Cours Angular

4.1 | TD Hackers

Routage – Service – Stockage des données

1] Principes de la nouvelle application

L'application va porter sur la recherche de l'origine des hackers cherchant à pirater un serveur.

À partir d'une IP fournie par un outil comme fail2ban (http://www.fail2ban.org/wiki/index.php/Main_Page), l'application effectuera une recherche sur le pays d'origine via une API externe de localisation : https://freegeoip.appjson/.

- ***Test de l'API***

T1.1 Enregistrez-vous sur le site **FreeGeoIP** et à l'aide d'un navigateur, mettez une URL de type `https://freegeoip.app/json/x.x.x.x` où bien sûr vous remplacerez les quatre x par une IPv4 publique ; si vous en avez une mettez la votre, sinon vous pouvez mettre **167.71.210.244** qui a été un de mes derniers « attaquants » sur mon serveur.

Celle-ci donne en retour :

```
{"ip":"167.71.210.244","country_code":"SG","country_name":"Singapour","region_code":"","region_name":"","city":"Singapour","zip_code":"62","time_zone":"Asia/Singapore","latitude":1.3078,"longitude":103.6818,"metro_code":0}
```

Avec une présentation plus structurée avec **Firefox** :

- ***À nouvelle structure, nouvelle application***

Classiquement une application comporte plusieurs pages et donc un menu, ce sera le principe du **routage** sous Angular. Pour le routage, n'hésitez pas à consulter la documentation officielle : https://angular.io/guide/router.

T1.2 Dans le répertoire des projets **Angular**, créez la nouvelle application cette fois avec **obligatoirement** le routage :

```
angular$ ng new hackers --style=css --routing=true
```

T1.3 Installez le framework **Bulma** comme vu dans les précédents TDs, lancez l'application et générez tout de suite les composants `header`, `footer`, `home` et `liste` dans un dossier `components` (à créer et pour avoir une meilleure structure).

Nous avons deux composants « template » (`header`, `footer`) et deux composants « pages » (`home`, `liste`) correspondant aux liens du menu.

À la création de l'application, la mention de l'option pour le routage a permis de créer entre autres le fichier `app-routing.module.ts`. C'est dans ce fichier que nous indiquerons dans un tableau les routes possibles.

- ***Mise en place du template de l'application***

Classiquement d'abord le composant `header`, ensuite le composant `footer` et au final la page `home`.

T1.4 Mettez le code suivant pour le template du `header` :

```
<nav class="navbar" role="navigation" aria-label="main navigation">
  <div class="navbar-brand">
     <img src="assets/images/logo_hackers.png">
    <h1 class="title is-1"> Hackers Base</h1>
  </div>

  <div id="menu" class="navbar-menu">
    <div class="navbar-end">
      <div class="navbar-item">
        <div class="buttons">
          <a class="button is-light">
            Home
          </a>
          <a class="button is-light">
            Liste
          </a>
        </div>
      </div>
    </div>
  </div>
</nav>
```

Et celui-ci dans le template du `footer` :

```
<footer class="footer">
  <div class="container">
    <div class="content has-text-centered">
      <p>
        Template réalisé à partir du framework CSS <a href="https://bulma.io">Bulma</a>
      </p>
    </div>
  </div>
</footer>
```

Et enfin, celui-ci dans le template du composant `home` sachant que pour l'occasion, j'ai récupéré deux images (à mettre dans `assets/images`) : une pour le logo et une autre pour le fond (prenez en d'autres si vous le voulez sinon vous les trouverez sur le site de cours) :

```
<section class="section">
  <div class="container">
    <div class="card-image has-text-centered">
      <figure class="image is-inline-block">
        <img class="is-rounded" src="assets/images/fond_hackers.jpg" />
      </figure>
    </div>
  </div>
</section>
```

- ***Construction du routage***

T1.5 Changez le contenu du fichier dédié au routage `app-routing.module.ts` en y ajoutant/modifiant les lignes suivantes :

```
…
import { HomeComponent } from "./components/home/home.component";
import { ListeComponent } from "./components/liste/liste.component";

const routes: Routes = [
      { path: '', component: HomeComponent },
      { path: 'liste', component: ListeComponent },
];
…
```

On voit bien dans ces lignes la liaison entre le **composant** et le **chemin** servant de descripteur dans le menu. Les changements suivants vont être au niveau du template du `header` à l'aide de la directive **RouterLink**.

Notez l'**absence de "/" au niveau du path** alors qu'il est présent dans la directive **RouterLink** pour la page principale.

