

Fms Academy 2022

Formation Java Spring Angular

M09 : Angular



Sommaire

Angular les bases

- Rappel M08 Web
- C'est quoi un framework (front/back) ?
- Historique Angular
- Les Atouts d'Angular
- Architecture Globale & Protocole HTTP
- Introduction à TypeScript
- Install NodeJs & Angular
- 1^{ère} appli Angular
- Structure d'un projet Angular
- Chargement d'une appli Angular
- App.Module
- App.Component
- Lancement de notre 1^{ère} Appli
- Ajout d'un nouveau composant
- Structurer nos données, les exploiter et utiliser des directives
- Ajout Framework Css
- MVVM & Data Binding
- Service & Injection de dépendance
- Mise en œuvre d'un formulaire
- Les outils de débogages
- Sites développé avec Angular
- Ressources

Angular (suite)

- Synthèse et révision
- Protocole http et Api Rest
- Mise en œuvre de Json Server
- Api Rest & Bdd
- Utilisation d'un client Rest
- Programmation réactive
- Cycle de vie d'un composant & hooks
- Pipes
- Différence entre Ngcontainer & ngtemplate
- FormsModule, ReactiveFoms & FormBuilder
- RxJs

Rappel M08 Web

Html/Css/Js/Dom/Event/Frameworks Css Maquette/interface web statique & dynamique

Ma Boutique en Ligne Home

All

SmartPhone

Tablet

Pc

Liste des articles en stock

ID	NAME	BRAND	PRICE	QUANTITE
1	S8	Samung	200	<input type="text" value="2"/>
2	S9	Samung	250	<input type="text" value="3"/>
3	Iphone 10	Apple	500	<input type="text" value="1"/>

Contenu de mon Panier

id:1 S8 Samung 200€ Quantité : 2
id:2 S9 Samung 250€ Quantité : 3
id:3 Iphone 10 Apple 500€ Quantité : 1
prix total de la commande : 1650

Suivant

Nom :

el babili

Adresse :

14 rue du pecheur

Telephone :

0607080910

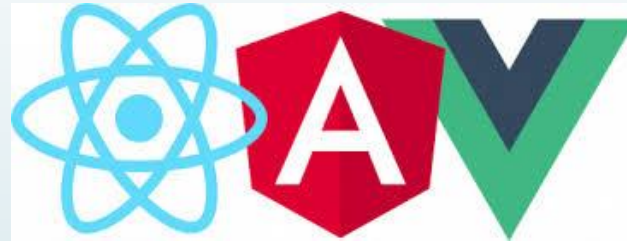
Email :

mohamed.el-babili@fms-ea.com

Commander

C'est quoi un framework (front/back) ?

- Les Frameworks répondent à une problématique connue de tous les développeurs : « *Il est difficile de réaliser une application qui respecte toutes les exigences fonctionnelles et techniques avec un coût faible sans utiliser l'expérience des autres* »



Historique d'Angular(JS)

- [Angular JS](#)(2009 à ce jour) **VS** [Angular](#)(2016 à ce jour)
- **AngularJS** est développé et maintenu par Google afin de simplifier le dev et les tests d'applis web utilisant Javascript...
- **Angular** est la réécriture d'AngularJS utilisant Typescript, il est maintenu par Google et une communauté(particulier et prof), la dernière version est la 13, il a été pensé pour le mobile avec notamment les SPA, sa syntaxe diffère par ex sur les directives ou la gestion du binding, il utilise directement le DOM pour les évènements... Pour plus de détails voir le comparatif [ici](#)

Features	Angular 9	Angular 7	AngularJS
 DOM Content loaded	561 ms	907 ms	1035 ms
 Visual complete	802 ms	1037 ms	1187 ms
 First visual change	802 ms	1037 ms	1187 ms
 Speed index	903 ms	1161 ms	1291 ms
 Last visual change	2239 ms	2519 ms	2281 ms

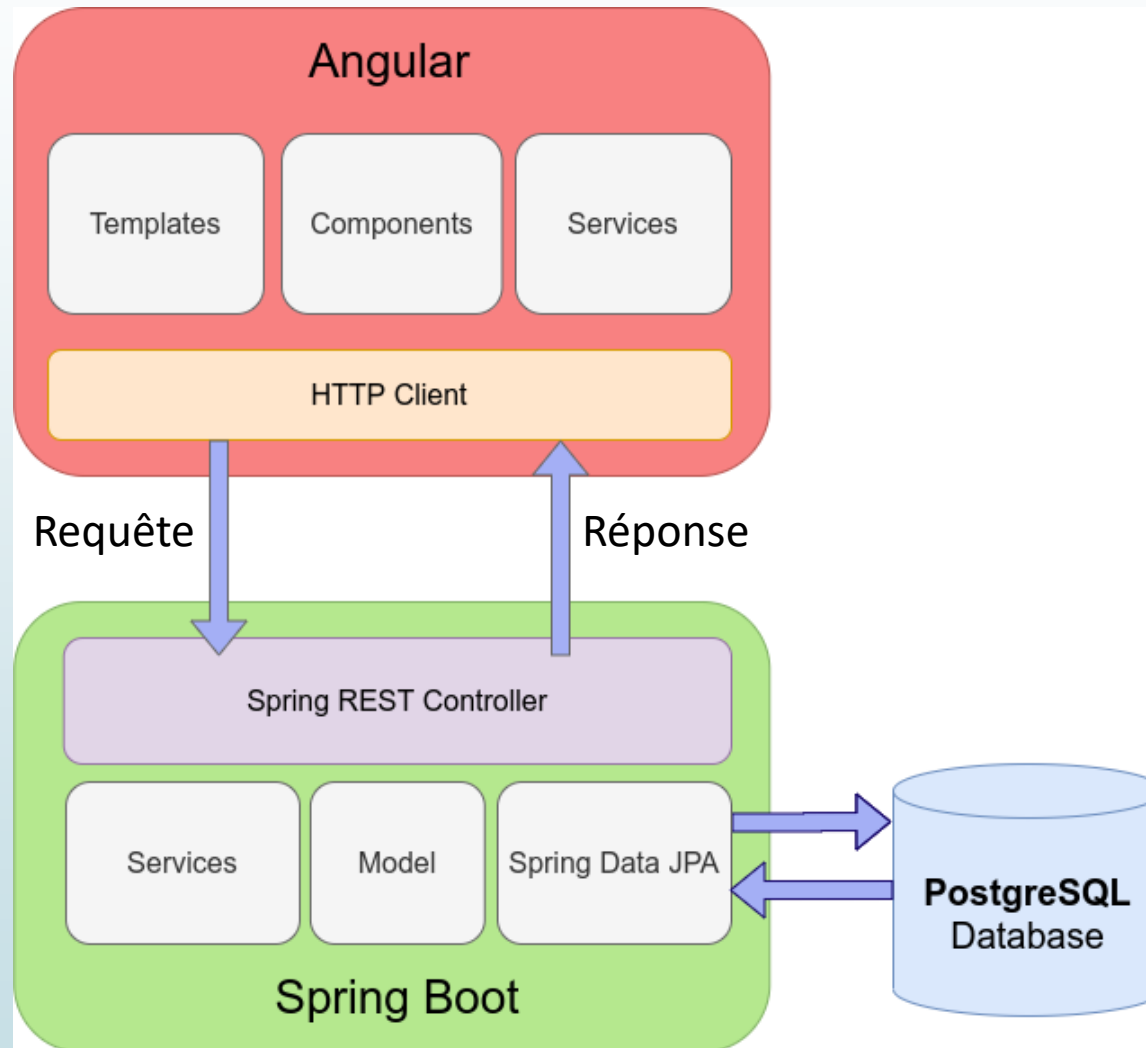
Les Atouts d'Angular

- **Single Page Application** : une seule page est chargée, toute l'appli tourne dans index.html
- Avec Angular, on ne parle plus de modèle MVC mais de **programmation orientée Web Component** (concept de composant web introduit par html 5 et repris ici)
- L'idée ici est donc de créer une page sous forme d'ensemble de composants
- Angular se base sur la **programmation réactive**, l'appli n'est pas bloqué en attente d'une réponse à une action
- **L'appli est autonome** et seules des interactions existent avec des API Rest par ex (compatible avec tous les éco systèmes back)
- L'utilisation de **Typescript** (typage + poo)
- L'utilisation de **l'inversion de contrôle** et **l'injection des dépendances**
- S'agissant enfin des différentes versions d'Angular (en dehors de la différence majeure avec AngularJS), il s'agit essentiellement d'amélioration continue.

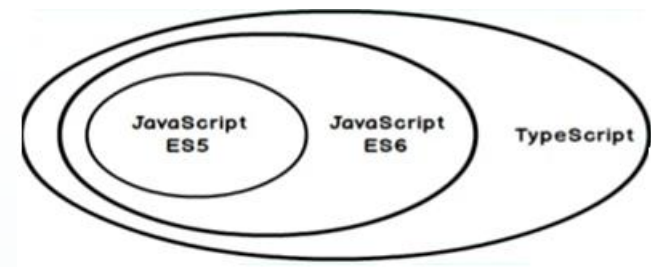
Architecture globale & protocole HTTP

Client/User

Serveur/Api



Introduction à TypeScript



- **JavaScript** est un langage de script côté client permettant de gérer l'interaction avec un utilisateur à l'aide de formulaire et autre évènements... tous les navigateurs contiennent un moteur JS qui interprète le code. **Mais en réalité Javascript n'a pas été prévu initialement pour gérer des applications front volumineuses.**
- **Typescript** est donc né de ce besoin, développé par des devs de Microsoft en 2012 dont Anders Hejlsberg (principal inventeur de C#), il s'agit d'un sur-ensemble de Javascript. Le code TS est transcompilé en JS et donc interprétable par n'importe quel navigateur. Typescript permet un typage statique optionnel des variables et fonctions et offre la possibilité de mettre en œuvre la POO. Il supporte la spécification EcmaScript 6.

