

FRAMEWORK ANGULAR

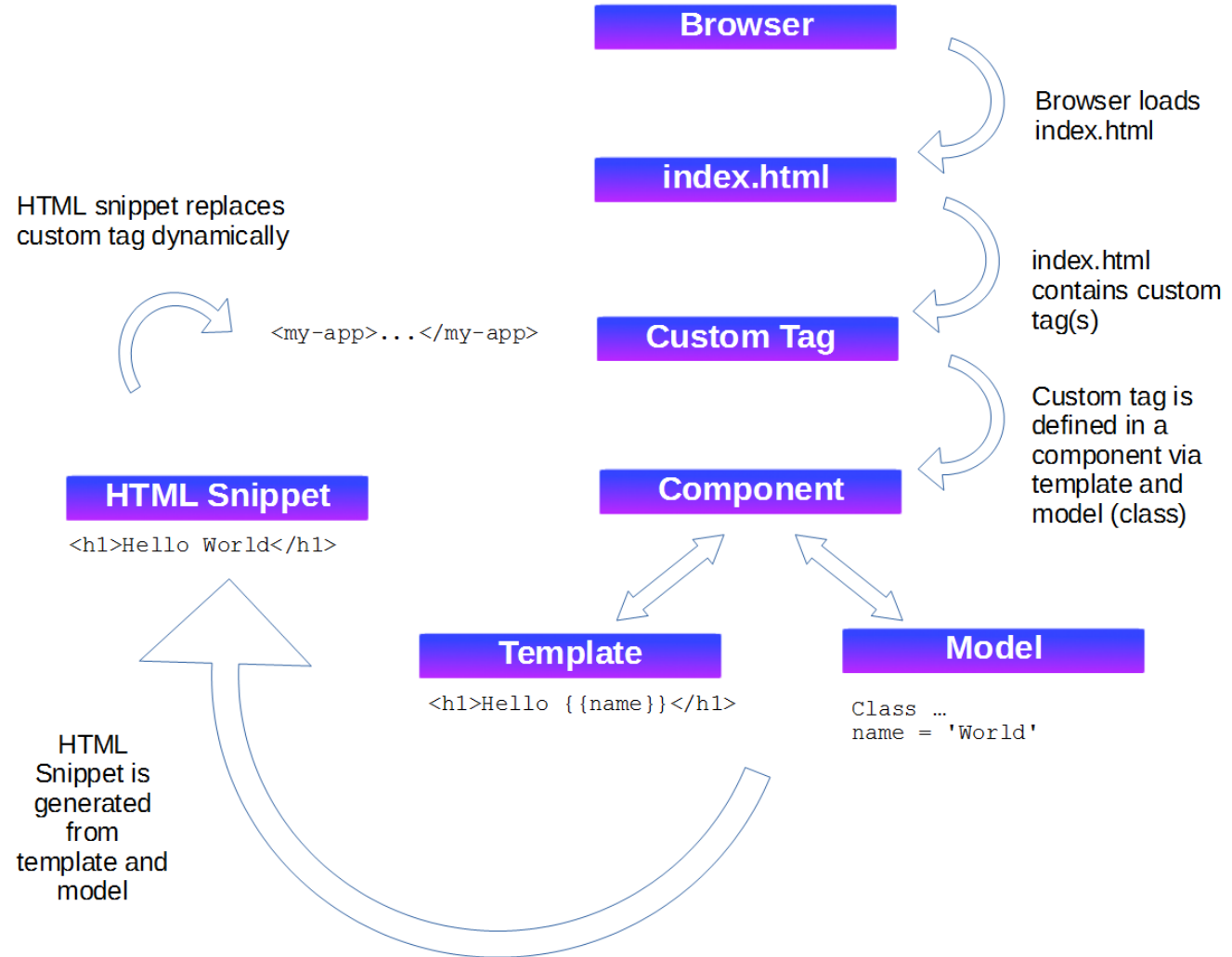
Utilisation de Framework

Notion de composant web

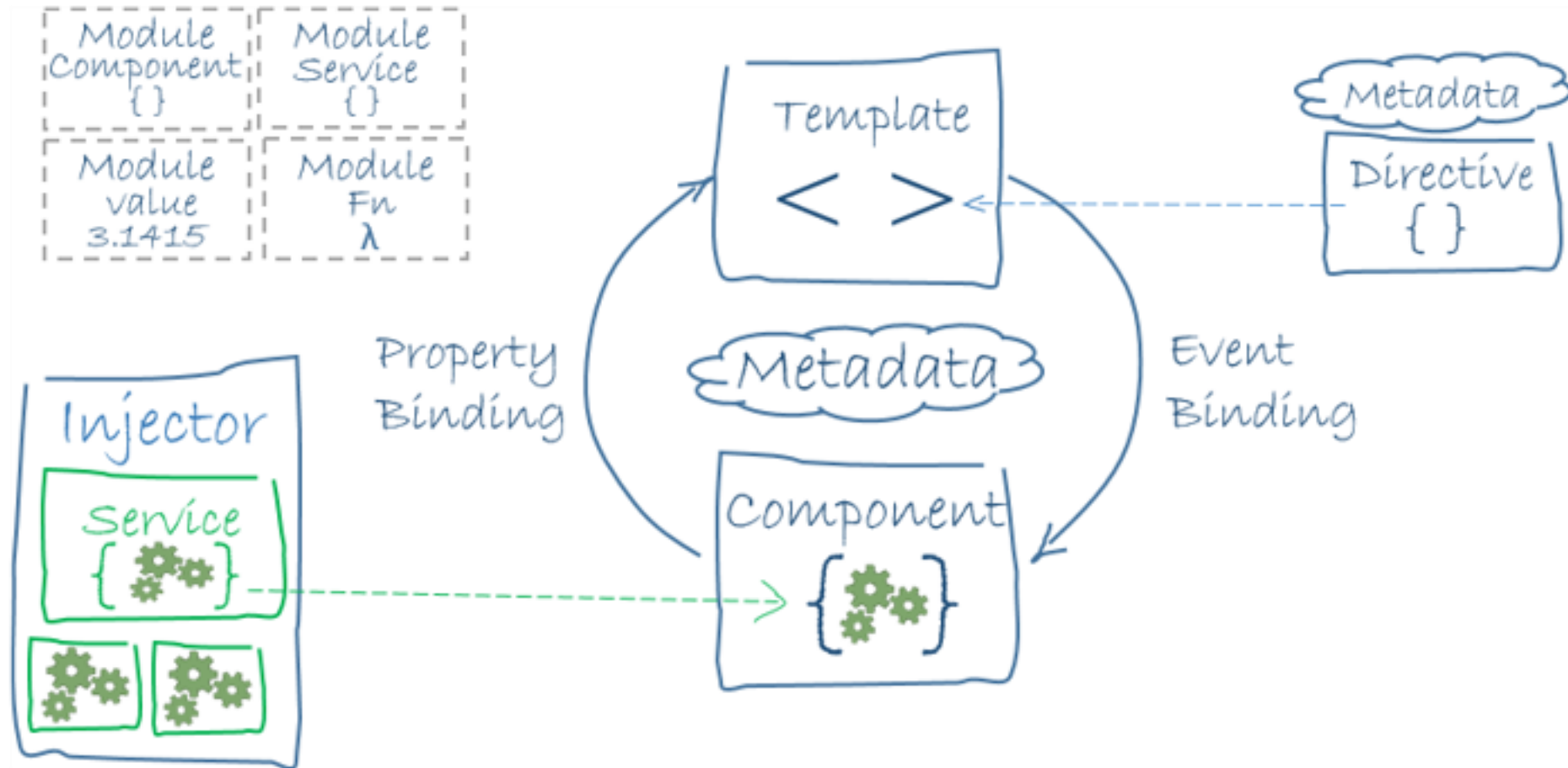
Concept d'Angular

ANGULAR

→ Angular s'appuie fortement sur la notion de web component



ANGULAR



ANGULAR : CONCEPTS

- **Components** : élément de base d'une application Angular, permet de définir la manière dont l'utilisateur interagit avec la vue.
- **Templates** : rendu HTML du composant sur une page.
- **Data bindings** : relation entre les données d'un composant (modèle) et les valeurs affichés dans le template.
- **Metadata** : information permettant de relier des éléments angular (template et component par exemple pour former la vue)
- **Component interaction** : lien entre les différents composants (échange d'information)

ANGULAR : CONCEPTS

- **Dependency injection / service** : implémentation du pattern IoC (inversion des contrôles) afin de gérer les dépendances d'une application.
- **Routing** : gérer l'aspect navigation d'une SPA.
- **Forms** : gestion de la saisie utilisateur.
- **Pipe** : transformation de la valeur d'un élément avant de l'afficher dans la vue (e.g. date).
- **Modules** : organisation d'une application en bloc fonctionnel.

FRAMEWORK ANGULAR

Utilisation de Framework

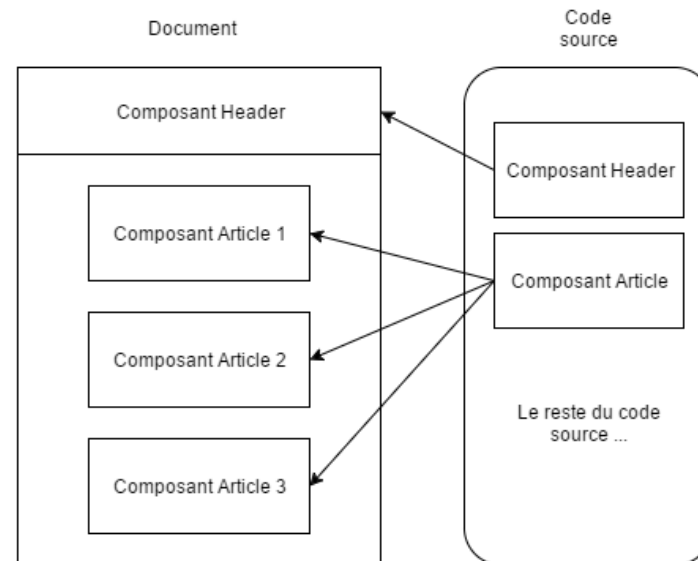
Notion de composant web

Concept d'Angular

Component / Template

ANGULAR : COMPOSANT

- Structure fondamentale d'Angular (et d'autres frameworks et Web Component).
- Une application est découpée en composant qui peuvent contenir eux-mêmes d'autres composants.
- Avantages : réutilisation des composants et découpage logique.



