FMS ACADEMY BY FMS

FACYLITIES MULTI SERVICES

Une entreprise ordinaire à l'agilité extraordinaire

FORMATION JAVA SPRING ANGULAR

ANGULAR NIV 2

mohamed.el-babili@fms-ea.com

33+ 628 111 476

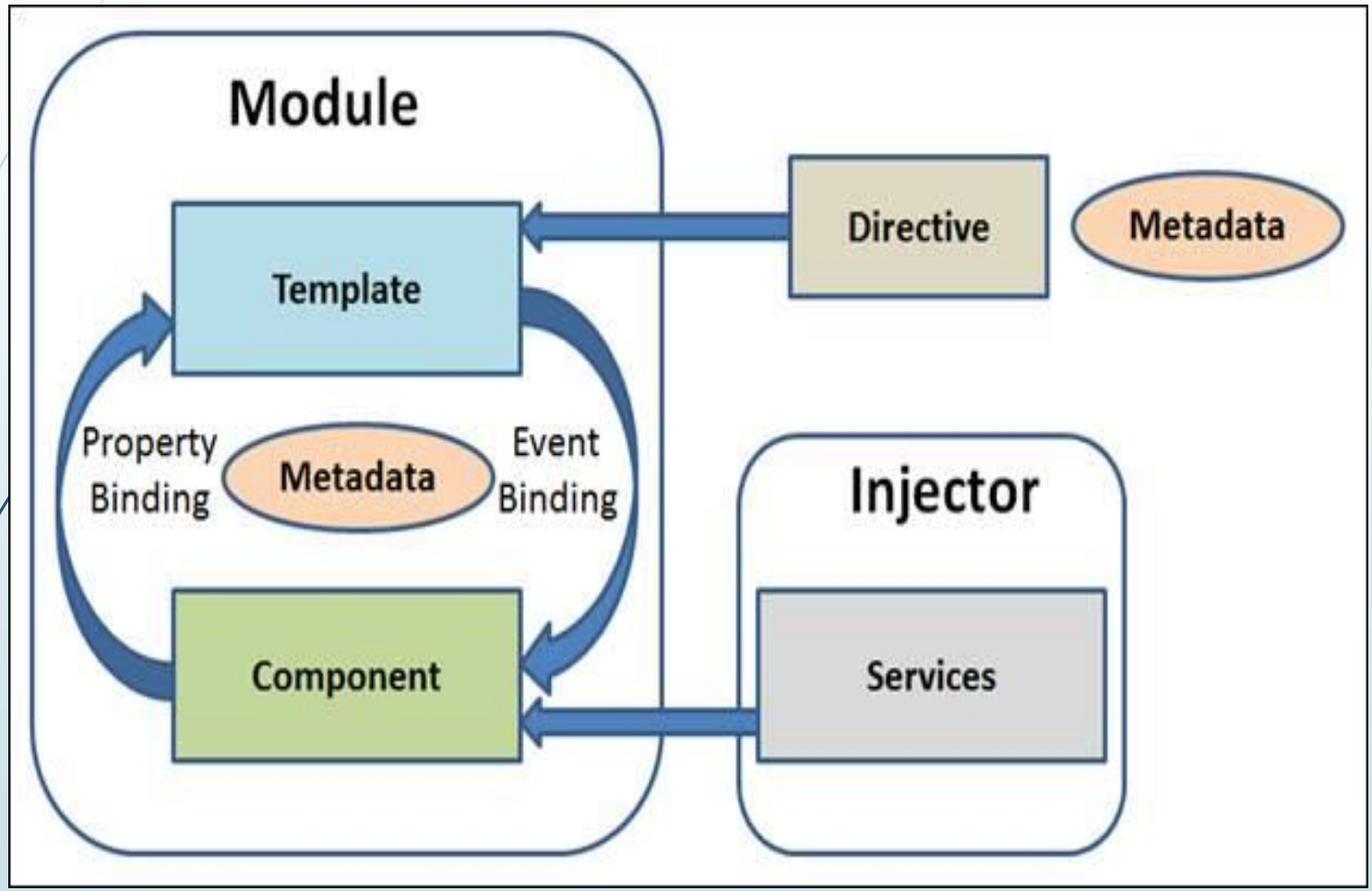
Version : 2.0

DMAJ : 16/04/23

Sommaire

ANGULAR avancé