Install NodeJs & Angular

- Install [Node](#) :

- Éviter d'installer les outils complémentaires
- Pour vérifier que c'est fait
- Les chemins d'exécution sont ajoutés pendant l'install aussi vous n'avez pas à le faire, si ce n'est pas le cas, il faut vérifier le path.

```
C:\Users\EI-BabiliM>node --version
v16.13.1
```

```
C:\Users\EI-BabiliM>npm --version
8.1.2
```

- Install [Angular](#) :

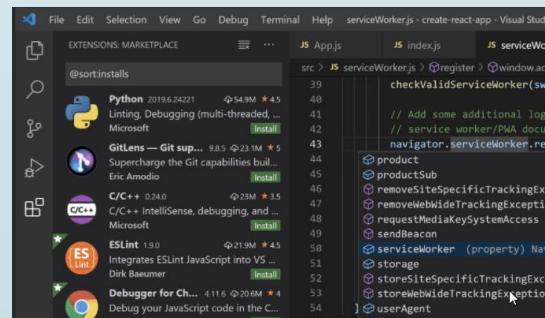
- npm install -g @angular/cli

```
Angular CLI
Angular CLI: 13.0.4
Node: 16.13.1
Package Manager: npm 8.2.0
OS: win32 x64

Angular: 13.0.3
... animations, common, compiler, compiler-cli, core, forms
... platform-browser, platform-browser-dynamic, router

Package                                Version
-----
@angular-devkit/architect              0.1300.4
@angular-devkit/build-angular         13.0.4
@angular-devkit/core                  13.0.4
@angular-devkit/schematics            13.0.4
@angular/cli                          13.0.4
@schematics/angular                  13.0.4
rxjs                                  7.4.0
typescript                           4.4.4
```

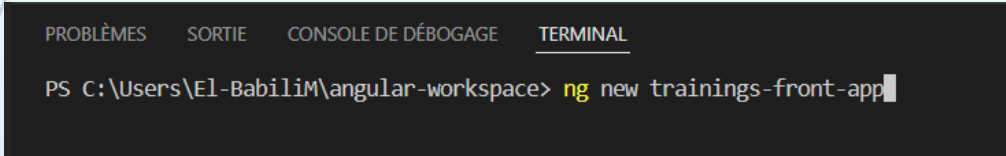
- Choisir un [Ide](#) : VSC



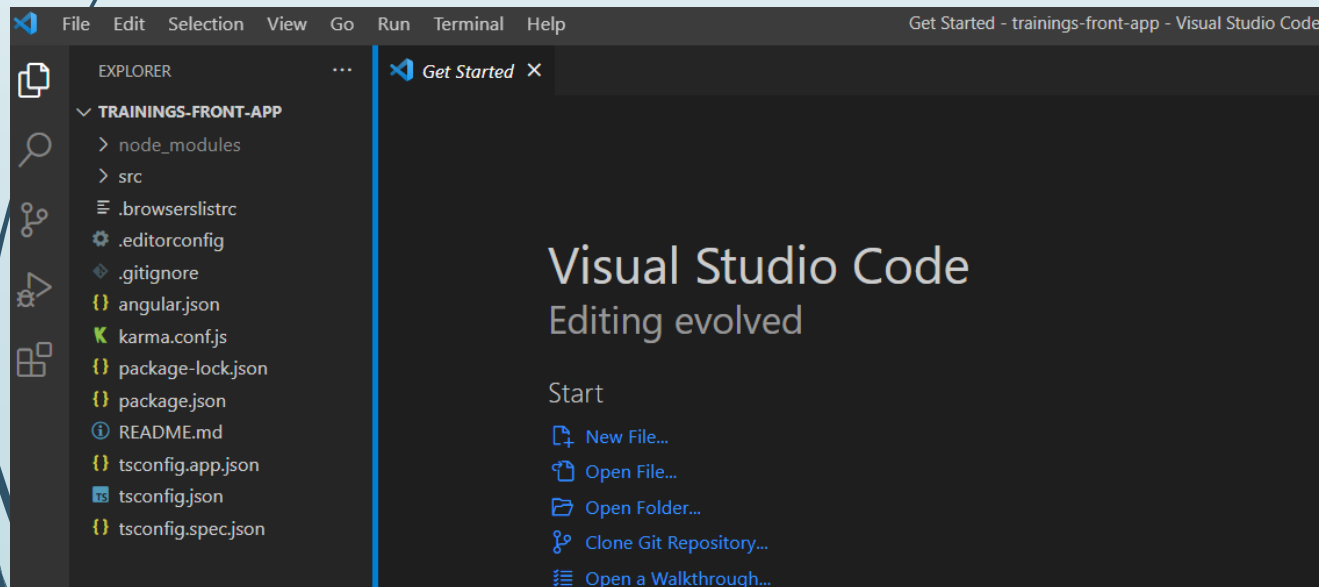
1^{ère} appli Angular

10

Expressions des besoins : « Soit une application de vente de formation en ligne »

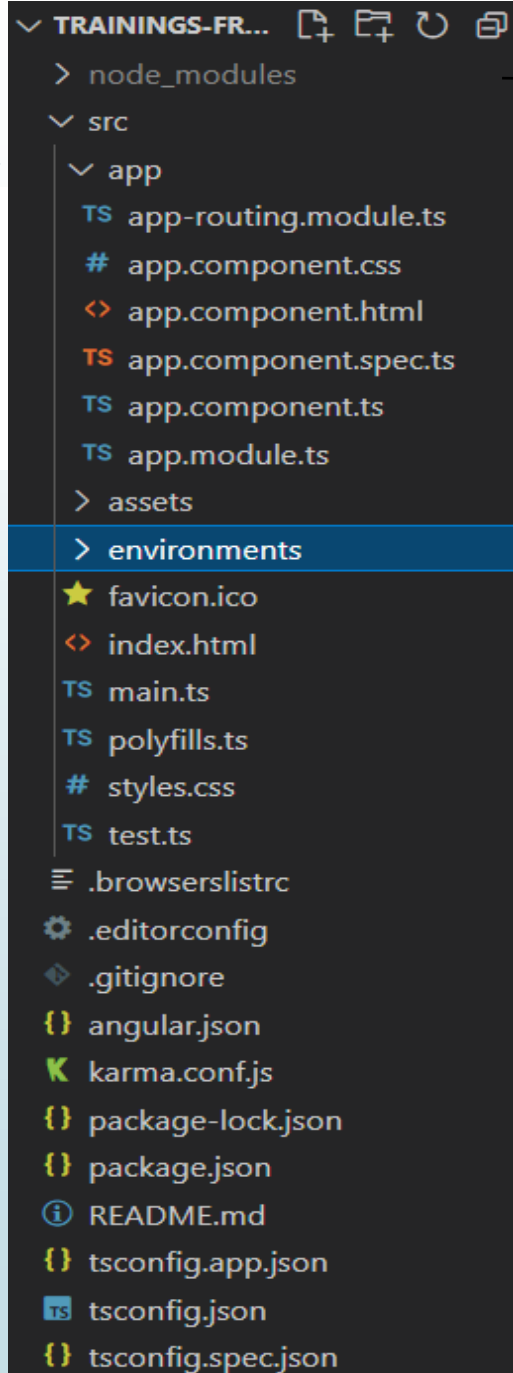
- 1.1 Création d'un espace de travail (s'il n'existe pas) : « Angular-workspace »
- 1.2 Dans ce workspace, création d'un projet et ouverture dans un IDE
 - 

```
PS C:\Users\EI-BabiliM\angular-workspace> ng new trainings-front-app
```
- 1.3 Ouvrir ce projet/dossier dans votre ide



