TP Angular

Hervé Le Cornec herve.le.cornec@open-world-wide.com

L'objectif de ce TP est d'apprendre les fondamentaux les plus importants d'Angular, ceux qui seront requis dans la très grande majorité des applications, et qui sont au cœur de son fonctionnement (asynchronisme, two-way data biding, dépendences en cascade, routage, lazy loading, emport de Bootstrap et Ng-Bootstrap, service api, service store global, pipes, formulaire, …). Pour ce faire on développe une application qui utilise tous ces concepts.

Création de l'application « myapp » :

Dans une console entrez les commandes suivantes :

```
# ng new myapp --routing

……
# cd myapp
# ng serve
```

Ouvrir un onglet dans le navigateur et lui indiquer l'url **`localhost:4200`**, la nouvelle application apparaît.

A noter :

```
# ng serve --port 4561
```

ouvrira l'application sur **`localhost:4561`**

Editer **`app.component.html`** et tout supprimer sauf :

```
<h1>Hello</h1>
<router-outlet></router-outlet>
```

Enregistrer le fichier, le ng serve compile et recharge la page automatiquement.

Structure de base du filesystem

Il est nécessaire d'utiliser l'arborescence de fichiers la plus simple possible

```
app
_______header
_______pages
_______________home
_______________news
_______lib
_______________class
_______________pipes
_______________services

_______app-routing.module.ts
_______app.component.css
_______app.component.html
_______app.component.ts
_______app.module.ts
```

Création du composant header

Tout site possède un header, indépendant des pages

```
# cd src/app
# ng generate component header
CREATE src/app/header/header.component.css (0 bytes)
CREATE src/app/header/header.component.html (25 bytes)
CREATE src/app/header/header.component.spec.ts (628 bytes)
```

```
CREATE src/app/header/header.component.ts (269 bytes)
UPDATE src/app/app.module.ts (475 bytes) <<< le composant header a été déclaré
                                                dans app.module.ts
```

Préparer un container dans le fichier **header.component.html** écrire :

```
<div class="header"></div>
```

Puis déclarer la classe css dans le fichier **header.component.css**

```
.header {
	position : fixed ;
	top : 0 ;
	left : 0 ;
	z-index : 999 ;
	width : 100 %;
	height : 60px ;
	background : blue ;
}
```

Intégrer le header dans le composant **app.component.html** :

```
<app-header></app-header>
<div class="appContainer">
	<h1>Hello</h1>
	<router-outlet></router-outlet>
</div>
```

Le titre « Hello » n'est plus visible, car il est situé sous le header. Il faut donc encapsuler le contenu visible du **app.component.html** dans un div de classe **appContainer** définie ainsi :

```
.appContainer : {
	margin-top : 60px ; /*hauteur du header*/
}
```

Enregistrer tout et regarder le résultat.

Création de la page d'accueil

Il s'agit de créer un composant home

```
# mkdir pages
# cd pages
# ng generate component home
```

Le composant home sera une fois encore embarqué automatiquement dans **`app.module.ts`**.

Embarquement de la page d'accueil dans le routing.module

- Déplacer l'import du composant home depuis **`app.module.ts`**, vers **`app-routing.module.ts`**
- Renseigner la route vers home dans **`app-routing.module.ts`** :

```
const routes: Routes = [
	{
		path: '',
		component: HomeComponent
	},
	{
		path : '**',
		redirectTo : '',
		pathMatch: 'full'
	}
];
```

Enregistrer tout et observer que le nouveau composant s'affiche dans la balise **router-outlet.**

Création de la page « news »

On pratique comme pour la page home

```
# cd pages
# ng generate component news
```

Le composant home sera une fois encore embarqué automatiquement dans **`app.module.ts`**. On le déplace dans le **`app-routing.module.ts`**, comme on l'a fait pour la page home, et on y déclare une route « news ».

```
	...
	{
		path: 'news',
		component: NewsComponent
	},
	...
```

Par le navigateur on peut désormais afficher **`localhost:4200/news`** et vérifier que tout fonctionne.

