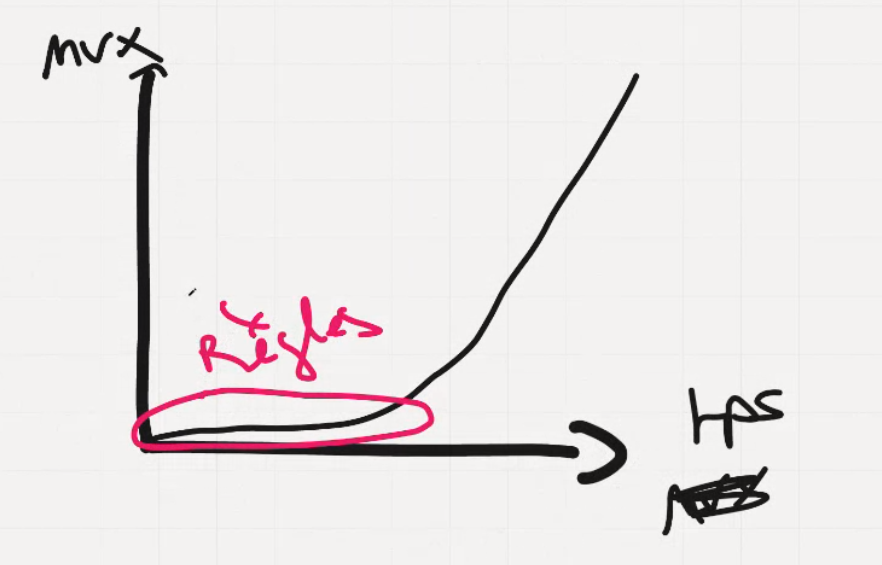
# Jour 1 – 09/12/2021

[Angular (notion.site)](https://paint-tempo-501.notion.site/Angular-b0a34d1af37c4fd6849ebc36676ef3e3)

Thomas Chevalier

## Présentation

On utilisera rarement Angular pour faire un site vitrine (trop lourd), il n'est pas fait pour..



React et vueJS + rapide à apprendre (librairies et pas framework, aucunes règles)

Meilleurs pour portfolio par exemple.

Angular mauvais en SEO

2 releases par an

Typescript -> Sur couche sémantic -> sécurise le js -> Type les variables

Typescript non lisable par navigateur -> Ensuite est traduit en vanilla js

* Transpilation (comme SASS)

Un exemple de typage strict:

class Main {

id: number; // Variable de classe typée nombre

title: string; // Variable de classe typée chaine de caractères

color: Color; // Variable de classe typée instance de classe personnalisée

constructor() {

this.id = 0;

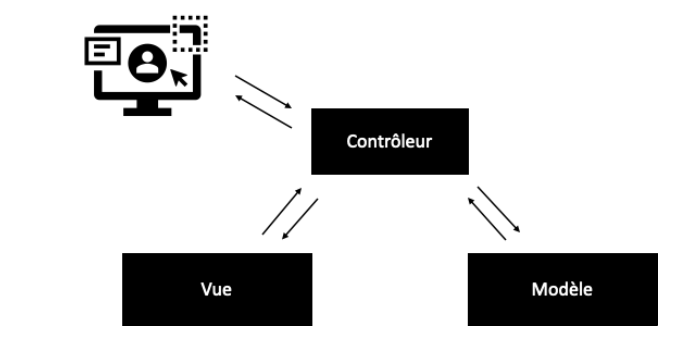
this.title = 'Mon titre';

this.color = new Color();

}

}

## Modèle MVC



### Modèle

Le modèle est la représentation d'un élément en BDD. C'est lui qui va faire la relation entre le contrôleur et la base de données.

Si on possède un site d'agence de voyages avec des offres, on possèdera un modèle qui sera le squelette d'une offre et qui nous permettra, lorsqu'on récupérera une ou plusieurs offres de structurer la donnée sous forme de classe. Tout ceci grâce au modèle.

### Contrôleur

Le contrôleur est là pour réguler et rediriger le trafic mais il aussi le cerveau des opérations.

Lorsqu'un utilisateur demande à accéder à une page, le serveur va renvoyer la demande sur le contrôleur associé. Celui-ci va ensuite récupérer les informations en BDD à travers le ou les modèles associés et va ensuite demander à la vue de faire le rendu avec toutes les informations récupérées. Il renverra ensuite le rendu à l'utilisateur

### Vue

La vue est finalement l'élément visible de l'interface, c'est elle qui va afficher le résultat de tout ce qu'aura fait le contrôleur.

Elle aura aussi le rôle de mettre à jour le modèle avec par exemple un formulaire ou un bouton de suppression.

## Installation

Extensions VSC : [Les extensions utiles sous VSCode (notion.site)](https://paint-tempo-501.notion.site/Les-extensions-utiles-sous-VSCode-dcd08bbbf4654afa9b3db22ef01d4f3b)

NodeJS : [Download | Node.js (nodejs.org)](https://nodejs.org/en/download/)

Angular npm i -g @angular/cli typescript

Ng --version

## Nouveau projet

**ng new hello-world --routing --style=css**

ng est le nom de la commande qui nous suivra sur tout le reste du cours.

new est le mot clé permettant de dire que l'on souhaite créer un nouveau projet.

hello-world est le nom de notre projet en question

--routing représente une option permettant de dire à Angular que l'on veut utiliser un router

—-style=css représente une option permettant de dire à Angular que l'on veut utiliser du CSS Vanilla dans notre projet

Une fois le projet créé, on va pouvoir se déplacer à l’intérieur :

**cd hello-world**

Puis, on va lancer le projet avec la commande suivante :

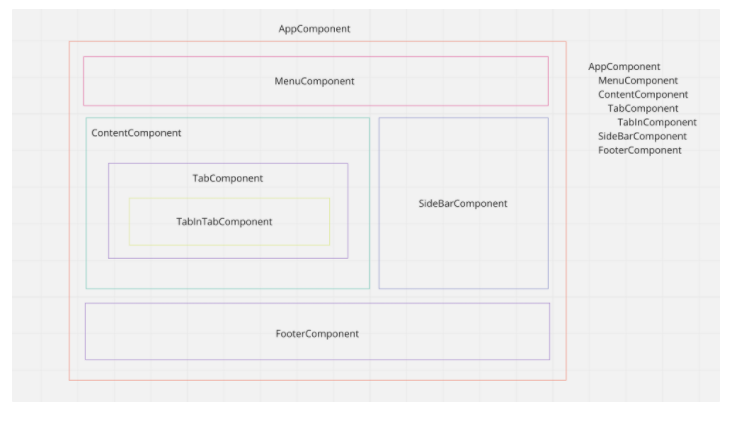
**ng serve**

app\_component a ne jamais supprimer -> fichier de base d’angular

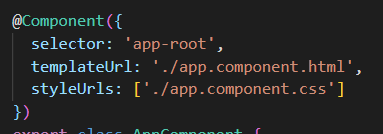
Autres composants : composants nested

### Structure

SPA = Single Page Component



App.component.ts :



Décorateur -> Instructions pour décorer la classe

**NE JAMAIS MODIFIER INDEX.HTML**



Relation avec le selecteur de app.component.ts

Appel des composants dans app.component.html

Fichier spec = fichier de test

### Génération et fonctionnement

Pour générer un composant, on utilisera la commande suivante (toujours à partir du chemin de notre projet):

ng generate component nom-du-composant ou **ng g c nom-du-component**

Un composant comportera toujours 4 fichiers:

- la vue

- le contrôleur

- le style

- les tests

La commande de génération effectuera 5 actions:

- Création d'un fichier HTML qui représente la vue du composant

- Création d'un fichier TS qui représente le contrôleur du composant

- Création d'un fichier CSS qui permet de styliser le composant

- Création d'un fichier SPEC.TS qui permet de tester le code TS

- Mise à jour du fichier `app.module.ts` pour rendre disponible le composant dans l'application

Ctrl+C pour quitter le serveur

On retrouve:

* ngOnInit() => appelé une fois seulement à la création du composant
* ngDoCheck() => détection de modifications par le framework
* ngOnChanges() => appelé quand il y a un changement de données sur un champ input
* ngAfterContentInit() => appelé une fois que le ng-content soit chargé
* ngAfterContentChecked() => appelé lors de la détection de ng-content
* ngAfterViewInit() => appelé une fois que la vue est chargée
* ngAfterViewChecked() => appelé à la détection de changement dans la vue
* ngOnDestroy() => appelé juste avant la suppression du composant

## Intéragir avec le DOM

**Ne jamais importer JQuery.**

Le contrôleur (partie TS) et la vue (partie HTML) d'un composant sont en perpétuel communication. C'est ce qu'on appelle le databinding.

