

Par NIDHAL JELASSI
nidhal.jelassi@fsegt.utm.tn

PROGRAMMATION WEB AVANCÉE

ANGULAR

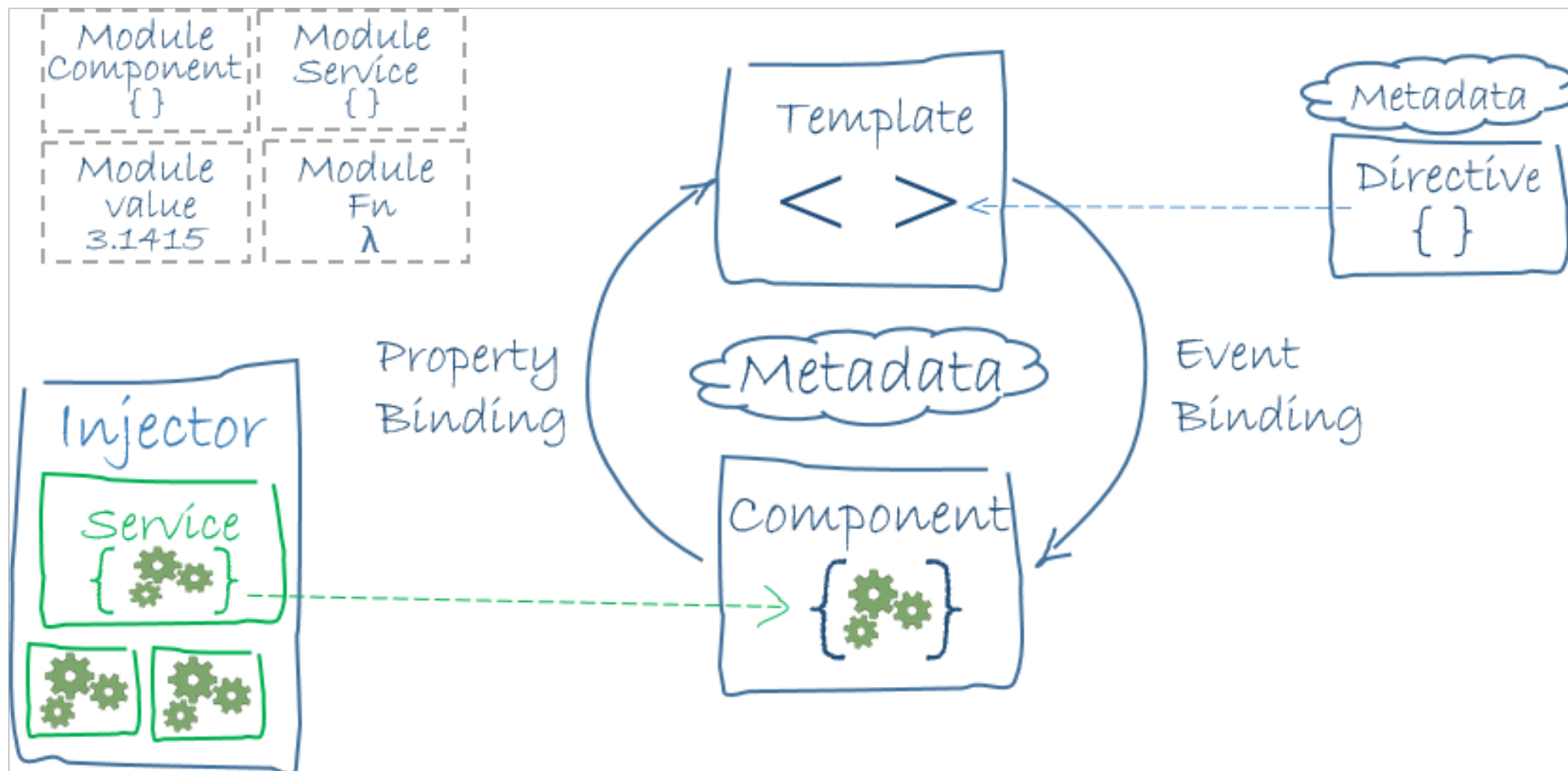


Chapitre 2 : Les composants

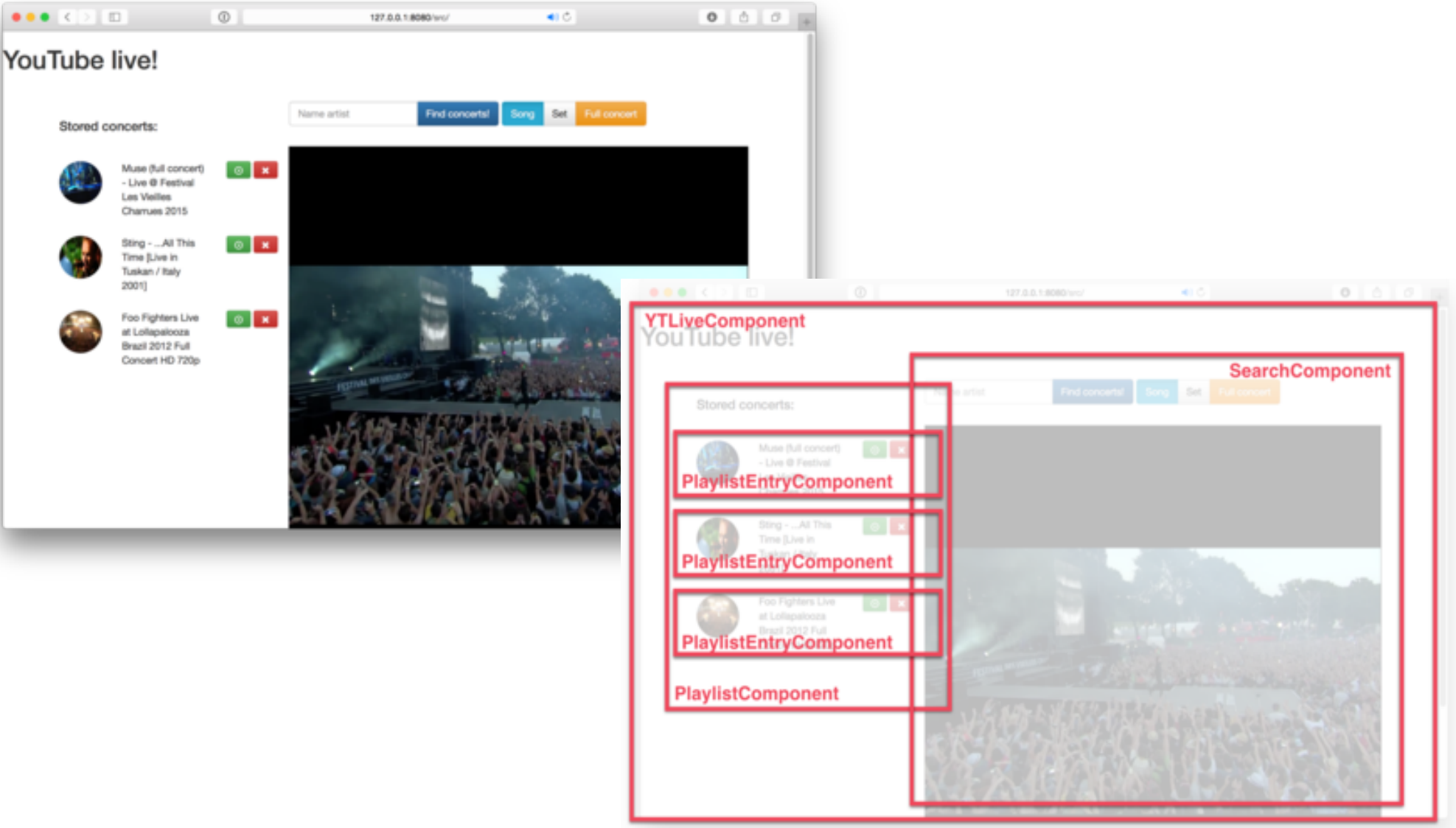
OBJECTIFS

- ▶ Comprendre la définition du composant
- ▶ Assimiler et pratiquer la notion de Binding
- ▶ Gérer les interactions entre composants.