Synthèse

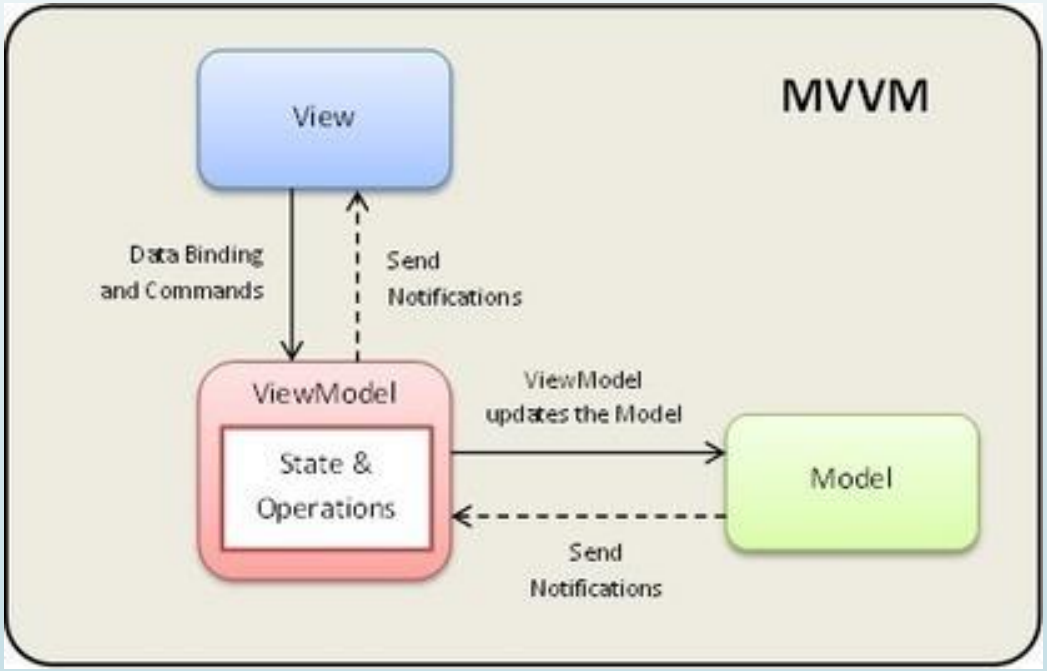


MVVM

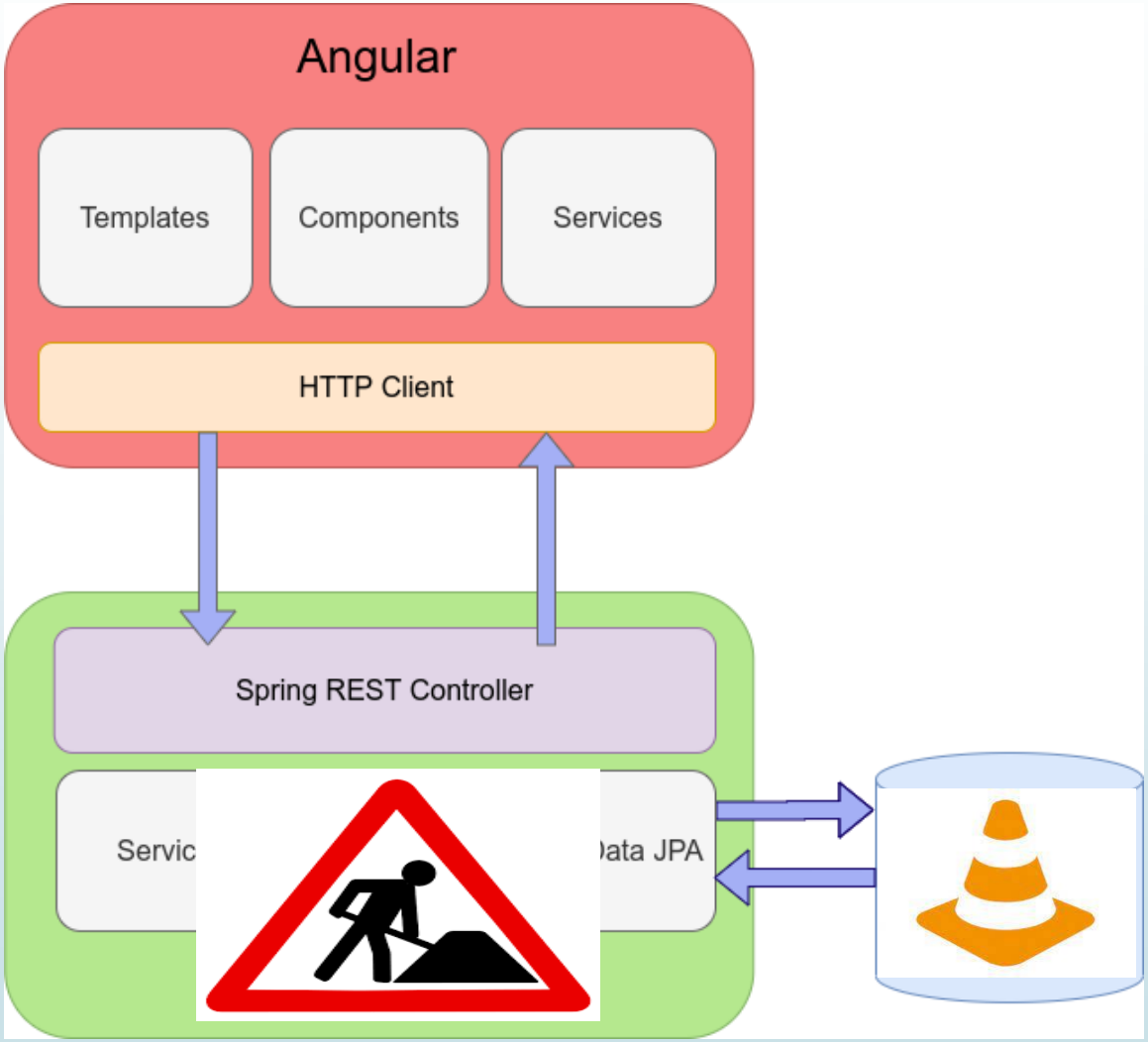
L'architecture MVVM (Model-View-ViewModel) est une approche de conception logicielle qui vise à séparer les données d'une application (le modèle) de sa présentation (la vue).

Le ViewModel est une sorte de pont entre le modèle et la vue, il fournit une interface que la vue peut utiliser pour accéder aux données du modèle de manière appropriée.

Cette architecture permet de rendre les vues plus indépendantes du modèle, ce qui facilite la maintenance et l'évolution de l'application.