T1.6 Insérez les liens pour le menu dans `header.component.html` pour `home` et `liste` :

```
<a [routerLink]="['/']" class="button is-light">
```

```
<a [routerLink]="['/liste']" class="button is-light">
```

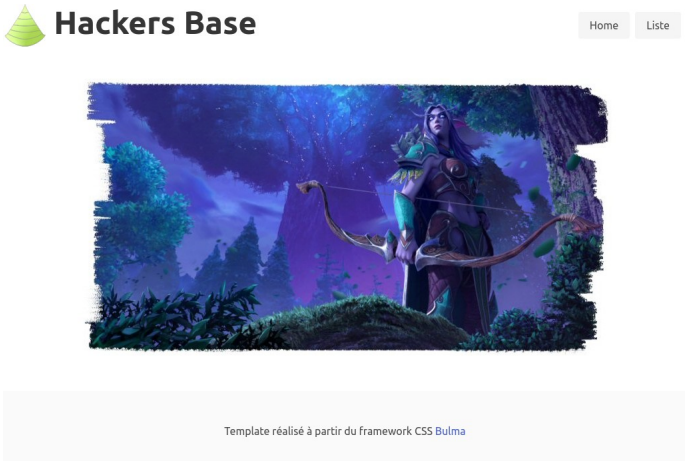
Tout fonctionnera à priori, mais il faut dire à Angular où **injecter le contenu du template** du composant lié (ou « amené ») du menu. Pour le faire, il faut utiliser une autre directive : **RouterOutlet**.

Voici le lien pour la documentation officielle : https://angular.io/api/router/RouterOutlet.

T1.7 Dans le template du composant principal `app.component.html`, **ne laissez que le code suivant**, assez facile à comprendre :

```
<app-header></app-header>

<router-outlet></router-outlet>

<app-footer></app-footer>
```

Testez, cela doit fonctionner (redémarrez bien le serveur), avec **notamment les deux liens**. Voici mon exemple :



2] Utilisation d'un modèle de données

- ***Création d'une classe***

L'application **stockera** et **affichera** les résultats des requêtes successives. Il est pertinent de créer un modèle (une classe objet) contenant les informations retournées par une requête.

Nous allons donc utiliser le concept de classe sous Angular, concept que l'on retrouve de façon similaire dans d'autres frameworks MVC.

T2.1 Créez la nouvelle classe `hacker` **après** avoir (dans un souci de structure) créé un sous-répertoire `classes` dans `app`:

```
hackers$ ng generate class classes/hacker
```

La nouvelle classe créée (vide bien sûr) se trouve au niveau du fichier `hacker.ts`. Nous ne prendrons pas toutes les informations retournées par la requête pour notre objet.

T2.2 Complétez le modèle avec les informations suivantes :

```
export class Hacker {
  constructor(
    public ip: string,
    public country_name: string,
    public region_name: string,
    public city: string
  ) { }
}
```

Ce modèle sera disponible dans l'ensemble de l'application. Par contre la gestion de celui devra passer par un autre concept : le **service**.

- ***Principes d'un service***

Le **principe premier** d'un service est de pouvoir réutiliser du code ou en d'autres termes : « stocker de la logique applicative réutilisable ». Dans cette optique, on peut définir deux caractéristiques essentielles :

(1) Le service est unique (singleton) pour toute l'application
(2) Le service est global à l'application

Exemple : *l'authentification des utilisateurs.*

Le **deuxième principe** réside dans mise en commun de données pour les composants de l'application ou en d'autres termes : « stocker ou obtenir des données communes et partagées ».

Dans cette optique, on peut définir deux caractéristiques essentielles :

(3) Le service gère la logique applicative du stockage (local ou externe)
(4) Le service permet l'accès et la modification des données par les composants

Exemple : *Un panier d'articles pour un site de commerce en ligne.*

Dans notre cas cela sera pour **partager les informations entre les composants**. Vous avez vu l'échange de données entre un composant parent et un composant enfant ; il s'agit ici d'**utilisation des données par plusieurs composants enfants** à l'aide d'un modèle de données.

- ***Création du service***

T2.3 Générez le service que l'on nommera comme le modèle, `hacker` et bien sûr dans un sous-répertoire `services` à créer :

```
hackers$ ng generate service services/hacker
```

Ce service aura pour rôles de stocker la liste des objets `hacker` et la gestion de celle-ci.

Pour pouvoir instancier un service, **Angular a besoin d'un "provider"** (littéralement : fournisseur) lui indiquant **comment produire l'instance** de ce service.