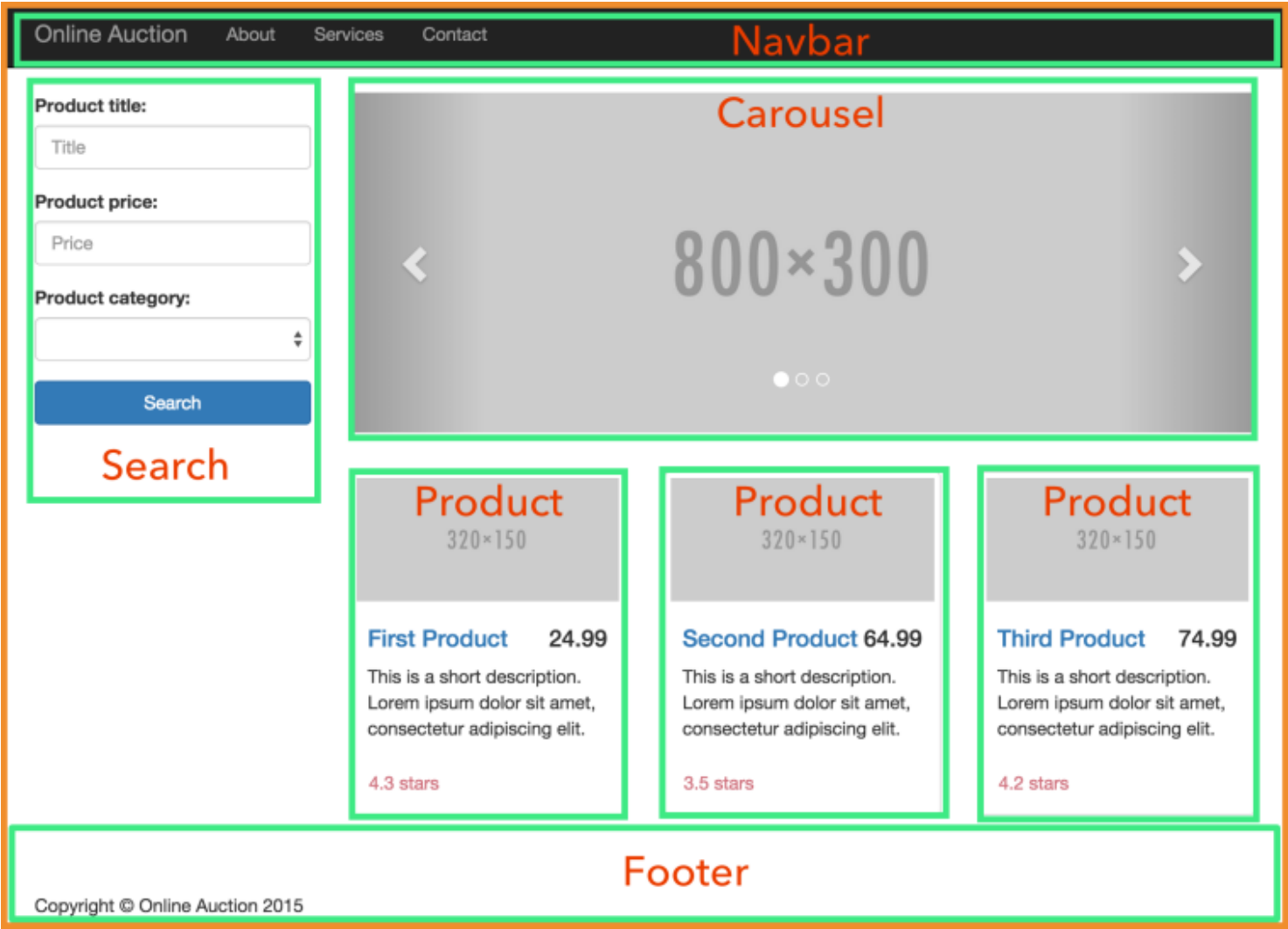
ANGULAR EN... UN SCHÉMA



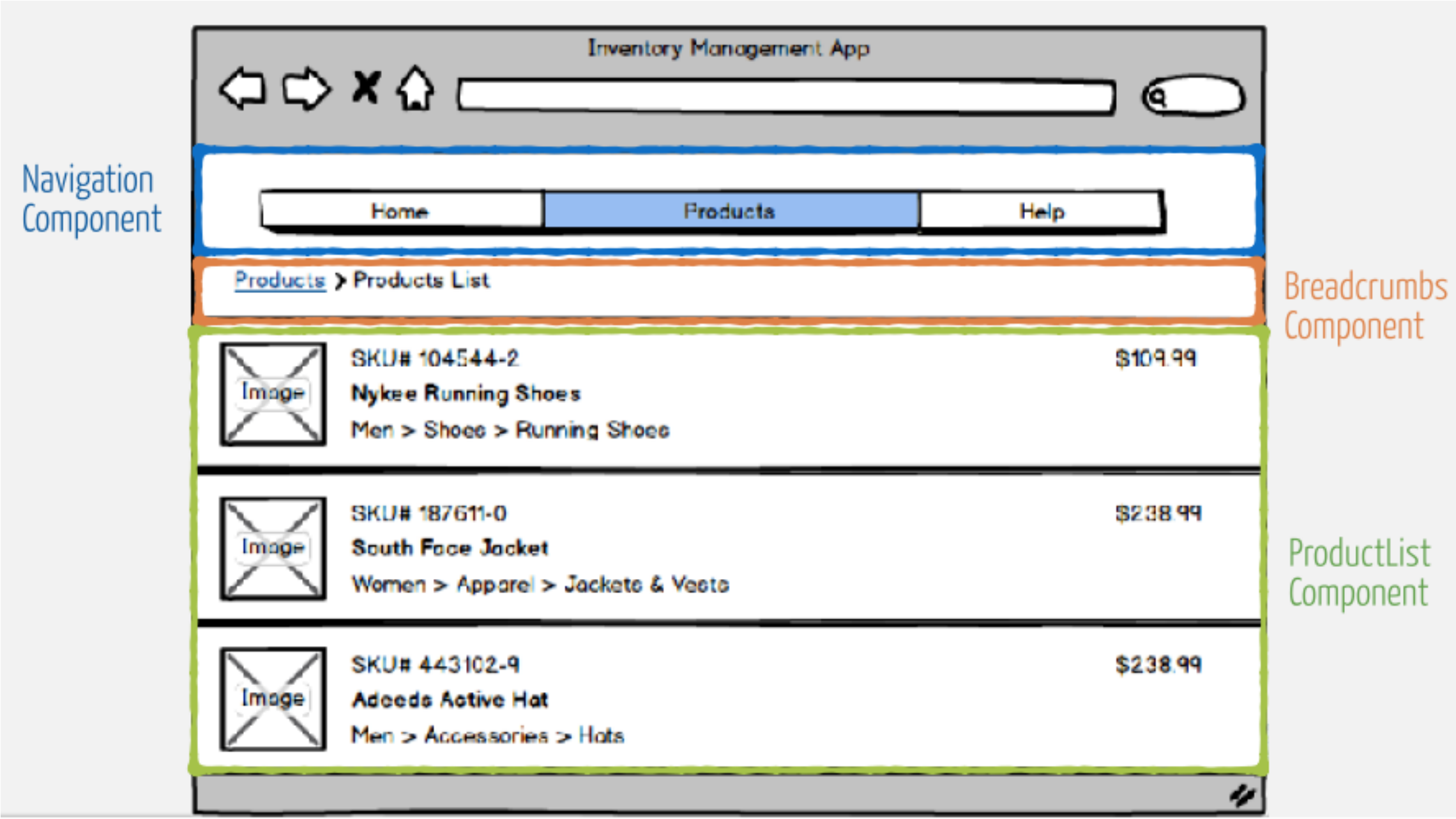
EXAMPLE 1



EXEMPLE 2



EXAMPLE 3



QU'EST CE QU'UN COMPOSANT (COMPONENT) ?

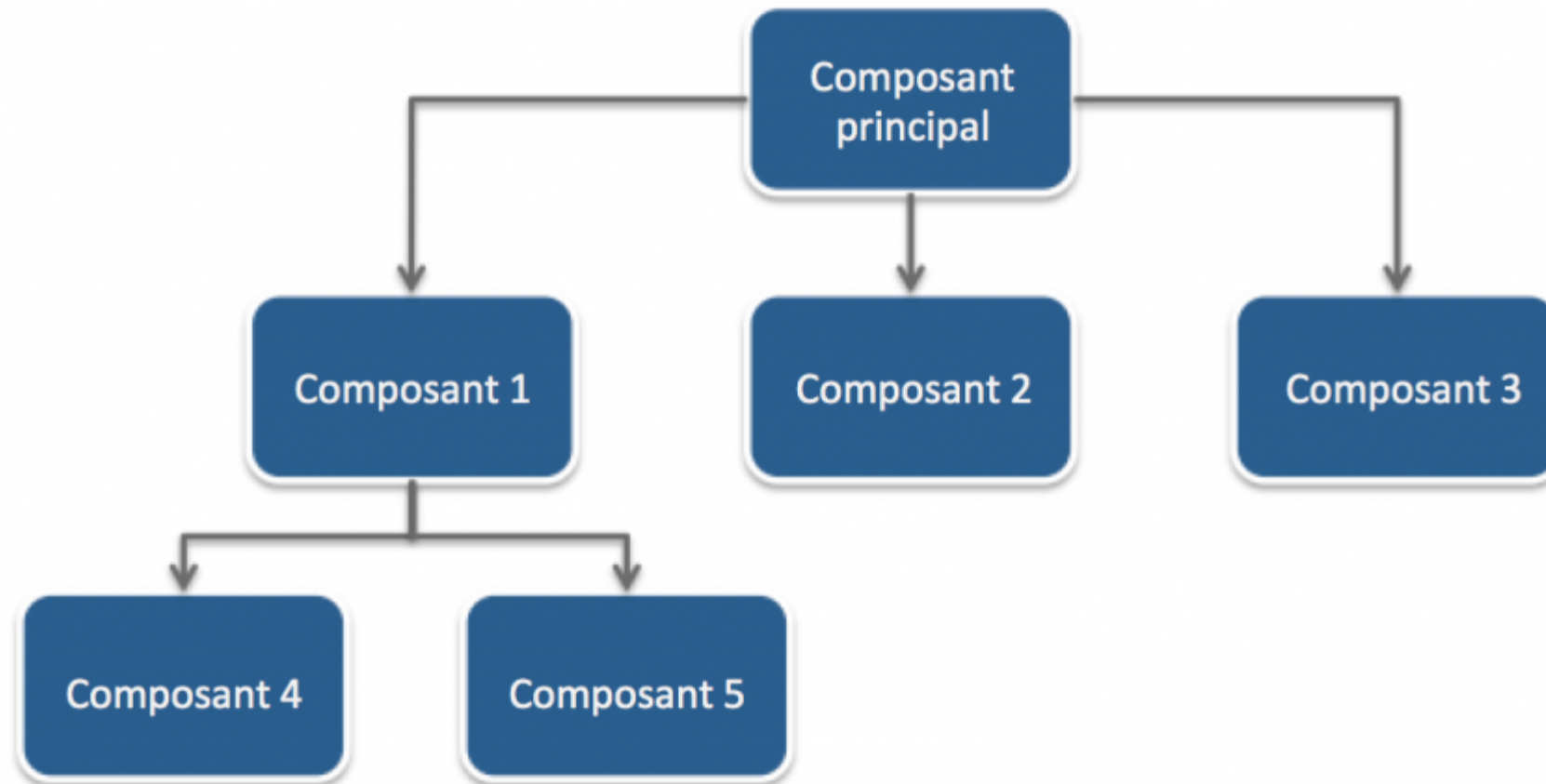
- ▶ Un composant est une **classe** qui permet de gérer une **vue**. Il se charge uniquement de cette vue-là.
- ▶ Plus simplement, un composant est un fragment HTML géré par une classe JS (component en angular et controller en angularJS)
- ▶ Une application Angular est un arbre de composants
- ▶ La racine de cet arbre est l'application lancée par le navigateur au lancement.

QU'EST CE QU'UN COMPOSANT (COMPONENT) ?