ANGULAR : COMPOSANT

→ Fichier app.component.ts

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

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

export class AppComponent {
  title = 'Premier exemple de cours';
}
```

→ Fichier app.component.html

```
<header class="container-fluid bg-light p-2">
  <h1>{{title}}</h1>
</header>
```


ANGULAR : COMPOSANT

→ Fichier app.component.ts

Décorateur **@Component** :
permet de déclarer le composant

Classe liée au composant

→ Fichier app.component.html

Permet d'importer le décorateur Component

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

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

```
export class AppComponent {  
  title = 'Premier exemple de cours';  
}
```

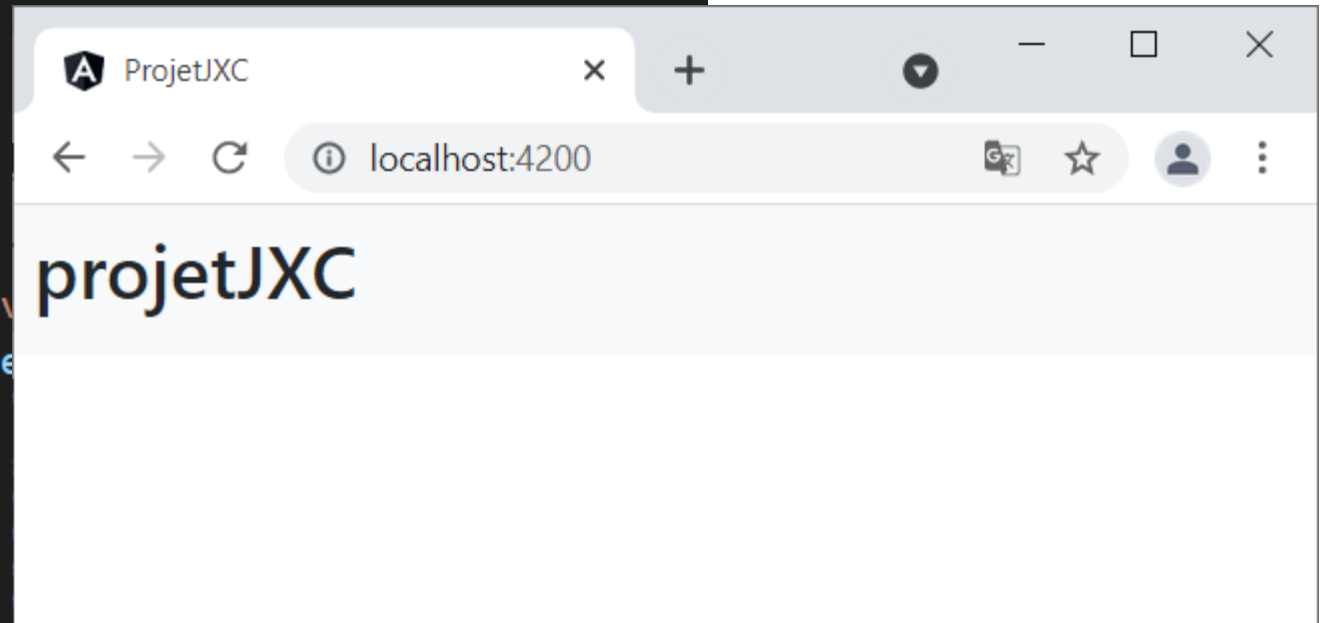
```
<header class="container-fluid bg-light p-2">  
  <h1>{{title}}</h1>  
</header>
```

Interpolation du texte

ANGULAR : COMPOSANT

→ Fichier **index.html** avec utilisation de notre composant racine (selecteur **app-root**).

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>ProjetJXC</title>
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <app-root></app-root>
</body>
</html>
```



ANGULAR : COMPOSANT

→ Possibilité de créer **des classes** et **des interfaces** pour les données afin de les utiliser dans les composants.

```
projetJXC > src > app > TS Personnage.ts > ...  
1  export interface Personnage {  
2      id: number;  
3      name: string;  
4  }
```

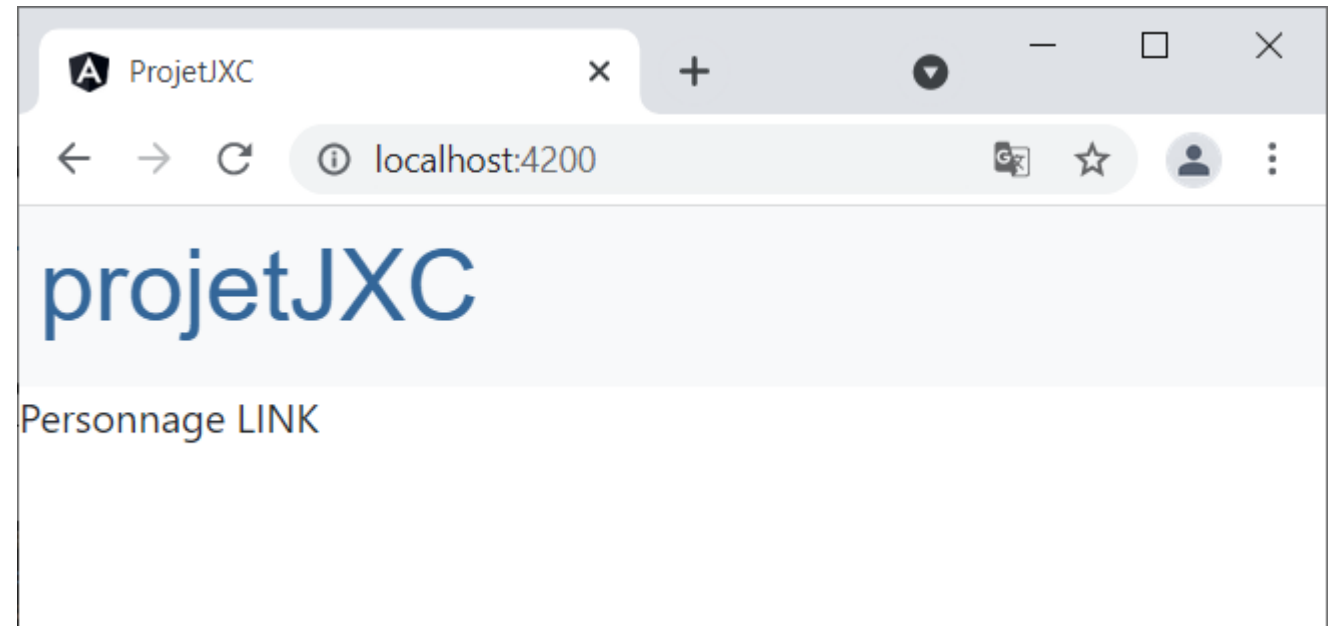
```
projetJXC > src > app > personnages > TS personnages.component.ts > ...  
1  import { Component, OnInit } from '@angular/core';  
2  import { Personnage } from '../Personnage';  
3  
4  @Component({  
5      selector: 'app-personnages',  
6      templateUrl: './personnages.component.html',  
7      styleUrls: ['./personnages.component.css']  
8  })  
9  export class PersonnagesComponent implements OnInit {  
10     perso: Personnage = {  
11         id: 1,  
12         name: 'Link'  
13     }  
14     constructor() { }  
15     ngOnInit(): void {  
16     }  
17 }
```

ANGULAR : COMPOSANT

→ Possibilité de créer **des classes** et **des interfaces** pour les données afin de les utiliser dans les composants.

```
projetJXC > src > app > TS Personnage.ts > ...  
1  export interface Personnage {  
2      id: number;  
3      name: string;  
4  }
```

```
projetJXC > src > app > personnages > <> personnages.component.html > ..  
1  <p>Personnage {{perso.name | uppercase}}</p>
```



ANGULAR : COMPOSANT

- Utilisation d'une liste de personnages dans le composant.
- Création d'une constante (liste de personnages) pour simuler la récupération des données depuis un serveur.

```
projetJXC > src > app > TS mock-personnages.ts > ...  
1  import { Personnage } from "../Personnage";  
2  
3  export const PERSONNAGES: Personnage[] = [  
4      {id:1, name:'Link'},  
5      {id:2, name:'Zelda'},  
6      {id:3, name:'Revali'},  
7      {id:4, name:'Urbosa'},  
8      {id:5, name:'Sidon'},  
9      {id:6, name:'Mipha'}  
10 ];
```