Créer une navigation dans le header

- Modifier **`header.componen.html`** :

```
<div class="header">
  <h1>Header de l'application</h1>
  <ul>
    <li>
      <a [routerLink]="['news']">News</a>
    </li>
    <li>
        <a routerLink="">Home</a>
    </li>
  </ul>
</div>
```

- Embarquer le RouterModule dans **`app.module.ts`**

```
import { RouterModule } from '@angular/router';
...
  imports: [
    ...
    RouterModule
  ],
```

- Vérifier qu'on navigue d'une page à l'autre en cliquant sur les liens

Créer un service d'api récupérant les informations sur un user

- Créer un service api

```
# cd app
# mkdir lib
# cd lib
# mkdir services
# cd services
# ng generate service api
```

- Comme ce service sera global à l'application, le déclarer dans **`app.module.ts`**

```
...
import { ApiService } from './lib/services/api.service';
...
providers : [
	ApiService
]
...
```

- Créer un fichier **`user.json`** dans le répertoire **`src/asset`**

```
{
    "id": 123,
    "nom": "Doe",
    "prenom": "John",
    "code" : "abc123def456",
    "role": "user",
```

```
    "dateNaissance": "23/05/2003"
}
```

- Importer HttpClienModule dans **`app.module.ts`** :

```
...
import { HttpClientModule }  from '@angular/common/http';
...
imports: [
       ...
       HttpClientModule
]
...
```

- Importer HttpClient dans **`api.service.ts`**

```
...
import { HttpClient }  from '@angular/common/http';
...
constructor(
    private http: HttpClient
) {
...
```

- Ajouter une méthode « getUser » au service qui récupère les données user

```
getUser = () => {

    this.http.get('assets/user.json').subscribe({
      next: result => { console.log(result); },
      error: err => { console.error('Error: ' + err); },
      complete: () => { console.log('call ended'); }
    });

  }
```

Créer un store global à l'application pour y stocker les données globales

- Créer un service store :

```
# cd lib/services
# ng generate service store
```

- Modifier le service en déclarant qu'on y stokera des données et un user

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

@Injectable({
  providedIn: 'root'
})
export class StoreService {

  data: any = {}
  user: any = {}

  constructor() { }
}
```

- Déclarer le service dans **`app.module.ts`**, comme il a été fait pour ApiService
- Importer StoreService dans ApiService, afin de stocker le résultat d'un call api

```
import { Injectable } from '@angular/core';
import { HttpClient }  from '@angular/common/http';
import { StoreService }  from './store.service';

@Injectable({
  providedIn: 'root'
})
export class ApiService {

  constructor(
    private http: HttpClient,
    private store: StoreService
  ) {
```

```
  }

  getUser = () => {

    setTimeout( () => {
      this.http.get('assets/user.json').subscribe({
        next: result => {
          this.store.user = result;
        },
        error: err => { console.error('Error: ' + err); },
        complete: () => { console.log('-- call ended --'); }
      });
    },2000); //on impose un setTimeout pour simuler un réseau très lent

  }
}
```

Utiliser le StoreService dans le header

- Importer le service dans le composant **`header.component.ts`**

```
import { Component, OnInit } from '@angular/core';
import { ApiService } from '../lib/services/api.service';
import { StoreService } from '../lib/services/store.service';

@Component({
  selector: 'app-header',
  templateUrl: './header.component.html',
  styleUrls: ['./header.component.css']
})
export class HeaderComponent implements OnInit {

  user: any;

  constructor(
    private api: ApiService,
    public store: StoreService
  ) {
     this.api.getUser();
   }

  ngOnInit() {}
}
```

- Utiliser directement le store pour afficher le nom et le prénom de l'utilisateur dans le header, et donc insérer le code suivant dans **`header.component.html`**

```
  <div class="bonjour">
    Bonjour
    <span [innerHtml]="store.user?.prenom"></span>  <span
[innerHtml]="store.user?.nom"></span>
  </div>
```

Afficher les caractéristiques du user aussi sur la page « news »

- Importer le StoreService dans le composant **`news.component.ts`**

```
import { Component, OnInit } from '@angular/core';
import { StoreService } from '../../lib/services/store.service';