- ▶ Ainsi, un composant est :
- ▶ Composable (évidemment puisque c'est un composant 😄)
- ▶ Réutilisable
- ▶ Hiérarchique (n'oublier pas c'est un arbre)

NB : Dans le reste du cours les mots composant et component représentent toujours un composant Angular.

ARBRE DE COMPOSANTS



VOTRE PREMIER COMPOSANT

- ▶ Chargement de la classe **Component**
- ▶ Le décorateur **@Component** permet d'ajouter un comportement à notre classe et de spécifier que c'est un Composant Angular.
- ▶ c'est un Composant Angular.
 - ▶ **selector** permet de spécifier le tag (nom de la balise) associé ce composant
 - ▶ **templateUrl**: spécifie l'url du template associé au composant
 - ▶ **styleUrls**: tableau des feuilles de styles associé à ce composant
- ▶ **Export** de la classe afin de pouvoir l'utiliser

```
import { Component } from '@angular/core';

@Component({
  selector: 'liste-cours',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Angular App for ISIE 1st year';
}
```

CRÉATION D'UN COMPOSANT

Pour créer un composant, il existe 2 méthodes possibles :

▶ Manuelle

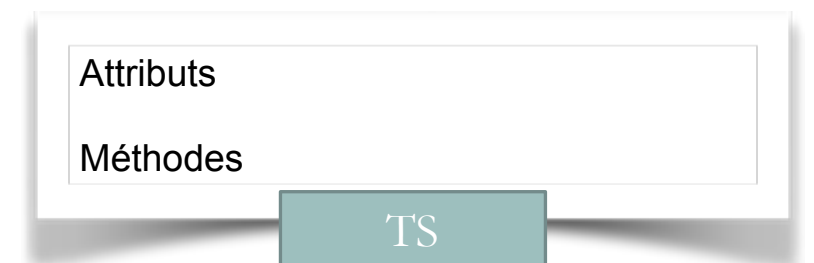
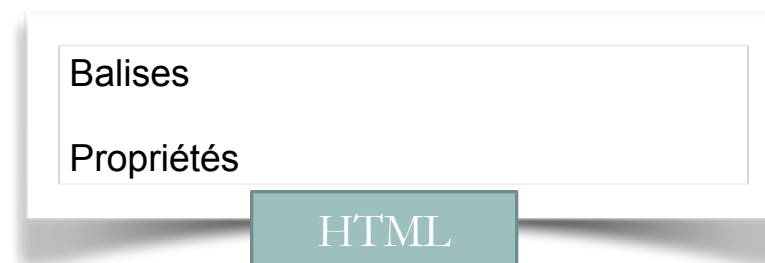
- ▶ Créer la classe
- ▶ Importer Component
- ▶ Ajouter l'annotation et l'objet qui la décore
- ▶ Ajouter le composant dans le AppModule(app.module.ts) dans l'attribut declarations

▶ Cli

- ▶ Avec la commande **ng generate component my-new-component** ou son raccourci **ng g c my-new-component**

PROPERTY BINDING

- ▶ Binding unidirectionnel.
- ▶ Permet aussi de récupérer dans le DOM des propriétés du composant.
- ▶ La propriété liée au composant est interprétée avant d'être ajoutée au Template.
- ▶ Deux possibilités pour la syntaxe:
 - ▶ [propriété]
 - ▶ **bind**-propriété



PROPERTY BINDING

```
<div [style.backgroundColor]="color">
  Color
</div>

<input [(ngModel)]="color"
       type="text"
       class="form-control"
>
le contenu de la propriété color est {{color}}
<button (click)="loggerMesData()">log data</button>
<br>
<a (click)="goToCv()">Go to Cv</a>
```

HTML

```
@Component({
  selector: 'app-color',
  templateUrl: './color.component.html',
  styleUrls: ['./color.component.css'],
  providers: [PremierService]
})
export class ColorComponent implements OnInit {
  color = 'red';
  constructor() { }

  ngOnInit() {}
  processReq(message: any) {
    alert(message);
  }
  loggerMesData() {
    this.premierService.logger('test');
  }
  goToCv() {
    const link = ['cv'];
    this.router.navigate(link);
  }
}
```

TS

EVENT BINDING

- ▶ Binding unidirectionnel.
- ▶ Permet d'interagir du DOM vers le composant.
- ▶ L'interaction se fait à travers les événements.
- ▶ Deux possibilités pour la syntaxe :
 - ▶ (evenement)
 - ▶ **on**-evenement

PROPERTY BINDING ET EVENT BINDING

```
import { Component } from '@angular/core';

@Component({
  selector: 'tests',
  templateUrl: './tests.component.html',
  styleUrls: []
})
export class TestsComponent {
  nom:string = 'Nidhal Jelassi';
  age: number = 35;
  adresse:string = 'Quelque part...';

  getName() {
    return this.nom;
  }
  setName(nvNom) {
    this.nom = nvNom;
    console.log(this.nom);
  }
}
```

Component

```
<hr>
Nom : {{ nom }} <br>
Age : {{ age }} <br>
Adresse : {{ adresse }} <br>

<input #name [value]="getName()">
<br>
<button (click)="setName(name.value)">
  Modifier Nom</button>
<hr>
```