Cela permet de faire transiter les informations dans un sens (TS vers HTML) comme dans l'autre (HTML vers TS)

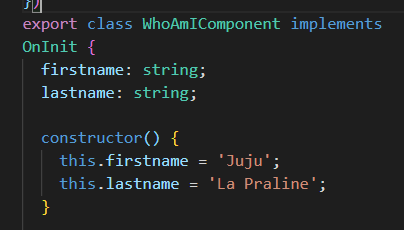
Ce système s'appelle le databinding ou liaison de données.

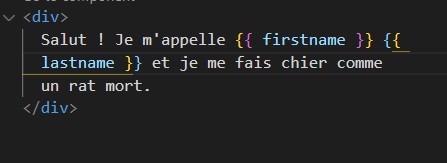
Cette liaison est possible dans deux directions différentes (mentionnées au-dessus):

* du contrôleur vers la vue (on utilisera des outils comme le property binding et la string interpolation)
* de la vue vers le contrôleur (on utilisera l'event binding)

Sa notation côté HTML est **{{ nomVariable }}.**

Varible défini avant le constructeur





ngOnInit(): void {

    setTimeout(() => {

      this.firstname = 'Plus Juju,';

      this.lastname = 'je suis décédé';

    }, 2000);

  }

### Le property binding

Le property binding ou la liaison de données sur une propriété est un concept qui va nous permettre de lier une propriété d'une balise HTML5 à une variable de notre contrôleur.

Par exemple, un bouton HTML5 peut avoir la propriété disabled ou encore hidden. Si elle sont présentes sur la balise, respectivement, elles désactiveront ou cacheront le bouton. Si ne sont pas présentes, il ne se passera rien.

En Javascript Vanilla, on pourrait ajouter à la volée cette propriété, avec Angular, on va pouvoir la lié à un booléen disponible dans le contrôleur.

Sa notation côté HTML est [propriété]="variableContrôleur".

<input type="checkbox" [checked]="isChecked" />

<button [disabled]="isDisabled">

Tempo

Affichage dans component

Var ville

Boutton qui change contenu de la ville au click

### L'event binding

L'event binding ou la liaison de données sur un évènement permet de se rapprocher de l'utilisation d'un addEventListener en Javascript Vanilla.

La différence réside encore une fois dans le fait que la vue et le contrôleur vont discuter en temps réel et que l'on va pouvoir lier un évènement côté vue à une fonction côté contrôleur.

Les événements sont identiques à ceux qui existent en Javascript Vanilla, comme le click, le mousemove, le mouseup, le mousedown, le input, le submit, le keypress, le keyup, le keydown, etc.

Sa notation côté HTML est (événement)="fonctionContrôleur".

<button (click)="onClickShowAge(45)" [disabled]="isDisabled">

  Afficher mon âge !

</button>

onClickShowAge(age: Number): void {

    alert(age);

  }

Void ne retourne pas de valeur( car pas de return).

## Propriétés personnalisées

On reconnait 2 propriétés:

* l'input ⇒ permet de faire passer une info du parent vers l'enfant
* l'output ⇒ permet de faire remonter une info de l'enfant vers le parent (sous forme d'événement)

### Input

L'`input est la propriété la plus simple à prendre en main.

On peut la comparer aux paramètres d'une fonction ou encore aux attributs d'une classe.

Notre composant va donc avoir des `inputs` (on peut dire que c'est des paramètres) qui vont permettre de le construire avec des valeurs différentes.

App component html (2 possibilités)

<app-who-am-i firstname="Paul" lastname="Doe"> </app-who-am-i>

<app-who-am-i

  [firstname]="firstNameToInput"

  [lastname]="lastNameToInput"

></app-who-am-i>

Who I am component ts

export class WhoAmIComponent implements OnInit {

  @Input() firstname: string;

  @Input() lastname: string;

//Valeurs par défauts si pas de input dans le parent

  constructor() {

    this.firstname = 'Juju';

    this.lastname = 'Le tordu';

app component ts

export class AppComponent {

  //title = 'hello-world';

  firstNameToInput: string;

  lastNameToInput: string;

  constructor() {

    this.firstNameToInput = 'Jojo';

    this.lastNameToInput = 'Lapin';

  }

}

Importer input : Ctrl + espace

### Output

L'`output` est la propriété la plus pénible à prendre en main et finalement utile que dans des cas très précis. Cependant, il est toujours bon de savoir la manipuler !

Son rôle est de permettre d'émettre un événement qui sera écoutable par le composant parent. Ainsi, le parent sera libre d'exécuter l'action qu'il souhaite lors du déclenchement de cette événement.

Prenons l'exemple d'un formulaire !

Imaginons que je possède un formulaire pour ajouter un utilisateur.

Ce formulaire contiendra le nom et le prénom de l'utilisateur (formulaire très simple)

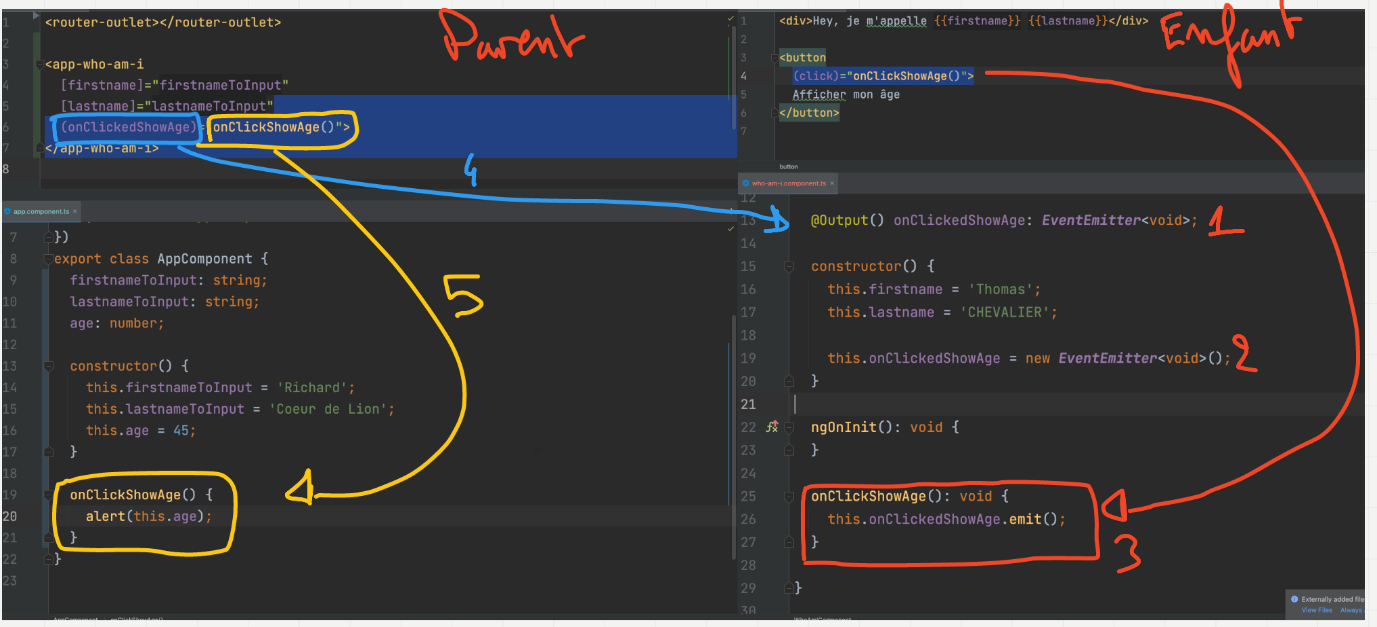
Imaginons maintenant que je possède un formulaire pour modifier un utilisateur.

Ce formulaire contiendra les mêmes informations à savoir le nom et le prénom.

À ce moment là, on ne va pas recréer 2 fois le formulaire, ce serait une erreur en terme de maintenabilité. On va donc en faire un composant !

Oui mais.. Notre formulaire, lorsqu'il va être soumis devra faire 2 actions différentes, d'un côté ajouter l'utilisateur et de l'autre le modifier.

Et c'est là qu'intervient l'`output`. Plutôt que d'exécuter une fonction dans le composant de formulaire, ce dernier va renvoyer l'événement pour dire que la demande de soumission a été envoyé et c'est à chaque parent d'écouter sur cet événement et de lancer sa propre fonction.