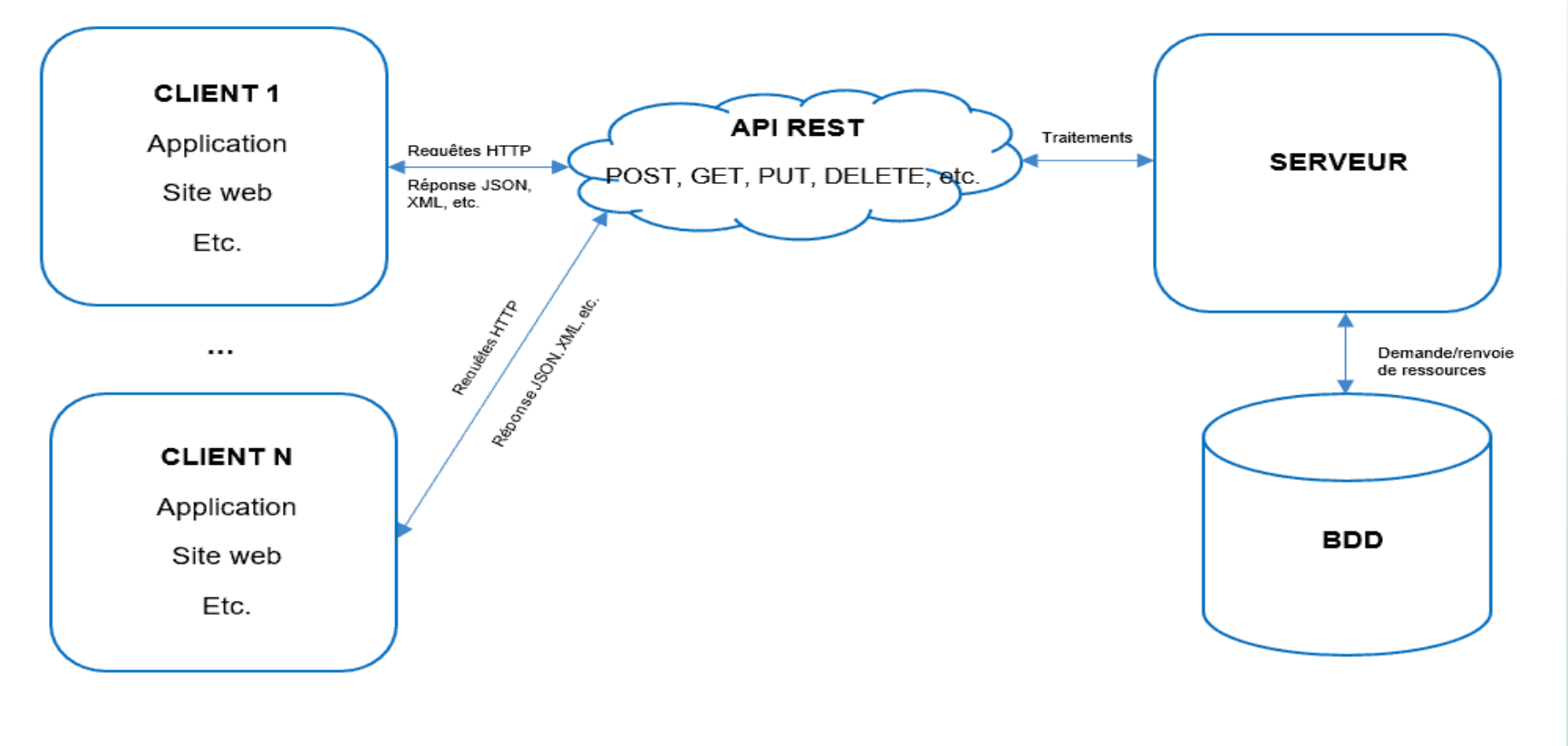
C’est bien beau tout ça mais comment je vais simuler l’activité finale de mon application quand l’appli/api côté back n’est pas prête !



Protocole Http & Api Rest

- L'Hypertext Transfert Protocol(http) est un protocole de transfert hypertexte, de communication client serveur développé pour le World Wilde 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.