Template

EXERCICE D'APPLICATION

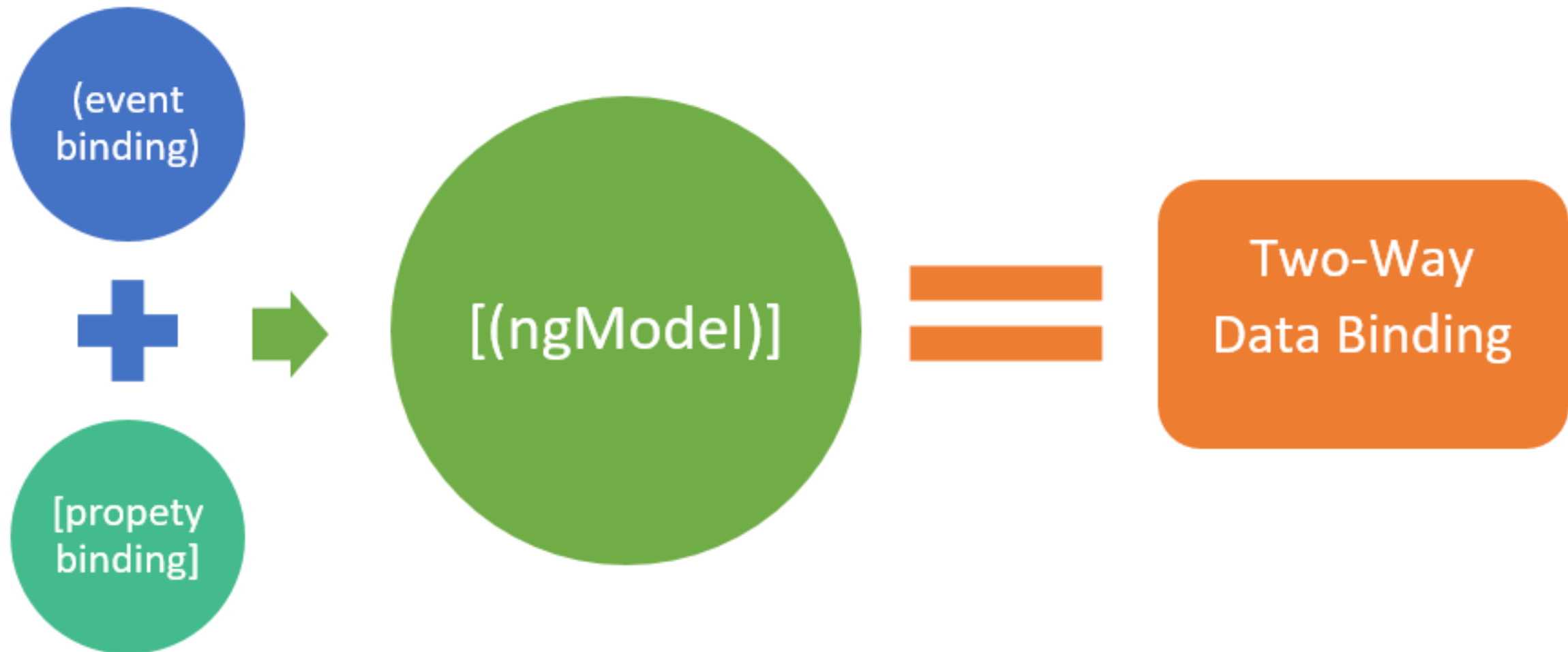
- ▶ Créer un nouveau composant. Ajouter y un Div et un input de type texte.
- ▶ Fait en sorte que lorsqu'on écrit une couleur dans l'input, ca devienne la couleur du Div.
- ▶ Ajouter un bouton. Au click sur le bouton, il faudra que le Div reprenne sa couleur par défaut.
- ▶ Ps : pour accéder au contenu d'un élément du DOM, utilisez #nom dans la balise et accéder ensuite à cette propriété via ce nom.
- ▶ Pour accéder à une propriété de style d'un élément on peut binder la propriété **[style.nomPropriété]** : exemple **[style.backgroundColor]**

TWO-WAY BINDING

- ▶ Binding **Bi-directionnel**
- ▶ Permet d'interagir du DOM vers le composant et du composant vers le DOM.
- ▶ Se fait à travers la directive **ngModel** (*on reviendra sur le concept de directive plus en détail*)
- ▶ Syntaxe :
- ▶ **[(ngModel)]**=property

N.B : Afin de pouvoir utiliser ngModel vous devez importer le **FormsModule** du **@angular/forms** dans **app.module.ts**

PROPERTY BINDING ET EVENT BINDING



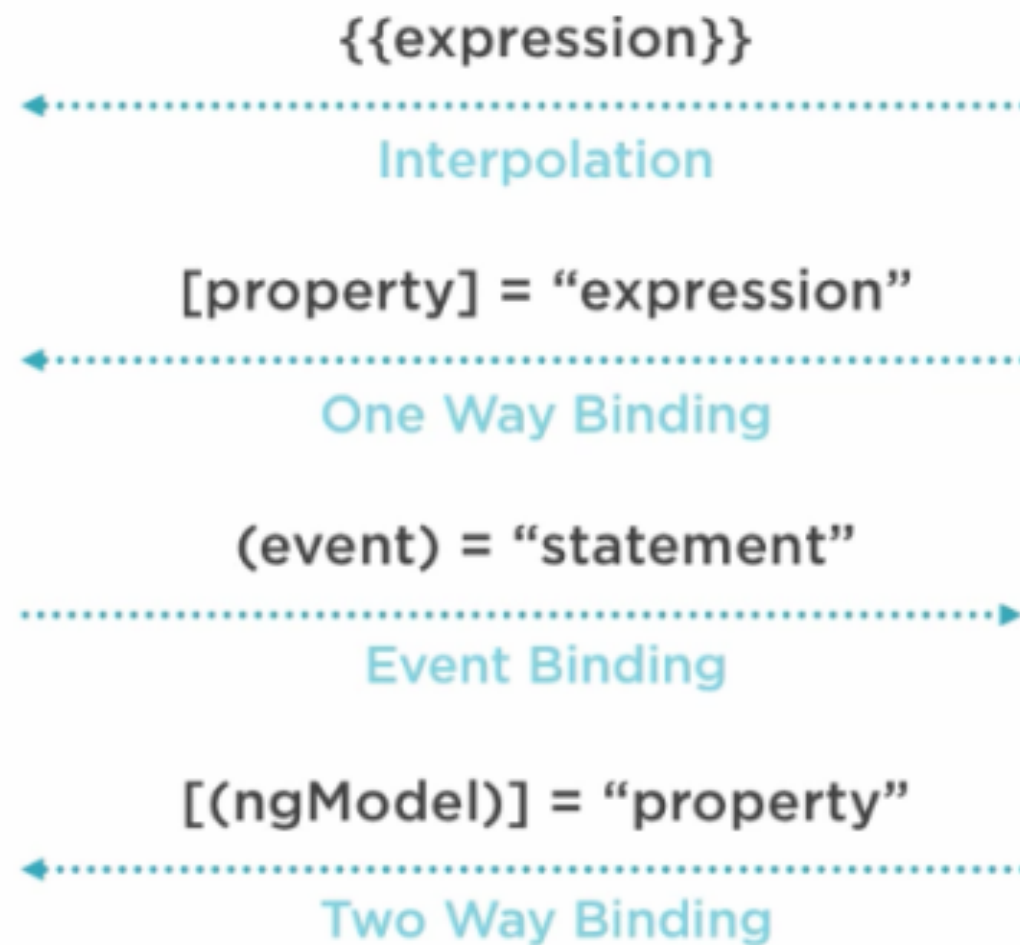
```
<hr>  
Change me <input [(ngModel)]="nom">  
<br>My new name is {{nom}}
```

Template

RÉCAPITULONS... AVANT D'AVANCER



DOM



Component

MVVM DESIGN PATTERN

- ▶ Avec le two-ways-binding, Angular adopte une architecture MVVM.
- ▶ Le contrôleur (le C dans MVC) est remplacé par le ViewModel (la VM dans MVVM). Le ViewModel, qui est comme un contrôleur, est responsable du maintien de la relation entre la vue et le modèle.
- ▶ Cependant, la différence ici est que, si nous mettons à jour quoi que ce soit en vue, il est mis à jour dans le modèle. De même, changez quoi que ce soit dans le modèle, cela apparaît dans la vue.
- ▶ En MVC, les informations de la vue sont envoyés au serveur pour mettre à jour le modèle. Or, dans Angular, nous fonctionnons côté client, nous pouvons donc mettre à jour les objets associés en temps réel.

