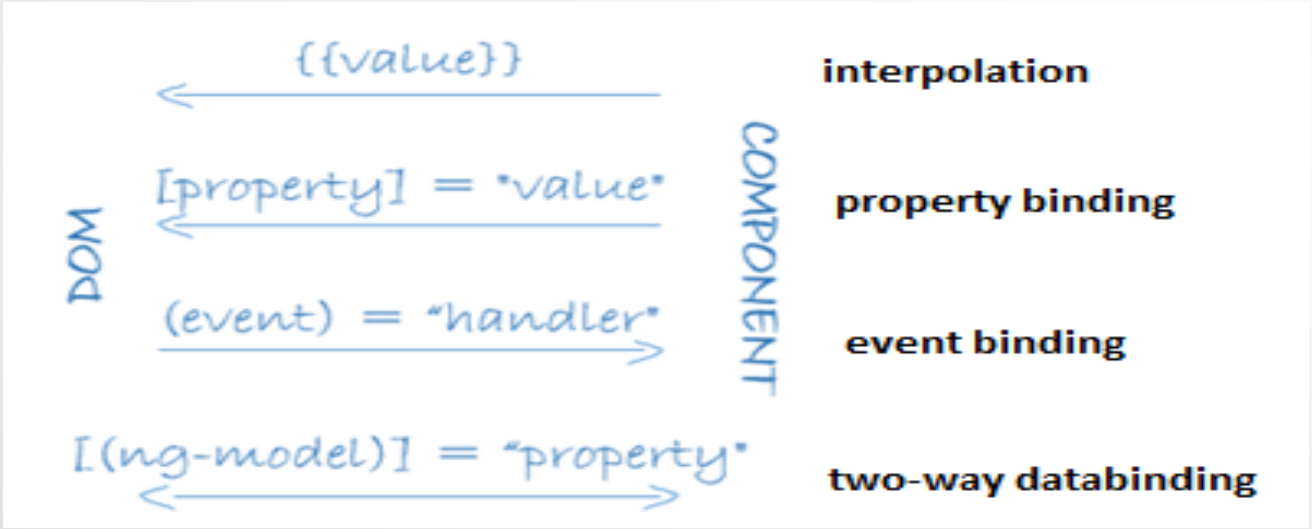
Data-Binding

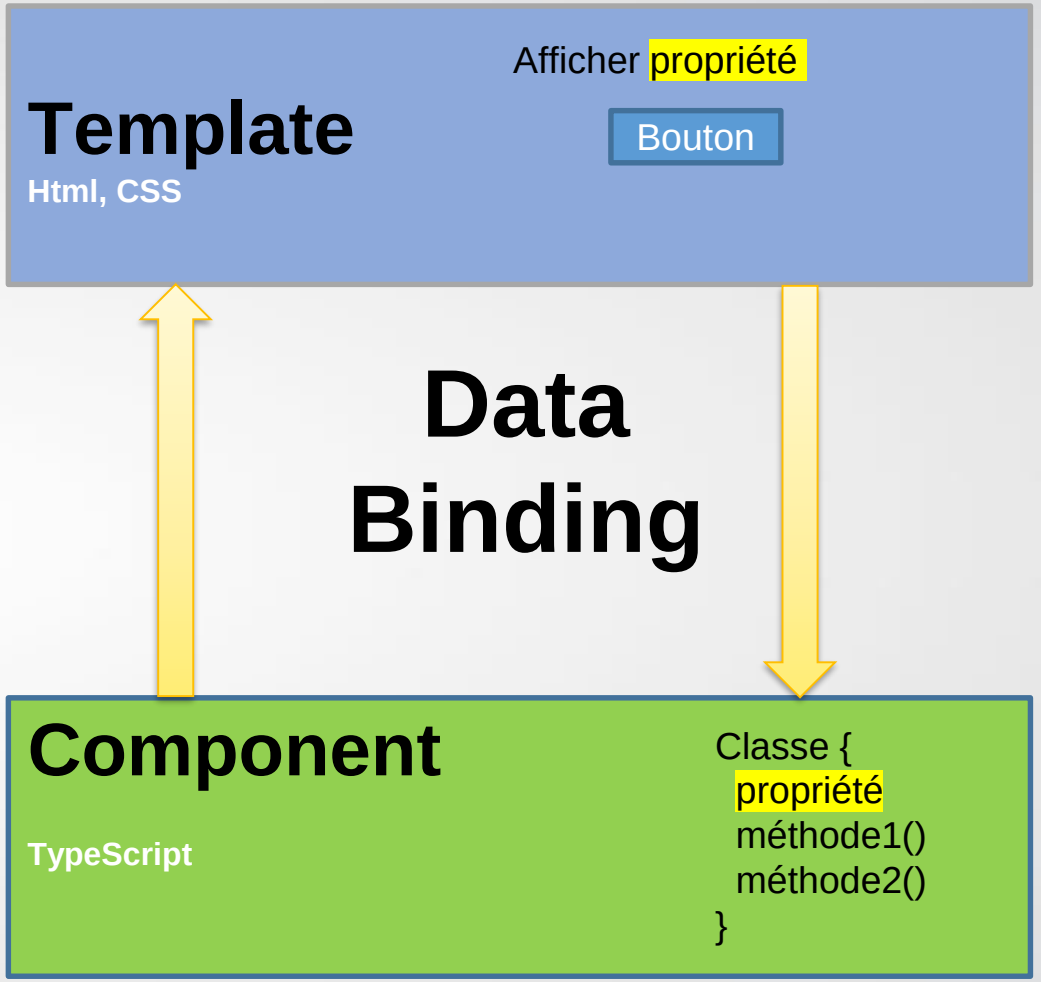
- Modules
- Data-Binding
- Components
- Directives
- Templates
- Services
- MetaData
- Dependency Injection