# Jour 2 – 10/12/2021

## DOM partie 1 exo 2 Tableau

<!-- app component html -->

<div>

  <app-table></app-table>

</div>

<!-- table component html -->

<table class="table">

  <thead>

    <tr>

      <th scope="col">ID</th>

      <th scope="col">Username</th>

      <th scope="col">Adresse e-mail</th>

    </tr>

  </thead>

  <tbody>

    <tr

      app-row

      \*ngFor="let user of users"

      [id]="user.id"

      [username]="user.username"

      [mail]="user.email"

    ></tr>

  </tbody>

</table>

// table component ts

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

@Component({

  selector: 'app-table',

  templateUrl: './table.component.html',

  styleUrls: ['./table.component.css'],

})

export class TableComponent implements OnInit {

  users: Array<any> = [];

  constructor() {}

  ngOnInit(): void {

    this.users = [];

    this.users.push({

      id: 1,

      username: 'toto',

      email: 'toto@gmail.com',

    });

    this.users.push({

      id: 2,

      username: 'toto2',

      email: 'toto2@gmail.com',

    });

    this.users.push({

      id: 3,

      username: 'toto3',

      email: 'toto3@gmail.com',

    });

    this.users.push({

      id: 4,

      username: 'toto4',

      email: 'toto4@gmail.com',

    });

  }

}

<!-- Row component html -->

<th scope="row">{{ id }}</th>

<td>{{ username }}</td>

<td>{{ mail }}</td>

//row component ts

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

@Component({

  selector: '[app-row]',

  templateUrl: './row.component.html',

  styleUrls: ['./row.component.css'],

})

export class RowComponent implements OnInit {

  @Input() id: string;

  @Input() username: string;

  @Input() mail: string;

  constructor() {

    this.id = '0';

    this.username = 'Jojo86';

    this.mail = 'toto@gmail.com';

  }

  ngOnInit(): void {}

}

### Transformation balise en attribut

Transformer app-row avec [] pour transformer de balise à attribut

### Boucle for :

<tr

      app-row

      \*ngFor="let user of users"

      [id]="user.id"

      [username]="user.username"

      [mail]="user.email"

    ></tr>

## DOM partie 1 exo 3 boutton + compteur

### V1 sans output

<!-- app component html -->

<router-outlet></router-outlet>

<div class="container mt-4">

  <div>

    <app-button></app-button>

  </div>

</div>

// button component ts

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

@Component({

  selector: 'app-button',

  templateUrl: './button.component.html',

  styleUrls: ['./button.component.css'],

})

export class ButtonComponent implements OnInit {

  compteur: number;

  constructor() {

    this.compteur = 0;

  }

  ngOnInit(): void {}

  onClickIncrementCompteur(): void {

    this.compteur++;

  }

  onClickDecrementCompteur(): void {

    this.compteur--;

  }

}

<!-- button component html -->

<p>Valeur du compteur: {{ compteur }}</p>

<button

  type="button"

  (click)="onClickIncrementCompteur()"

  class="btn btn-primary"

>

  Incrémentation

</button>

<button

  type="button"

  (click)="onClickDecrementCompteur()"

  class="btn btn-primary"

>

  Décrémentation

</button>

### V2 avec output

<!-- app component html -->

<router-outlet></router-outlet>

<div class="container mt-4">

  <div>

    <p>Valeur du compteur: {{ compteur }}</p>

  </div>

  <app-button

    (clickButtonEmitter)="incCnt()"

    labelButton="Incrémentation"

  ></app-button>

  <app-button

    (clickButtonEmitter)="decCnt()"

    labelButton="Décrémentation"

  ></app-button>

</div>

//app component ts

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

@Component({

  selector: 'app-root',

  templateUrl: './app.component.html',

  styleUrls: ['./app.component.css'],

})

export class AppComponent {

  title = 'exo-dom3';

  compteur: number;

  constructor() {

    this.compteur = 0;

  }

  incCnt() {

    this.compteur++;

  }

  decCnt() {

    this.compteur--;

  }

}

<!-- button component html -->

<button type="button" (click)="onClickButton()" class="btn btn-primary">

  {{ labelButton }}

</button>

// button component ts

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

@Component({

  selector: 'app-button',

  templateUrl: './button.component.html',

  styleUrls: ['./button.component.css'],

})

export class ButtonComponent implements OnInit {

  @Input() labelButton: string;

  //Envoie l'évènement du boutton vers le parent(click)

  @Output() clickButtonEmitter = new EventEmitter();

  constructor() {

    this.labelButton = 'Boutton';

  }

  ngOnInit(): void {}

  onClickButton() {

    this.clickButtonEmitter.emit();

  }

}

## DOM partie 1 exo 4 cri animaux

<!-- app component html -->

<router-outlet></router-outlet>

<app-cards

  \*ngFor="let user of animals"

  [name]="user.name"

  (clickButtonEmitter)="criOnClick(user.cri)"

>

</app-cards>

// app ts

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

@Component({

  selector: 'app-root',

  templateUrl: './app.component.html',

  styleUrls: ['./app.component.css'],

})

export class AppComponent {

  title = 'exo-dom4-animaux';

  animals: Array<any> = [];

  constructor() {}

  ngOnInit(): void {

    this.animals = [];

    this.animals.push({

      name: 'Chien',

      cri: 'Waouf',

    });

    this.animals.push({

      name: 'Chat',

      cri: 'Miaou',

    });

  }

  criOnClick(cri: string) {

    alert(cri);

  }

}

<!-- cards html -->

<div class="container">

  <div class="card" style="width: 18rem">

    <img src="..." class="card-img-top" alt="Image animaux" />

    <div class="card-body">

      <h5 class="card-title">{{ name }}</h5>

      <button (click)="onClickButton()" type="button" class="btn btn-primary">

        Quel est mon cri?

      </button>

    </div>

  </div>

</div>

// cards ts

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

import { ChildrenOutletContexts } from '@angular/router';

@Component({

  selector: 'app-cards',

  templateUrl: './cards.component.html',

  styleUrls: ['./cards.component.css'],

})

export class CardsComponent implements OnInit {

  @Input() name: string;

  @Input() cri: string;

  //Envoie l'évènement du boutton vers le parent(click)

  @Output() clickButtonEmitter = new EventEmitter();

  constructor() {

    this.name = '';

    this.cri = '';

  }

  ngOnInit(): void {}

  onClickButton() {

    this.clickButtonEmitter.emit();

  }

}

## DOM partie 1 exo 5

//Initialisation pour la valaur de base à l'affichage (on veut index 0 à l'ouverture):On le trigger manuellement

    this.newCarSelectedEmitter.emit(this.cars[0]);

<app-table (newCarSelectedEmitter)="updateSelectedCar($event)"></app-table>

  <!-- ($event) récupère la valeur initialisé dans table -->

<!-- app html -->

<router-outlet></router-outlet>

<div class="container pb-2">

  <div class="card mx-auto text-center mb-5">

    <app-card [marque]="car.marque" [modele]="car.modele"> </app-card>

  </div>

  <app-table (newCarSelectedEmitter)="updateSelectedCar($event)"></app-table>

  <!-- ($event) récupère la valeur initialisé dans table -->

</div>

// app ts

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

@Component({

  selector: 'app-root',

  templateUrl: './app.component.html',

  styleUrls: ['./app.component.css'],

})

export class AppComponent {

  title = 'exo-dom5-voitures';

  car: any;

  constructor() {

    this.car = {

      marque: '',

      modele: '',

    };

  }

  updateSelectedCar(car: string) {

    this.car = car;

  }

}

<!-- table component html -->

<table class="table">

  <thead>

    <tr>

      <th scope="col">ID</th>

      <th scope="col">Marque</th>

      <th scope="col">Modèle</th>

      <th scope="col">Actions</th>

    </tr>

  </thead>

  <tbody>

    <tr

      app-line

      \*ngFor="let car of cars"

      [id]="car.id"

      [marque]="car.marque"

      [modele]="car.modele"

      (showDetailEmitter)="showDetailsSelectedCar($event)" //Récupération de l’id de la ligne sélectionner

    ></tr>

  </tbody>

</table>

// table ts

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

@Component({

  selector: 'app-table',

  templateUrl: './table.component.html',

  styleUrls: ['./table.component.css'],

})

export class TableComponent implements OnInit {

  @Output() newCarSelectedEmitter = new EventEmitter();

  cars: Array<any> = [];

  constructor() {}

  ngOnInit(): void {

    this.cars = [];

    this.cars.push({

      id: 0,

      marque: 'Renault',

      modele: 'Clio',

    });

    this.cars.push({

      id: 1,

      marque: 'Peugeot',

      modele: '308',

    });

    //Initialisation pour la valaur de base à l'affichage (on veut index 0 à l'ouverture):On le trigger manuellement

    this.newCarSelectedEmitter.emit(this.cars[0]);

  }

  showDetailsSelectedCar(carId: number) {

    this.newCarSelectedEmitter.emit(this.cars[carId]);

  }

}

<!-- line html -->

<th scope="row">{{ id }}</th>

<td>{{ marque }}</td>

<td>{{ modele }}</td>

<td>

  <button type="button" (click)="onClickButton()" class="btn btn-primary">

    Détails

  </button>

</td>

// line ts

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

@Component({

  selector: '[app-line]',

  templateUrl: './line.component.html',

  styleUrls: ['./line.component.css'],

})

export class LineComponent implements OnInit {

  @Input() id: string;

  @Input() marque: string;

  @Input() modele: string;

  @Output() showDetailEmitter = new EventEmitter();

  constructor() {

    this.id = '';

    this.marque = '';

    this.modele = '';

  }

  ngOnInit(): void {}

  onClickButton() {

    this.showDetailEmitter.emit(+this.id);

  }

}

<!-- cards html -->

<div class="card" style="width: 18rem">

  <div class="card-body">

    <h5 class="card-title">{{ marque }}</h5>

    <p>{{ modele }}</p>

  </div>

</div>

// card ts

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

@Component({

  selector: 'app-card',

  templateUrl: './card.component.html',

  styleUrls: ['./card.component.css'],

})

export class CardComponent implements OnInit {

  @Input() marque: string;

  @Input() modele: string;

  constructor() {

    this.marque = '';

    this.modele = '';

  }

  ngOnInit(): void {}

}

## DOM partie 2 – cours

Pour ajouter une condition côté HTML, on utilisera la directive `\*ngIf`.