APPLICATION

Nous allons créer un aperçu de carte visite. Pour cela, vous devez :

- ▶ Créer un composant.
- ▶ Préparer une interface permettant de saisir d'un côté les données à insérer dans une carte visite. De l'autre côté et instantanément les données de la carte seront mis à jour.
- ▶ Préparer l'affichage de la carte visite. Vous pouvez utiliser ce thème gratuit : <https://www.creative-tim.com/product/rotating-css-card>

CYCLE DE VIE D'UN COMPOSANT

- ▶ Chaque composant a un cycle de vie géré par Angular lui-même.
 1. Angular crée le composant
 2. S'occupe de l'afficher
 3. Crée et affiche ses composants fils
 4. Vérifie quand les données des propriétés changent
 5. Détruit les composants, avant de les retirer du DOM quand cela est nécessaire.

CYCLE DE VIE D'UN COMPOSANT

- ▶ Angular offre aux développeurs la possibilité d'agir sur ces moments clefs (les phases du cycle de vie) quand ils se produisent, en implémentant une ou plusieurs interfaces, pour chacun des événements disponibles.
- ▶ Ces interfaces sont disponibles dans la librairie **core** d'Angular.
- ▶ Chaque interface permettant d'interagir sur le cycle de vie d'un composant fournit **une et une seule** méthode, dont le nom est le nom de l'interface préfixé par **ng**. Par exemple, l'interface **OnInit** fournit la méthode **ngOnInit**, et permet de définir un comportement lorsque le composant est initialisé.

CYCLE DE VIE D'UN COMPOSANT

- ▶ Ci-dessous la liste des méthodes disponibles pour interagir avec le cycle de vie d'un composant, **dans l'ordre chronologique** du moment où elles sont appelées par Angular.
- 1. **ngOnChanges** : C'est la méthode appelée en premier lors de la création d'un composant, avant même *ngOnInit*, et à chaque fois que Angular détecte que les valeurs d'une propriété du composant sont modifiées. La méthode reçoit en paramètre un objet représentant les valeurs actuelles et les valeurs précédentes disponibles pour ce composant.
- 2. **ngOnInit** : Cette méthode est appelée juste après le premier appel à *ngOnChanges*, et **elle initialise le composant après qu'Angular ait initialisé les propriétés du composant.**

CYCLE DE VIE D'UN COMPOSANT

3. **ngDoCheck**: On peut implémenter cette interface pour étendre le comportement par défaut de la méthode *ngOnChanges*, afin de pouvoir détecter et agir sur des changements qu'Angular ne peut pas détecter par lui-même.
4. **ngAfterViewInit**: Cette méthode est appelée juste après la mise en place de la vue d'un composant, et des vues de ses composants fils s'il en a. A partir de ce moment, les propriétés initialisées par `@ViewChild` et `@ViewChildren` sont valorisées. Il n'est appelé qu'une seule fois après `ngAfterContentChecked`.

CYCLE DE VIE D'UN COMPOSANT

5. **ngOnDestroy**: Appelée en dernier, cette méthode est appelée avant qu'Angular ne détruise et ne retire du DOM le composant. **Ex** : *Cela peut se produire lorsqu'un utilisateur navigue d'un composant à un autre par exemple. Afin d'éviter les fuites de mémoire, c'est dans cette méthode que nous effectuerons un certain nombre d'opérations afin de laisser l'application "propre" (nous détacherons les gestionnaires d'événements par exemple).*



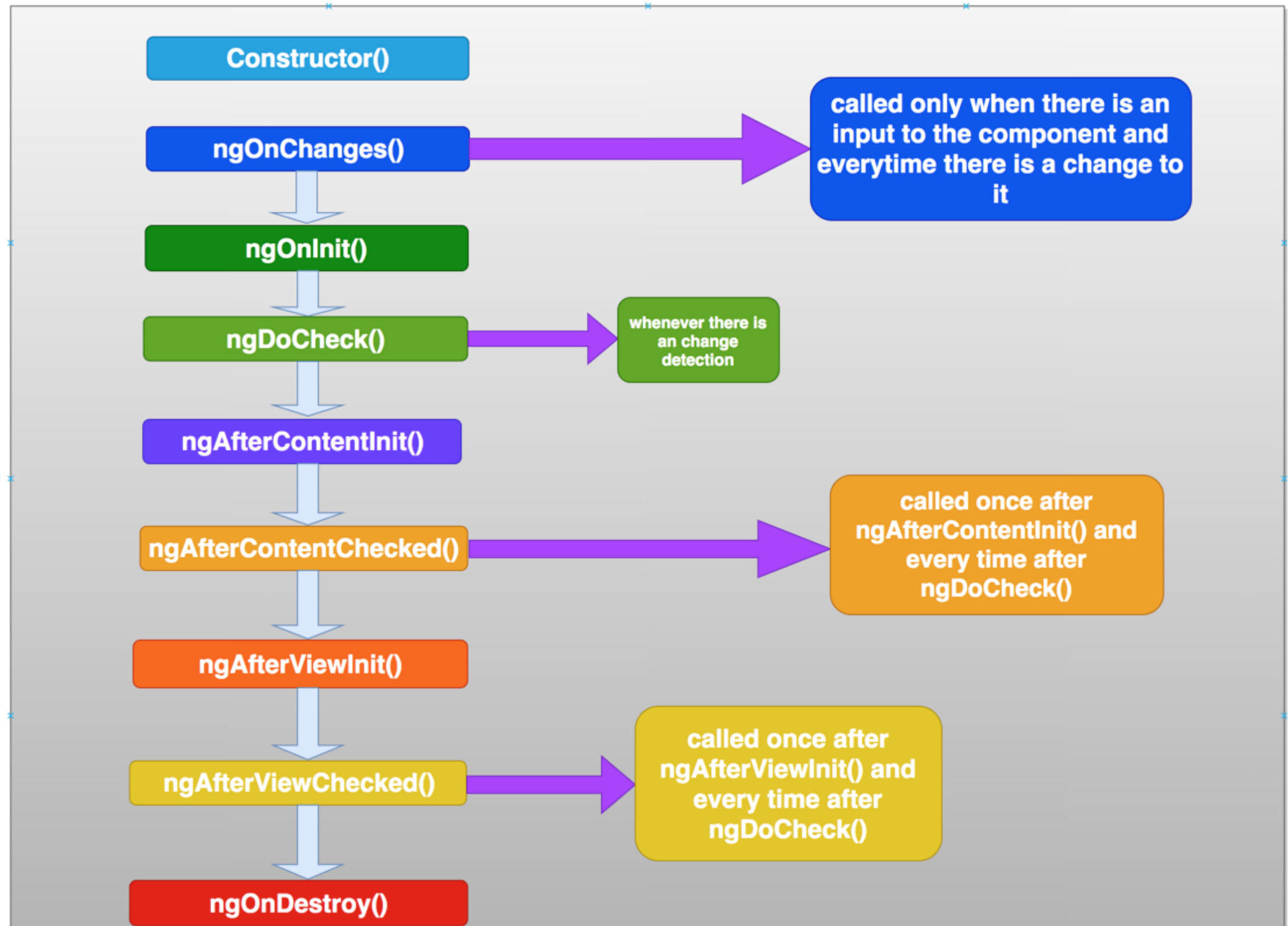
Les méthodes que vous utiliserez le plus seront certainement `ngOnInit` et `ngOnDestroy`, qui permettent d'initialiser un composant, et de le nettoyer proprement par la suite lorsqu'il est détruit.

