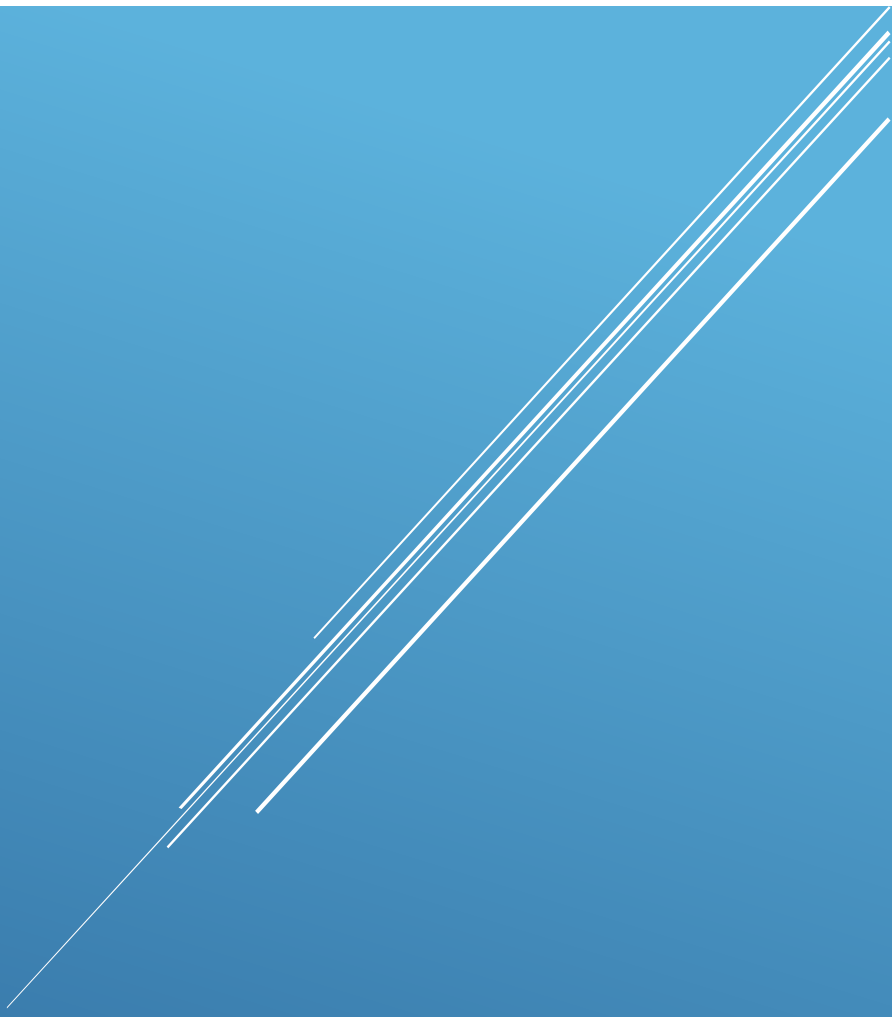


Adrien Vossough

Partie I : création d'un projet avec Angular CLI

1 – Créer un répertoire

2 – Se placer grâce à la console de Windows dans ce répertoire

3 – taper dans la console : ng g new exercice

- Ceci va créer un nouveau projet Angular
- Selon l'ordinateur, cela peut prendre du temps.

4 – Toujours dans la **console**, se placer dans le répertoire exercice (cd c:\chemin\vers\exercice\) et taper :

ng serve

- Cela permet d'exécuter un serveur http qui va pouvoir fournir l'application

6 – Dans un **navigateur**, aller à l'adresse : http://localhost:4200

- Le message suivant devrait s'afficher : app works !