Structure d'un projet Angular

Composant
et module
principal de
l'appli

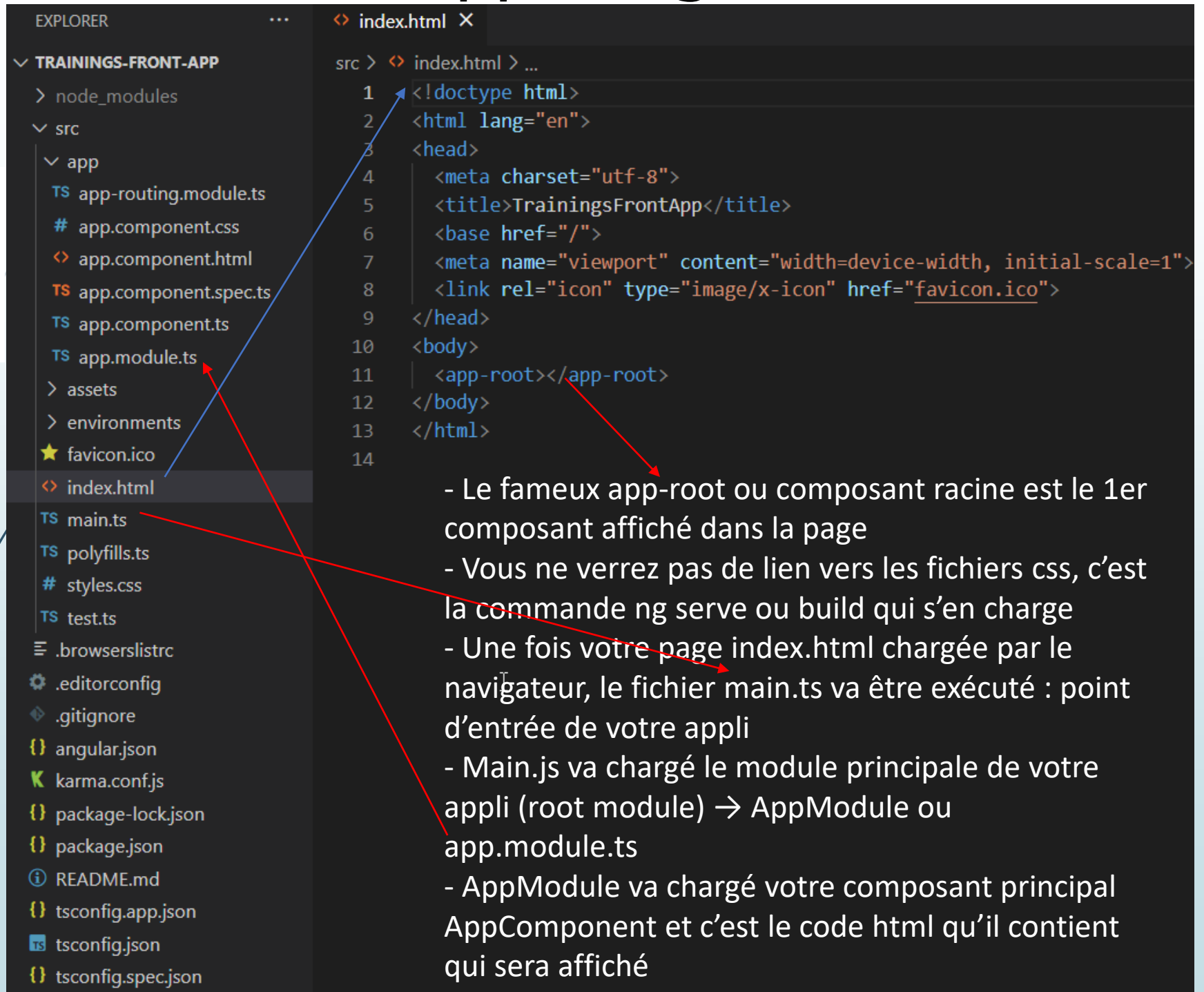


dépendances

Fichiers
sources

Fichiers
de config

Chargement d'une appli Angular



EXPLORER

TRAININGS-FRONT-APP

- node_modules
- src
 - app
 - app-routing.module.ts
 - app.component.css
 - app.component.html
 - app.component.spec.ts
 - app.component.ts
 - app.module.ts
 - assets
 - environments
 - favicon.ico
 - index.html
 - main.ts
 - polyfills.ts
 - styles.css
 - test.ts
- .browserslistrc
- .editorconfig
- .gitignore
- angular.json
- karma.conf.js
- package-lock.json
- package.json
- README.md
- tsconfig.app.json
- tsconfig.json
- tsconfig.spec.json

index.html

```
1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8">
5   <title>TrainingsFrontApp</title>
6   <base href="/">
7   <meta name="viewport" content="width=device-width, initial-scale=1">
8   <link rel="icon" type="image/x-icon" href="favicon.ico">
9 </head>
10 <body>
11   <app-root></app-root>
12 </body>
13 </html>
14
```

- Le fameux app-root ou composant racine est le 1er composant affiché dans la page
- Vous ne verrez pas de lien vers les fichiers css, c'est la commande ng serve ou build qui s'en charge
- Une fois votre page index.html chargée par le navigateur, le fichier main.ts va être exécuté : point d'entrée de votre appli
- Main.js va chargé le module principale de votre appli (root module) → AppModule ou app.module.ts
- AppModule va chargé votre composant principal AppComponent et c'est le code html qu'il contient qui sera affiché

App.module.ts

```
TS app-routing.module.ts
# app.component.css
<> app.component.html
TS app.component.spec.ts
TS app.component.ts
TS app.module.ts
```

```
app > TS 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';

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

Les
décorateur
s
permettent
de décrire
au
framework
comment
interpréter
un élément

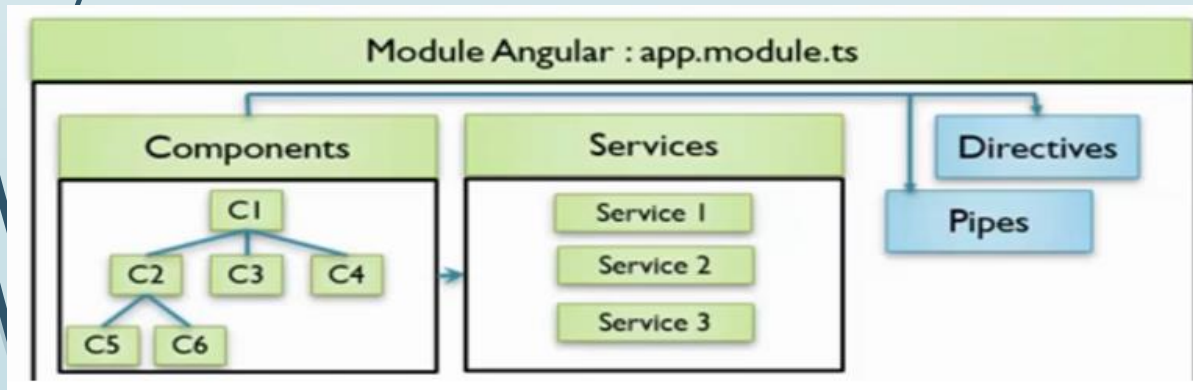
Mécanisme d'annotation ou décorateur
Permettant ici de spécifier un module

Tous les composants créés sont ajoutés ici

Tous les modules dont nous aurons besoin
Doivent être déclarés ici

Idem pour les services

On peut importer ce module ailleurs



Une application angular est
constitué d'un ou plusieurs
modules, chacun peut
contenir un ou plusieurs
composant/services +
directives + pipes...

App.component

Logique

Vue

```
TS app.component.ts X
src > app > TS app.component.ts > AppComponent > title
1  import { Component } from '@angular/core';
2
3  @Component({
4    selector: 'app-root',
5    templateUrl: './app.component.html',
6    styleUrls: ['./app.component.css']
7  })
8  export class AppComponent {
9    title = 'trainings-front-app';
10 }
11

<> app.component.html M X
src > app > <> app.component.html > p
Go to component
1  <p>
2    J'ai enlevé tout le bazar !!!
3  </p>
4
5  <router-outlet></router-outlet>
```

```
TS app-routing.module.ts
# app.component.css
<> app.component.html
TS app.component.spec.ts
TS app.component.ts
TS app.module.ts
```

App.component.spec.ts servira
aux tests unitaires et
App.component.css ...

Lancement de notre 1^{ère} Appli

```
PS C:\Users\EI-BabiliM\angular-workspace\trainings-front-app> ng serve
✓ Browser application bundle generation complete.
```

Initial Chunk Files	Names	Size
vendor.js	vendor	1.94 MB
polyfills.js	polyfills	334.03 kB
styles.css, styles.js	styles	212.41 kB
main.js	main	53.61 kB
runtime.js	runtime	6.87 kB
	Initial Total	2.53 MB

```
Build at: 2022-05-12T06:59:06.640Z - Hash: 6efefe401cec62b9 - Time: 7919ms
```

```
** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **
```

✓ Compiled successfully.



trainings-front-app app is running!

Resources

Here are some links to help you get started:

[Learn Angular >](#)[CLI Documentation >](#)

Angular Material >

[Angular Blog >](#)

Angular DevTools >

Next Steps

What do you want to do next with your app?



New Component



Angular Material



Add PWA Support



Add Dependency



Run and Watch Tests



Build for Production

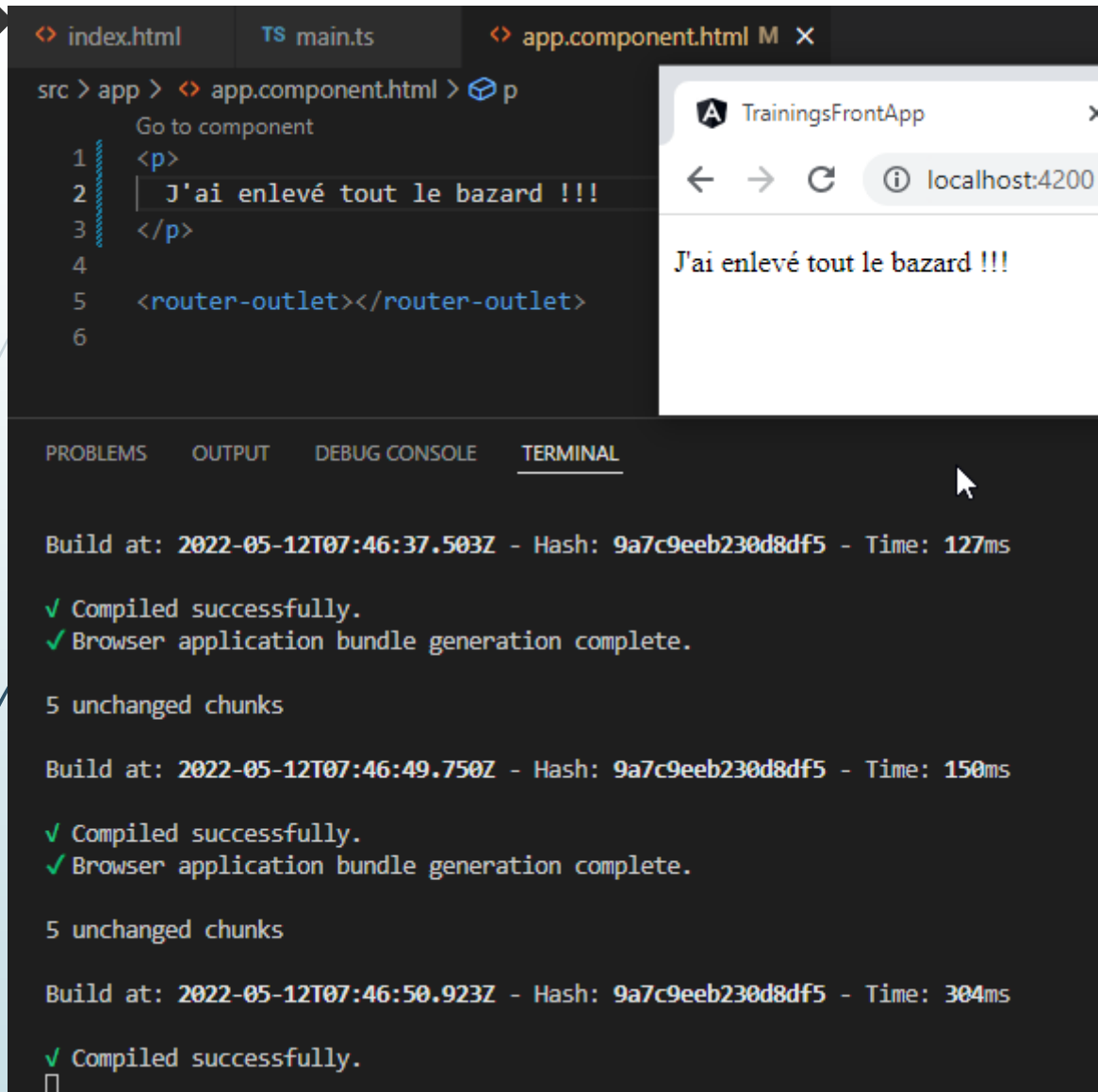
ng generate component xyz

Code Html par défaut du composant principal