Data binding: Définition

- Le databinding est un mécanisme de coordination entre le component et le Template dans un seul sens ou dans les deux sens.

- Il existe 4 formes de databinding



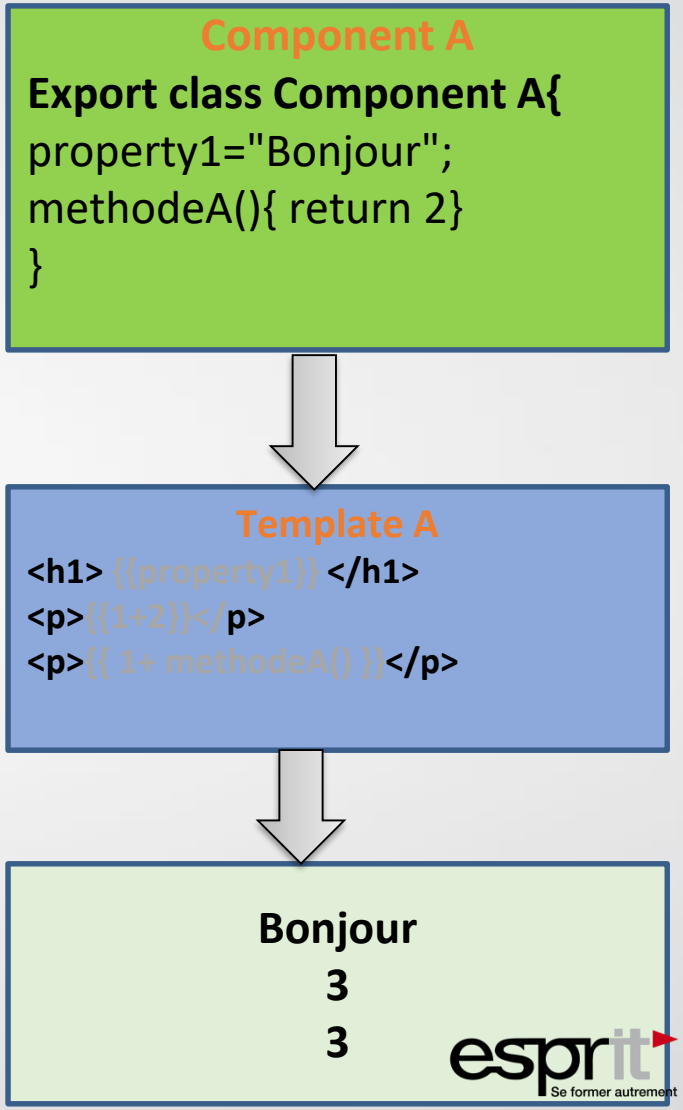


Data binding: Interpolation

- La syntaxe d'interpolation permet d'accéder directement aux propriétés du composant associé.
- L'interpolation permet d'évaluer une expression

