Cahier de Tps : Formation SpringBoot / Angular (Partie II)

IDE Recommandé : VSCode avec extensions TypeScript

II-1 : Démarrage avec Angular-CLI

Le premier TP a pour but de créer un premier projet avec Angular CLI.

1. Installation *Node.js* et *npm*

Si le poste n'est pas installé, référer vous aux procédures d'installation de *http://nodejs.org* en fonction de votre système d'exploitation

Vérifier la version avec ***node -v***

2. Installation Angular CLI

npm install -g @angular/cli@11.1.2

Vérifier avec :
ng –version

3. Création de projet
Placer vous dans un répertoire de travail et exécuter :

ng new tp1 --strict

L'option --strict augmente la maintenabilité d'un projet en forçant une utilisation stricte de TypeScript (voir https://angular.io/guide/strict-mode)

Utiliser les réponses par défaut et visualiser les fichiers générés

4. Exécution de l'application

Démarrer le Live Reload Server en exécutant
ng serve

Accéder à l'application http://localhost:4200/

Modifier le fichier *app.component.html* et observer le *Live Reload*

Exécuter ensuite
ng test

II-2 : Les composants

II-2.1 – Création de composant

1. Créer un composant ***ProductList*** avec Angular-cli
2. Visualiser les fichiers générés et la modification effectuée dans *app.module.ts*
3. Ajouter le composant dans l'arbre de composant
4. Accéder à l'application

II-2.2 Interpolation, binding, directives

- Ajouter une propriété ***name*** au composant précédemment créé et lui affecter une valeur en dur lors du constructeur
- Afficher la propriété dans le template

- Encapsuler l'affichage dans un paragraphe html <p>, modifier l'attribut class via un binding de propriété
- Définir la classe dans le fichier css local

- Ajouter un bouton dont l'attribut *disabled* est égal à une nouvelle propriété booléenne du composant
- Ajouter une case à cocher qui permet de contrôler l'activation/désactivation du bouton
- Afficher l'état du bouton en passant par une variable locale

- Ajouter une nouvelle propriété de type array, nommée *products*
- initialiser cette propriété avec des objets contenant un attribut *date* et un attribut *reference, vous pouvez utiliser la fonction ngInit*
- Afficher ce tableau dans une liste à puce
- Forcer l'affichage de la référence en majuscule

- Mettre en place un binding bidirectionnel entre un champ input et la propriété ***name*** du composant

II-2 : Services et injection de dépendances

II-2.1 – Création service, interface et implémentation

1. Créer une classe modèle ***Product*** via angular CLI.. Par exemple :
   `ng generate class model/product`

2. Y définir les attributs suivants :
   ```
   id : number
   reference : string
   nom : string
   description ? : string
   prixUnitaire : string
   availability : number
   ```

3. Créer une classe abstraite ProductService via angular CLI. Par exemple :
   `ng generate interface service/product-service`
   Puis modifier l'interface en classe abstraite

4. Y définir les méthodes abstraites suivantes :
   ```
   findAll(): Array<Product>
   findOne(id: number): Product
   create(product: Product): Product
   update(product: Product): Product
   delete(id: number): void
   ```

5. Créer une classe d'implémentation FakeService se basant sur le tableau de produits suivants
   ```
   products: Array<Product> = [
   {id: 1, reference: 'REF1', nom: 'Nom1', availability: 1,
   prixUnitaire: 10},
   {id: 2, reference: 'REF2', nom: 'Nom2', availability: 2,
   prixUnitaire: 20},
   {id: 3, reference: 'REF3', nom: 'Nom3', availability: 3,
   prixUnitaire: 30}
   ]
   ```

II-2.1 – Injection et utilisation du service

Configurer le module afin que la classe asbtraite soit résolue en l'implémentation ***FakeService***

Injecter la classe abstraite dans le composant ***ProductListComponent*** et l'utiliser.