```
> index.html TS mains.ts app.component.html X TS app.module.ts
```

```
src > app > > app.component.html > style > host  
297     @media screen and (max-width: 600px) {  
298         svg#rocket-smoke {  
299             display: none;  
300             visibility: hidden;  
301         }  
302     }  
303 </style>  
304  
305 <!-- Toolbar -->  
306 <div class="toolbar" role="banner">  
307       
312     <span>Welcome</span>  
313     <div class="spacer"></div>  
314     <a aria-label="Angular on twitter" target="_blank" rel="noopener" href="https://twitter.com/angular">  
315         <svg id="twitter-logo" height="24" data-name="Logo" xmlns="http://www.w3.org/2000/svg" viewBox="0  
316             <rect width="400" height="400" fill="none"/>  
317             <path d="M153.62,301.59c94.34,0,145.94-78.16,145.94-145.94,0-2.22,0-4.43-.15-6.63A104.36,104.36,  
318                 </svg> I  
319     </a>  
320     <a aria-label="Angular on YouTube" target="_blank" rel="noopener" href="https://youtube.com/angular">
```

Modif du code Html de AppComponent



The screenshot shows an IDE with three tabs: `index.html`, `TS main.ts`, and `app.component.html M`. The `app.component.html` file is open, showing the following code:

```
src > app > <> app.component.html > p
Go to component
1 <p>
2   J'ai enlevé tout le bazar !!!
3 </p>
4
5 <router-outlet></router-outlet>
6
```

Below the code editor, the `TERMINAL` tab is active, displaying the output of the build process:

```
Build at: 2022-05-12T07:46:37.503Z - Hash: 9a7c9eeb230d8df5 - Time: 127ms
✓ Compiled successfully.
✓ Browser application bundle generation complete.
5 unchanged chunks
Build at: 2022-05-12T07:46:49.750Z - Hash: 9a7c9eeb230d8df5 - Time: 150ms
✓ Compiled successfully.
✓ Browser application bundle generation complete.
5 unchanged chunks
Build at: 2022-05-12T07:46:50.923Z - Hash: 9a7c9eeb230d8df5 - Time: 304ms
✓ Compiled successfully.
```

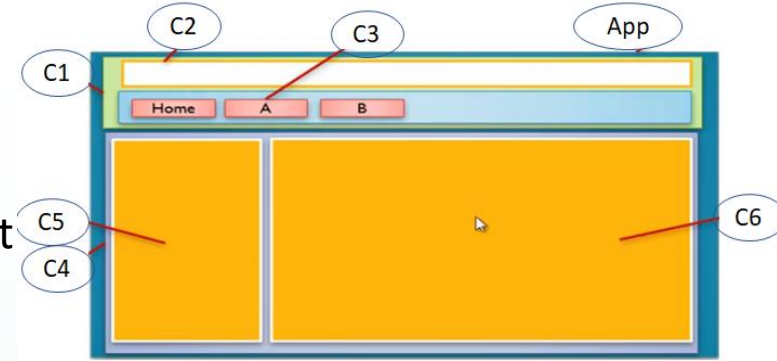
On the right, a browser window titled `TrainingsFrontApp` is open at `localhost:4200`, displaying the text: `J'ai enlevé tout le bazar !!!`.

NB : Au moindre changement sur un fichier source, seul le fichier est recompilé et le résultat instantané !

NB : Conformément au fichier `package.json/scripts`
On peut exécuter notre appli avec les commandes :
`Ng serve`
`Ng serve -o`
`Npm run start`
`Npm start`

Ajout d'un composant à notre appli

Une application Angular est constituée de nombreux composants dans le composant App-root



« Nous souhaitons maintenant afficher une liste de formation à l'aide d'un composant »

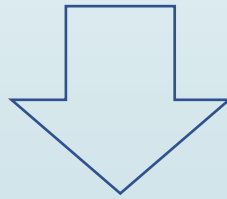
1/ Créer le composant (et le répertoire composants au passage)

```
PS C:\Users\EI-BabiliM\angular-workspace\trainings-front-app> ng g c components/trainings
CREATE src/app/components/trainings/trainings.component.html (24 bytes)
CREATE src/app/components/trainings/trainings.component.spec.ts (647 bytes)
CREATE src/app/components/trainings/trainings.component.ts (287 bytes)
CREATE src/app/components/trainings/trainings.component.css (0 bytes)
UPDATE src/app/app.module.ts (498 bytes)
```

2/ ajouter une route vers ce composant dans **app-routing.module.ts**

```
1 import { NgModule } from '@angular/core';
2 import { RouterModule, Routes } from '@angular/router';
3 import { TrainingsComponent } from '../components/trainings/trainings.component';
4
5 const routes: Routes = [
6   {
7     path: 'trainings',
8     component: TrainingsComponent
9   }
10 ];
11
12 @NgModule({
13   imports: [RouterModule.forRoot(routes)],
14   exports: [RouterModule]
15 })
16 export class AppRoutingModule { }
```

3/ modifier les fichiers app.component Et trainings.component



```
app.component.html M X
src > app > app.component.html > router-outlet
Go to component
1 <p>
2   J'ai enlevé tout le bazar !!!
3 </p>
4
5 <li>
6   <a routerLink="/trainings">liste des formations</a>
7 </li>
8
9 <router-outlet> </router-outlet>

trainings.component.html U X
src > app > components > trainings > trainings.component.html > p
Go to component
1 <p>C'est ici qu'on affiche la liste des formations</p>
2
```

Le composant 'trainings' est sollicité via le système de routage afin d'être inséré entre les balises <router-outlet>

J'ai enlevé tout le bazar !!!

- liste des formations

C'est ici qu'on affiche la liste des formations

Structurer nos données, les exploiter et utiliser des directives

2

```
import { Component, OnInit } from '@angular/core';
import { Training } from 'src/app/model/Training.model';

@Component({
  selector: 'app-trainings',
  templateUrl: './trainings.component.html',
  styleUrls: ['./trainings.component.css']
})
export class TrainingsComponent implements OnInit {
  listTrainings: Training[] | undefined;
  constructor() {}

  ngOnInit(): void {}

  displayTrainings() {
    this.listTrainings = [
      {id:1,name:'Java',description:'Formation Java SE 8 sur 5 jours',price:1500 },
      {id:2,name:'DotNet',description:'Formation DotNet 3 jours',price:1000 },
      {id:3,name:'Python',description:'Formation Python/Django 5 jours',price:1500 }
    ];
  }
}
```

1

```
app > model > TS training.model.ts > Training > constructor
export class Training {
  id : number;
  name : string;
  description : string;
  price : number;

  constructor(id:number,name:string,description:string,price:number) {
    this.id = id;
    this.name = name;
    this.description = description;
    this.price = price;
  }
};
```

4

5

```
app > components > trainings > trainings.component.html > div > ng-container > table > tr
Go to component
<div>
  <button (click)="displayTrainings()"> Formations</button>
  <ng-container *ngIf="listTrainings">
    <table>
      <tr>
        <th>ID</th> <th>NOM</th> <th>DESCRIPTION</th> <th>PRIX</th>
      </tr>
      <tr *ngFor="let training of listTrainings">
        <td>{{training.id}}</td>
        <td>{{training.name}}</td>
        <td>{{training.description}}</td>
        <td>{{training.price}}</td>
      </tr>
    </table>
  </ng-container>
</div>
```

localhost:4200/trainings

J'ai enlevé tout le bazar !!!

- [liste des formations](#)

Formations

ID	NOM	DESCRIPTION	PRIX
1	Java	Formation Java SE 8 sur 5 jours	1500
2	DotNet	Formation DotNet 3 jours	1000
3	Python	Formation Python/Django 5 jours	1500

Quel résultat ici ?