Pour boucler et créer un élément selon un tableau, on utilisera la directive `\*ngFor`.

### Utilisation \*ngIf

Comme on a pu le voir avec le `property binding`, il est possible d'interpréter du Javascript entre les `""`. (Ex: `[disabled]="duJsIci"`)

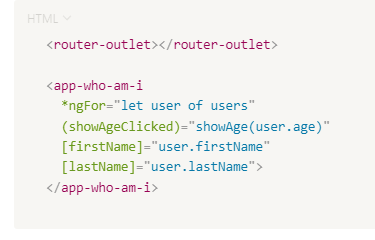
Le fonctionnement du `\*ngIf` va être assez similaire, on va lui donner la condition à l'intérieur des `" "`.



On peut voir ici que la condition se fait sur la variable firstName qui est l'input reçu depuis le parent. Selon sa valeur, je modifierais ou non le champ firstName.



### Utilisation \*ngFor



On peut voir que la directive \*ngFor se positionne sur l'élément que l'on souhaite dupliquer grâce à la boucle.

users: any[];

any = tous types de valeures

users: any[];

  constructor() {

    this.users = [];

    // this.firstNameToInput = 'Jojo';

    // this.lastNameToInput = 'Lapin';

    // this.age = 35;

    const userOne = {

      firstname: 'Julien',

      lastname: 'La Malice',

      age: 35,

    };

    this.users.push(userOne);

    const userTwo = {

      firstname: 'Bruno',

      lastname: 'Le malin',

      age: 38,

    };

    this.users.push(userTwo);

  }

onClickShowAge(age: number) {

    alert(age);

  }

onClickAddUser() {

    this.users.push({

      firstname: 'Toto',

      lastname: 'Le bargeot',

      age: 12,

    });

  }

<app-who-am-i

  \*ngFor="let user of users"

  [firstname]="user.firstname"

  [lastname]="user.lastname"

  (onClickedShowAge)="onClickShowAge(user.age)"

></app-who-am-i>

<button (click)="onClickAddUser()">Ajouter un user</button>

### Les directives par attribute

Contrairement aux directives structurelles, les directives par attribut vont permettre de modifier un composant une fois qu'il aura été créé.

On reconnait plusieurs de ces directives mais on va en aborder 2:

- `ngStyle` pour assigner dynamiquement du style

- `ngClass` pour assigner dynamiquement des classes

#### Utilisation de ngStyle

L'attribut `ngStyle` permet d'appliquer du style de manière dynamique sur un élément.

C'est un attribut mis à disposition par Angular qui acceptera un objet Javascript dont les clés seront les propriétés CSS et les valeurs, les valeurs à appliquer.

Cet attribut accepte évidemment le `property binding` et va nous permettre par exemple d'utiliser une fonction pour renvoyer un style généré selon certaines conditions.

Nous allons faire en sorte de mettre en majuscule les index impairs et en minuscule les index pairs





\*ngFor="let user of users; let i = index" i=0

Let I = index pour récupérer l’index de la boucle

Ou :

index as i

<app-who-am-i

  \*ngFor="let user of users; let i = index" i=0

  [ngStyle]="{ color: i % 2 === 0 ? 'blue' : 'red' }"

Si I pair -> ligne en bleu, sinon ligne en rouge

On peut utiliser un tableau de style qui va retourner une valeure

  generateStyles(index: number): any {

    const styles = {

      color: 'blue',

      textTransform: '',

    };

    if (index % 2 !== 0) {

      styles.color = 'red';

    } else {

      styles.textTransform = 'uppercase';

    }

    return styles;

  }

[ngStyle]="generateStyles(i)"

#### Utilisation de ngClass

Nécessite Bootstrap

L'attribut `ngClass` permet d'attribuer dynamiquement des classes à un élément.

Tout comme le `ngStyle`, c'est un élément qui est mis à disposition par Angular.

Son fonctionnement est le suivant: il accepte un objet dans lequel les clés seront les classes à appliquer et les valeurs seront les conditions pour appliquer les clés.

[ngClass]="{

'toto': users[0].firstname === 'Julien',

'tata': users[0].firstname !== 'Julien'}

"

Ajoute class toto si Julien, sinon Tata

#### Animation grace à ngstyle

L'attribut ngStyle permettant de changer du style à la volée. Il est possible de l'utiliser pour modifier par exemple la position d'un élément en temps réel.

Ainsi, si on change à de multiples reprises une position d'un élément, celui va se déplacer.

On peut donc se servir d'une ngStyle et de Javascript pur comme la fonction setInterval pour animer un élément.

# Jour 3 – 13/12/2021

## DOM partie 2 exo1

<!-- app html -->

<router-outlet></router-outlet>

<div class="container">

  <app-table></app-table>

</div>

<!-- table html -->

<table class="table">

  <thead>

    <tr>

      <th scope="col">ID</th>

      <th scope="col">Username</th>

      <th scope="col">Adresse e-mail</th>

    </tr>

  </thead>

  <tbody>

    <tr

      app-line

      \*ngFor="let user of users; index as i"

      [ngStyle]="generateStyles(i)"

      [id]="user.id"

      [username]="user.username"

      [mail]="user.mail"

    ></tr>

  </tbody>

</table>

// table ts

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

@Component({

  selector: 'app-table',

  templateUrl: './table.component.html',

  styleUrls: ['./table.component.css'],

})

export class TableComponent implements OnInit {

  users: Array<any> = [];

  constructor() {}

  ngOnInit(): void {

    this.users = [];

    this.users.push({

      id: 1,

      username: 'toto',

      mail: 'toto@gmail.com',

    });

    this.users.push({

      id: 2,

      username: 'toto2',

      mail: 'toto2@gmail.com',

    });

    this.users.push({

      id: 3,

      username: 'Robert',

      mail: 'toto3@gmail.com',

    });

    this.users.push({

      id: 4,

      username: 'toto4',

      mail: 'toto4@gmail.com',

    });

  }

  generateStyles(index: number): any {

    const styles = {

      color: 'blue',

      textTransform: '',

    };

    if (index % 2 !== 0) {

      styles.color = 'green';

    }

    return styles;

  }

}

<!-- Line component html -->

<th scope="row">{{ id }}</th>

<td \*ngIf="username !== 'Robert'">{{ username }}</td>

<td \*ngIf="username === 'Robert'">Tom</td>

<td>{{ mail }}</td>

// line ts

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

@Component({

  selector: '[app-line]',

  templateUrl: './line.component.html',

  styleUrls: ['./line.component.css'],

})

export class LineComponent implements OnInit {

  @Input() id: string;

  @Input() username: string;

  @Input() mail: string;

  constructor() {

    this.id = '';

    this.username = '';

    this.mail = '';

  }

  ngOnInit(): void {}

}

## Cours animation

### Square with arrow

<!-- square html -->

<div class="container mt-5">

  <div class="text-center">

    <div class="d-flex justify-content-center mb-2">

      <button (click)="move('top')" class="btn btn-outline-dark">T</button>

    </div>

    <div class="d-flex justify-content-center mb-2">

      <button (click)="move('left')" class="btn btn-outline-dark me-4">

        L

      </button>

      <button (click)="move('right')" class="btn btn-outline-dark ms-4">

        R

      </button>

    </div>

    <div class="d-flex justify-content-center">

      <button (click)="move('bottom')" class="btn btn-outline-dark">B</button>

    </div>

  </div>

  <div [ngStyle]="squareStyles"></div>

</div>

// square component ts

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

import { timeout } from 'rxjs';