@Component({
  selector: 'app-news',
  templateUrl: './news.component.html',
  styleUrls: ['./news.component.css']
})
export class NewsComponent implements OnInit {

  constructor(
    private store: StoreService
  ) { }

  ngOnInit() {
  }

}
```

- Modifier le template **`news.component.html`**

```
<p>
  The user
  <span [innerHtml]="store.user?.prenom"></span>
  <span [innerHtml]="store.user?.nom"></span>
  has id: <span [innerHtml]="store.user?.id"></span>
</p>
```

Installer sur la home page un bouton qui déclenche une modal Bootstrap

- Installer bootstrap (voir https://ng-bootstrap.github.io) en lançant la commande

```
# npm i --save @ng-bootstrap/ng-bootstrap
```

- Importer Bootstrap dans le fichier **`src/style.css`** en y ajoutant

```
@import "../node_modules/bootstrap/dist/css/bootstrap.css"
```

- Créer un composant partageable « modal », donc embarqué dans son module

```
# cd /lib
# mkdir modules
# cd modules
# ng g module modal
# ng g component modal
CREATE src/app/lib/modules/modal/modal.component.css (0 bytes)
CREATE src/app/lib/modules/modal/modal.component.html (24 bytes)
CREATE src/app/lib/modules/modal/modal.component.spec.ts (621 bytes)
CREATE src/app/lib/modules/modal/modal.component.ts (265 bytes)
UPDATE src/app/lib/modules/modal/modal.module.ts (255 bytes) <<< le composant
ModalComponent est embarqué dans le ModalModule
```

- Exporter le composant ModalComponent depuis le ModalModule :

```
...
  exports: [
    ModalComponent
  ]
```

- Importez le ModalModule dans le module important aussi le composant home page, c'est à dire **`app-routing.module.ts`**

```
import { ModalModule } from './lib/modules/modal/modal.module';
...
  imports: [
    ...
    ModalModule
  ],
...
```

- Afficher le composant modal sur la home page, c'est à dire modifier **`home.component.html`**

```
<p>
  home works!
</p>
<app-modal></app-modal>
```

- Vérifier que tout fonctionne bien.

- Allez sur https://ng-bootstrap.github.io et cliquez sur Components > Modal. Choisissez une modal et recopiez ses codes HTML et Typescript.

Afficher un message dans la modal

- Modifier **`home.component.html`** pour indiquer au composant modal qu'elle lui envoie un message:

```
<app-modal [message] = "message"></app-modal>
```

- Modifier le **`home.component.ts`** pour déclarer la variable message :

```
  message: string;

  constructor() {
    this.message = 'hello les amis !';
  }
```

Afficher en plus le nom de l'utilisateur dans la modal

- On importe le StoreService dans **`home.component.ts`**

```
import { StoreService } from '../../lib/services/store.service';
...
  constructor(
    private store: StoreService
  ) { }
```

- On modifie **`home.component.html`** pour qu'il envoie le user du store au composant modal :

```
<app-modal [user] = "store.user"></app-modal>
```

- On modifie le **`modal.component.ts`** pour lui ajouter la capacité à recevoir des informations

```
import { Component, OnInit, Input } from '@angular/core';
...
export class ModalComponent implements OnInit {
  @Input() user: any;
  constructor(...
```

- On insère le code suivant dans le template du composant modal, **`modal.component.html`** :

```
 <p>Bonjour <span [innerHtml]="user.prenom"></span> <span
[innerHtml]="user.nom"></span> !</p>
```

Afficher le jour de la semaine de naissance de l'utilisateur, grâce à un pipe

- Modifier **`news.component.html`** en y inscrivant

```
<p>
  <span [innerHtml]="store.user?.prenom"></span> <span
[innerHtml]="store.user?.nom"></span>
  (id: <span [innerHtml]="store.user?.id"></span>) est né un
  {{ store.user.dateNaissance | weekDayOfBirth }}
</p>
```

- Créons le filtre attendu « weekDayOfBirth », qui sera automatiquement déclaré dans **`app.module.ts`**