ANGULAR : COMPOSANT

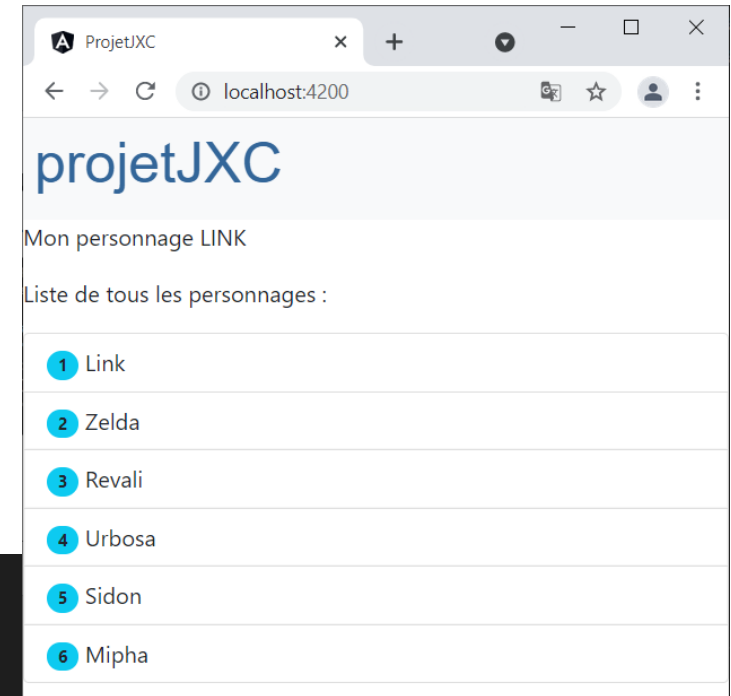
- Utilisation d'une liste de personnages dans le composant.
- Création d'une constante (liste de personnages) pour simuler la récupération des données depuis un serveur.

```
projetJXC > src > app > personnages > TS personnages.component.ts > ...  
 1  import { Component, OnInit } from '@angular/core';  
 2  import { PERSONNAGES } from '../mock-personnages';  
  
 9  export class PersonnagesComponent implements OnInit {  
10  
11    listPersonnages = PERSONNAGES;  
12    myPerso = this.listPersonnages[0];
```

ANGULAR : COMPOSANT

Directive *ngFor

```
projetJXC > src > app > personnages > <> personnages.component.html > ...
1  <p>Mon personnage {{myPerso.name | uppercase}}</p>
2  <p>Liste de tous les personnages :</p>
3  <ul class="list-group">
4    <li *ngFor="let perso of listPersonnages" class="list-group-item">
5      <span class="badge rounded-pill bg-info text-dark">{{perso.id}}</span>
6      {{perso.name}}
7    </li>
8  </ul>
```



ANGULAR : COMPOSANT

Cycle de vie des composants

Possibilité d'intercepter les différentes étapes du cycle de vie d'un composant (hook).

- **ngOnChanges()** : appelée à la création du composant, puis à chaque changement d'un attribut scalaire décoré par @Input.
- **ngOnInit()** : appelée lors de la création d'un composant juste après le premier appel de ngOnChange (souvent appel aux API pour récupérer les données).
- **ngDoCheck()** : mise en œuvre pour connaître les changements des valeurs internes d'objets ou de listes (ceux non identifiables par ngOnChanges)
- **ngOnDestroy()** : appelée juste avant que le composant soit désalloué.
- Et d'autres : ngAfterContentInit(), ngAfterContentChecked(), etc.

ANGULAR : COMPOSANT

Gestion de l'encapsulation

Possibilité de spécifier une *View Encapsulation* dans un composant.

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

```
@Component({  
  selector: 'app-personnages',  
  templateUrl: './personnages.component.html',  
  styleUrls: ['./personnages.component.css'],  
  encapsulation: ViewEncapsulation.None  
})
```

- **None** : aucune encapsulation de style réalisée, le style d'un composant est global à toute l'application.
- **Emulated** : le Shadow DOM n'est pas utilisé mais une encapsulation émulée est fait pour les styles de composants (permet de limiter les styles des composants)
- **ShadowDom** : utilisation du shadow DOM du navigateur (rendu plus rapide mais attention au support des navigateurs).

ANGULAR : TEMPLATE

- Correspond à la vue du composant dans la page de l'application Angular.
- Possibilité d'utiliser des directives, de déclencher un appel d'évènement, d'afficher les données mise à jour des composants, d'instancier d'autres composants et bien d'autres.



FRAMEWORK ANGULAR

Utilisation de Framework

Notion de composant web

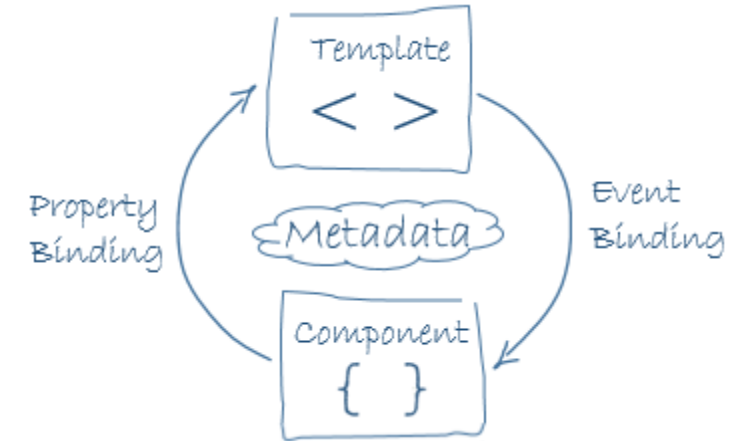
Concept d'Angular

Component / Template

Data Binding

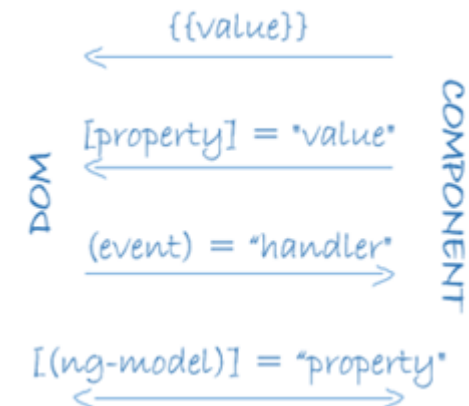
ANGULAR : DATA BINDING

- Permet de créer une relation entre les données d'un composant et les valeurs correspondantes affichées dans la vue.



Différentes catégories de binding :

- Du composant à la vue []
- De la vue au composant ()
- Et dans les deux directions - two-way binding [()]



ANGULAR : DATA BINDING

Interpolation :

Une variable scalaire est injectée dans le template.

→ Si un composant modifie les données, la vue sera automatiquement mise à jour.

```
projetJXC > src > app > personnages > <> personnages.component.html >  
1   <p>Mon personnage {{myPerso.name | uppercase}}</p>  
2   <img src={{myPerso.urlImage}}/>
```

projetJXC

Mon personnage LINK



ANGULAR : DATA BINDING

Property binding :

Si la variable à interpoler dans le template est la valeur d'un attribut d'une balise HTML, la gestion de cet attribut peut être délégué à Angular.

→ Attribut HTML encadré par des crochets, devient une directive de l'attribut.

```
projetJXC > src > app > personnages > <> personnages.component.html >  
1   <p>Mon personnage {{myPerso.name | uppercase}}</p>  
2   <img src={{myPerso.urlImage}}/>  
3   <img [src]="myPerso.urlImage"/>
```



ANGULAR : DATA BINDING

Property binding : **[target]="expression"**

Le target peut être une propriété d'un élément, d'un composant ou d'une directive.

```
<img [src]="heroImageUrl">  
<app-hero-detail [hero]="currentHero"></app-hero-detail>  
<div [ngClass]="{'special': isSpecial}"></div>
```

160

Cette syntaxe utilisé également pour les attributs, classe et style :

```
<button [attr.aria-label]="help">help</button>  
<div [class.special]="isSpecial">Special</div>  
<button [style.color]="isSpecial ? 'red' : 'green'"></button>
```

ANGULAR : DATA BINDING

Event binding :

Permet à Angular d'exécuter du code ou des actions lorsqu'un évènement est levé.

→ Clics, mouvement de souris, frappes au clavier, manipulations tactiles, etc.

`<button (click)="onSave()">Save</button>`

target event name

template statement

```
<button (click)="onSave()">Save</button>
<app-hero-detail (deleteRequest)="deleteHero()"></app-hero-detail>
<div (myClick)="clicked=$event" clickable>click me</div>
```

→ Evènement sur un élément, un composant ou un directive.

ANGULAR : DATA BINDING

Event binding : Exemple

```
projetJXC > src > app > personnages > <> personnages.component.html > ...
1  <p>Liste de tous les personnages :</p>
2  <ul class="list-group personnages">
3      <li *ngFor="let perso of listPersonnages"
4          (click)="selectedPersonnage(perso)"
5          [class.selected]="perso === myPerso"
6          class="list-group-item">
7          <span class="badge rounded-pill bg-info text-dark">{{perso.id}}</span>
8          {{perso.name}}
9      </li>
10 </ul>
11 <div *ngIf="myPerso">
12     <p>Mon personnage {{myPerso.name | uppercase}}</p>
13     <img [src]="myPerso.urlImage"/>
14 </div>
```