@Component({

  selector: 'app-square-with-arrows',

  templateUrl: './square-with-arrows.component.html',

  styleUrls: ['./square-with-arrows.component.css'],

})

export class SquareWithArrowsComponent implements OnInit {

  squareStyles: any;

  constructor() {

    this.squareStyles = {

      position: 'relative',

      width: '100px',

      height: '100px',

      backgroundColor: 'black',

      top: '0px',

      left: '0px',

    };

    let i = 0;

    const squareInit = setInterval(() => {

      this.squareStyles.top = parseInt(this.squareStyles.top) + 10 + 'px';

      this.squareStyles.left = parseInt(this.squareStyles.left) + 10 + 'px';

      i++;

      if (i === 25) {

        clearInterval(squareInit);

      }

    }, 100);

  }

  ngOnInit(): void {}

  move(to: string) {

    switch (to) {

      case 'left':

        this.squareStyles.left = parseInt(this.squareStyles.left) - 10 + 'px';

        break;

      case 'top':

        this.squareStyles.top = parseInt(this.squareStyles.top) - 10 + 'px';

        break;

      case 'bottom':

        this.squareStyles.top = parseInt(this.squareStyles.top) + 10 + 'px';

        break;

      case 'right':

        this.squareStyles.left = parseInt(this.squareStyles.left) + 10 + 'px';

        break;

      default:

        console.log('erreur');

    }

  }

}

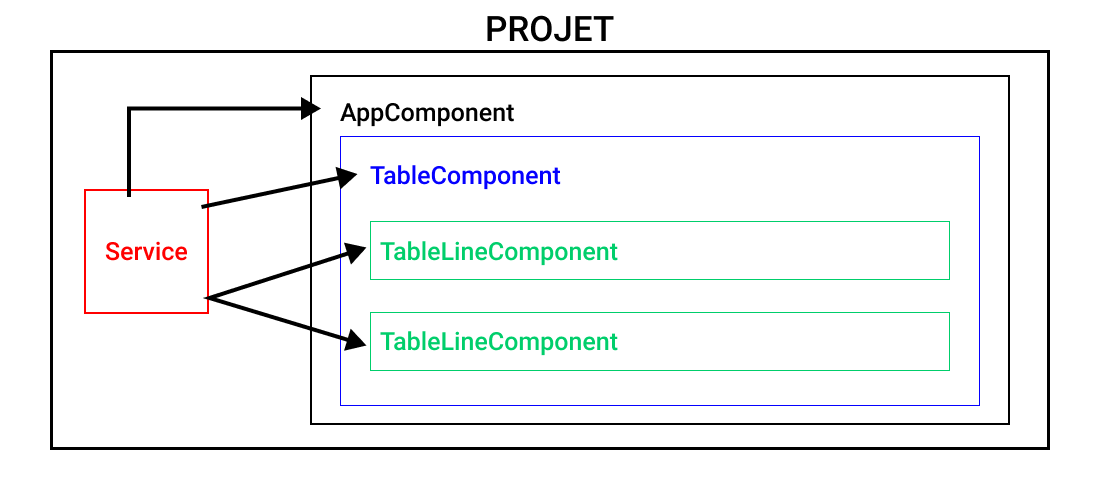
### Services et observables

Nous avons vu jusqu'à présent comment faire pour communiquer des informations entre deux composants dits "parent-enfant" notamment grâce aux `inputs`. Mais comment faire lorsque les composants n'ont aucunes hiérarchies ou encore qu'on souhaite mettre à jour facilement les données ?

Et bien, il existe ce qu'on appelle les services !

Ce sont des éléments qui sont des electrons libres et qui vont être disponibles partout dans notre application.

On va pouvoir les appeler d'où on le souhaite et ils vont centraliser les données pour nous.



On peut voir que le service fait partie du projet mais qu'il est disponible dans n'importe quel composant.

**Un service par type de donnée**

#### Les services

Pour créer un service, on utilisera la commande suivante:

ng generate service services/nomDuDossier/nomDuService

Par convention, on nommera nomDuDossier et nomDuService au singulier.

Si on souhaite créer un service pour gérer des livres, on aura donc la commande suivante:

ng generate service services/book/book

Par défaut, la commande ne fera que la création de 2 fichiers:

* le service en lui-même
* le fichier de test unitaire associé

On reconnait 3 grands types de services:

* utilitaire & contrôleur
* de données
* guard

Pour utiliser les services, il est nécessaire de dire à Angular qu'ils existent, c'est ce qu'on appelle "les injecter".

Pour cela, il y a 2 possibilités:

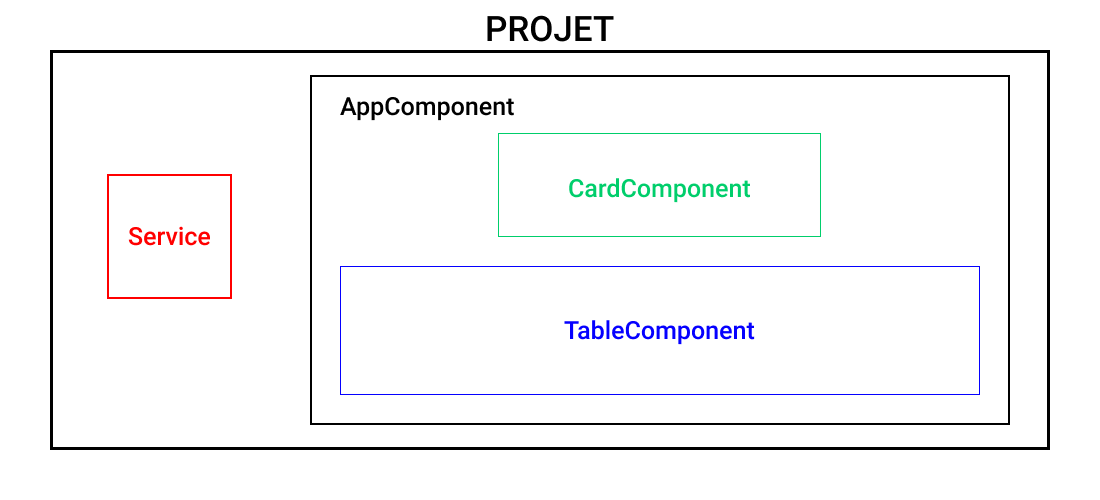
* à partir du fichier app.module.ts
* à partir d'un composant

La différence réside dans les possibilités d'accès à une information partagée.

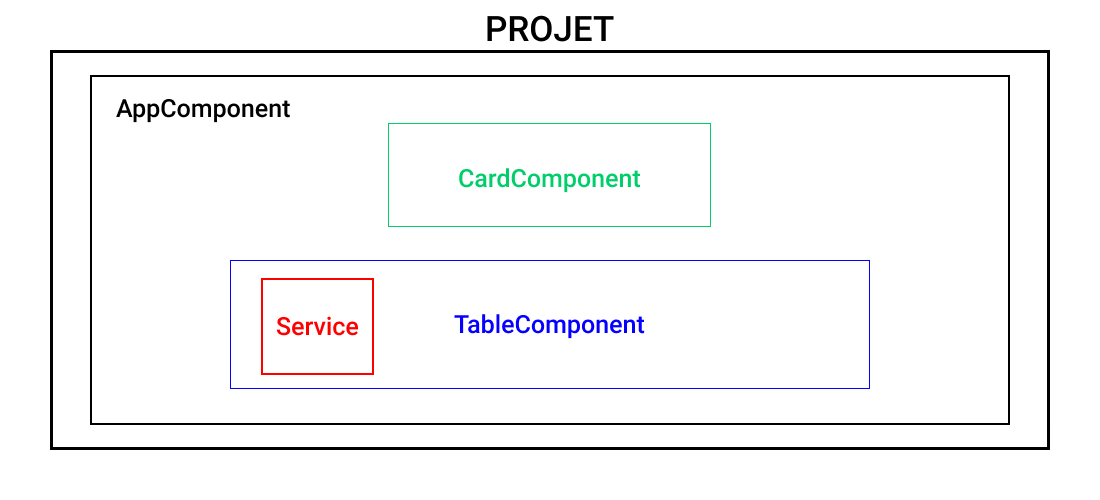
Dans le cas d'une injection dans le app.module.ts toute l'application aura accès à la même instance du service, donc tout le monde aura accès à la même chose. (la méthode la plus utilisée)

Dans le cas d'une injection dans un composant spécifique, seul le composant et ses enfants partageront la même instance. Tous les autres auront une autre instance et donc potentiellement des données différentes.