{{ expression_template }}

- Angular évalue l'expression ensuite convertit le résultat en chaine de caractères.
- Exemples:
  - ✓ {{ property1 }} : property 1 est une propriété du composant
  - ✓ {{ 1 + 2 }} : le résultat affiché est 3
  - ✓ {{ 1+ methode A() }} : le résultat affiché est la somme de 1 avec e résultat de la méthode getval()



Data binding: Property Binding

Property binding permet de modifier la valeur d'une propriété d'un élément du DOM par la valeur d'une propriété du composant

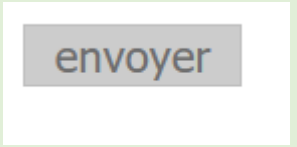
Template A

```
<button [disabled]= 'property'  />
Envoyer
</button>
```

Component A

```
export class A{
  property: boolean = 'true'
}
```

Rendu Final



Data binding: Event Binding

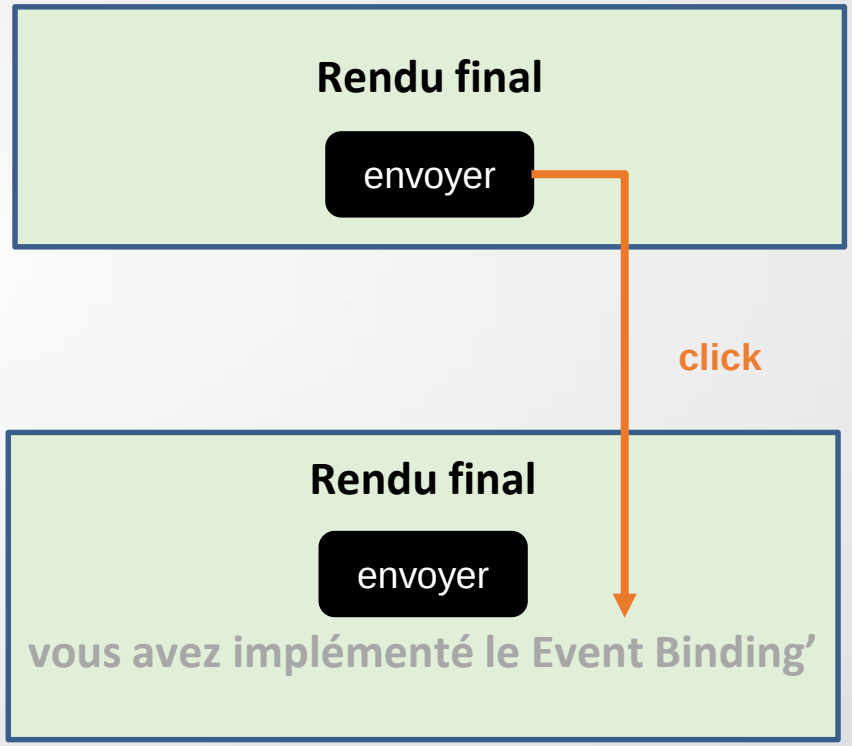
Event binding permet d'appeler une méthode du composant suite à une action faite par l'utilisateur au niveau du template

Template A

```
<button (click)="methodeA()"> envoyer</button>
<p>{{ message}} </p>
```