ANGULAR : DATA BINDING

Event binding : Exemple

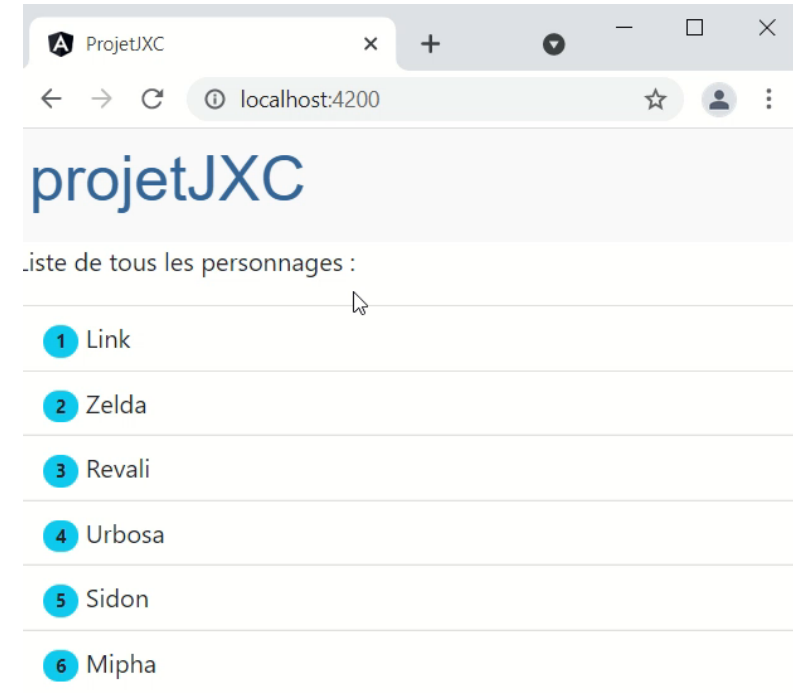
```
10  export class PersonnagesComponent implements OnInit {  
11  
12      listPersonnages = PERSONNAGES;  
13      myPerso = this.listPersonnages[0];  
14  
15      constructor() { }  
16      ngOnInit(): void {  
17      }  
18  
19      selectedPersonnage(perso: Personnage){  
20          this.myPerso = perso;  
21      }  
22  }
```

ANGULAR : DATA BINDING

Event binding : Exemple

```
projetJXC > src > app > personnages > # personnages.component.css > ...
```

```
1  .personnages li{
2      cursor: pointer;
3  }
4  .personnages li.selected {
5      background-color: #rgb(247, 149, 174);
6      color: white;
7  }
```



ANGULAR : DATA BINDING

Event binding :

- Possibilité d'avoir l'objet **\$event** comme paramètre à la méthode.
- Le type de l'objet \$event dépend du target (DOM element event)

InputEvent

```
<div *ngIf="myPerso">
  <p>Mon personnage {{myPerso.name | uppercase}}</p>
  <img [src]="myPerso.urlImage"/>
  <input [value]="myPerso.name"
    | (input)="myPerso.name=getValue($event)"/>
</div>
```

```
getValue(event: Event): string {
  console.dir(event);
  return (event.target as HTMLInputElement).value;
}
```

ANGULAR : DATA BINDING

Event binding :

→ Résultat :

projetJXC

Liste de tous les personnages :

1	Link
2	Zelda
3	Revali
4	Urbosa
5	Sidon
6	Mipha

Console

1

top

Filter

Default levels

1 Issue: 1

ngOnInit

personnages.component.ts:19

ngDoCheck

personnages.component.ts:25

Angular is running in

core.js:28059

development mode. Call enableProdMode()

to enable production mode.

ngDoCheck

personnages.component.ts:25

[WDS] Live Reloading

index.js:52

enabled.

ANGULAR : DATA BINDING

Two-way binding : `[(target)]="expression"`

Permet de lier une propriété de la classe TypeScript implémentant le composant avec une interface de saisie/sélection du template (input, select, textarea, etc.)

- Une modification dans le template de la valeur de la zone de saisie met à jour immédiatement la valeur de la propriété dans le classe.
- La mise à jour de la variable au sein de la propriété est immédiatement répercutée dans le template.

```
<input [(ngModel)]="name">
```

FRAMEWORK ANGULAR

Utilisation de Framework

Notion de composant web

Concept d'Angular

Component / Template

Data Binding

Component Interaction

ANGULAR : INTERACTION ENTRE COMPOSANT

Une application *Angular* est composée de plusieurs composants.

→ De quelle manière les composants peuvent interagir entre eux ?

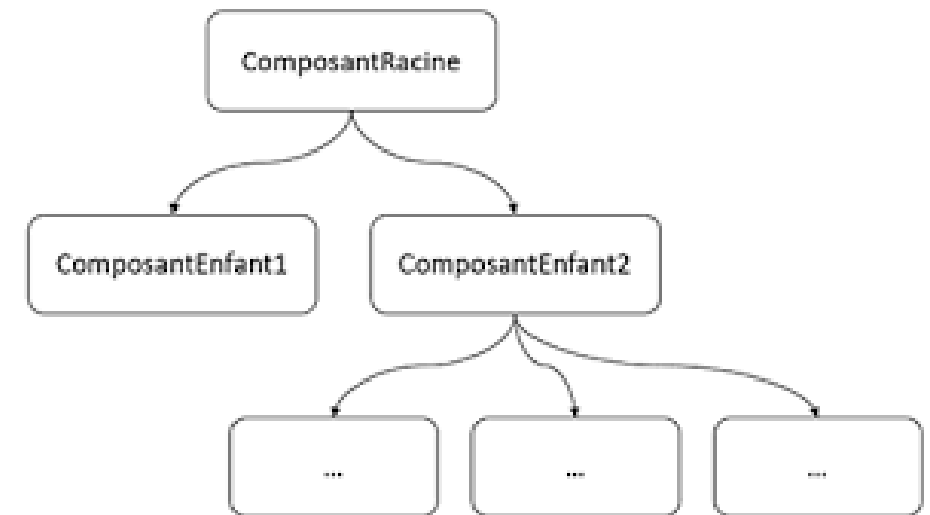
→ **Déclarer les inputs et outputs d'un composant**

→ **EventEmitter**

→ **Getter et Setter**

→ **Variable locale**

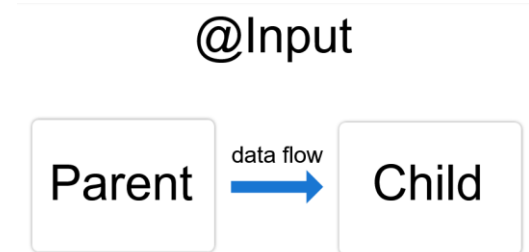
→ **Décorateur ViewChild**



ANGULAR : INTERACTION ENTRE COMPOSANT

@Input :

- Permet d'identifier une propriété du composant en tant qu'input.
- Permet au composant parent de mettre à jour une donnée du composant enfant.



TS du composant enfant :

```
import { Component, Input } from '@angular/core';
export class ItemDetailComponent {
  @Input() item = '';
}
```

Template du composant parent :

```
<app-item-detail [item]="currentItem"></app-item-detail>
```

Source (propriété du parent)

Target (propriété de l'enfant)

ANGULAR : INTERACTION ENTRE COMPOSANT

@Output



@Output :

- Permet de notifier le composant parent qu'un évènement s'est produit au sein du composant enfant.
- Utilisation de la classe **EventEmitter** afin de notifier le composant parent (évènements personnalisés)

172

TS du composant enfant :

```
import { Output, EventEmitter } from '@angular/core';
export class ItemOutputComponent {

  @Output() newItemEvent = new EventEmitter<string>();

  addNewItem(value: string) {
    this.newItemEvent.emit(value);
  }
}
```

ANGULAR : INTERACTION ENTRE COMPOSANT

Template du composant enfant :

```
<label for="item-input">Add an item:</label>
<input type="text" id="item-input" #newItem>
<button (click)="addNewItem(newItem.value)">Add to parent's list</button>
```

Template du composant parent :

```
<app-item-output (newItemEvent)="addItem($event)"></app-item-output>
```

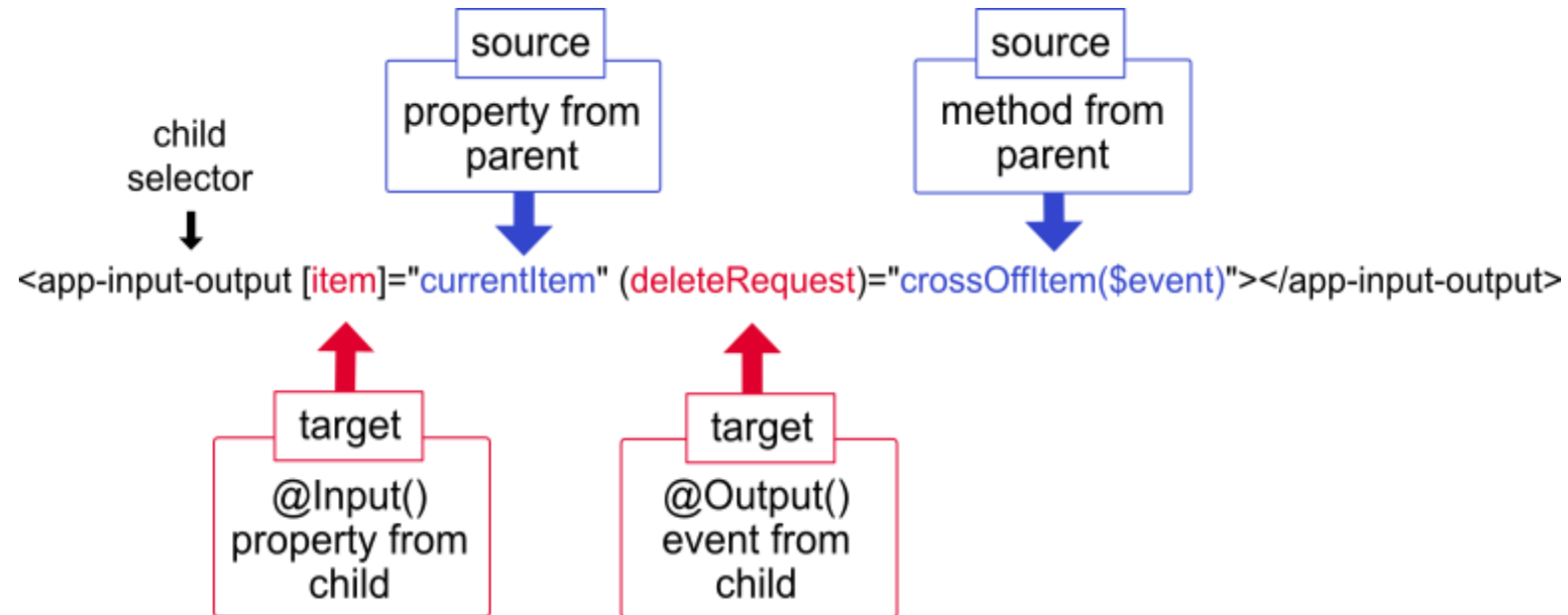
TS du composant parent :

```
export class AppComponent {
  items = ['item1', 'item2', 'item3', 'item4'];

  addItem(newItem: string) {
    this.items.push(newItem);
  }
}
```

ANGULAR : INTERACTION ENTRE COMPOSANT

Possibilité de combiner **@Input** et **@Output** :



ANGULAR : EXEMPLE

- Création d'un nouveau composant : personnage-detail
- Composant parent : personnages, composant enfant : personnage-detail
- Utilisation du décorateur @Input et @Output
- But :

Séparer la liste des personnages et le détail d'un personnage,
ajouter des nouvelles armes à un personnage (ajout d'une donnée arme).