Exemple de schéma avec une injection dans le app.module.ts:



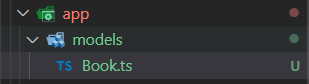
Exemple de schéma avec une injection dans un composant spécifique:



## Exo bibliothèque / Création de classe

ng generate service services/book/book

Classe à créer dans un dossier models (manuellement



Classe est le « moule », pour contrôler la forme du produit. Elle est injectée dans le service

export class Book {

  //attributs qu'on veut conserver

  //private : élément accessible seulement dans l'objet

  private \_id: number;

  private \_title: string;

  private \_description: string;

  private \_author: string;

  private \_available: boolean;

 constructor(

    id: number,

    title: string,

    description: string,

    author: string,

    available: boolean

  ) {

    //attributs qu'on veut conserver

    //private : élément accessible seulement dans l'objet

    this.\_id = id;

    this.\_title = title;

    this.\_description = description;

    this.\_author = author;

    this.\_available = available;

  }

Ctrl+shift+p generate all getter and setters

/\*\*

   \* Getter id

   \* @return {number}

   \*/

  get id(): number {

    return this.\_id;

  }

  /\*\*

   \* Setter id

   \* @param {number} value

   \*/

  set id(value: number) {

    this.\_id = value;

  }

  /\*\*

   \* Getter title

   \* @return {string}

   \*/

  get title(): string {

    return this.\_title;

  }

  /\*\*

   \* Getter description

   \* @return {string}

   \*/

  get description(): string {

    return this.\_description;

  }

  /\*\*

   \* Getter author

   \* @return {string}

   \*/

  get author(): string {

    return this.\_author;

  }

  /\*\*

   \* Getter available

   \* @return {boolean}

   \*/

  get available(): boolean {

    return this.\_available;

  }

  /\*\*

   \* Setter title

   \* @param {string} value

   \*/

  set title(value: string) {

    this.\_title = value;

  }

  /\*\*

   \* Setter description

   \* @param {string} value

   \*/

  set description(value: string) {

    this.\_description = value;

  }

  /\*\*

   \* Setter author

   \* @param {string} value

   \*/

  set author(value: string) {

    this.\_author = value;

  }

  /\*\*

   \* Setter available

   \* @param {boolean} value

   \*/

  set available(value: boolean) {

    this.\_available = value;

  }

##### Création de service

// Book.service.ts

import { Injectable } from '@angular/core';

import { Book } from 'src/app/models/Book';

@Injectable({

  providedIn: 'root',

})

export class BookService {

  books: Book[];

  constructor() {

    this.books = [];

    //Création du tableau

    for (let i = 1; i < 11; i++) {

      this.books.push(

        new Book(i, 'Titre ' + i, 'Description ' + i, 'Auteur ' + i, true)

      );

    }

  }

##### Injecter le service dans le tableau Providers

// app module .ts

import { NgModule } from '@angular/core';

import { BrowserModule } from '@angular/platform-browser';

import { AppRoutingModule } from './app-routing.module';

import { AppComponent } from './app.component';

import { BookService } from './services/book/book.service';

import { BooksListComponent } from './views/books-list/books-list.component';

@NgModule({

  declarations: [AppComponent, BooksListComponent],

  imports: [BrowserModule, AppRoutingModule],

  providers: [BookService], //Importer le bookservice

  bootstrap: [AppComponent],

})

export class AppModule {}

##### Création d’un component books-list dans dossier views

// books list ts

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

import { Book } from 'src/app/models/Book';

import { BookService } from 'src/app/services/book/book.service';

@Component({

  selector: 'app-books-list',

  templateUrl: './books-list.component.html',

  styleUrls: ['./books-list.component.css'],

})

export class BooksListComponent implements OnInit {

  books!: Book[]; //! permet d'overrider a la compilation pour instancier + tard (dans le ngOnInit)

  constructor(private bookService: BookService) {}

  ngOnInit(): void {

//Les livres sont ceux crées dans le service

    this.books = this.bookService.books;

  }

}

<!-- books-list html -->

<div class="container mt-5">

  <h1 class="text-center my-4">Books Manager</h1>

  <table class="table">

    <thead class="table-dark">

      <tr>

        <th scope="col">ID</th>

        <th scope="col">Title</th>

        <th scope="col">Author</th>

        <th scope="col">Available</th>

      </tr>

    </thead>

    <tbody>

      <tr \*ngFor="let book of books">

        <th scope="row">{{ book.id }}</th>

        <td>{{ book.title }}</td>

        <td>{{ book.author }}</td>

        <td>{{ book.available ? "Dispo" : "Pas dispo" }}</td>

      </tr>

    </tbody>

  </table>

</div>

##### Boutons a ajouter pour switcher tous les status

<div class="d-flex justify-content-center my-4">

    <button

      (click)="onClickUpdateAllStatus(true)"

      class="btn btn-outline-dark mx-2"

    >

      Tous dispo

    </button>

    <button

      (click)="onClickUpdateAllStatus(false)"

      class="btn btn-outline-dark mx-2"

    >

      Aucun de dispo

    </button>

  </div>

 // books-list .ts

  //Method called when user want to switch all books status

  onClickUpdateAllStatus(newStatus: boolean) {

    this.bookService

      .updateAllStatus(newStatus) //Promesse (type de retour attendu)call asynchrone lié à une fonction callback -> Travail avec un serveur

      .then((books: Book[]) => {

        this.books = books; //promesse tenue -> Succès

      })

      .catch(() => {

        //promesse non tenue -> erreurs (ici vide, a supprimer)

      });

  }

//Book.service.ts

  //Change  all books status

  updateAllStatus(newStatus: boolean): Promise<Book[]> {

    return new Promise<Book[]>((res, rej) => {

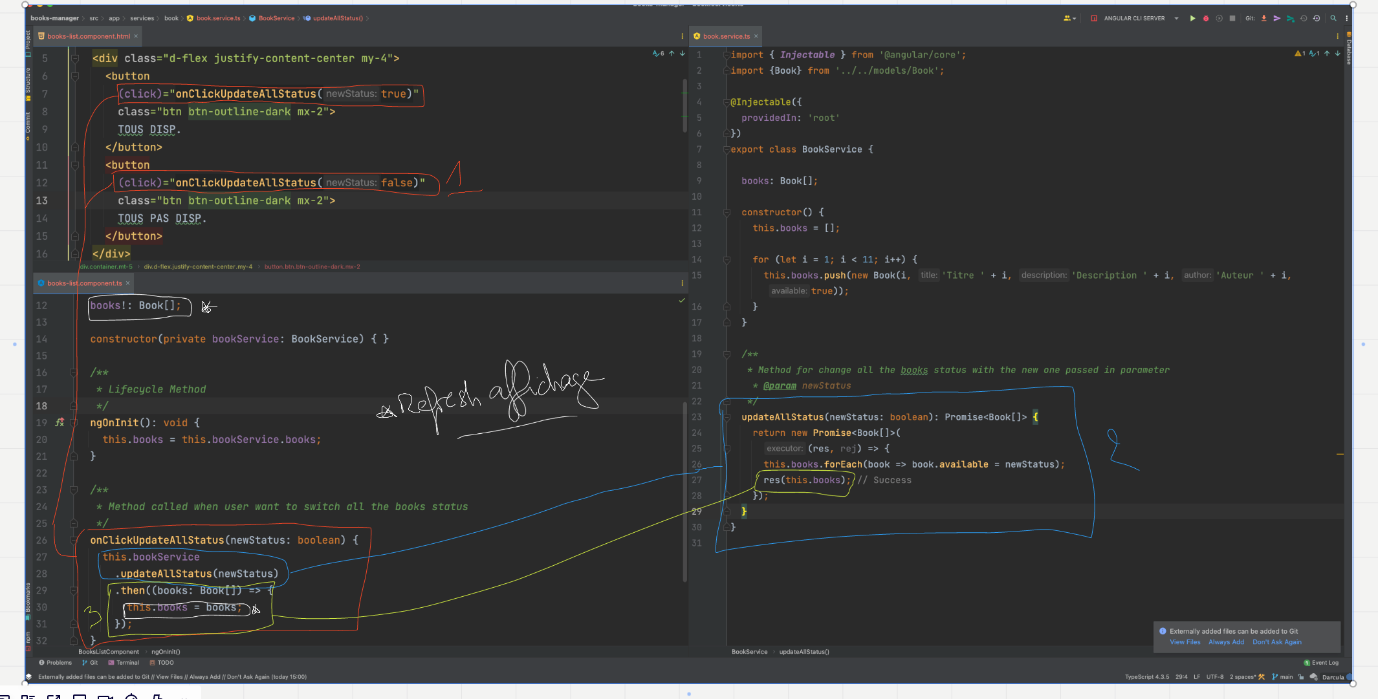
      //res= resolve rej=reject

      this.books.forEach((Book) => (book.available = newStatus));

      res(this.books);

    });