```
app > <> app.component.html > app-trainings
Go to component
<p>
  J'ai enlevé tout le bazar !!!
</p>

<li>
  <a routerLink="/trainings">liste des formations</a>
</li>

<router-outlet>    </router-outlet>

<app-trainings></app-trainings>
```

← → ↻ ⓘ localhost:4200/trainings

J'ai enlevé tout le bazar !!!

- [liste des formations](#)

Formations

ID	NOM	DESCRIPTION	PRIX
1	Java	Formation Java SE 8 sur 5 jours	1500
2	DotNet	Formation DotNet 3 jours	1000
3	Python	Formation Python/Django 5 jours	1500

Formations

ID	NOM	DESCRIPTION	PRIX
1	Java	Formation Java SE 8 sur 5 jours	1500
2	DotNet	Formation DotNet 3 jours	1000
3	Python	Formation Python/Django 5 jours	1500

Ajout d'un framework css

1/ Installer le framework de votre choix : `npm install bootstrap --save`

2/ ajouter dans le fichier `angular.json` :

```
"styles": [  
  "src/styles.css",  
  "node_modules/bootstrap/dist/css/bootstrap.min.css"  
],
```

3/ Ajouter une barre de navigation dans le composant principal avec un lien qui renvoi vers le composant 'trainings'

4/ Le composant doit chargé une liste de formation dès qu'il est sollicité

```
export class TrainingsComponent implements OnInit {  
  listTrainings : Training[] | undefined;  
  constructor() { }  
  
  ngOnInit(): void {  
    this.listTrainings = [  
      {id:1,name:'Java',description:'Formation Java SE 8 sur 5 jours',price:1500 },  
      {id:2,name:'DotNet',description:'Formation DotNet 3 jours',price:1000 },  
      {id:3,name:'Python',description:'Formation Python/Django 5 jours',price:1500 }  
    ];  
  }  
}
```

5/ Modifier la vue

```
<div class="container">  
  <ng-container *ngIf="listTrainings">  
    <table class="table table-striped table-hover">  
      <thead>  
        <tr>  
          <th>ID</th> <th>NOM</th> <th>DESCRIPTION</th> <th>PRIX</th>  
        </tr>  
      </thead>  
      <tbody>  
        <tr *ngFor="let training of listTrainings">  
          <td>{{training.id}}</td>  
          <td>{{training.name}}</td>  
          <td>{{training.description}}</td>  
          <td>{{training.price}}</td>  
        </tr>  
      </tbody>  
    </table>  
  </ng-container>  
</div>
```

```
<nav class="navbar navbar-expand-sm bg-light navbar-light">  
  <a class="navbar-brand" href="#">TRAININGS</a>  
  
  <ul class="navbar-nav">  
    <li class="nav-item">  
      <a class="nav-link" routerLink="/trainings">Formations</a>  
    </li>  
  </ul>  
</nav>  
<router-outlet> </router-outlet>
```

Résultat



ID	NOM	DESCRIPTION	PRIX
1	Java	Formation Java SE 8 sur 5 jours	1500
2	DotNet	Formation DotNet 3 jours	1000
3	Python	Formation Python/Django 5 jours	1500

MVVM & Data Binding

Model ou partie Typescript du composant

```
ngOnInit(): void {  
  this.listTrainings = [  
    {id:1,name:'Java',description:'Formation Java SE 8 sur 5 jours',price:1500,quantity:1 },  
    {id:2,name:'DotNet',description:'Formation DotNet 3 jours',price:1000,quantity:1 },  
    {id:3,name:'Python',description:'Formation Python/Django 5 jours',price:1500,quantity:1 }  
  ];  
}  
  
onAddToCart(training:Training){  
  console.log(training.name);  
  console.log(training.quantity);  
}
```

String interpolation
Property Binding
Event Binding
Two Way Binding

View ou partie Html du composant

```
<tr *ngFor="let training of listTrainings">  
  <td>{{training.id}}</td>  
  <td>{{training.name}}</td>  
  <td>{{training.description}}</td>  
  <td>{{training.price}}</td>  
  <td><input type="number" name="quantity" style="width: 60px" [(ngModel)]="training.quantity" >  
  <td><a class="btn btn-primary" (click)="onAddToCart(training)">Ajouter </a></td>
```

ID	NOM	DESCRIPTION	PRIX	QUANTITE	PANIER
1	Java	Formation Java SE 8 sur 5 jours	1500	<input type="text" value="6"/>	<button>Ajouter</button>
2	DotNet	Formation DotNet 3 jours	1000	<input type="text" value="1"/>	<button>Ajouter</button>
3	Python	Formation Python/Django 5 jours	1500	<input type="text" value="1"/>	<button>Ajouter</button>

Ajouter **FormsModule**
Dans les imports de
app.module

21

Service & Injection de dépendance

Une **dépendance** est généralement l'instance d'une classe permettant de **factoriser certaines fonctionnalités** ou d'**accéder à un état** permettant ainsi aux composants de communiquer entre eux. Dans le vocabulaire Angular, ces classes sont appelées "**services**". Les services sont le plus souvent **des singletons**.

1/ Ajouter un service qui va permettre de gérer notre Panier avec les particularités métier qui vont bien...

```
PS C:\Users\EI-BabiliM\angular-workspace\trainings-front-app> ng g s services/cart
CREATE src/app/services/cart.service.spec.ts (347 bytes)
CREATE src/app/services/cart.service.ts (133 bytes)
```

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

@Injectable({
  providedIn: 'root'
})
export class CartServiceService {
  constructor() { }
}
```

2/ Injection du service(dépendance) dans le composant puis utilisation de celui-ci pour ajouter des formations au panier

```
import { CartService } from 'src/app/services/cart.service';

@Component({
  selector: 'app-trainings',
  templateUrl: './trainings.component.html',
  styleUrls: ['./trainings.component.css']
})
export class TrainingsComponent implements OnInit {
  listTrainings : Training[] | undefined;
  constructor(private cartService : CartService) { }

  ngOnInit(): void {
    this.listTrainings = [
      {id:1,name:'Java',description:'Formation Java SE 8 sur 5 jours',price:1500,quantity:1 },
      {id:2,name:'DotNet',description:'Formation DotNet 3 jours',price:1000,quantity:1 },
      {id:3,name:'Python',description:'Formation Python/Django 5 jours',price:1500,quantity:1 }
    ];
  }

  onAddToCart(training:Training){
    this.cartService.addTraining(training);
  }
}
```

Routage et Navigation

Compte tenu du fait que nous avons une seule page index.html (Single Page Application) Il s'agit du moyen pour passer de l'affichage d'un composant à un autre.

Reprenons notre exemple et imaginons vouloir afficher un composant contenant tous les éléments du panier à chaque fois qu'on insère de nouvelles données.

```
import { Router } from '@angular/router';

@Component({
  selector: 'app-trainings',
  templateUrl: './trainings.component.html',
  styleUrls: ['./trainings.component.css']
})
export class TrainingsComponent implements OnInit {
  listTrainings : Training[] | undefined;
  constructor(private cartService : CartService, private router : Router) { }

  onAddToCart(training: Training){
    this.cartService.addTraining(training);
    this.router.navigateByUrl('cart');
```

TRAININGS Liste des Formations Contenu du Panier

ID	Name	Quantity	Price	
1	Java	3.00	1,500.00	<button>Supprimer</button>
3	Python	2.00	1,500.00	<button>Supprimer</button>
Total			7,500.00	

Commander

```
const routes: Routes = [
  { path : 'trainings', component : TrainingsComponent },
  { path : 'cart' , component : CartComponent },
  { path : '' , redirectTo : 'trainings', pathMatch : 'full' },
  { path: '404', component: NotFoundComponent},
  { path: '**', redirectTo: '/404' }
];
```


Mise en œuvre d'un formulaire

NB : si c'est pas déjà fait,
ajouter **FormsModule**
Dans les imports de
app.module

```
<div class="container">
  <div class="col-md-3">
    <form #f="ngForm" (ngSubmit)="onSaveCustomer(f.value)">
      <div class="form-group">
        <label class="control-label">Nom : </label>
        <input type="text" name="name" required class="form-control" [(ngModel)]="cartService.getCustomer().name">
      </div>
      <div class="form-group">
        <label class="control-label">Prénom : </label>
        <input type="text" name="firstName" class="form-control" [(ngModel)]="cartService.getCustomer().firstName">
      </div>
      <div class="form-group">
        <label class="control-label">Adresse : </label>
        <input type="text" name="address" class="form-control" [(ngModel)]="cartService.getCustomer().address">
      </div>
      <div class="form-group">
        <label class="control-label">Tel : </label>
        <input type="text" name="phone" class="form-control" [(ngModel)]="cartService.getCustomer().phone">
      </div>
      <div class="form-group">
        <label class="control-label">Email : </label>
        <input type="text" name="email" required class="form-control" [(ngModel)]="cartService.getCustomer().email">
      </div>
      <button type="submit" [disabled]="!f.valid" class="btn btn-primary">Next</button>
    </form>
  </div>
</div>
```

```
export class CustomerComponent implements OnInit {
  constructor(public cartService : CartService) {
  }

  ngOnInit(): void {
  }

  onSaveCustomer(customer : Customer){
    console.log(customer);
  }
}
```

Nom : el babili

Prénom : mohamed

Adresse : 14 rue du pecheur

Tel : 0607080910

Email : mohamed.el-babili@fms-ea.com

Next

Console: {name: 'el babili', firstName: 'mohamed', address: '14 rue du pecheur', phoneNumber: '0607080910', email: 'mohamed.el-babili@fms-ea.com'}

Une autre manière de faire la même
chose sans binding bidirectionnel

```
<form #f="ngForm" (ngSubmit)="onSaveCustomer(f.value)">
  <div class="form-group">
    <label class="control-label">Nom : </label>
    <input type="text" ngModel name="name" required class="form-control" >
  </div>
  <div class="form-group">
    <label class="control-label">Prénom : </label>
    <input type="text" ngModel name="firstName" class="form-control" >
  </div>
```

[disabled]="!f.valid"

Exemple d'utilisation d'une property binding ici pour désactiver le bouton next tant que le formulaire est incomplet