TS du composant enfant (personnages-detail) :


```
src > app > personnages-detail > TS personnages-detail.component.ts > ...
1  import { Component, OnInit, Input, Output, EventEmitter } from '@angular/core';
2  import { Personnage } from '../Personnage';
3
4  @Component({
5    selector: 'app-personnages-detail',
6    templateUrl: './personnages-detail.component.html',
7    styleUrls: ['./personnages-detail.component.css']
8  })
9  export class PersonnagesDetailComponent implements OnInit {
10
11    @Input() perso?: Personnage;
12
13    @Output() newArmeEvent = new EventEmitter<string>();
14
15    constructor() { }
16    ngOnInit(): void {
17    }
18    changerArme(value: string){
19      this.newArmeEvent.emit(value);
20    }
21  }
```

ANGULAR : EXEMPLE

Template du composant enfant (personnages-detail) :

```
src > app > personnages-detail > <> personnages-detail.component.html > ...
1   <div *ngIf="perso">
2       <p>Détail :</p>
3       <p>Personnage {{perso.name | uppercase}}</p>
4       <img [src]="perso.urlImage"/>
5       <label for="arme-input">Quelle arme ? </label>
6       <input type="text" id="arme-input" #nouvelleArme>
7       <button (click)="changerArme(nouvelleArme.value)">Valider</button>
8   </div>
9
```

TS du composant parent
(personnages) :

```
src > app > personnages > TS personnages.component.ts >  PersonnagesComponent
 1  ∨ import { Component, OnInit } from '@angular/core';
 2    import { PERSONNAGES } from '../mock-personnages';
 3    import { Personnage } from '../Personnage';
 4
 5  ∨ @Component({
 6    selector: 'app-personnages',
 7    templateUrl: './personnages.component.html',
 8    styleUrls: ['./personnages.component.css']
 9  })
10  ∨ export class PersonnagesComponent implements OnInit {
11    listPersonnages = PERSONNAGES;
12    myPerso: Personnage = this.listPersonnages[0];
13    couleur = "red";
14    constructor() { }
15  ∨ ngOnInit(): void {
16    |   console.log("ngOnInit");
17    |
18  ∨   selectedPersonnage(perso: Personnage): void{
19    |     this.myPerso = perso;
20    |   }
21  ∨   modifierArme(newArme: string) {
22    |     this.myPerso.arme.push(newArme);
23    |   }
24  } }
```


ANGULAR : EXEMPLE

Template du composant parent (personnages) :

```
src > app > personnages > <> personnages.component.html > ...
1  <p>Liste de tous les personnages :</p>
2  <ul class="list-group personnages">
3    <li *ngFor="let perso of listPersonnages"
4      (click)="selectedPersonnage(perso)"
5      [class.selected]="perso === myPerso"
6      class="list-group-item">
7      <span class="badge rounded-pill bg-info text-dark">{{perso.id}}</span>
8      {{perso.name}}
9      <span *ngFor="let arme of perso.arme">{{arme}} </span>
10    </li>
11  </ul>
12
13  <app-personnages-detail [perso]="myPerso" (newArmeEvent)="modifierArme($event)"></app-personnages-detail>
14
```

ANGULAR :

Résultat :

projetJXC

Liste de tous les personnages :

1 Link

2 Zelda

3 Revali

4 Urbosa

5 Sidon

6 Mipha

Détail :

Personnage LINK



Quelle arme ?

épée

Valider

ANGULAR : INTERACTION ENTRE COMPOSANT

→ Possibilité d'intercepter un changement de valeur grâce à un **setter**

```
export class NameChildComponent {  
  @Input()  
  get name(): string { return this._name; }  
  set name(name: string) {  
    this._name = name;  
  }  
  private _name = '';  
}
```

ANGULAR : EXEMPLE

→ .Template du composant parent

Two-way binding

```
src > app > test-get-set-parent > <> test-get-set-parent.component.html > ...  
1  <div>  
2    <input [(ngModel)]="text" placeholder="Texte"/>  
3    <hr/>  
4    <app-test-get-set [name]="text"></app-test-get-set>  
5  </div>
```

One-way binding : property binding

Parent

Enfant

Mon nom :

Historique :

-

ANGULAR : EXEMPLE

→ TS du composant parent :

```
src > app > test-get-set-parent > TS test-get-set-parent.component.ts > ...
1  import { Component, OnInit } from '@angular/core';
2  @Component({
3      selector: 'app-test-get-set-parent',
4      templateUrl: './test-get-set-parent.component.html',
5      styleUrls: ['./test-get-set-parent.component.css']
6  })
7  export class TestGetSetParentComponent implements OnInit {
8
9      text: string = '';
10
11     constructor() { }
12     ngOnInit(): void {
13     }
14 }
```

ANGULAR :

→ TS du composant enfant

```
src > app > test-get-set > TS test-get-set.component.ts > ...
1  import { Component, OnInit, Input } from '@angular/core';
2  √ @Component({
3      selector: 'app-test-get-set',
4      templateUrl: './test-get-set.component.html',
5      styleUrls: ['./test-get-set.component.css']
6  })
7  √ export class TestGetSetComponent implements OnInit {
8      historyName: string[] = [];
9      private _name: string = '';
10  √ get name(): string {
11      |     return this._name;
12      | }
13      @Input()
14  √ set name(value: string){
15      |     this._name = value;
16      |     this.historyName.push(value);
17      | }
18      constructor() { }
19      ngOnInit(): void {
20      }
21  }
```

ANGULAR : EXEMPLE

→ Template du composant enfant

```
src > app > test-get-set > <> test-get-set.component.html > ...  
1  <p>Mon nom : {{ name }}</p>  
2  <div>  
3    <p>Historique :</p>  
4    <ul>  
5      <li *ngFor="let oldName of historyName">  
6        {{oldName}}  
7      </li>  
8    </ul>  
9  </div>
```

ANGULAR : INTERACTION ENTRE COMPOSANT

→ Utilisation d'une **variable locale** (template référence variable) :

Permet d'utiliser une donnée déclaré dans un template à un autre endroit de ce template.

→ Peut se référer à :

Un élément du DOM dans le template (e.g. input),

Un composant

Une directive

#templateVariable

ANGULAR : INTERACTION ENTRE COMPOSANT

→ Composant enfant :

```
src > app > testlocalvar-child > TS testlocalvar-child.component.ts > ...
  8  export class TestlocalvarChildComponent {
  9      text: string = '';
 10  showMessage() {
 11      alert(this.text);
 12  }
 13  }
```

```
src > app > testlocalvar-child > <> testlocalvar-child.component.html > ...
 1  <div>
 2      <p class="text-danger fw-bold">Partie enfant</p>
 3      <input [(ngModel)]="text" placeholder="Texte de l'enfant"/>
 4      <p>Texte de l'enfant : {{text}}</p>
 5  </div>
```

ANGULAR : INTERACTION ENTRE COMPOSANT

→ Composant parent :