```
# cd lib
# mkdir pipes
# cd pipes
#ng g pipe weekDayOfBirth
CREATE src/app/lib/pipes/week-day-of-birth.pipe.spec.ts (222 bytes)
CREATE src/app/lib/pipes/week-day-of-birth.pipe.ts (217 bytes)
UPDATE src/app/app.module.ts (1022 bytes) <<< emport du pipe dans app.module
```

- Modifions le pipe **`week-day-of-birth.pipe.ts`**

```
 ...
  transform(value: any, args?: any): any {
```

```
    let weekDay: string = ‘jour indéfini’;

    if(value && value.match(/[0-9]{2}\/[0-9]{2}\/[0-9]{4}/)) {
      const arr = value.split('/').map((x: string) => parseInt(x, 10));
      const laDate = new Date(arr[2], arr[1], arr[0], 0, 0, 0, 1);
      switch(laDate.getDay()) {
        case 0: weekDay = 'dimanche'; break;
        case 1: weekDay = 'lundi'; break;
        case 2: weekDay = 'mardi'; break;
        case 3: weekDay = 'mercredi'; break;
        case 4: weekDay = 'jeudi'; break;
        case 5: weekDay = 'vendredi'; break;
        case 6: weekDay = 'samedi'; break;
      }

    }

    return weekDay;
  }
...
```

Rendre un pipe partageable en l’installant dans un module

Un pipe n’est reconnu que dans le module (et ses sous modules) dans lequel il est importé. Le lazy loading (voir plus bas) casse cet héritage, et un pipe reconnu dans le routage principal ne le sera plus dans une route montée en lazy loading. Il faudrait réimporter le pipe dans la lazy route, mais ce n’est pas possible de monter un pipe dans deux modules. Il faut donc utiliser le module qui emporte le pipe, mais pas le pipe directement.

- Créer un module WeekDayOfBirthModule

```
# cd lib/pipes
# ng g module week-day-of-birth
# mv week-day-of-birth.pipe.* week-day-of-birth <<< on déplace tous les fichiers concernant le pipe dans le répertoire de son nom.
```

- Importez et exportez le pipe dans le module en modifiant **`week-day-of-birth.module.ts`**

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { WeekDayOfBirthPipe } from './week-day-of-birth.pipe';

@NgModule({
  declarations: [
    WeekDayOfBirthPipe
  ],
  imports: [
    CommonModule
  ],
  exports: [
    WeekDayOfBirthPipe
  ]
})
export class WeekDayOfBirthModule { }
```

- Décharger **`app.module.ts`** de l’emport de WeekDayOfBirthPipe, et importez le module du pipe WeekDayOfBirthModule dans **`app-routing.module.ts`**

```
import { WeekDayOfBirthModule } from './lib/pipes/week-day-of-birth/week-day-of-birth.module';
...
  imports: [
    ...
    WeekDayOfBirthModule
  ],
...
```

- Vérifier que le résultat est le même que sans module d’emport du pipe.

Afficher « Loading ... » sur la home page tant que l'api n'a pas retourné de résultats

- Dans **`app-routing.module.ts`**, importer les CommonModule :

```
...
import { CommonModule } from '@angular/common';
...
  imports: [
    ...
    CommonModule
  ],
...
```

- Dans **`api.service.ts`**, déclarer un status dans le store qui fasse état du loading en cours de chaque api appelée

```
  getUser = () => {

    this.store.status.loading++;

    setTimeout( () => {
      this.http.get('assets/user.json').subscribe({
        next: result => { this.store.user = result; },
        error: err => { console.error('Error: ' + err); },
        complete: () => {
          console.log('-- call ended --');
          this.store.status.loading--;
        }
      });
    },2000); //on impose un setTimeout pour simuler un réseau très lent

  }
```

- Déclarer la branche status.loading dans **`store.service.ts`**

```
  status: any = {
    loading: 0
  };
```

- Comme on importe déjà le store dans home.component, nous n'avons qu'à modifier le template **`home.component.html`**

```
<div *ngIf="store.status.loading">
    <p>
      Loading ...
    </p>
</div>
<div *ngIf="!store.status.loading">
  <p>
    Bienvenue !
  </p>
  <app-modal [user] = "store.user"></app-modal>
</div>
```