Partie II : création des modules et composants des pages

0 – Créer un **répertoire site** dans **src/app**

1 – Avec la **console**, se placer à la racine du projet et taper :

ng g module site\home
ng g module site\about
ng g module site\films

- Chacun des 3 modules représentent une page de notre application

2 – Ajouter ces 3 modules dans le module principal qui se trouve dans le répertoire **app** et se nomme **app.module.ts** :

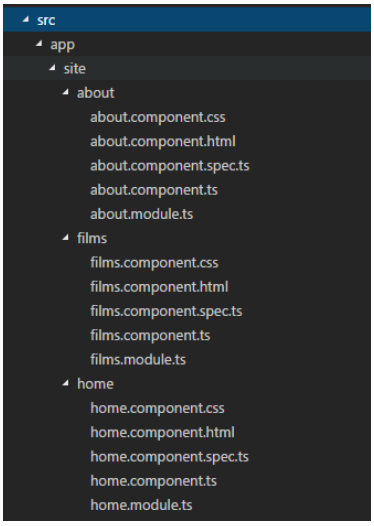
app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { HttpModule } from '@angular/http';

import { AppComponent } from './app.component';

import { HomeModule } from './site/home/home.module';
import { AboutModule } from './site/about/about.module';
import { FilmsModule } from './site/films/films.module';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpModule,

    HomeModule,
    AboutModule,
    FilmsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

3 – Ajouter les composants qui serviront à afficher chacune des pages :

ng g component site\home
ng g component site\about
ng g component site\films

- Nous devons avoir la structure ci-dessous



4 – Nous devons retirer les instructions ajoutées par Angular CLI dans le module principal

app.module.ts

```
import { HomeModule } from './site/home/home.module';
import { AboutModule } from './site/about/about.module';
import { FilmsModule } from './site/films/films.module';
import { HomeComponent } from './site/home/home.component';
import { AboutComponent } from './site/about/about.component';
import { FilmsComponent } from './site/films/films.component';


@NgModule({
 declarations: [
   AppComponent,
   HomeComponent,
   AboutComponent,
   FilmsComponent,
 ],
 imports: [
   BrowserModule,
   FormsModule,
   HttpModule,
   HomeModule,
   AboutModule,
   FilmsModule
 ],
```

5 – Ajouter les composants dans les modules de chacune des pages qui ont été créé plus tôt.

- Chacune des pages a un module et un composant

home.module.ts

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { HomeComponent } from './home.component';

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

about.module.ts

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { AboutComponent} from './about.component';

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

films.module.ts

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { FilmsComponent} from './films.component';

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

Partie III : Mise en place des Routes

1 –Indiquer au module principal que nous allons utiliser des routes.

app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { HttpModule } from '@angular/http';

import { RouterModule } from '@angular/router';

import { AppComponent } from './app.component';
import { HomeModule } from './site/home/home.module';
import { AboutModule } from './site/about/about.module';
import { FilmsModule } from './site/films/films.module';

@NgModule({
  declarations: [
    AppComponent,
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpModule,

    RouterModule.forRoot([]),

    HomeModule,
    AboutModule,
    FilmsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

2 – Ajouter les routes aux modules enfants

- Le module racine ne contiendra pas les routes ceci pour séparer au maximum les composants/modules
- Le module racine chargera les routes indiquées par ses enfants.
- Pour configurer des routes dans des modules autre que le module racine nous devons utiliser la méthode **forchild** :
    - Exemple : **RouterModule.forChild([{ path: '*ROUTE_VERS_COMPOSANT*', component: *COMPOSANT_A_CHARGER* }])**

home.module.ts

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';

import { RouterModule } from '@angular/router';

import { HomeComponent } from './home.component';

@NgModule({
 imports: [
  CommonModule,
  RouterModule.forChild([
   { path: '', component: HomeComponent }
  ])
 ],
 declarations: [
  HomeComponent
 ]
})
export class HomeModule { }
```

about.module.ts

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';

import { RouterModule } from '@angular/router';

import { AboutComponent } from './about.component';

@NgModule({
 imports: [
  CommonModule,
  RouterModule.forChild([
   { path: 'about', component: AboutComponent }
  ])
 ],
 declarations: [
  AboutComponent
 ]
})
export class AboutModule { }
```

films.module.ts

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';

import { RouterModule } from '@angular/router';

import { FilmsComponent } from './films.component';