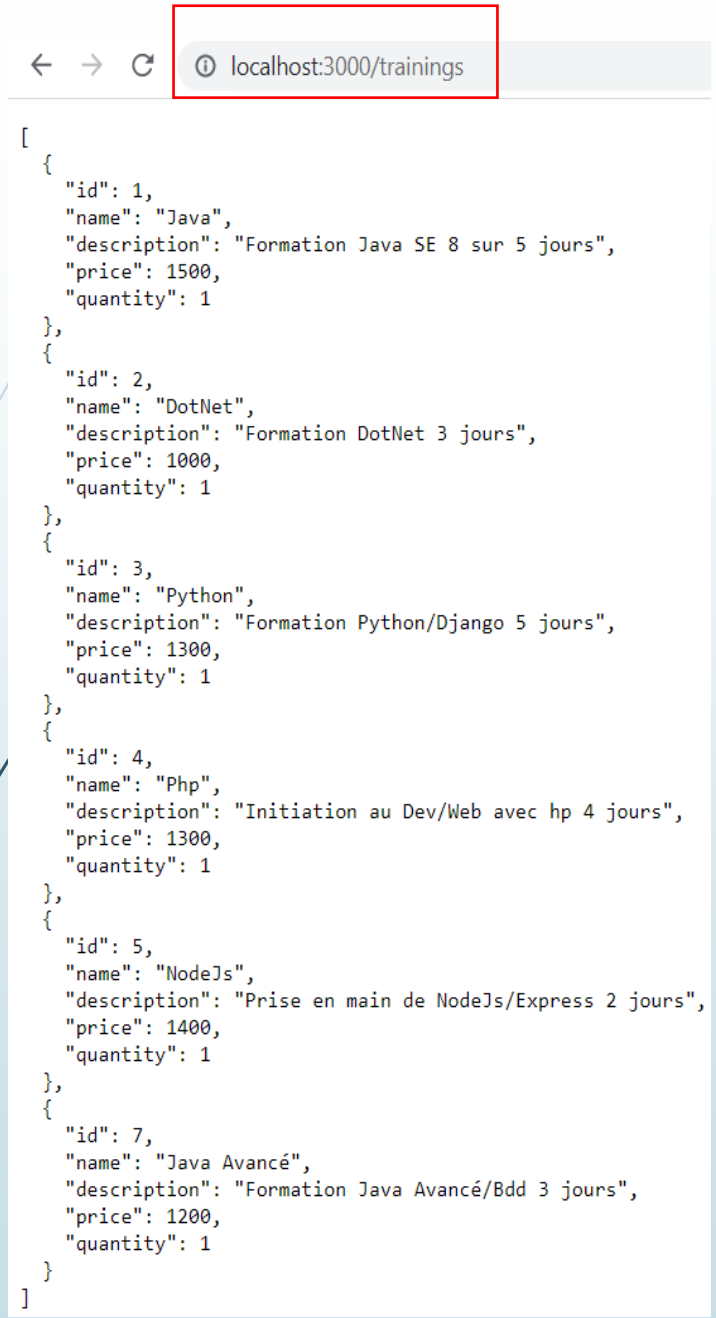


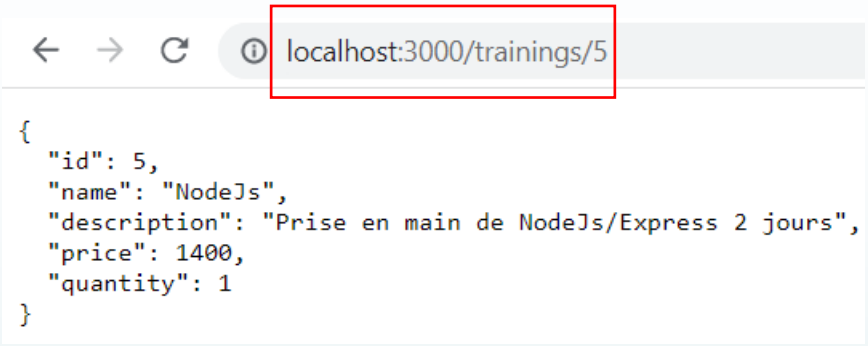
- [Comparatif Api Rest, Soap, GraphQL et Rpc]

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`
  - Modif fichier “Package.json”
    - `"start": "concurrently \"ng serve\"  \"json-server --watch db.json\"",`
    - Permet de lancer en parallèle les 2 serveurs
- Ajouter une base de donnée “db.json” à la racine du projet
- Tester : npm start