- Le Loading ne s'efface que si store.status.loading = 0 ;

Boucle ngFor sur la page des news

- Dans assets ajoutons le fichier **`documents.json`**, contenant une liste (tableau) de données

```
[
    {
```

```
        "id": 123,
        "titre": "Tintin au Tibet",
        "resume": "Un jeune reporter se lance à la recherche du Yeti avec le capitaine
Haddock",
        "auteur": "Hergé"
    },
    {
        "id": 345,
        "titre": "Asterix le gaulois",
        "resume": "Les aventures d'un guerrier Gaulois doté d'une potion magique rendant
invincible",
        "auteur": "Gosigny"
    },
    {
        "id": 678,
        "titre": "le Capital",
        "resume": "Etude économique sur le travail et le capital",
        "auteur": "K. Marx"
    },
    {
        "id": 910,
        "titre": "Mécanique Analytique",
        "resume": "Les bases de la physique classique",
        "auteur": "J.L. Lagrange"
    },
    {
        "id": 112,
        "titre": "La byciclette Bleue",
        "resume": "Je sais pas, je l'ai pas lu",
        "auteur": "Deforge"
    }

]
```

- Créer le service api qui ira lire ces données, comme nous avons lu le user, dans ApiService :

```
  getDocuments = () => {
    this.store.status.loading++;
    setTimeout( () => {
      this.http.get('assets/documents.json').subscribe({
        next: result => { this.store.data.documents = result; },
        error: err => { console.error('Error: ' + err); },
        complete: () => {
          console.log('-- call ended --');
          this.store.status.loading--;
        }
      });
    },2000); //on impose un setTimeout pour simuler un réseau très lent
  }
```

A noter qu'à la place des 2 fonctions getUser et getDocument, il serait possible de ne fabriquer qu'une seule fonction « getData(url, storePlace) » qui reçoit en entrée l'url (user.json ou document.json), et la place dans le store (store.user ou store.data.documents). Tout dépend de la politique API que vous choisirez.

- Importer le service api dans **`news.component.ts`**

```
import { ApiService } from '../../lib/services/api.service';
...
  constructor(
    private store: StoreService,
    private api: ApiService
  ) {

    if(!this.store.data.documents) { //On ne se sert de l'API que si le store est vide
      this.api.getDocuments();
    }
  }
...
```

- Modifier **`news.component.html`** pour afficher la liste des documents :

```
<ul>
  <li *ngFor="let item of store.data.documents">
    <p class="titre">{{item.titre}}</p>
    <p class="resume">{{item.resume}}</p>
  </li>
</ul>
```

Lazy loading de la page « news »

- Se placer dans le répertoire de la page « news » et créer un module pour cette page :

```
# cd pages/news
# ng g module news
# mv news/news.module.ts . <<< on place le module directement dans le répertoire news.
# rm -r  news <<< on détruit le répertoire devenu inutile
```

- Se placer dans le répertoire de la page « news » et créer un module de routage :

```
# cd pages/news
# ng g module news-routing
# mv news-routing/news-routing.module.ts . <<< on place le module de routage directement
                                              dans le répertoire news.
# rm -r  news-routing <<< on détruit le répertoire devenu inutile
```

!! Note : relancez VSCode, qui s'embrouille avec ces commandes manuelles

- Embarquer le NewsRoutingModule dans le NewsModule en modifiant **`news.module.ts`**

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { NewsRoutingModule } from './news-routing.module';

@NgModule({
  declarations: [],
  imports: [
    CommonModule,
    NewsRoutingModule
  ]
})
export class NewsModule { }
```

- Modifier **`news-routing.module.ts`** pour embarquer NewsComponent et WeekDayOfBirthModule, qui contient le pipe utile sur la page news :

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { RouterModule, Routes } from '@angular/router';
import { NewsComponent } from './news.component';
import { WeekDayOfBirthModule } from '../../lib/pipes/week-day-of-birth/week-day-of-
birth.module';

const routes: Routes = [
  {
     path: '',
     component: NewsComponent
  }
];


@NgModule({
  declarations: [
    NewsComponent
  ],
  imports: [
    CommonModule,
    RouterModule.forChild(routes),
    WeekDayOfBirthModule
  ]
})
export class NewsRoutingModule { }
```

!! Notez le routing « forChild » au lieu de « forRoot », nécessaire pour une route secondaire en lazy loading.