  }



##### Ajout bouton pour switcher individuellement

Ajout de l’index = i dans la boucle for et ajout d’un bouton

 <tr \*ngFor="let book of books; index as i">

        <th scope="row">{{ book.id }}</th>

        <td>{{ book.title }}</td>

        <td>{{ book.author }}</td>

        <td>{{ book.available ? "Dispo" : "Pas dispo" }}</td>

        <td>

          <button

          (click)="onClickUpdateStatus(i)"

          class="btn btn-outline-dark">

            Switch en {{ book.available ? "Pas Dispo" : "Dispo" }}

          </button>

        </td>

      </tr>

 // books-list .ts

  //Method called when user want to switch to a specific book status

  onClickUpdateStatus(arrayIndex: number){

this.bookService

      .updateStatusForBookId(this.books[arrayIndex].id)

      .then((book: Book) => {

        this.books[arrayIndex] = book;

      });

  }

// Book.service.ts

  //Toggle the status

  updateStatusForBookId(bookId: number): Promise<Book> {

    return new Promise<Book>((res, rej) => {

      //for (let book of this.books)

      for (let [index, book] of this.books.entries()) {

        if (book.id === bookId) {

          this.books[index].available = !this.books[index].available;

          res(this.books[index]);

          break;

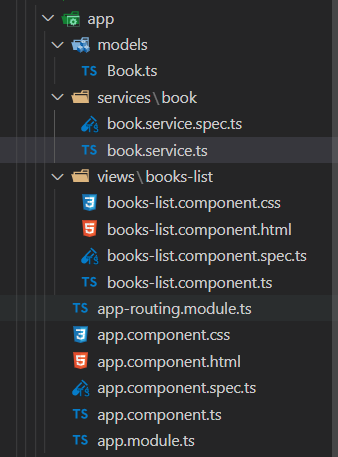
        }

      }

    });

  }

##### Résumé de l’arbre





# Jour 4 14/12/2021

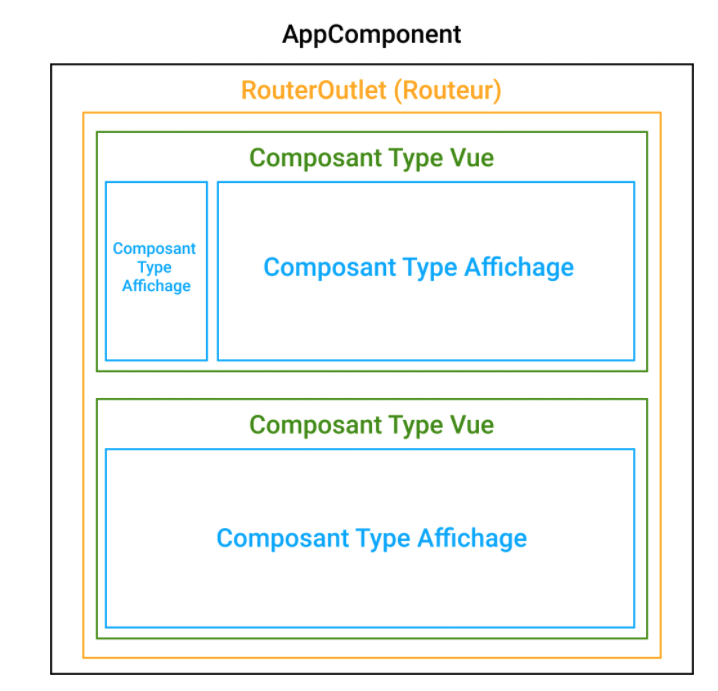
## Cours : Le routeur et les vues

### Composant Conteneur VS Composant de Présentation

On a vu jusqu'à présent que l'on pouvait créer des composants et on en a créé plusieurs.

Pour la suite du cours et de l'apprentissage, il est nécessaire de comprendre qu'un composant peut avoir 2 fonctions différentes:

* celle de contenir d'autres composants ⇒ composant dit "container" (ou que j'appellerais des vues)
* celle d'afficher des informations ⇒ composant dit "presentational" (ou que j'appellerais des composants d'affichage)



### Les composants type vue

**Ce type de composant ne sera utilisé que par le routeur** dans la grande majorité des cas.

Leur rôle est de regrouper d'autres composants de type composant et de s'afficher lorsque le routeur en aura besoin.

Chaque composant de type vue sera relié à une URL que l'utilisateur final pourra requêter.

Ils ont donc un rôle de conteneur.

Par convention, on mettra ce type de composant dans un dossier views:

**ng generate component views/maVue**

### Les composants type affichage

C'est le type de composant que l'on a utilisé jusqu'à présent.

C'est à dire qu'ils représentent un élément précis à afficher et ils seront réutilisables soit dans d'autres composants d'affichage soit directement dans des composants de type vue.

Par convention, on mettra ce type de composant dans un dossier components:

**ng generate component components/maVue**

### Le routeur

Le routeur est comme l'aiguillage sur une voie de chemin de fer:

Son rôle va être de récupérer la demande de l'utilisateur (l'URL demandée) et d'afficher correctement la bonne partie de l'application.

C'est tout ! (Et c'est déjà beaucoup)

Pour rappel, nous sommes dans une SPA ce qui signifie que la page ne se rechargera jamais !

Et c'est la qu'intervient le routeur puisque pour chaque URL renseignée et demandée par le client, notre routeur va tout simplement cacher le composant de type conteneur qui était affiché et va afficher à la place le composant de type conteneur qui correspond à l'URL.

Ce paramètrage se fait dans le fichier **app-routing.module.ts**, tout en haut du fichier, il existe une constante **routes** qui est un tableau qui va contenir des objets représentant les routes.

Les objets auront au moins 2 clés:

* path qui représente le chemin que l'on veut dans notre application pour une vue
* component qui représente le composant de type vue à afficher lorsque l'URL est requêté

Un exemple vaut 1000 mots:

**const routes: Routes = [**

**{path: '', component: HomeViewComponent},**

**{path: 'auth', component: AuthViewComponent},**

**{path: 'not-found', component: ErrorViewComponent},**

**{path: '\*\*', redirectTo: 'not-found'}**

**];**

Dans l'exemple, on a deux URLs qui pointent vers un composant et une URL avec un redirectTo.

De plus, son path est assez spécifique.

Cette route là est une route "joker", cela signifie que si aucune des routes listées ne correspond à la route demandée par l'utilisateur, alors, on rentrera dans cette route qui renverra l'utilisateur sur une route connue (dans notre cas, une route qui affichera un message de page non trouvée)

### Sécuriser une route avec un guard

On a vu dans la partie du cours précédent qu'il existait un type de service nommé le guard.

On peut comparer ce type de service à un videur en boite de nuit. Son rôle va être d'effectuer une vérification afin de savoir si l'utilisateur peut accéder à une URL ou pas.

Par exemple, est-ce que l'utilisateur est connecté ou pas ? Ou bien est-ce qu'il possède les bons droits pour accéder à cette URL.

Le guard va ainsi implémenter une méthode spécifique qui se nomme **canActivate**.

Elle porte assez bien son nom puisqu'elle va permettre de savoir (via les actions qu'on lui donnera à l'intérieur) si la route peut-être utilisée (activée) par l'utilisateur ou non.

Il faudra ensuite référencer ce guard dans notre routeur et pour chaque route que l'on souhaite sécuriser avec ce dernier.

Exemple si nous possédons un guard nommé AuthGuard:

const routes: Routes = [

{path: '', **canActivate: [AuthGuard]**, component: HomeViewComponent},

{path: 'auth', component: AuthViewComponent},

{path: 'not-found', component: ErrorViewComponent},

{path: '\*\*', redirectTo: 'not-found'}

];

## TP Bibliothèque partie 2

### Création des vues/services/components

$ ng g c views/auth

$ ng g c views/error

$ ng g c views/single-book

$ ng g s services/auth/auth

$ ng g c components/header

$ ng g g guards/auth/auth (sur git bash c’est canActivate par défaut) ou :

ng g g guards/auth/auth --implements CanActivate

### Router

App-routing.module.ts

Ajout des paths

const routes: Routes = [

  { path: '', component: AuthComponent }, //root

  { path: 'books', component: BooksListComponent },

  { path: 'books/:id', component: SingleBookComponent }, //accès à un livre en particulier ( localHost:4200/books/1) ':id' pour paramètre sinon il cherche la route 'id'

  { path: 'not-found', component: ErrorComponent },

  { path: '\*\*', redirectTo: 'not-found' }, // redirect sur route not-found en cas d'erreur

];

App.module.ts

Ajout de AuthService dans providers

@NgModule({

  declarations: [AppComponent, BooksListComponent, AuthComponent, ErrorComponent, SingleBookComponent, HeaderComponent],

  imports: [BrowserModule, AppRoutingModule],

  providers: [AuthService, BookService], //Importer le bookservice et le AuthService

  bootstrap: [AppComponent],

})