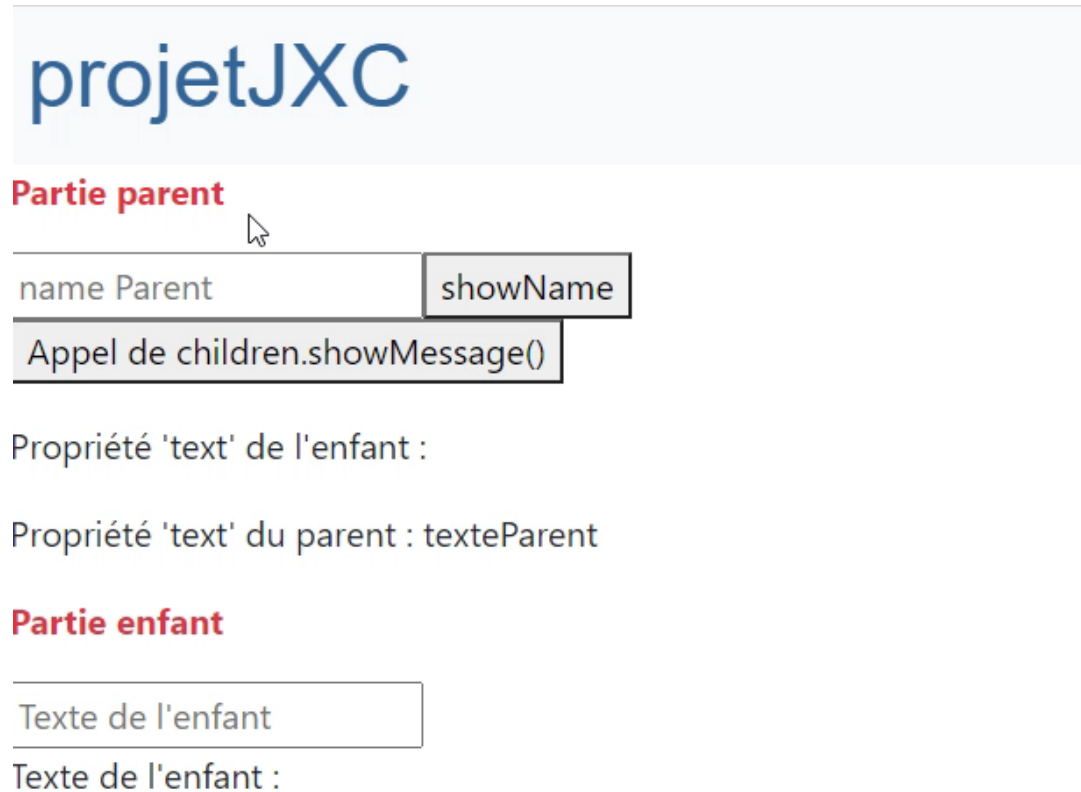
```
src > app > testlocalvar-parent > TS testlocalvar-parent.component.ts > ...
  8   export class TestlocalvarParentComponent {
  9       text: string = 'texteParent';
 10       showName(value: string){
 11           alert(value);
 12       }
 13   }
```

188

```
src > app > testlocalvar-parent > <> testlocalvar-parent.component.html > ...
 1   <div>
 2       <p class="text-danger fw-bold">Partie parent</p>
 3       <input #name placeholder="name Parent" />
 4       <button (click)="showName(name.value)">showName</button>
 5       <button (click)="children.showMessage()">Appel de children.showMessage()</button>
 6       <p>Propriété 'text' de l'enfant : {{children.text}}</p>
 7       <p>Propriété 'text' du parent : {{text}}</p>
 8       <app-testlocalvar-child #children></app-testlocalvar-child>
 9   </div>
```

ANGULAR : INTERACTION ENTRE COMPOSANT

→ Résultat



ANGULAR : INTERACTION ENTRE COMPOSANT

@ViewChild :

Permet de récupérer les données du premier composant fils à partir d'un composant parent.

```
export class PereComponent implements OnInit, AfterViewInit {  
  @ViewChild(FilsComponent)  
  private fils!: FilsComponent;  
  constructor(){}  
  ngAfterViewInit() {};  
  ngOnInit(){};  
}
```

190

@ViewChildren :

Possibilité à un composant parent de récupérer les données de ses composant enfants (QueryList).

ANGULAR : EXEMPLE

→ Modification de l'exemple précédent (même résultat)

```
export class TestlocalvarParentComponent implements AfterViewInit{
  @ViewChild (TestlocalvarChildComponent, {static: false})
  childComp !: TestlocalvarChildComponent;
  text: string = 'texteParent';
  showName(value: string){
    alert(value);
  }
  callChildren(){
    this.childComp.showMessage();
  }
  ngAfterViewInit(){
    //composant disponible ici
  }
}
```

ANGULAR : EXEMPLE

→ Modification de l'exemple précédent (même résultat)

```
src > app > testlocalvar-parent > <> testlocalvar-parent.component.html > ...
1   <div>
2       <p class="text-danger fw-bold">Partie parent</p>
3       <input #name placeholder="name Parent" />
4       <button (click)="showName(name.value)">showName</button>
5       <p><button (click)="callChildren()">Appel de children.showMessage()</button></p>
6       <app-testlocalvar-child></app-testlocalvar-child>
7   </div>
```

FRAMEWORK ANGULAR

Utilisation de Framework

Notion de composant web

Concept d'Angular

Component / Template

Data Binding

Component Interaction

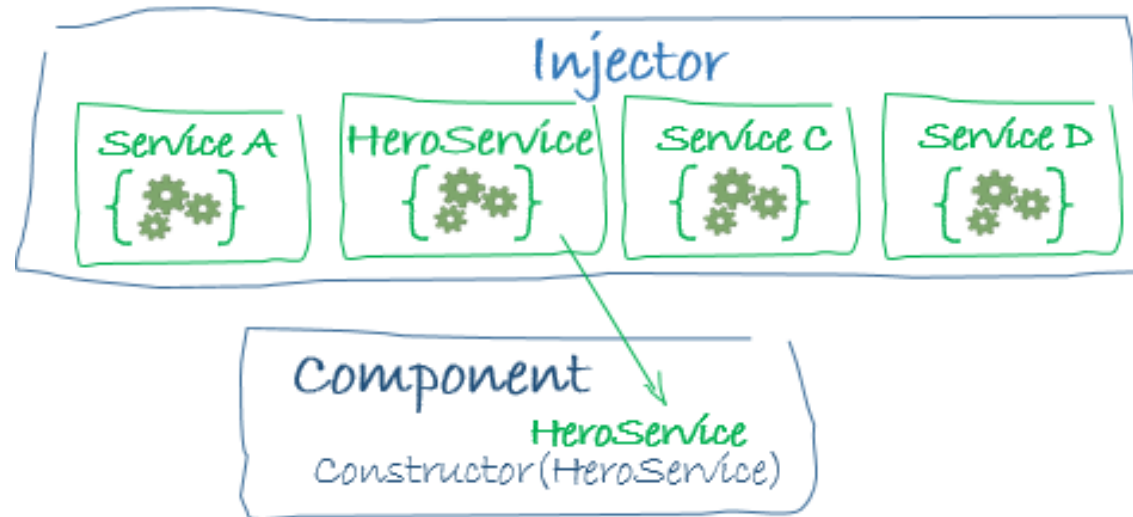
Service

ANGULAR : SERVICE

- Un composant doit se focaliser sur la représentation des données (user expérience). Il ne doit pas se soucier des mécanismes et transformations mis en place pour récupérer ces données.
- Délégation du traitement des données au service (fetching data, validation d'une saisie utilisateur, système de log).
- Un service permet de partager facilement des informations entre classes qui n'ont pas de lien.
- **Permet d'augmenter la modularité et la réutilisation des composants.**

ANGULAR : INJECTION DE DÉPENDANCE

- Les dépendances sont des services ou des objets dont une classe à besoin.
- Les injections de dépendances (DI) est un design pattern.



ANGULAR : SERVICE

Déclaration d'un service (à partir de Angular 6)

Décorateur : Angular peut
utiliser cette classe dans le DI

```
src > app > TS personnage.service.ts > ...
1  import { Injectable } from '@angular/core';
2
3  @Injectable({
4    providedIn: 'root'
5  })
6  export class PersonnageService {
7
8    constructor() { }
9  }
```

Visible dans toute
l'application.

Ajout au provider.

ANGULAR : SERVICE

Déclaration d'un service (avant Angular 6)

```
import { Injectable } from '@angular/core';  
@Injectable()  
export class PersonneService {  
  constructor() { }  
}
```

ANGULAR : INJECTION DE DÉPENDANCE

Injection d'un service

Component *Service*
{Constructor(service)}

- Pas d'utilisation du **new** !
- Dans le constructeur du composant ayant besoin du service :

```
constructor(private persoService:PersonnageService) { }
```



Singleton de PersonnageService

ANGULAR : INJECTION DE DÉPENDANCE

Autre manière possible (versions Angular inférieures à 6)

- Utilisation de **providers** (metadata) pour spécifier les services dont le composant a besoin.

Dans un composant :

```
@Component({
  selector: 'app-personnages',
  templateUrl: './personnages.component.html',
  styleUrls: ['./personnages.component.css'],
  providers: [PersonnageService]
})
export class PersonnagesComponent implements OnInit {
```

Dans un module :

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

ANGULAR : EXEMPLE

→ Récupération du **mock des personnages** dans un service et utilisation d'un système de log :

Service Logger :

```
src > app > TS logger.service.ts > ...
1  import { Injectable } from '@angular/core';
2
3  @Injectable({
4    providedIn: 'root'
5  })
6  export class LoggerService {
7    logs: string[] = []; // capture logs
8
9    constructor() { }
10
11    log(message: string) {
12      this.logs.push(message);
13      console.log(message);
14    }
15  }
```

200

ANGULAR : EXEMPLE

Service Personnage :

```
src > app > TS personnage.service.ts > ...
1  import { Injectable } from '@angular/core';
2  import { LoggerService } from '../logger.service';
3  import { PERSONNAGES } from '../mock-personnages';
4  import { Personnage } from '../Personnage';
5
6  @Injectable({
7    providedIn: 'root'
8  })
9  export class PersonnageService {
10    //Injection du service de log
11    constructor(private logger: LoggerService) { }
12    getPersonnages(): Personnage[] {
13      this.logger.log('Getting personnages ...')
14      return PERSONNAGES;
15    }
16  }
```

Composant Personnages :

```
src > app > personnages > TS personnages.component.ts > PersonnagesComponent >
1  import { Component, OnInit } from '@angular/core';
2  //import { PERSONNAGES } from '../mock-personnages';
3  import { Personnage } from '../Personnage';
4  import { PersonnageService } from '../personnage.service';
5  @Component({
6    selector: 'app-personnages',
7    templateUrl: './personnages.component.html',
8    styleUrls: ['./personnages.component.css']
9  })
10 export class PersonnagesComponent implements OnInit {
11   listPersonnages!: Personnage[]; // = PERSONNAGES;
12   myPerso!: Personnage; // = this.listPersonnages[0];
13   couleur = "red";
14   constructor(private persoService: PersonnageService) { }
15   ngOnInit(): void {
16     console.log("ngOnInit");
17     this.getHeros();
18     this.selectedPersonnage(this.listPersonnages[0]);
19   }
20   getHeros(): void {
21     this.listPersonnages = this.persoService.getPersonnages();
22   }
23   selectedPersonnage(perso: Personnage): void{
```


ANGULAR : RXJS

Dans un réelle application : besoin de services asynchrones.