Component A

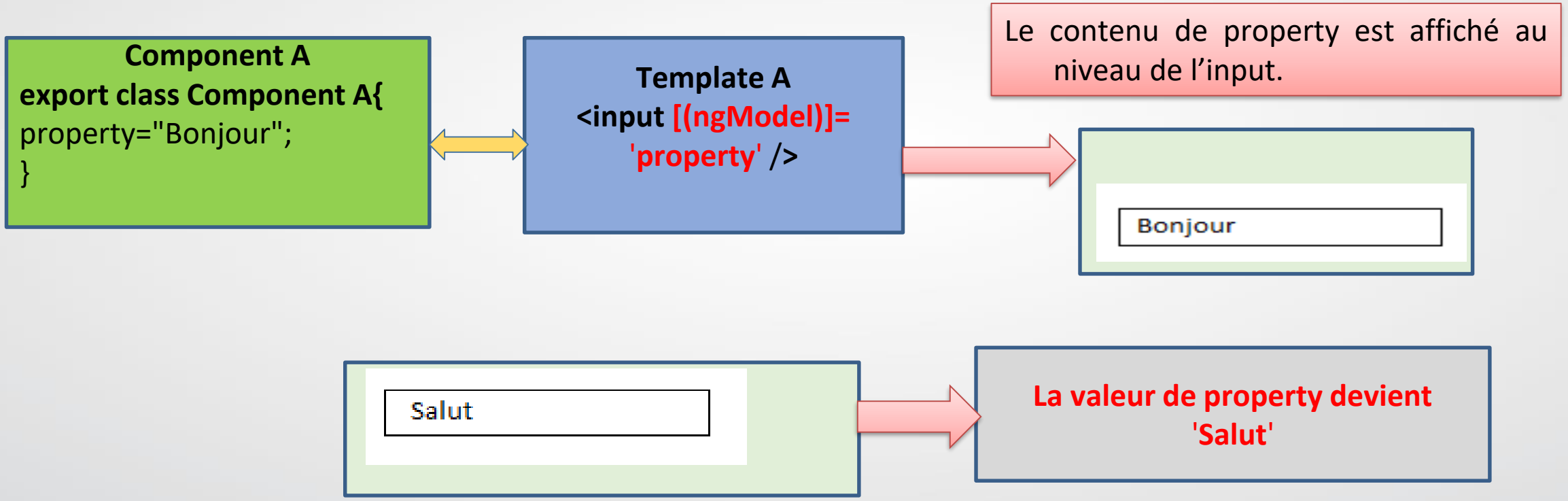
```
export class A {
message: string = ' ';
methodeA(){
   this.message = 'vous avez implémenté le Event
Binding'
}
}
```