Les outils de débogages



Solution 1

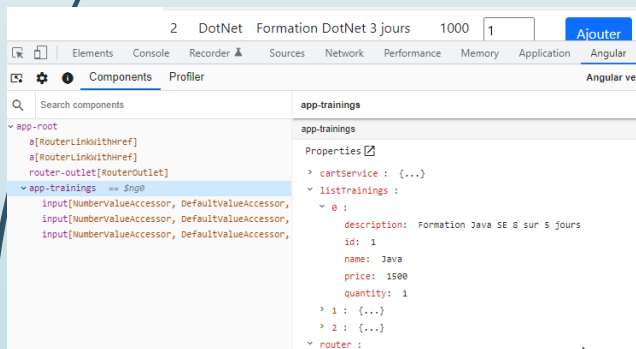
Utiliser des messages sur la console

```
console.log('total : ' + this.amount);
```

Puis accéder aux outils de dev via votre navigateur : F12

Solution 3

Angular Dev Tool
(extension Chrome)




Solution 2

Utiliser le débugueur de Vsc



Configurer un fichier launch.json
("url": http://localhost:4200)

puis mettre le point d'arrêt où vous souhaitez
Ng serve puis

EXÉCUTER ET DÉBOGUER  Launch Chrome against local



Solution 4

Logger ou Journalisation

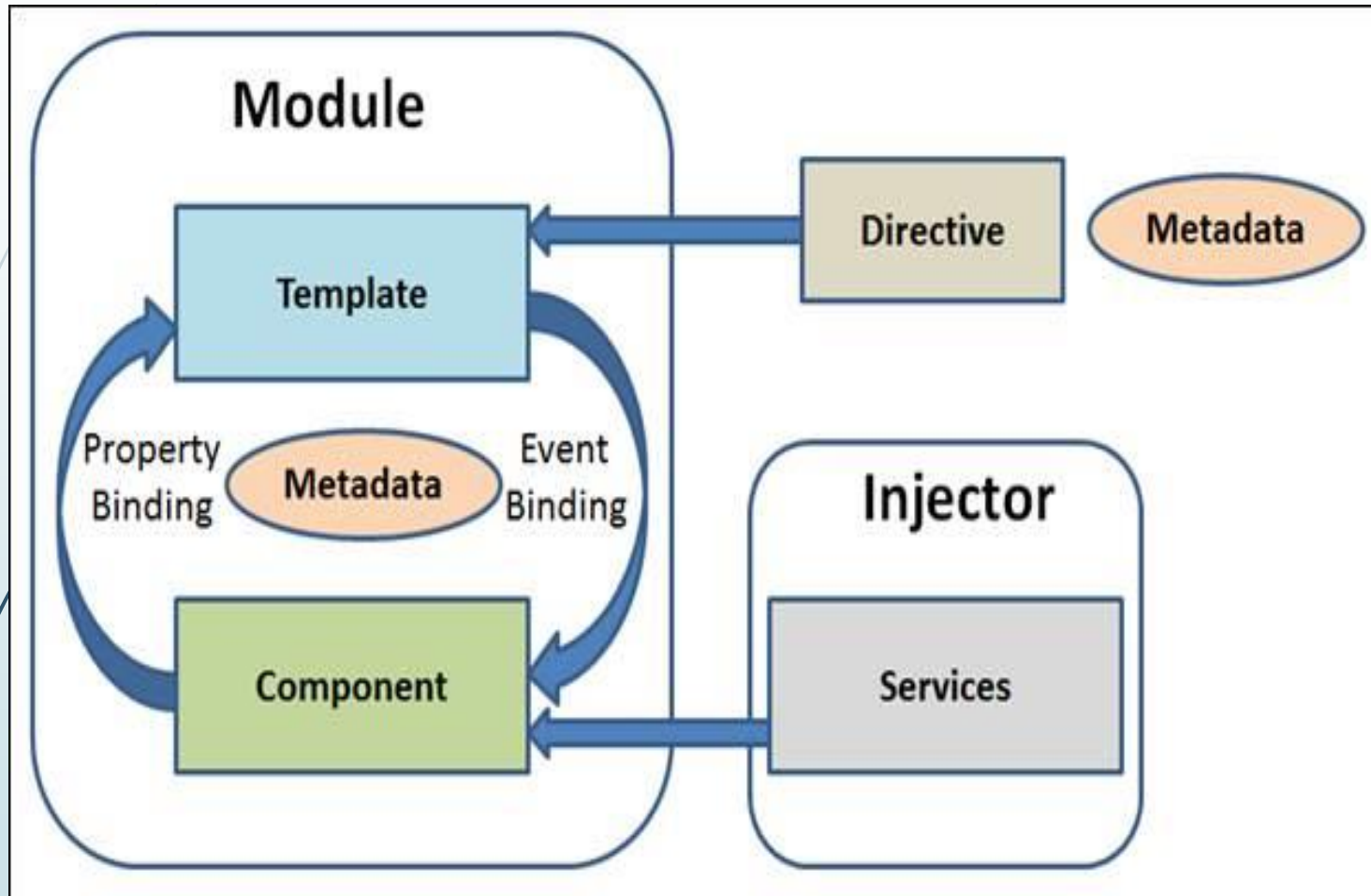
Sites développés avec Angular



Les Ressources

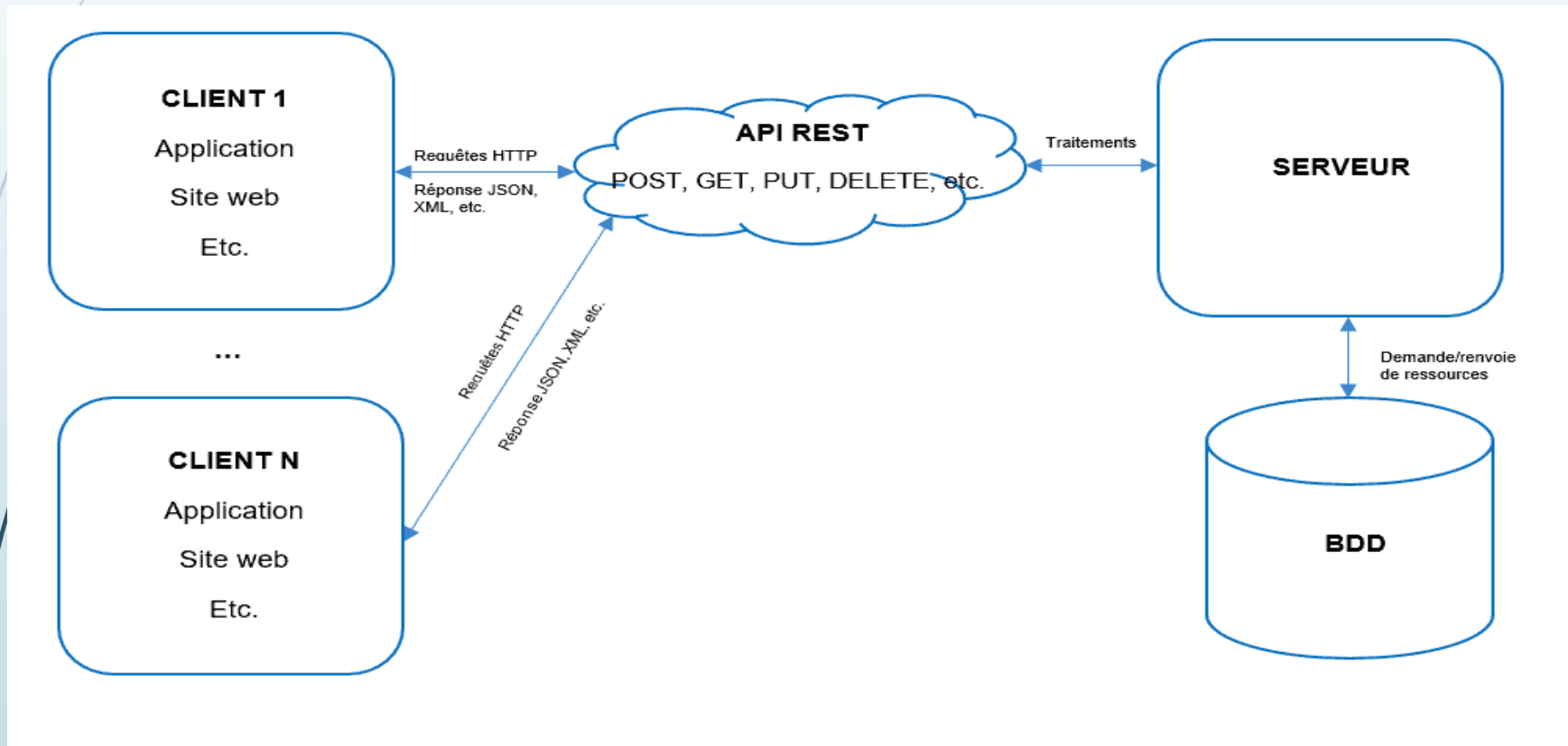
- <https://angular.io/>
- <https://www.typescriptlang.org/>
- Livre Angular et Node.js de Pierre POMPIDOR
- <https://talks.freelancerepublik.com/angular-vs-angularjs-version-choisir-2021/>
- <https://fr.slideshare.net/mohamedyousfi9/support-de-cours-angular>
- <https://code.visualstudio.com/docs/nodejs/angular-tutorial>
- <https://angular.io/guide/devtools>

Synthèse & révision



Protocole Http & Api Rest

- L'Hypertext Transfert Protocol(http) est un protocole de transfert hypertexte, de communication client serveur développé pour le World Wide Web. Html définit comment un site est construit, http détermine comment la page est transmise du serveur au client.
- Une Api est une interface de programmation d'application (Web ou pas), c'est un contrat entre un utilisateur(appel ou requête) et un fournisseur d'informations(réponse). S'il s'agit d'une Api Rest, elle respecte les contraintes d'architecture REST (Representational State Transfer) et donc permet d'interagir avec les services web RESTfull.