CYCLE DE VIE D'UN COMPOSANT



- ▶ Il est important à noter que même si vous ne définissez pas explicitement des méthodes de cycle de vie dans votre composant, ces méthodes sont appelées en interne par Angular pour chaque composant.
- ▶ Lorsqu'on utilise ces méthodes, on vient donc juste surcharger tel ou tel événement du cycle de vie d'un composant.

CYCLE DE VIE D'UN COMPOSANT



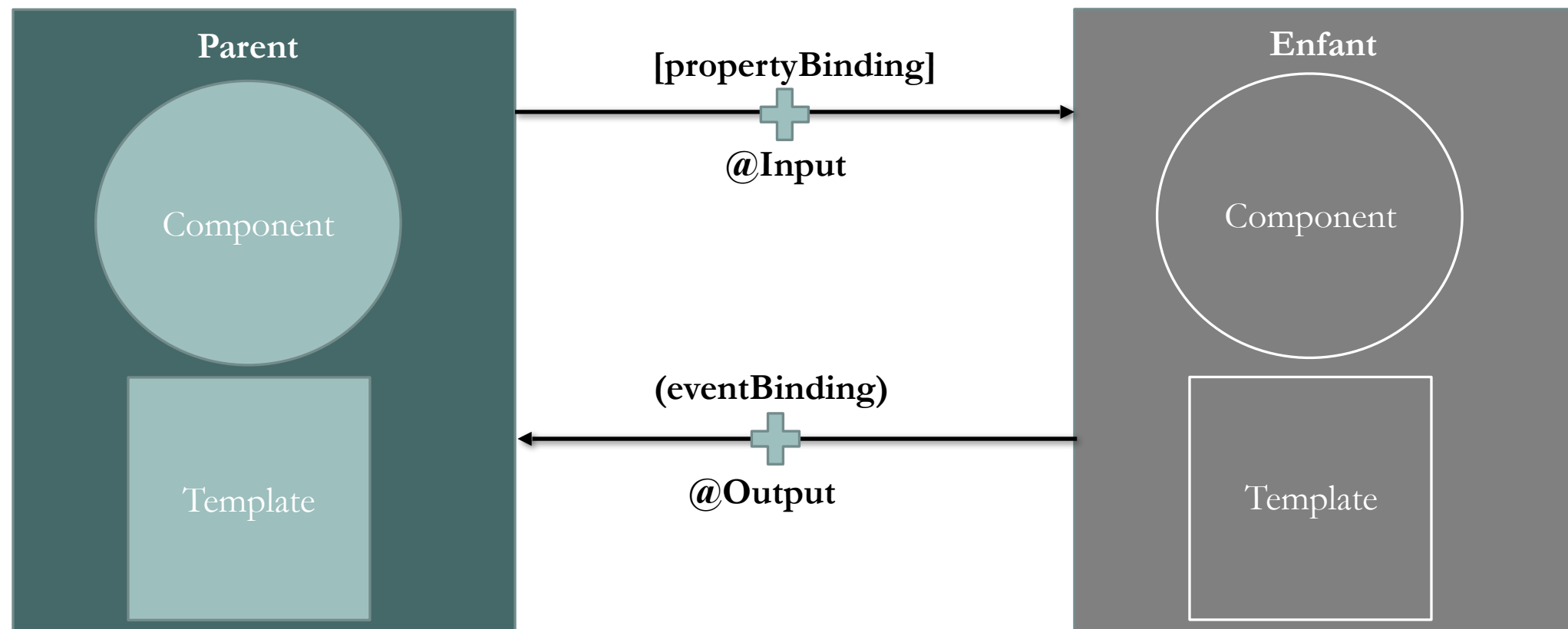
ngOnInit VS CONSTRUCTOR

- ▶ Il est assez courant de confondre les deux. Il est donc fortement recommandé de comprendre à quoi sert chacun deux.
- ▶ *ngOnInit* est appelée quand le composant est initié. Plus précisément, quand les propriétés sont affichées et que les propriétés **@Input** sont initiés.
- ▶ La méthode **constructor** n'a absolument rien à voir avec Angular. Il s'agit d'une méthode liée aux classes TypeScript.
- ▶ Le framework Angular n'a absolument aucun contrôle sur ce constructor et son appel.

ngOnInit VS CONSTRUCTOR

- ▶ Pour faire court donc :
- ▶ Le seul intérêt d'utiliser le **constructor** dans vos classes est lors de l'injection de dépendance.
- ▶ En effet, rappelons que le constructor est appelé par le moteur JavaScript. Par conséquent, il pourra dire au framework Angular de quelles dépendances, il va avoir besoin.
- ▶ Pour l'initialisation des attributs de vos composants, passez plutôt par ngOnInit.

INTERACTION ENTRE LES COMPOSANTS



INTERACTION ENTRE LES COMPOSANTS

- ▶ Relation entre un composant **parent** et un composant **enfant** :
- ▶ Le parent voit l'enfant mais ne peut pas voir ses propriétés.
- ▶ **Solution** : Faire du property binding avec @input

```
import { Component, OnInit, Input, Output, EventEmitter } from '@angular/core';

@Component({
  selector: 'app-child-first',
  templateUrl: './child-first.component.html',
  styleUrls: ['./child-first.component.css']
})
export class ChildFirstComponent implements OnInit {
  @Input() colour:string;
```


INTERACTION ENTRE LES COMPOSANTS

- ▶ Relation entre un composant **enfant** et un composant **parent** :
- ▶ L'enfant ne voit pas le parent car il ne sait simplement pas par quel composant il a été appelé.
- ▶ **Solution** : 1. Faire du event binding avec @output

```
import { Component, OnInit, Input, Output, EventEmitter } from '@angular/core';

@Component({
  selector: 'app-child-first',
  templateUrl: './child-first.component.html',
  styleUrls: ['./child-first.component.css']
})
export class ChildFirstComponent implements OnInit {
  @Output() sendRequest = new EventEmitter();
}
```