App-component html

Supprimer le app-books-list, ne laisser que le routeur-outlet

<!-- app component html -->

<div class="container mt-5">

  <app-header></app-header>

  <router-outlet></router-outlet>

</div>

Possible aussi de rajouter un footer

### Mise en place de la structure html avec le header

Books-list html

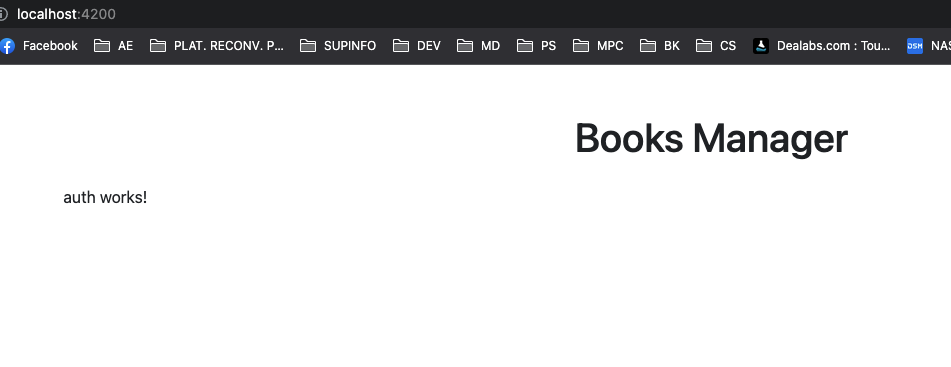
Supprimer le container et le h1

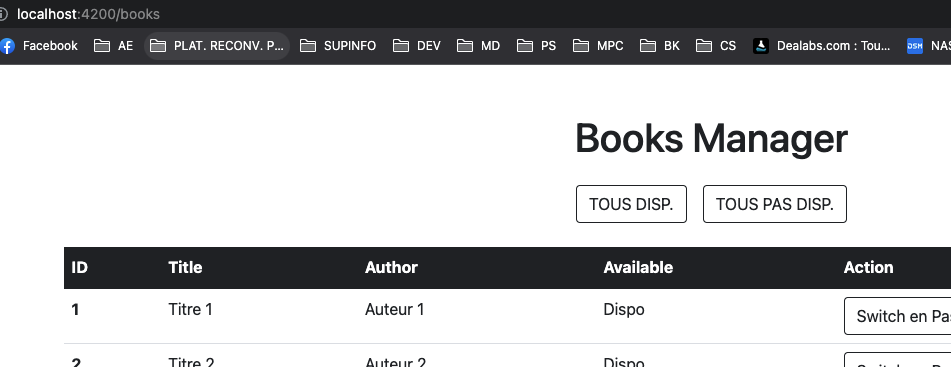
Header html

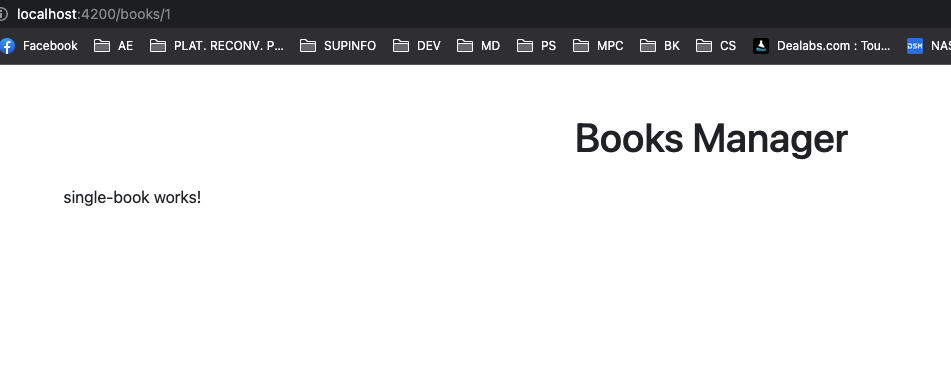
Mettre le h1

<!-- header html -->

<h1 class="text-center my-4">Books Manager</h1>







### Page d’authentification

**Auth component html**

<!-- auth component html -->

<div class="row mt-5">

  <div class="col-6 card shadow mx-auto p-3">

    <h3 class="text-center">Connexion</h3>

    <form>

      <div class="mb-3">

        <label for="email">Email</label>

        <input id="email" class="form-control" placeholder="Email" />

      </div>

      <div class="mb-3">

        <label for="password">Email</label>

        <input id="password" class="form-control" placeholder="password" />

      </div>

      <div class="row col-4 mx-auto">

        <button

          type="button"

          (click)="onSubmitAuthForm()"

          class="btn btn-dark btn-lg"

        >

          Me connecter

        </button>

      </div>

    </form>

  </div>

</div>

**Auth component ts**

Dans l’export class on Init

onSubmitAuthForm(): void {}

On injecte le Auth Service dans le constructor :

constructor(private authService: AuthService) {}

Appel du Auth service dans la fonction

onSubmitAuthForm(): void {

    this.authService

    .signIn('contact@juju.com', 'azerty')

    .then(() => {

    })

  }

**Auth service ts**

// auth service ts

import { Injectable } from '@angular/core';

@Injectable({

  providedIn: 'root',

})

export class AuthService {

  token!: string;

  constructor() {}

  signIn(email: string, password: string): Promise<void> {

    return new Promise<void>((res, rej) => {

      if (email === 'contact@juju.com' && password === 'azerty') {

        this.token = 'qsdfghj'; //donne une valeur au token pour verif + tard si le user est connecté

        res();

      }

    });

  }

}

**Auth component ts**

Ajout du lien vers router dans le constructor et le then

export class AuthComponent implements OnInit {

  constructor(private authService: AuthService, private router: Router) {}

  ngOnInit(): void {}

  onSubmitAuthForm(): void {

    this.authService.signIn('contact@juju.com', 'azerty').then(() => {

      //redirect user to books view (utilisation du router)

      this.router.navigateByUrl('/books');

    });

  }

}

Connecté au compte mais si on va sur une autre page on est plus connecté.

On va donc sécuriser les routes

### Guard

**Auth guard ts**

Ajout du constructor

constructor(private authService: AuthService, private router: Router) {}

Modif du canActivate (ajout du if et du redirect à la place du return true)

canActivate(

    route: ActivatedRouteSnapshot,

    state: RouterStateSnapshot

  ):

    | Observable<boolean | UrlTree>

    | Promise<boolean | UrlTree>

    | boolean

    | UrlTree {

    if (this.authService.token) {

      //verif si un token de connexion est présent

      return true;

    }

    //Redirect to login view

    return this.router.navigateByUrl('');

  }

**App routing module ts**

Ajout des guards

{ path: 'books', canActivate: [AuthGuard], component: BooksListComponent },

  {

    path: 'books/:id',

    canActivate: [AuthGuard],

    component: SingleBookComponent,

  }

### Page d’erreur

Href va recharger la page ce qu’il ne faut pas faire sur angular -> Utiliser router link

<!-- error html -->

<div class="row text-center">

  <div class="col-md-12">

    <h1>Oups !</h1>

    <h2>404 Page Introuvable</h2>

    <div class="my-2">

      <p style="font-size: large">

        Oups, une erreur est survenue, la page demandée n'a pas été trouvée !

      </p>

    </div>

    <div class="mx-1">

      <a routerLink="/" class="btn btn-dark btn-large">Retour a l'accueil</a>

    </div>

  </div>

</div>

### Aller au détail d’un livre

**Books list html**

Ajout d’un bouton détails

[router link] pour renvoyer vers la page, concaténation /books/ :id

<button

          [routerLink]="['/books', book.id]"

          class="btn btn-outline-dark mx-2"

        >

          Détails

        </button>

### Récupérer infos d’un livre

**Single book component ts**

Récup de l’id

Variable book, parametres constructor, const id, function vers service

export class SingleBookComponent implements OnInit {

  book!: Book;

  constructor(

    private bookService: BookService,

    private route: ActivatedRoute

  ) {}

  ngOnInit(): void {

    //Récup de l'id

    const id = this.route.snapshot.params['id']; // car :id dans app routing module ts, return a string

    this.bookService

      .getBookById(+id) //+ before a string cast it to a number (ParseInt)

      .then((book: Book) => {

        this.book = book;

      });

  }

}

**book service ts**

Récup du livre

 getBookById(bookId: number): Promise<Book> {

    return new Promise<Book>((res, rej) => {

      //for (let book of this.books)

      for (let book of this.books) {

        if (book.id === bookId) {

          res(book);

          break;

        }

      }

    });

  }

**Single book html**

<!-- single book html -->

<div class="row col-6 card shadow mx-auto">

  <div class="card-body text-center">

    <h5 class="card-title">{{ book.title }} - {{ book.author }}</h5>

    <img src="https://picsum.photos/200/200" class="card-img-top" />

    <button routerLink="/books" class="btn btn-dark mt-2">RETOUR</button>

  </div>

</div>