- [Comparatif Api Rest, Soap, GraphQL et Rpc](#)

Mise en œuvre de Json Server

- Comment mettre en œuvre une api en local pour tester une application sans attendre que le back ou la vraie Api soit opérationnelle, bref ne pas rester bloqué : Réponse : **Json Server** !
 - `npm install json-server concurrently`
 - Modifier fichier "Package.json"
 - ```
"start": "concurrently \"ng serve\" \"json-server --watch db.json\""
```
    - Permet de lancer en parallèle les 2 serveurs
- Ajout d'une base de donnée "db.json" à la racine du projet
- Tester : `npm start`

# Api Rest & Bdd



localhost:3000/trainings

```
[
 {
 "id": 1,
 "name": "Java",
 "description": "Formation Java SE 8 sur 5 jours",
 "price": 1500,
 "quantity": 1
 },
 {
 "id": 2,
 "name": "DotNet",
 "description": "Formation DotNet 3 jours",
 "price": 1000,
 "quantity": 1
 },
 {
 "id": 3,
 "name": "Python",
 "description": "Formation Python/Django 5 jours",
 "price": 1300,
 "quantity": 1
 },
 {
 "id": 4,
 "name": "Php",
 "description": "Initiation au Dev/Web avec hp 4 jours",
 "price": 1300,
 "quantity": 1
 },
 {
 "id": 5,
 "name": "NodeJs",
 "description": "Prise en main de NodeJs/Express 2 jours",
 "price": 1400,
 "quantity": 1
 },
 {
 "id": 7,
 "name": "Java Avancé",
 "description": "Formation Java Avancé/Bdd 3 jours",
 "price": 1200,
 "quantity": 1
 }
]
```



localhost:3000/trainings/5

```
{
 "id": 5,
 "name": "NodeJs",
 "description": "Prise en main de NodeJs/Express 2 jours",
 "price": 1400,
 "quantity": 1
}
```

Comment faire pour ajouter des formations, supprimer ou mettre à jour ?

# Utilisation d'un client Rest : ARC

Accueil > Applications > Advanced REST client



Advanced REST client

advancedrestclient.com

★★★★★ 11916 | Extensions |

Method Request URL  
GET http://localhost:3000/trainings

Parameters

200 OK 2099.20 ms



```
[Array[6]]
-0: {
 "id": 1,
 "name": "Java",
 "description": "Formation Java SE 8 sur 5 jours",
 "price": 1500,
 "quantity": 1
},
-1: {
 "id": 2,
 "name": "DotNet",
 "description": "Formation DotNet 3 jours",
 "price": 1000,
 "quantity": 1
},
-2: {
 "id": 3,
 "name": "Python",
 "description": "Formation Python/Django 5 jours",
 "price": 1300,
 "quantity": 1
},
-3: {
 "id": 4,
 "name": "Php",
 "description": "Initiation au Dev/Web avec hp 4 jours",
 "price": 1300,
 "quantity": 1
}
```

Request

2

Method Request URL  
GET http://localhost:3000/trainings/7

Parameters

200 OK 2068.90 ms



```
{
 "id": 7,
 "name": "Java Avancé",
 "description": "Formation Java Avancé/Bdd 3 jours",
 "price": 1200,
 "quantity": 1
}
```

Method Request URL  
DELETE http://localhost:3000/trainings/7

3

Parameters

200 OK 2085.50 ms



```
{}
```

Method Request URL  
POST http://localhost:3000/trainings/

4

Parameters

Headers

Body

Variables

Body content type Editor view  
application/json Raw input

FORMAT JSON MINIFY JSON

```
{
 "id": 10,
 "name": "PowerBi",
 "description": "Business Intelligence 10 jours",
 "price": 5000,
 "quantity": 1
}
```

201 Created 2082.40 ms



```
{
 "id": 10,
 "name": "PowerBi",
 "description": "Business Intelligence 10 jours",
 "price": 5000,
 "quantity": 1
}
```

Method Request URL  
PUT http://localhost:3000/trainings/10

5

Parameters

Headers

Body content type Editor view  
application/json Raw input

FORMAT JSON MINIFY JSON

```
{
 "id": 10,
 "name": "PowerBi",
 "description": "Business Intelligence 10 jours",
 "price": 3000,
 "quantity": 1
}
```

200 OK 2058.10 ms



```
{
 "id": 10,
 "name": "PowerBi",
 "description": "Business Intelligence 10 jours",
 "price": 3000,
 "quantity": 1
}
```

6

```
{
 "id": 5,
 "name": "NodeJs",
 "description": "Prise en main de NodeJs/Express 2 jours",
 "price": 1400,
 "quantity": 1
},
{
 "id": 10,
 "name": "PowerBi",
 "description": "Business Intelligence 10 jours",
 "price": 3000,
 "quantity": 1
}
]
```

```
[1] POST /trainings/10 404 24.107 ms - 2
[1] GET /trainings/ 200 31.646 ms - 679
[1] POST /trainings/ 201 26.099 ms - 120
[1] PUT /trainings/ 404 23.935 ms - 2
[1] PUT /trainings/10 200 28.353 ms - 120
[1] GET /trainings/ 200 20.949 ms - 815
```



# Programmation réactive

- Angular se base sur la programmation réactive ou événementiel
- En effet, sans bloquer l'appli, les données arrivent progressivement et de manière asynchrone, la vue se met à jour automatiquement dès réception des datas.

## Mise en œuvre d'un service dédié à la communication avec l'api :

- 1/ Ajout du module HttpClientModule (from '@angular/common/http';)
- 2/ Ajout d'un service de communication avec l'api (services/trainings)

```
export class TrainingsService {

 constructor(private http:HttpClient) { }

 public getTrainings(){
 return this.http.get<Training[]>(environment.host+"/trainings");
 }
}
```

```
src
├── app
├── assets
├── environments
│ ├── environment.prod.ts
│ └── environment.ts
└── M
 5 export const environment = {
 6 production: false,
 7 host : "http://localhost:3000"
 8 };
 9
 10 /*
```

- 3/ Appel du service dans un composant

```
export class TrainingsComponent implements OnInit {
 listTrainings : Training[] | undefined;
 error = null;

 constructor(private cartService : CartService, private router : Router,
 private trainingsService : TrainingsService) {
 }

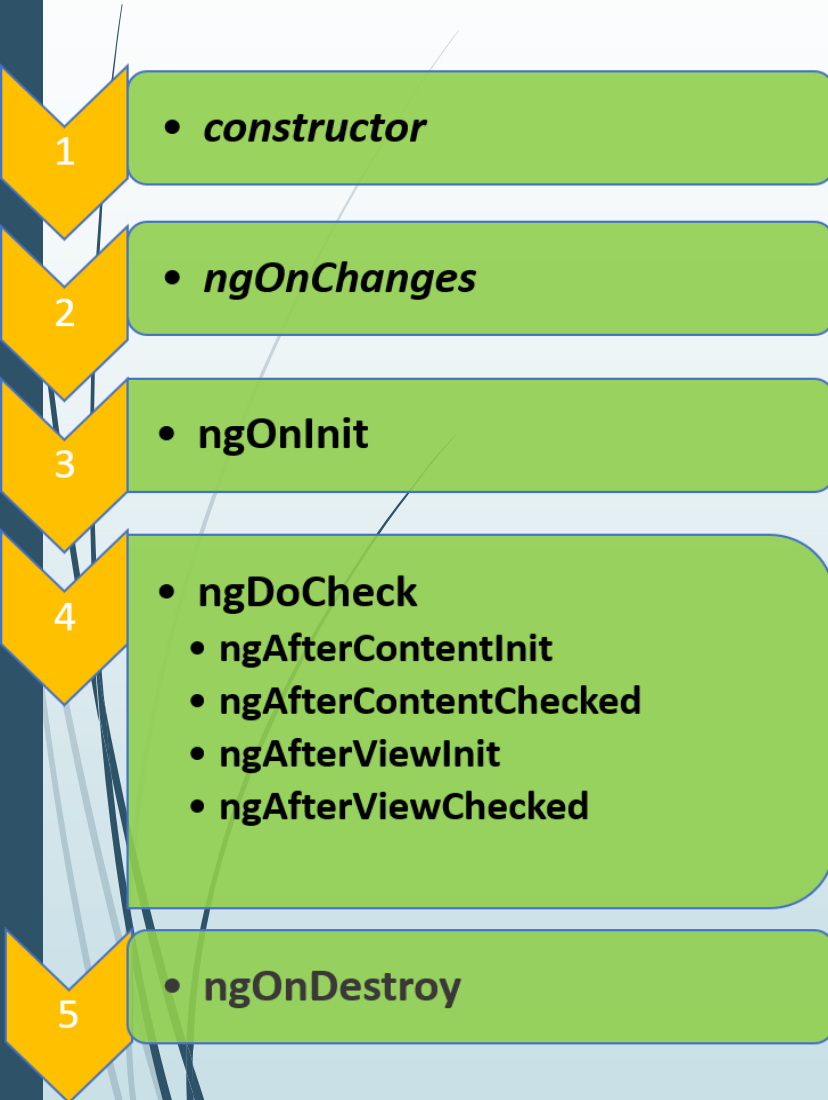
 ngOnInit(): void {
 this.getAllTrainings();
 }

 getAllTrainings() {
 this.trainingsService.getTrainings().subscribe(
 next : (data) => this.listTrainings = data,
 error : (err) => this.error = err.message,
 complete : () => this.error = null
)
 }
}
```

| ID | NOM     | DESCRIPTION                             | PRIX | QUANTITE                       | PANIER                  |
|----|---------|-----------------------------------------|------|--------------------------------|-------------------------|
| 1  | Java    | Formation Java SE 8 sur 5 jours         | 1500 | <input type="text" value="1"/> | <a href="#">Ajouter</a> |
| 2  | DotNet  | Formation DotNet 3 jours                | 1000 | <input type="text" value="1"/> | <a href="#">Ajouter</a> |
| 3  | Python  | Formation Python/Django 5 jours         | 1300 | <input type="text" value="1"/> | <a href="#">Ajouter</a> |
| 4  | Php     | Initiation au Dev/Web avec hp 4 jours   | 1300 | <input type="text" value="1"/> | <a href="#">Ajouter</a> |
| 5  | NodeJs  | Prise en main de NodeJs/Express 2 jours | 1400 | <input type="text" value="1"/> | <a href="#">Ajouter</a> |
| 10 | PowerBi | Business Intelligence 10 jours          | 3000 | <input type="text" value="1"/> | <a href="#">Ajouter</a> |