Api Rest & Bdd





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

1

Method: GET
Request URL: 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
```


2

Request

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

Parameters

200 OK 2068.90 ms DETAILS


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


3

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

Parameters

200 OK 2085.50 ms DETAILS

{}

4

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

Parameters

Headers | Body | Variables

Body content type: application/json
Editor view: 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
}
```


5

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

Parameters

Headers

Body content type: application/json
Editor view: 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

Il est question maintenant d'exploiter notre api dans notre application

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

Ajouter un service dédié

```
export class ApiService {

  constructor(private http:HttpClient) { }

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

  public getTraining(id : number) {
    return this.http.get<Training>(environment.host+"/trainings/"+id);
  }
```

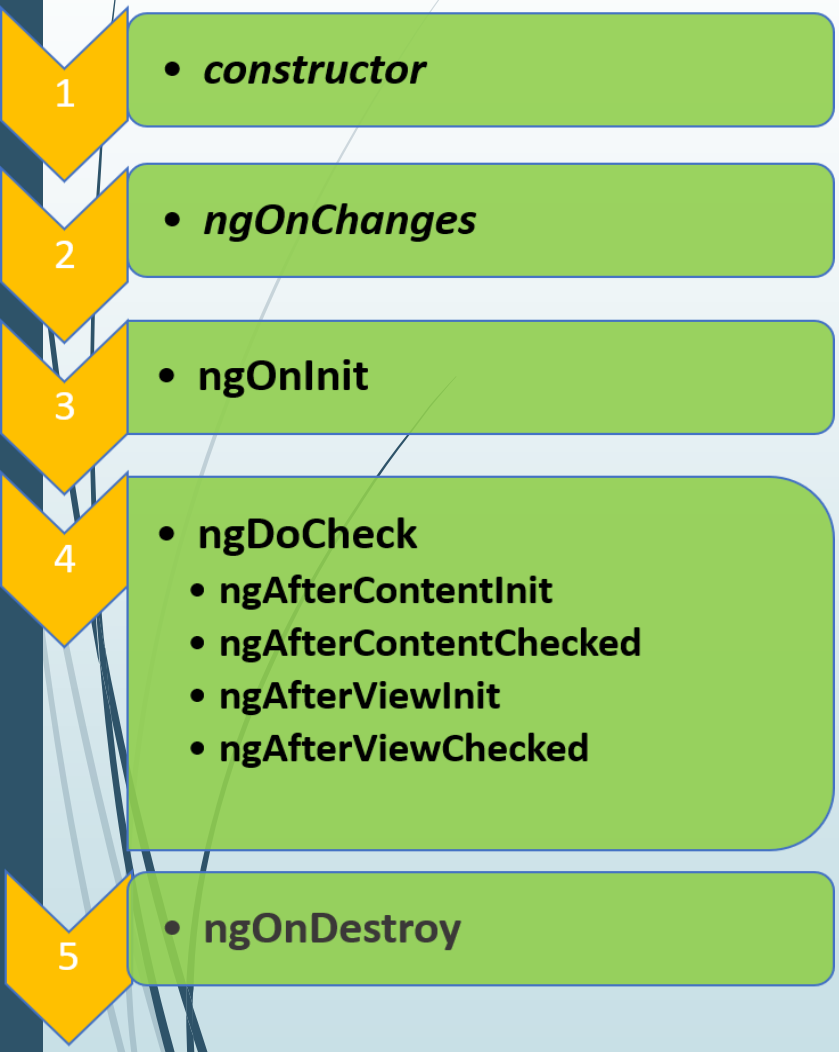
Environnement.ts

```
export const environment = {
  production: false,
  host : "http://localhost:3000"
};
```

Ajouter le module : `HttpClientModule`

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 quatres hooks les plus importants :