@NgModule({
 imports: [
  CommonModule,
  RouterModule.forChild([
   { path: 'films', component: FilmsComponent }
  ])
 ],
 declarations: [
  FilmsComponent
 ]
})
export class FilmsModule { }
```

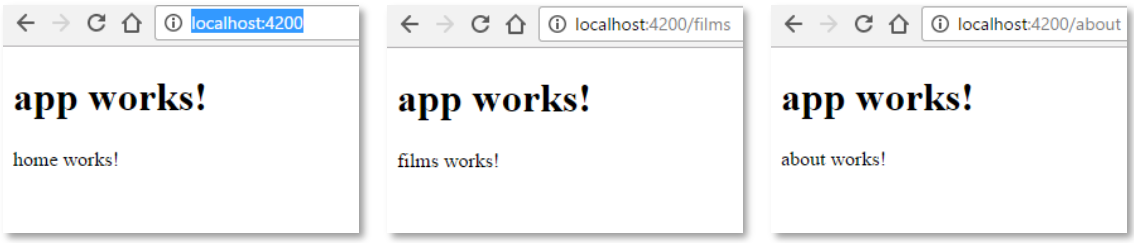
3 – Indiquer le point où se chargera les templates des composants des pages

- Lorsque le module Root charge les sous-modules, il prend en compte les routes indiquées dans celles-ci.

app.component.html

```
<div class="container">
  <h1>{{title}}</h1>
</div>
<router-outlet></router-outlet>
```

4 – Tester les routes :



5 – Reprendre ce qui a été fait et ajouter un module qui gère les pages pour les erreurs (404, 401, etc.)

- Exemple de configuration du module pour les erreurs d'URL

errors.module.ts

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { RouterModule } from '@angular/router';

import { ErrorsComponent } from './errors.component';

@NgModule({
 imports: [
  CommonModule,
  RouterModule.forChild([
   { path: '404', component: ErrorsComponent }
  ])
 ],
 declarations: [
  ErrorsComponent
 ]
})
export class ErrorsModule { }
```

- Il est possible de rediriger toutes les adresses inexistantes vers les pages d'erreurs.
- Dans ce dernier cas, le module erreur doit être le dernier à être chargé par le module root !!!

errors.module.ts

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { RouterModule } from '@angular/router';

import { ErrorsComponent } from './errors.component';

@NgModule({
 imports: [
  CommonModule,
  RouterModule.forChild([
   { path: '404', component: ErrorsComponent },
   { path: '**', redirectTo: '/404'}
  ])
 ],
 declarations: [
  ErrorsComponent
 ]
})
export class ErrorsModule { }
```

Partie IV : Ajout de Bootstrap

1 – Ajouter la librairie Bootstrap (fichiers CSS seulement) dans la balise **<head></head>** du fichier **index.html**

- Cela va permettre de designer facilement les différents éléments et de ne pas se préoccuper du CSS

Index.html

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>Projet</title>
  <base href="/">

  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
  <!-- Latest compiled and minified CSS -->
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"  integrity="sha384-
BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u" crossorigin="anonymous">

  <!-- Optional theme -->
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap-theme.min.css"  integrity="sha384-
rHyoN1iRsVXV4nD0JutlnGaslCJuC7uwjduW9SVrLvRYooPp2bWYgmgJQIXwl/Sp" crossorigin="anonymous">
</head>
<body>
  <app-root>Loading...</app-root>
</body>
</html>
```

Partie V : Ajout d'un menu

1 – Dans la console taper : **ng g module site\shared**

2 – Dans la console taper : **ng g component site\shared\navmenu**

- Le répertoire \shared\ contiendra les éléments partagés

3 – Comme dans la partie II section 4, nous devons **retirer** l'ajout automatique du **composant** au module Root (**app.module.ts**)