# Cycle de vie d'un composant & Hooks

Un composant passe par différentes étapes de sa naissance à sa mort. Il est possible d'intercepter ces étapes et d'y associer du code via des méthodes prédéfinies (et déclarées dans des interfaces de programmation). Ces méthodes sont appelées des hooks



Voici la liste des quatre hooks les plus importants :

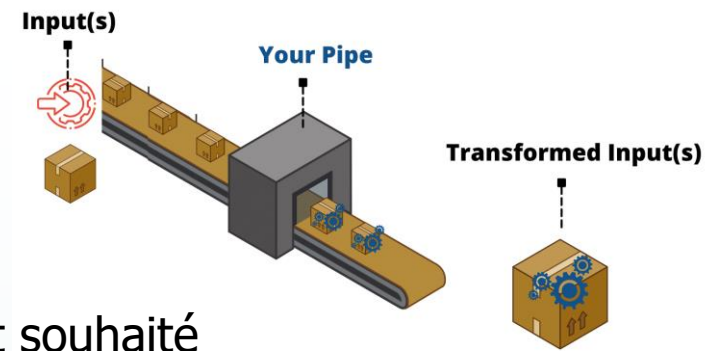
• `ngOnChanges()` : cette méthode est appelée à la création du composant, puis à chaque changement de valeur d'un attribut scalaire décoré par `@Input()`.

• `ngOnInit()` : cette méthode est appelée lors de la création du composant (juste après le premier appel de `ngOnChanges()`).

• `ngDoCheck()` : cette méthode est mise en œuvre pour connaître les changements des valeurs internes d'objets ou de listes, changements de valeurs non identifiables par le hook `ngOnChanges()`.

• `ngOnDestroy()` : cette méthode est appelée juste avant que le composant ne soit désalloué.

# Pipes



Servent à mettre en forme des données dans le format souhaité  
Pour ce faire, il y a l'opérateur |

Nous pouvons appliquer des pipes à n'importe quelle vue ou modèle et à toute donnée d'entrée.

Voici quelques exemples d'utilisation :

```
<tr *ngFor="let p of cartService.getCart()">
 <td>{{p.id}}</td>
 <td>{{p.name}}</td>
 <td>{{p.quantity}}</td>
 <td>{{p.price | number:'0.2'}}</td>
</tr>
```

```
<td>{{cartService.getAmount() | currency : 'EUR'}}</td>
```

```
<div class="form-group">
 <label>Date de la commande : </label>
 <label> {{dateOrder | date:"MM/dd/yy"}}</label>
</div>
```

ID	Nom	Quantité	Prix
1	Java	1	1,500.00
2	DotNet	1	1,000.00
Total :			€2,500.00

Date de la commande :05/23/22



# Template Driven Form Vs Reactive Forms

Jusqu'ici nous avons utilisé les formulaires basés sur un modèle (**FormsModule**), plutôt pratique pour des formulaires simples d'authentification, de contacts ou autre. En bref, ils sont parfaitement **adaptés à l'utilisation de cas simples**.

Les formulaires réactifs (**ReactiveFormsModule**) offrent des **formulaires dynamiques** dont les valeurs changent au fil du temps. Ils peuvent paraître complexe dans leur mise en œuvre mais s'avère très puissants et flexibles. Compte tenu de leur approche, toute la déclaration et la structuration des formulaires est à faire dans le code Typescript exploité côté Html.

NB : Ajouter

**ReactiveFormsModule**

Dans les imports de app.module

```
import { FormControl, FormGroup } from '@angular/forms';

@Component({
 selector: 'app-customer',
 templateUrl: './customer.component.html',
 styleUrls: ['./customer.component.css']
})
export class CustomerComponent implements OnInit {
 myForm : FormGroup;
 constructor(public cartService : CartService, private router : Router) {
 let customer = this.cartService.getCustomer();
 this.myForm = new FormGroup({
 name : new FormControl(customer.name),
 firstName : new FormControl(customer.firstName),
 address : new FormControl(customer.address),
 phone : new FormControl(customer.phone),
 email : new FormControl(customer.email)
 })
 }

 ngOnInit(): void {
 }

 onSaveCustomer(form : FormGroup){
 if(form.valid){
 this.cartService.saveCustomer(new Customer(form.value.name, form.value.firstName,
 form.value.address, form.value.phone, form.value.email));
 this.router.navigateByUrl('order');
 }
 }
}
```

```
<form [formGroup]="myForm" (ngSubmit)="onSaveCustomer(myForm)">
 <div class="form-group">
 <label class="control-label">Nom : </label>
 <input type="text" formControlName="name" class="form-control">
 </div>
```

Variante avec un FormBuilder

```
ngOnInit(): void {
 this.myForm = this.formBuilder.group({
 name : [this.customer.name, Validators.required],
 firstName : [this.customer.firstName, Validators.required],
 address : [this.customer.address, [Validators.required, Validators.minLength(25)]],
 phone : [this.customer.phone, [Validators.required, Validators.maxLength(10)]],
 email : [this.customer.email, [Validators.required, Validators.pattern('[a-z0-9.@]*')]]
 })
}
```



- *Reactive extensions for Javascript* est une librairie permettant de travailler sur des **flux de données asynchrones** représentés à l'aide de **séquence d'observables**.
- Il faut être à l'aise avec les **Arrow functions** (fonctions anonymes ou expressions lambda) pour aborder sereinement la notion **d'Observables** !
- En effet, on parle ici de **programmation réactive** qui se base sur le **Design pattern Observer** avec l'idée que des **observables** (subject) vont émettre des événements asynchrones, interceptés par les **observateurs** (observers)

Voir dossier labo/manip opérateurs...

```
const data$ = new Observable(observer => { // on définit ici un observable (qui peut être observé)
 observer.next('first call' + counter++); //voici l'appel qui sera sollicité par les observateurs ayant souscrit
 setTimeout(() => {
 observer.next('second call' + counter++); //2ème appel 5 secondes plus tard ...
 }, 5000);
});

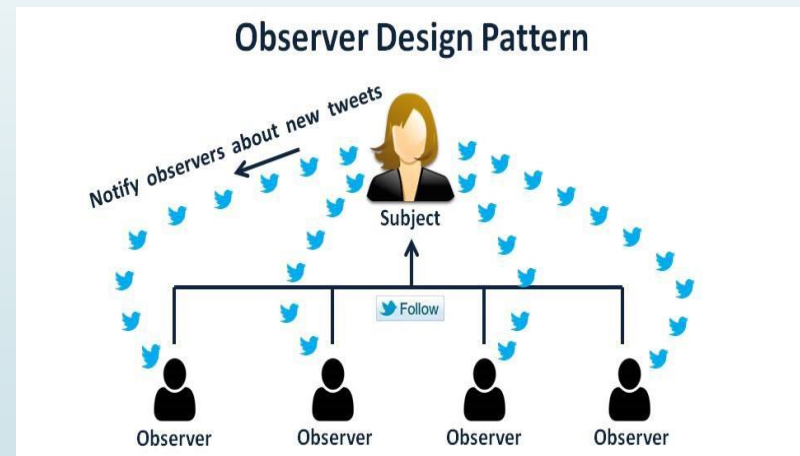
let counter = 0;
const observer1 = data$.subscribe({ //l'observateur1 souscrit à l'observable et gère 2 cas :
 next: value => console.log('1-' + value), //chaque fois qu'une valeur est émise par l'observable, il l'affiche
 error: err => console.log('1-' + err), //affichage en cas d'erreur
});

//et puis un autre observateur sur le même Observable
data$.subscribe(value => {
 console.info('2--' + value);
});

//création d'un observable à partir d'un tableau de string
const dataTables = from(['monday', 'tuesday', 'wednesday', 'thursday', 'friday', 'saturday', 'sunday']);
dataTables.subscribe(val => console.log(val)); //souscription à celui-ci

//voilà donc un tuyau qui permet d'appliquer plusieurs traitements via les opérateurs tap, map, ...
data$.pipe(
 tap(item => console.log('3---' + item)), //lère affichage + renvoi de la source
 map(dat => dat + ' end') //map : renvoi un observable avec chaque donnée transformée
).subscribe(
 dis => console.log('hi ' + dis)
);

//of crée un Observable sur une liste de chiffres qui passent par un tuyau qui filtre(pair) puis modifie les datas avec map
//avant de renvoyer un observable auquel il faut souscrire si on veut avoir un affichage
const obs$ = Observable<number> = of(1, 2, 3, 4, 5, 6, 7, 8, 9, 10).pipe(filter(v => v % 2 === 0), map(v => v * 10));
obs$.subscribe(
 value => console.log(value)
);
```



## Pourquoi les patterns ?

- Des « recettes d'expert » ayant fait leurs preuves
- Un vocabulaire commun pour les architectes logiciels
- Incontournable dans le monde de la P.O.O