- Modifier **`app-routing.module.ts`** pour lui indiquer que la route « news » sera en lazy loading,et qu'il n'a plus besoin d'embarquer NewsComponent

```
// import { NewsComponent } from './pages/news/news.component';
...
/*{
    path: 'news',
    component: NewsComponent
  },*/
  { path: 'news',
      loadChildren:'./pages/news/news.module#NewsModule',
  },
...
```

Page profil avec formulaire, two-way data binding

- Créer la page profil

```
# cd pages
# ng g component profil
CREATE src/app/pages/profil/profil.component.css (0 bytes)
CREATE src/app/pages/profil/profil.component.html (25 bytes)
CREATE src/app/pages/profil/profil.component.spec.ts (628 bytes)
CREATE src/app/pages/profil/profil.component.ts (269 bytes)
UPDATE src/app/app.module.ts (1043 bytes) <<<  le composant est automatiquement embarqué
dans app.module.ts
```

- On déplace l'import du composant ProfilComponent depuis **`app.module.ts`**, vers **`app-routing.module.ts`**, et on déclare la nouvelle route

```
import { ProfilComponent } from './pages/profil/profil.component';
...
const routes: Routes = [
  ...
  {
    path: 'profil',
    component: ProfilComponent
  },
...
```

- Modifier la navigation du header pour atteindre la page profil sur un clic, **`header.component.html`** :

```
...
    <li>
      <a [routerLink]="['profil']">Profil</a>
    </li>
...
```

- Vérifier que la page fonctionne

- Embarquer les modules standard Angular FormsModule et ReactiveFormsModule dans **`app.module.ts`**

```
import { FormsModule, ReactiveFormsModule } from '@angular/forms';
...
  imports: [
    ...
    FormsModule,
    ReactiveFormsModule
  ],
...
```

- Déclarer le formulaire dans le composant profil, **`profil.component.ts`**

```
import { Component, OnInit } from '@angular/core';
import { FormGroup, FormControl } from '@angular/forms';
import { StoreService } from '../../lib/services/store.service';

@Component({
  selector: 'app-profil',
  templateUrl: './profil.component.html',
  styleUrls: ['./profil.component.css']
```

```
})
export class ProfilComponent implements OnInit {

  myform: FormGroup;

  constructor(
       private store: StoreService
  ) { }

  ngOnInit() {
    this.myform = new FormGroup({
      nom : new FormControl(),
      prenom: new FormControl()
    });
  }

  onSubmit() {
    console.log(this.myform.value);

    // On met à jour le store.user avec les nouvelles données du formulaire
    this.store.user.prenom = this.myform.value.prenom;
    this.store.user.nom = this.myform.value.nom;
  }

}
```

- Mettre en place le formulaire HTML dans **`profil.component.html`**

```
<div class="formulaire">
  <form [formGroup]="myform" (ngSubmit)="onSubmit()">

    <div class="form-group">
      <label>Prénom</label>
      <input  type="text"
              class="form-control"
              placeholder="indiquez votre prénom usuel"
              formControlName="prenom"
              required
      />
      <div class="form-group">
        <label>Nom</label>
        <input  type="text"
                class="form-control"
                placeholder="indiquez votre nom de famille"
                formControlName="nom"
                required
        />
      </div>
      </div>
    <button type="submit" class="btn btn-primary">Valider</button>
  </form>
</div>
```

- Le fichier css associé, **`profil.component.css`**

```
.formulaire  {
    margin: auto;
    width: 50%;
}
```

- Vérifier que tout fonctionne, et qu'une modification des nom et prénom du user se répercute instantanément dans le header et les autres pages (popin sur home, affichage sur news).