4 – Ajouter **shared.module.ts** au module Root.

app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { HttpModule } from '@angular/http';
import { RouterModule } from '@angular/router';

import { AppComponent } from './app.component';

import { SharedModule } from './site/shared/shared.module';

import { HomeModule } from './site/home/home.module';
import { AboutModule } from './site/about/about.module';
import { FilmsModule } from './site/films/films.module';
import { ErrorsModule } from './site/errors/errors.module';
import { NavmenuComponent} from './site/shared/navmenu/navmenu.component';

@NgModule({
  declarations: [
    AppComponent,
    NavmenuComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpModule,
    RouterModule.forRoot([]),

    SharedModule,

    HomeModule,
    AboutModule,
    FilmsModule,
    ErrorsModule,
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

4 – Ajouter le composant au module shared.module.ts

- Nous ajoutons aussi : **exports: [NavmenuComponent]**
- **exports: [composant, module, …]** Indique que le module exporte des éléments vers le module parent. Le module parent pourra donc utiliser directement les éléments exportés.

shared.module.ts

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';

import { NavmenuComponent } from './navmenu/navmenu.component';

@NgModule({
 imports: [
  CommonModule
 ],
 declarations: [
  NavmenuComponent
 ],
 exports: [NavmenuComponent]
})

export class SharedModule { }
```

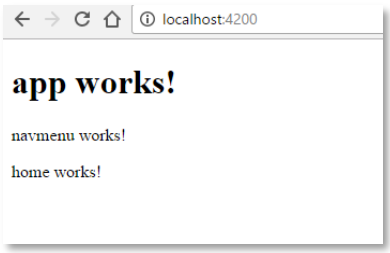
5 – Ajouter le composant menu au template principal.

- Ainsi le menu s'affichera sur toutes les pages

app.component.html

```
<h1>
 {{title}}
</h1>

<app-navmenu></app-navmenu>

<router-outlet></router-outlet>
```

6 – Vérifier que le composant navmenu s'affiche :



7 – Importer RouterModule à shared.module.ts

- RouterModule embarque des directives pour faire des menus

shared.module.ts

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { RouterModule } from '@angular/router';
import { NavmenuComponent } from './navmenu/navmenu.component';

@NgModule({
  imports: [
    CommonModule,
    RouterModule
  ],
  declarations: [
    NavmenuComponent
  ],
  exports: [NavmenuComponent]
})
export class SharedModule { }
```

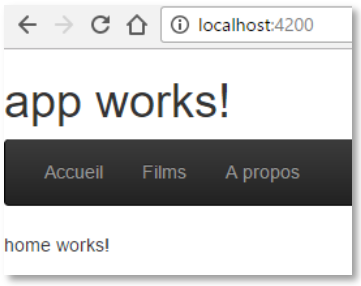
8 – Modifier le template **navmenu.component.html** pour y ajouter un menu avec les liens vers les différentes pages

- Les classes utilisées ci-dessous sont des classes Bootstrap qui permettent de mettre facilement en forme un menu

navmenu.component.html

```
<div class="container">

  <nav class="navbar navbar-inverse">
    <ul class="nav navbar-nav">
      <li>
        <a [routerLink]="['/']">Accueil</a>
      </li>
      <li>
        <a [routerLink]="['/films']">Films</a>
      </li>
      <li>
        <a [routerLink]="['/about']">A propos</a>
      </li>
    </ul>
  </nav>

</div>
```

9 – Vérifier que le menu fonctionne



Partie VI : Ajout d'un formulaire

1 – Créer un nouveau composant : **ng g component site\films\searchform**

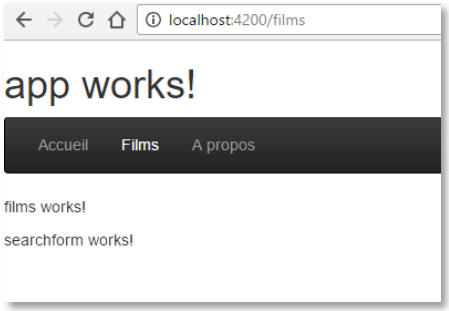
- Ce composant est un formulaire pour rechercher des films
- Angular CLI a ajouté ce composant au module films.module.ts