II-3 : Mise en place de *ng-bootsrap*

II-3.1 – Installation

```
ng add @ng-bootstrap/ng-bootstrap
```

Vérifier les importations de composants dans le module

II-3.2 – Mise en place d'un layout et d'un barre de navigation

Mettre en place un layout basique faisant apparaître une zone de navigation latérale dans le composant *Application* comme suit :

```
<div class="container-fluid">
<div class="row">
<div class="col-1">
        Zone de menu
</div>
<div class="col-sm">
        <app-product-list> </app-product-list>
</div>
</div>
</div>
```

Créer un composant *NavigationBar* dont le template utilise une barre de navigation bootstrap :

```
<nav class="navbar navbar-expand-lg navbar-dark bg-dark">
  <a class="navbar-brand" href="#">Catalogue Produits</a>
  <button class="navbar-toggler" type="button" data-toggle="collapse" data-
target="#navbarNav" aria-controls="navbarNav" aria-expanded="false" aria-
label="Toggle navigation">
    <span class="navbar-toggler-icon"></span>
  </button>
  <div class="collapse navbar-collapse" id="navbarNav">
    <ul class="navbar-nav mr-auto">
      <li class="nav-item">
        <a class="nav-link" href="#">Fournisseurs <span
class="sr-only">(current)</span></a>
       </li>
       <li class="nav-item active">
      <a class="nav-link" href="#">Produits</a>
       </li>
     </ul>
    <ul class="form-inline my-2 my-lg-0">
      <li>
        <div class="btn-group">
```

```
    <button type="button" class="btn btn-info dropdown-toggle" data-
toggle="dropdown" aria-haspopup="true" aria-expanded="false">
User Loggé
   </button>
   <div class="dropdown-menu">
     <a class="dropdown-item" href="/logout">Logout</a>
    </div>
  </div>
  </li>
</ul>
</div>
</nav>
```

Ajouter ce composant dans l’application

II-4 : Routing

II-3.1 – Routing simple

Dans le moule applicatif, mettre en place une règles de routage afin que l'url par défaut de l'application ainsi que ***/products*** affiche la liste des produits

Mettre à jour la barre de navigation

II-3.1 – Routing enfant

Définir une deuxième niveau de navigation utilisé par le menu latéral permettant d'afficher dans la zone centrale la liste de produit ou un composant détail

II-4 : Formulaire

II-4.1 – Mise en place

Ajouter ***ReactiveFormsModule***

II-4.2 – Formulaire produits

Le composant *formulaireProduit* gère la création et la mise à jour de produit.

Se faire injecter :

- *FormBuiler* pour construire le formulaire
- *ActivatedRoute* afin de récupérer un éventuel paramètre de l'identifiant produit

Construire le formulaire avec les règles de validation suivante :

- Référence obligatoire maximum 4 caractères
- Nom obligatoire
- Description optionnelle limité à 255 caractères
- Prix : chiffre > 0

En fonction de la présence du paramètre, faire un appel à fakeService et initialiser le formulaires avec le produit retourné par le service

II-4.3 – Liste produits et navigation

Sur la liste des produits, ajouter des liens permettant d'éditer un produit, tester une mise à jour

II-5 : Communication back-end

II-5.1 – Mise en place

Démarrer l'application SpringBoot
Ajouter le module ***HttpClientModule***
Définir dans un fichier ***app.constant.ts*** l'URL du serveur comme suit :

```
import { environment } from 'src/environments/environment';
export const SERVER_API_URL = environment.SERVER_API_URL+'/api';
```

Définir SERVER_API_URL dans les 2 fichiers d'environnements

Changer la définition de l'interface ***ProductService*** en utilisant des Observable

Modifier l'implémentation de ***FakeService*** et l'utilisation de ***ProductService*** dans les composants

Tester votre application.

II-5.2 – Implémentation du service back-end

Créer un nouveau service product-back-service qui implémente ProductService.
Implémenter les appels REST de type GET vers l'application SpringBoot.

Modifier la configuration pour utiliser la nouvelle classe (vous pouvez utiliser la propriété useFactory)

II-5.2 – Accès aux ressources sécurisées

Implémenter un service Authentication proposant les méthodes

- ***authenticate(username, password)*** : La méthode effectue un post pour récupérer un token et les informations utilisateurs et les stocke dans la session (sessionStorage)
- ***isLoggedIn*** : Teste session storage sur le présence de l'utilisateur
- ***logout*** : Nettoie sessionStorage

Implémenter un interceteur ***BasicAuthHtppInterceptorService***
qui teste la présence du token en session et si il est présent l'ajoute dans l'entête ***Authorization*** de la requête
Enregistrer l'intercepteur dans le module