→ Utilisation de callback, de **Promise** ou de **Observable** (bibliothèque **RxJS**).

203

Observable :

→ Objet permettant un échange d'information.

→ Utilisé lors d'évènements, par la module HttpClient (get() retourne un Observable), etc.

→ Méthode *subscribe()* : abonne un traitement à l'observable.

→ Méthode *unsubscribe()* : désabonne un traitement à l'observable.

ANGULAR : RXJS

Méthode *subscribe()* : trois callbacks en paramètre

- next : se déclenche à chaque fois que l'Observable émet de nouvelles données (données en tant qu'argument)
- error : se déclenche si l'Observable émet une erreur (erreur en tant qu'argument)
- complete : se déclenche si l'Observable s'achève (aucun argument)

```
// Subscribe to begin listening for async result
data.subscribe({
  next(response) { console.log(response); },
  error(err) { console.error('Error: ' + err); },
  complete() { console.log('Completed'); }
});
```

ANGULAR : RXJS

RxJS propose des fonctions permettant de créer des nouveaux **Observables** :

```
// Create an Observable out of a promise
const data = from(fetch('/api/endpoint'));
// Create an Observable that will publish a value on an interval
const secondsCounter = interval(1000);
// Create an Observable that will publish mouse movements
const el = document.getElementById('my-element');
const mouseMoves = fromEvent<MouseEvent>(el, 'mousemove');
// Create an Observable that will create an AJAX request
const apiData = ajax('/api/data');
```

ANGULAR : EXEMPLE

→ Modification de l'exemple précédent utilisant un **Observable** (asynchrone)

Service Personnage:

```
import { Observable, of } from 'rxjs';
@Injectable({
  providedIn: 'root'
})
export class PersonnageService {

  constructor(private logger: LoggerService) { }

  getPersonnages(): Observable<Personnage[]>{
    this.logger.log('Getting personnages ...')
    const persos = of(PERSONNAGES);
    return persos;
  }
}
```

ANGULAR : EXEMPLE

composant Personnage:

```
export class PersonnagesComponent implements OnInit {
  listPersonnages!: Personnage[]; // = PERSONNAGES;
  myPerso!: Personnage;
  couleur = "red";

  constructor(private persoService: PersonnageService) { }

  ngOnInit(): void {
    console.log("ngOnInit");
    this.getHeros();
    this.selectedPersonnage(this.listPersonnages[0]);
  }

  getHeros(): void {
    //this.listPersonnages = this.persoService.getPersonnages();
    this.persoService.getPersonnages()
      .subscribe(personnages => this.listPersonnages = personnages,
        error => console.log("Error"),
        () => console.log("terminé"));
  }
}
```

FRAMEWORK ANGULAR

Utilisation de Framework

Notion de composant web

Concept d'Angular

Component / Template

Data Binding

Component Interaction

Service

Rooting

ANGULAR : ROUTING

- Dans une **SinglePage Application**, le contenu de la page est modifié au fur et à mesure (affichage de différents composants).
- Notion de **navigation importante** pour les utilisateur (e.g. mimer le mécanisme de copier l'url, réaliser un retour en arrière dans le navigateur)
- Solution : **créer des routes** afin de définir comment l'utilisateur navigue d'une partie de l'application à une autre.

ANGULAR : ROUTING

- Mise en place d'un **routeur** : un module situé au niveau le plus haut de l'application dédié à la gestion des routes.
- Nom du module : **AppRoutingModule**.

210

```
src > app > TS app-routing.module.ts > ...
1  import { NgModule } from '@angular/core';
2  import { RouterModule, Routes } from '@angular/router';
3
4  const routes: Routes = [];
5
6  @NgModule({
7    imports: [RouterModule.forRoot(routes)],
8    exports: [RouterModule]
9  })
10 export class AppRoutingModule { }
```

Table de routes ←

Importation du module Router (au niveau de la racine de l'application) ←

Module disponible dans toute l'application →

ANGULAR : ROUTING

- Mise en place d'un **routeur** : un module situé au niveau le plus haut de l'application dédié à la gestion des routes.
- Nom du module : **AppRoutingModule**.

```
src > app > TS app-routing.module.ts > ...
1  import { NgModule } from '@angular/core';
2  import { RouterModule, Routes } from '@angular/router';
3
4  const routes: Routes = [];
5
6  @NgModule({
7    imports: [RouterModule.forRoot(routes)],
8    exports: [RouterModule]
9  })
10 export class AppRoutingModule { }
```

Importation du module
Router (au niveau de la
racine de l'application)

{enableTracing: true}

Permet de
garder une
trace de la
recherche
d'un chemin
(*debug*)

ANGULAR : ROUTING

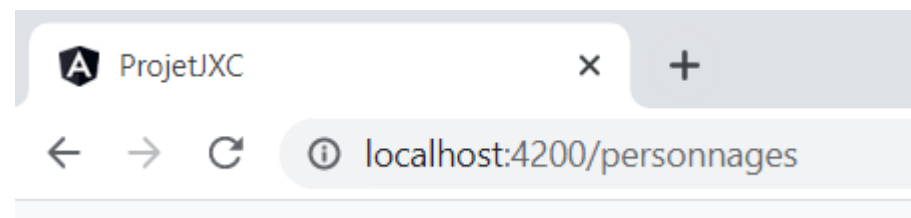
Forme basique d'une route :

- **Path** : chaîne de caractère pour l'URL
- **Component** : le composant associé à l'URL

212

```
const routes: Routes = [  
  {path: 'personnages', component: PersonnagesComponent}  
];
```

- Lorsque l'URL est demandée, le module de routage effectue le rendu du composant associé.



ANGULAR : ROUTING

Attention à l'ordre des routes (utilisation de la première route correspondante)

→ Possibilité d'effectuer une redirection :

```
const routes: Routes = [  
  {path: 'personnages', component: PersonnagesComponent},  
  {path: '', redirectTo: 'personnages', pathMatch: 'full'}  
];
```

213

→ Possibilité d'utiliser des URL dynamiques :

```
{path: 'personnage/:name', component: PersonnagesDetailComponent}
```

📄 localhost:4200/personnage/Mipha

ANGULAR : ROUTING

- Possibilité de rediriger l'utilisateur lorsqu'il rentre une URL qui n'existe pas (*wildcart route*)
- Création d'une page 404 (nouveau composant)

```
{path: '**', component: PageNotFoundComponent}
```

- A ajouter en dernier dans la table de routage !

ANGULAR : ROUTING

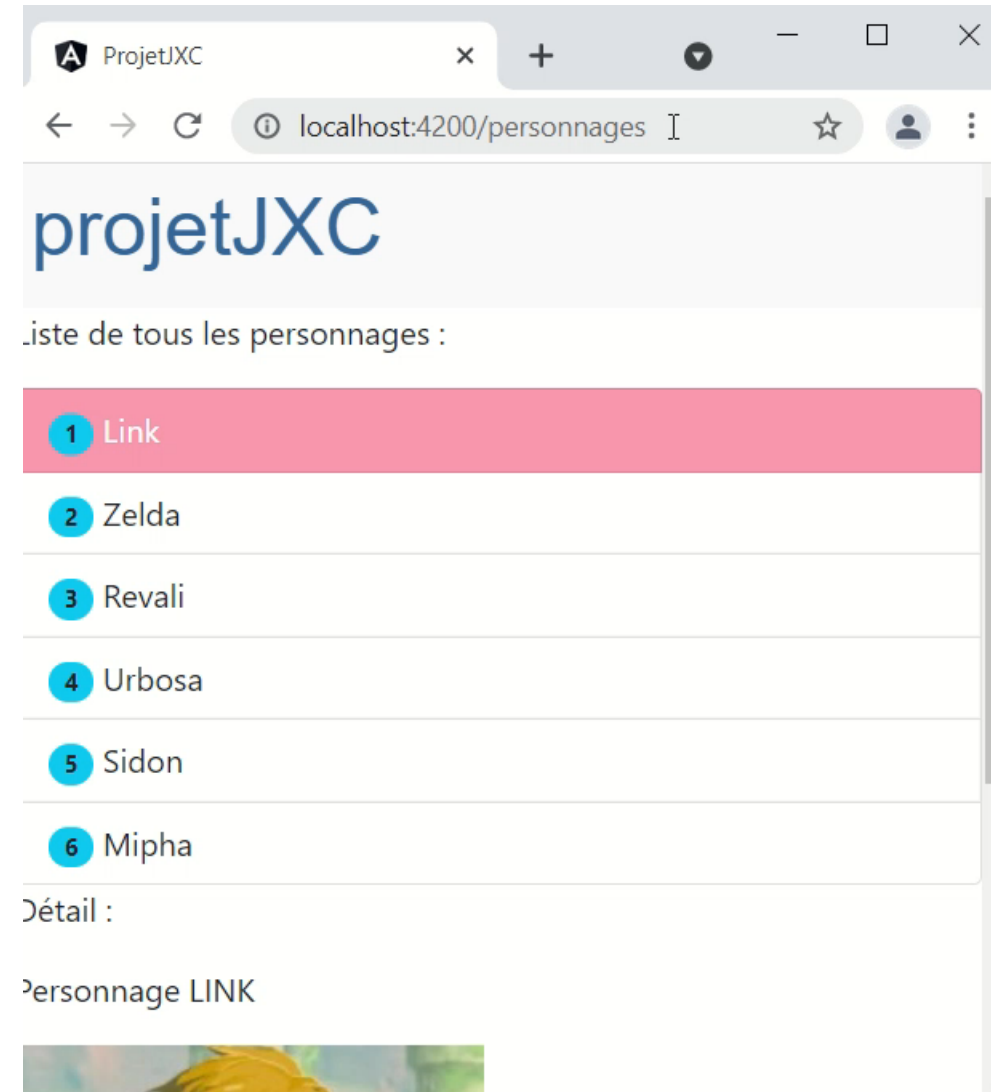
Comment intégrer les routes ?