T2.4 Complétez le fichier `app.module.ts` pour l'importation du service et sa déclaration dans la section `providers` :

```
…
import { HackerService } from './services/hacker.service';

@NgModule({
…
  providers: [
    HackerService,
  ],
…
```

T2.5 Remplacez ensuite l'intégralité du code `hacker.service.ts` par celui-ci (pour faire simple) :

```
import { Injectable } from '@angular/core';
import { Hacker } from '../classes/hacker';

@Injectable({
  providedIn: 'root'
})

export class HackerService {

  private hackers: Hacker[];

  constructor() {
    this.hackers = [];
  }

  getAllHackers() {
    return this.hackers;
  }

  addHacker(hacker: Hacker) {
    this.hackers.push(hacker);
  }
}
```

Explications :

→ L'**injection de dépendances** Angular constitue un des pivots de ce framework ; le décorateur @Injectable initialise un contexte « détectable » pour une utilisation d'un service dans un autre composant (voire un autre service) ; pour de plus amples informations sur le principe de l'injection de dépendances avec Angular, voyez le lien https://cdiese.fr/angular-depency-injection/.

→ L'injection est possible au niveau d'un composant, d'un module ou global à l'application avec le paramètre `provideIn` et la désignation du module `root` ou dans la partie `providers[]` de `app.module.ts`.
→ La classe comporte une propriété : `hackers`, tableau d'objets et pour l'instant deux méthodes, l'une permettant l'ajout d'un objet et l'autre retournant la totalité du tableau (pour la liste).

- ***Utilisation du service***

Pour tester notre requête, la suite ce passe au niveau du composant `Liste`.

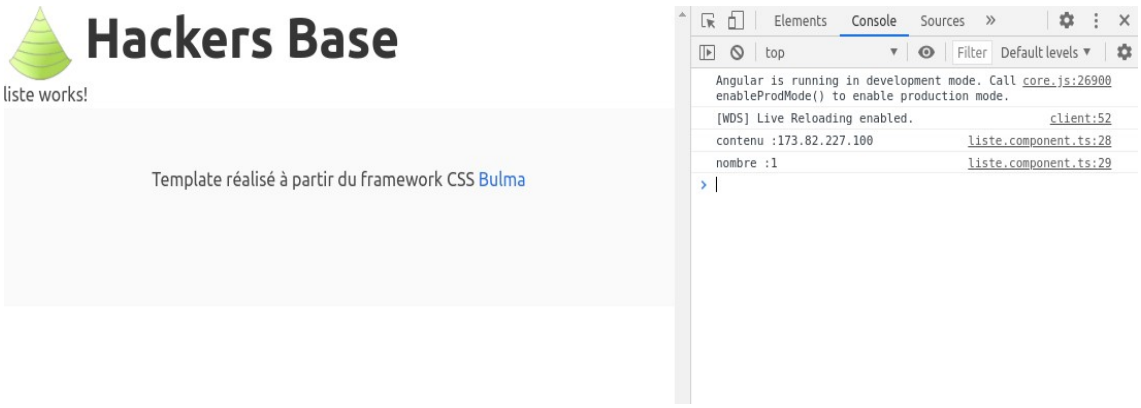
T2.6 Avec en plus des initialisations supplémentaires, intégrez au code du composant `liste.component.ts`, au moment de son initialisation, l'appel à la méthode `sendGetRequest()` du service, **injecté par ailleurs** dans le contrôleur :

```
import { Component, OnInit } from '@angular/core';
import { Hacker } from '../../classes/hacker';
import { HackerService } from '../../services/hacker.service';

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

export class ListeComponent implements OnInit {

  lesHackers: Hacker[];
  nbOflesHackers: number;

  constructor(private hackerService: HackerService) {
    this.lesHackers = this.hackerService.getAllHackers();
    this.nbOflesHackers = this.lesHackers.length;
  }

  ngOnInit(): void {
   // Lignes de code temporaires
   let testHacker = new Hacker('173.82.227.100','Etats-Unis','Californie','Los
Angeles');

   this.hackerService.addHacker(testHacker);
   this.lesHackers = this.hackerService.getAllHackers();
   this.nbOflesHackers = this.lesHackers.length;

   console.log('contenu :'+this.lesHackers[0].ip);
   console.log('nombre :'+this.nbOflesHackers);
  }
}
```

Le lancement de l'application doit faire apparaître à partir du menu `Liste`, l'objet ajouté en exemple dans la console de développement.

Comme indiqué dans le code, les lignes dans `ngOnInit()` sont ici uniquement pour vérifier le bon fonctionnement du service et du modèle. Elles seront changées par la suite.

Ce qui doit donner en visuel :


Hackers Base
liste works!
Template réalisé à partir du framework CSS Bulma
Elements
Console
Sources
top
Filter
Default levels
Angular is running in development mode. Call core.js:26900
enableProdMode() to enable production mode.
[WDS] Live Reloading enabled. client:52
contenu :173.82.227.100 liste.component.ts:28
nombre :1 liste.component.ts:29