esprit
Se former autrement

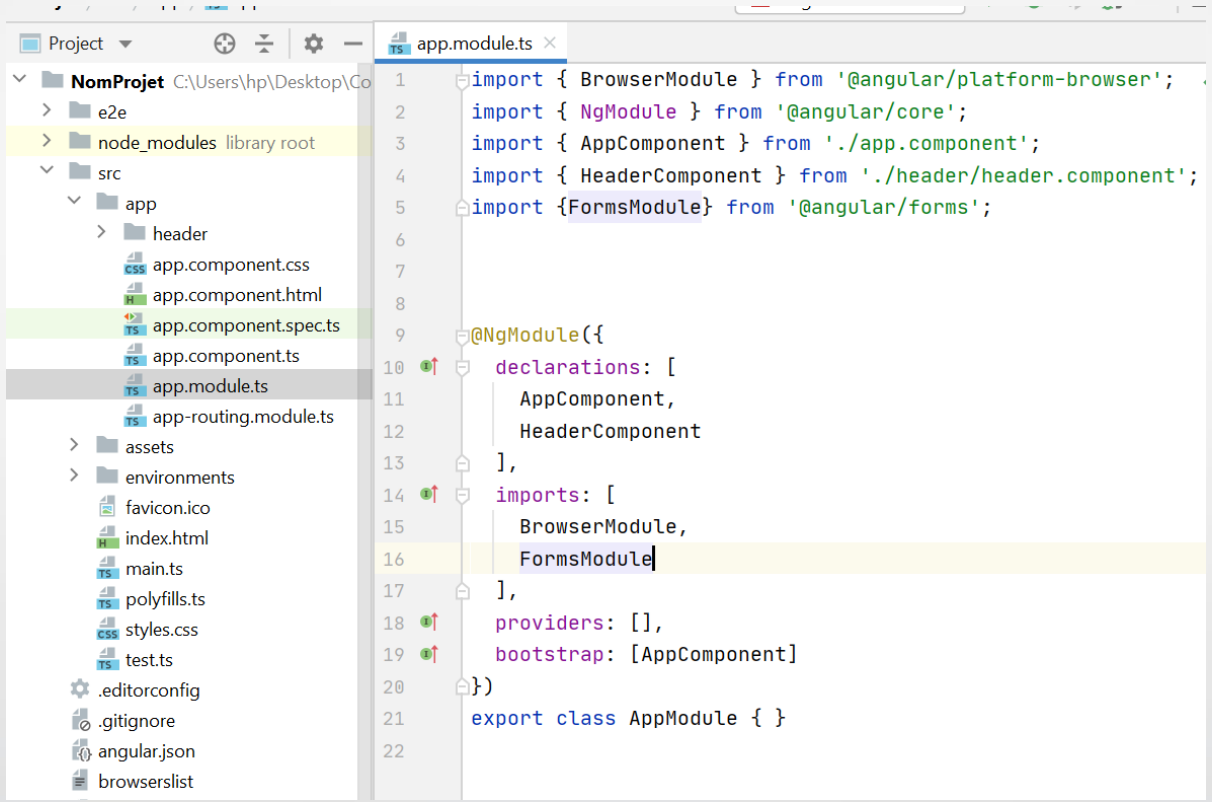
Data binding: Two way data binding

Two-way data binding: permet de récupérer une valeur à partir du template et l'envoyer vers une propriété du composant et vis versa. Ceci se fait grâce à la directive NgModel.



Data binding: Two way data binding

Pour utiliser la directive **NgModel**, il faut importer le module **FormsModule** depuis **@angular/forms** et le déclarer dans la liste des **imports** du module racine **AppModule**.



Les Directives et les pipes

Les directives - Définition

- **Une directive** est un décorateur qui marque une classe comme directive.
- Une directive applique une logique aux éléments du DOM.
- Il existe trois types de directives:
  1. Composant
  2. Directives structurelles
  3. Directives attributs

Les directives – Directives structurelles

- Directives **structurelles**: elles changent le DOM en ajoutant et retirant des éléments au template (NgIF, NgFor, NgSwitch)

```
<ul> <li *ngFor="let customer of customers">{{customer.name}}</li>
</ul>
```

- Customer1
- Customer2
- Customer 3

! Autant de puces que le nombre d'éléments dans la liste customers

Les directives – Directives attributs

- Directives d'**attribut**: change l'apparence et l'attitude d'un élément DOM, composant ou autre directive tels que : NgStyle, NgClass, NgModel, etc

```
<element [ngClass]="string|array|object">
</element>
```

```
<p [ngClass]="'first second'">… </p>
<p [ngClass]="['first', 'second']">… </p>
<p [ngClass]="{'first': true, 'second': true}">… </p>
```

```
<element [ngStyle]="objectExpression">
</element>
```

```
<p [ngStyle]="{'font-style': 'italic'}">… </p>
```

Les directives – Directives personnalisées

Nous pouvons développer notre propre directive en lui attribuant un comportement spécifique

Commande:

```
ng generate directive Nom_de_la_directive
```

Les directives – Directives personnalisées

Exemple:

```
import { Directive, ElementRef, HostListener, Input, Renderer2 } from '@angular/core';

@Directive({
  selector: '[appMyDirective]'
})
export class MyDirectiveDirective {

  @Input('appMyDirective') color: string;
  el: ElementRef;
  private _defaultbgColor = 'white';

  constructor(private _el: ElementRef, private _renderer: Renderer2) {
    this._renderer.setStyle(this._el.nativeElement, 'backgroundColor', this._defaultbgColor);
  }

  @HostListener('mouseenter') onMouseEnter() {
    console.log(this.color);
    this._renderer.setStyle(this._el.nativeElement, 'backgroundColor', this.color);
  }

  @HostListener('mouseleave') onMouseLeave() {
    this._renderer.setStyle(this._el.nativeElement, 'backgroundColor', this._defaultbgColor);
  }

}
```

<p [appMyDirective]=" 'red' "> text here </p>

Les pipes

- Les pipes transforment les données avant de les afficher.
- L’opérateur pipe passe le résultat d'une expression sur la gauche à une fonction pipe sur la droite tels que: DatePipe, UpperCasePipe, LowerCasePipe, ….

```
{{item.helpDate | date:'longDate'}}
```

- Vous pouvez chaîner des expressions via plusieurs pipes

```
{{title | uppercase | lowercase}}
```

Cycle de vie d’un composant

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 ».
- A chaque phase on peut implémenter une logique.
- Ces méthodes se trouvent dans des

interfaces dans la librairie « @angular/core »

Cycle de vie d’un composant

- 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/

Merci de votre attention