Directive RouterOutlet

- Utilisée comme un composant
- Permet de préciser au module de routage où il doit faire le rendu des composants associés aux routes.

```
src > app > <> app.component.html > ...  
1   <header class="container-fluid bg-light p-2">  
2   |   <h1>{{title}}</h1>  
3   </header>  
4   <!-- <app-personnages></app-personnages> -->  
5   <router-outlet></router-outlet>
```

Le composant chargé sera affiché ici



ANGULAR : ROUTING

Comment naviguer dans l'application ?

1. A travers des liens : **RouterLink**
2. Directement dans le code du composant

ANGULAR : ROUTING

Problème : l'élément sera en gras pour toutes les pages visitées

Comment naviguer dans l'application ?

1. A travers des liens : **RouterLink**


```
.active {  
  font-weight: bold;  
}
```

Route active : on peut lui définir un style

217

→ Cas pour une **URL non dynamique** :

```
<nav>  
  <ul>  
    <li><a routerLink="/first-component" routerLinkActive="active">First Component</a></li>  
    <li><a routerLink="/second-component" routerLinkActive="active">Second Component</a></li>  
  </ul>  
</nav>
```



→ Possibilité de créer un composant menu

ANGULAR : ROUTING

Comment naviguer dans l'application ?

1. A travers des liens : **RouterLink**


→ Cas pour une **URL non dynamique** :

La classe est uniquement ajoutée lorsque la route correspond exactement à la valeur de *routerLink*

```
[routerLinkActiveOptions] = "{exact: true}"
```

218

```
<nav>
  <ul>
    <li><a routerLink="/first-component" routerLinkActive="active">First Component</a></li>
    <li><a routerLink="/second-component" routerLinkActive="active">Second Component</a></li>
  </ul>
</nav>
```



→ Possibilité de créer un composant menu

ANGULAR : ROUTING

Comment naviguer dans l'application ?

1. A travers des liens : **RouterLink**

→ Cas pour une **URL dynamique**:

219

```
<nav>
  <ul>
    <li><a [routerLink]="['/first-composant', monObjet.id]" routerLinkActive="active">First Component ID</a></li>
    <li><a [routerLink]="['/second-composant', monObjet.name]" routerLinkActive="active">Second Component Name</a></li>
  </ul>
</nav>
```

→ /first-composant/:id

/second-composant/:name

ANGULAR : ROUTING

Comment naviguer dans l'application ?

1. A travers des liens : **RouterLink**

→ Cas pour une **URL dynamique**:

```
<a [routerLink]="[/personnage]" [queryParams]="{ id: '1', nom: 'Zelda' }">Personnage Zelda</a>
```

→ /first-composant?id=1&nom:Zelda

ANGULAR : ROUTING

Comment naviguer dans l'application ?

2. Directement dans le code du composant

→ Besoin d'injecter une instance de **Router** dans le composant

```
constructor(private router: Router) {}
```

→ Utilisation de la méthode **navigate / navigateByUrl**

```
goBack() {  
  this.router.navigate(['previousComposant']);  
}
```

Utile pour les retours !

ANGULAR : ROUTING

Comment naviguer dans l'application ?

2. Directement dans le code du composant

→ Autre possibilité pour aller à la vue précédente :
Utilisation du service **Location** de Angular.

```
constructor(private location: Location) { }
```

```
goBack(): void {  
  this.location.back();  
}
```

ANGULAR : ROUTING

Récupération des données de routage ?

Possibilité d'utiliser une route pour passer des informations d'un composant à un autre.

→ Utilisation de l'interface **ActivatedRoute**.

→ Utilisation d'un objet de cette classe dans la méthode **ngOnInit()**

1) Les propriétés **snapshot.params / paramMap** contiennent **tous les paramètres passés à la route**.

Solution avec les snapshot (instantanée)

Solution avec les observables (asynchrone)

ANGULAR : ROUTING

Comment récupérer
« 1 » dans l'URL
/personnage/1 ?

Récupération des données de routage ?

→ Récupération de la route de la forme *personnage/:id*

224

```
constructor(private route: ActivatedRoute,  
             private router: Router, private persoService: PersonnageService) {}  
  
ngOnInit(): void {  
    const id = +this.route.snapshot.params.id; //+ pour caster id en number  
    this.persoService.getPersonnageObservable(id).subscribe(perso => this.perso = perso);  
    //Observable pour récupérer le personnage avec le bon id  
}
```

Snapshot.params

→ Paramètres de routage passés sous forme de chaînes de caractères.

ANGULAR : ROUTING

Récupération des données de routage ?

→ Récupération de la route de la forme *personnage/:id*

225

```
constructor(private route: ActivatedRoute,  
             private router: Router, private persoService: PersonnageService) {}  
  
ngOnInit(): void {  
    this.route.paramMap.subscribe( res => {  
        const id = +res.get('id')!;  
        this.perso = this.persoService.getPersonnage(id);  
    });  
}
```

paramMap

→ Paramètres de routage passés sous forme de chaînes de caractères.

ANGULAR : ROUTING

Récupération des données de routage ?

Possibilité d'utiliser une route pour passer des informations d'un composant à un autre.

→ Utilisation de l'interface **ActivatedRoute**.

→ Utilisation d'un objet de cette classe dans la méthode **ngOnInit()**

2) Les propriétés **snapshot.queryParams / queryParams** permettent de récupérer les paramètres de l'URL **sans définition d'une route**.

Solution avec les snapshot (instantanée)

Solution avec les observables (asynchrone)

ANGULAR : ROUTING

Récupération des données de routage ?

→ Récupération de la route de la forme *personnage?id=value1&name=value2*

227

```
constructor(private route: ActivatedRoute,  
             private router: Router, private persoService: PersonnageService) {}  
  
ngOnInit(): void {  
    this.route.queryParamMap.subscribe(res => {  
        this.idQuery = res.get('id') ?? '';  
        this.nomQuery = res.get('nom') ?? '';  
    });  
}
```

queryParamMap

ANGULAR : ROUTING

Récupération des données de routage ?

→ Récupération de la route de la forme *personnage?id=value1&name=value2*

228

```
constructor(private route: ActivatedRoute,  
  private router: Router, private persoService: PersonnageService) {}  
  
ngOnInit(): void {  
  this.idQuery = this.route.snapshot.queryParams.id;  
  this.nomQuery = this.route.snapshot.queryParams.nom;  
  console.log(`queryParam : id = ${this.idQuery} et nom = ${this.nomQuery}`);  
}
```

Snapshot.queryParams

ANGULAR : ROUTING

Récupération des données de routage ?

Quelle méthode choisir ?

- **Snapshot** : Si la valeur initiale des paramètres est utilisée seulement à l'initialisation du composant et ne risque pas de changer.
- **Observable** : Si la route risque de changer tout en restant dans le même composant (l'initialisation du composant à travers *ngOnInit()* ne serait pas appelée à nouveau mais l'observateur sera notifié lorsque l'URL a été modifié).

Exemple de résultat de navigation :

- Création d'un menu avec `routerLink` (`/personnages`)
- Lien sur le composant `Personnage` avec un `routerLink` et un `id`
- Exemple de récupération de deux types d'URL

The screenshot shows a web browser at `localhost:4200/personnages` displaying the 'projetJXC' application. The page has a blue header with the title 'projetJXC' and a sidebar menu titled 'Personnages'. The menu lists six items: 'Link', 'Zelda', 'Revali', 'Urbosa', 'Sidon', and 'Mipha'. The 'Link' item is highlighted in pink and has a mouse cursor over it. Below the menu is a 'Messages' section with a 'Clear messages' button and two log messages: 'HeroService: fetched heroes' and 'PersonnagesComponant: Selected hero id=1'. The Chrome DevTools console is open on the right, showing a message about Angular running in development mode and a list of log messages corresponding to the application's state.

projetJXC

Personnages

Liste de tous les personnages :

- 1 Link
- 2 Zelda
- 3 Revali
- 4 Urbosa
- 5 Sidon
- 6 Mipha

Messages

Clear messages

HeroService: fetched heroes

PersonnagesComponant: Selected hero id=1

DevTools is now available in French!

Always match Chrome's language Switch DevTools to French Don't show again

Elements Console Sources Network Performance Memory >> 1

top Filter Default levels 1 Issue: 1

Angular is running in development mode. Call `enableProdMode()` to enable production mode. [core.js:28059](#)

ngOnInit [personnages.component.ts:21](#)

Getting personnages ... [logger.service.ts:13](#)

terminé [personnages.component.ts:30](#)

[WDS] Live Reloading enabled. [index.js:52](#)

ANGULAR : ROUTING

Resolve

- Lors de l'utilisation d'API, il peut y avoir un délai avant que les données qui doivent être affichées soient retournées du serveur (affichage d'une page blanche ou d'un message par défaut).

Utiliser **Resolve** permet :

- d'attendre le retour d'un observable avant d'initialiser ou mettre à jour un composant après une mise à jour de l'url.
- de passer un paramètre dynamique à une vue dans une route.