INTERACTION ENTRE LES COMPOSANTS

- ▶ **Solution** : 2. Configurer l'événement

```
sendEvent() {  
  this.sendRequest.emit('Accuse la réception de la couleur '+ this.colour);  
}
```

- ▶ 3. Récupérer l'évènement dans le composant parent

```
<app-child-first (sendRequest)="ReceivedEvent($event)"></app-child-first>
```

- ▶ 4. Traiter le message reçu de la part du composant enfant

```
ReceivedEvent(msg : any)  
{  
  alert(msg);  
}
```

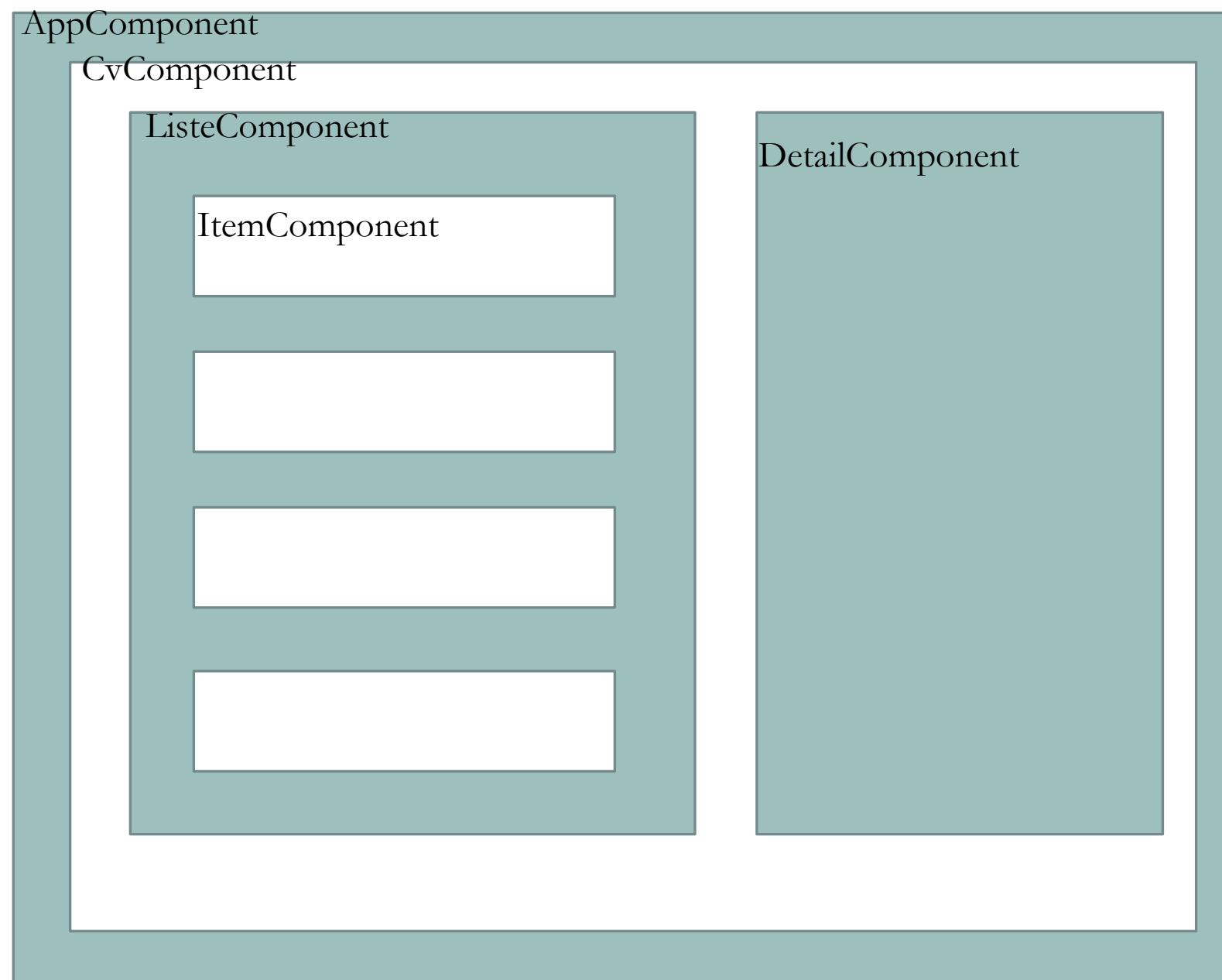
APPLICATION

- ▶ Créer un composant fils
- ▶ Récupérer la couleur du père dans le composant fils
- ▶ Faire en sorte que le composant fils affiche la couleur du background de son père.

- ▶ Ajouter une variable `myFavoriteColor` dans le composant du fils.
- ▶ Ajouter un bouton dans le composant Fils
- ▶ Au click sur ce bouton, la couleur de background du Div du père doit prendre la couleur favorite du fils.

DÉBUTONS NOTRE PROJET


- ▶ Application de gestion de CVs.




DÉBUTONS NOTRE PROJET

- ▶ L'objectif est de créer une mini plateforme de recrutement.
- ▶ La première étape est de créer la structure suivante avec une vue contenant deux parties :
- ▶ Liste des Cvs inscrits
- ▶ Détail du Cv qui apparaîtra au click
- ▶ Il vous est demandé juste d'afficher un seul Cv et de lui afficher les détails au click.
- ▶ Il faudra suivre l'architecture du diapo précédent.


- Un cv est caractérisé par : id, Nom, Prenom, Age, urlimage, metier, quote, description, mots-clés



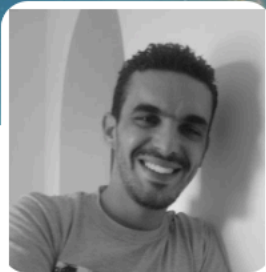
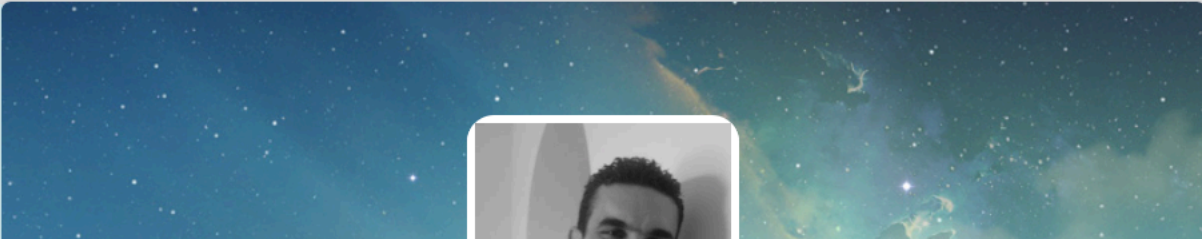
Nidhal Jelassi



Yasmine Jelassi



Senda Jelassi



Nidhal JELASSI
Enseignant
35
quote 1

↻ Auto Rotation