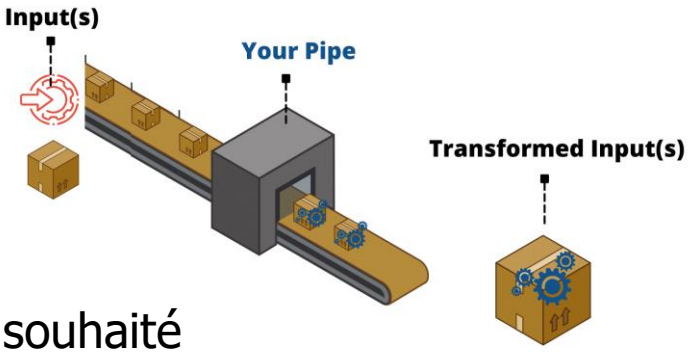
•`ngOnChanges()` : cette méthode est appelée à la création du composant, puis à chaque changement de valeur d'un attribut scalaire décoré par `@Input()`.

•`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

Différence entre ngContainer et ngTemplate

```
<div class="container">
    <ng-container *ngIf="listTrainings else loading">
        <table class="table table-striped">
            <thead>
                <tr>
                    <th>ID</th> <th>NOM</th> <th>DESCRIPTION</th> <th>PRIX</th> <th>QUANTITE</th> <th>PANIER</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>
                    <td> <input type="number" name="quantity" style="width: 60px" [(ngModel)]="training.quantity"
                    <td> <a class="btn btn-primary" (click)="onAddToCart(training)"> Ajouter </a></td>
                </tr>
            </tbody>
        </table>
    </ng-container>
    <ng-template #loading>
        <div>Loading...</div>
    </ng-template>
</div>
```

ng-container sert de conteneur pour les éléments qui peuvent accepter des directives structurelles mais ne sont pas ajoutés au DOM

ng-template permet de créer un contenu de modèle qui n'est pas ajouté tant que vous ne l'avez pas spécifiquement (conditionnellement ou directement) ajouté au DOM

En clair, il s'agit d'optimisation de gestion de l'utilisation du DOM

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 paraitre 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

```typescript
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');
```

```html
<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

```typescript
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.@]*')]]
  })
}
```

Guards

Il est parfois nécessaire d'empêcher l'accès vers une route ou composant de votre application pour des raisons de sécurité.
C'est possible à l'aide des guards ou gardes.
Par exemple, un composant d'administration que seules les admins peuvent avoir accès servant à créer des utilisateurs.

```
PS C:\Users\El-BabiliM\angular-workspace\trainings-front-app-auth> ng g g components/admin
? Which interfaces would you like to implement? (Press <space> to select, <a> to toggle all, <i> to invert selection, and <enter> to proceed)
>(*) CanActivate
 ( ) CanActivateChild
 ( ) CanDeactivate
 ( ) CanLoad
```

```
@Injectable({
  providedIn: 'root'
})
export class AdminGuard implements CanActivate {
  constructor(private authService:AuthenticateService, private router : Router){}
  canActivate(
    route: ActivatedRouteSnapshot,
    state: RouterStateSnapshot): Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean | UrlTree {

    if(this.authService.isAdmin()){
      return true;
    }
    else {
      this.router.navigateByUrl('/login');
      return false;
    }
```

App-routing.module.ts

```
    {
      path : 'admin' , component : AdminComponent,
      canActivate : [AdminGuard]      You, 19 minutes ago • Uncom
    },
    { path : '' , redirectTo : 'trainings', pathMatch : 'full' },
    { path: '404', component: NotFoundComponent},
    { path: '**', redirectTo: '/404'}
];
```

App.component.html

```
<li *ngIf="this.authService.isAdmin()" class="nav-item">
  <a class="nav-link" href="#" routerLink="/admin">Administration</a>
</li>
```

Du coup, grâce à la directive ici, le lien n'apparait que pour les admins. De plus, si qq tente d'accéder à la route correspondante, il est renvoyé vers le formulaire d'authentification

Les Ressources

- https://angular.io/
- https://angular.fr/routing/guard.html
- https://www.youtube.com/@CoulissesTech