2 – Ajouter le formulaire au template **films.component.html**

films.component.html

```
<p>
  films works!
</p>

<app-searchform></app-searchform>
```

3 – Vérifier que le résultat est le suivant :



4 – Créer un formulaire dans **searchform.component.html.**
Ce formulaire doit contenir un champ de **type texte** et un autre de type **number**.
Les valeurs du champ de type number devront être comprises entre **1900** et **2017.**

searchform.component.html

```
<div class="container">

  <form class="form-inline well well-sm">

    <div class="form-group col-md-5">
      <label>Veuillez entrer un titre de film</label>
      <input type="texte" class="form-control" placeholder="Titre du film">
    </div>

    <div class="form-group col-md-5">
      <label>Veuillez choisir une année</label>
      <input type="number" min="1900" max="2017" step="1" value="2016" class="form-control">
    </div>

    <button type="submit" class="btn btn-primary">Rechercher</button>
  </form>

</div>
```

5 – Ajouter au module **films.module.ts** les module **FormsModule, ReactiveFormsModule** du package **@angular/forms** et le service **FormBuilder**

films.module.ts

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { RouterModule } from '@angular/router';
import { FormsModule, FormBuilder, ReactiveFormsModule} from '@angular/forms';
import { FilmsComponent } from './films.component';
import { SearchformComponent } from './searchform/searchform.component';

@NgModule({
 imports: [
  CommonModule,
  FormsModule,
  ReactiveFormsModule,
  RouterModule.forChild([
   { path: 'films', component: FilmsComponent }
  ])
 ],
 declarations: [
  FilmsComponent,
  SearchformComponent
 ] ,
 providers: [
  FormBuilder
 ]

})
```

6 – Ajouter à **searchform.component.ts** : **import { FormBuilder, FormGroup } from '@angular/forms';**

- **FormBuilder** permet de créer plus facilement des « **FormControl** » (objet lié à un champ) et des « **FormGroup** » (objet lié à un formlaire).

7 – En utilisant l'injection par le constructeur de **searchform.component.ts**, récupérer un objet de type **FormBuilder.**

8 – Grace à l'objet FormBuilder (qui est un **service**), fabriquer un objet de type **FormGroup** qui contiendra deux **FormControl**, l'un pour le titre, l'autre pour l'année qui aura pour valeur par défaut : 2016.

searchform.component.ts

```
import { Component, OnInit } from '@angular/core';
import { FormBuilder, FormGroup } from '@angular/forms';

@Component({
 selector: 'app-searchform',
 templateUrl: './searchform.component.html',
 styleUrls: ['./searchform.component.css']
})
export class SearchformComponent implements OnInit {
 searchForm: FormGroup;

 constructor( fb: FormBuilder ) {
  this.searchForm = fb.group({
   title: '',          // la chaîne de caractères est la valeur par défaut du champ
   year: ''
  });
 }

 ngOnInit() {
 }
}
```

9 – Lier le formulaire **searchform.component.html** au composant **searchform.component.ts** en utilisant l'objet de type **FormGroup**.

10 – Ajouter une méthode (fonction) **startSearch** au composant qui sera appelé à chaque soumission du formulaire.
La méthode **startSearch** affiche les valeurs du formulaire en utilisant l'objet **FormGroup**.

searchform.component.ts

```
import { Component, OnInit } from '@angular/core';
import { FormBuilder, FormGroup } from '@angular/forms';

@Component({
 selector: 'app-searchform',
 templateUrl: './searchform.component.html',
 styleUrls: ['./searchform.component.css']
})
export class SearchformComponent implements OnInit {
 searchForm: FormGroup;

 constructor(fb: FormBuilder) {
  this.searchForm = fb.group({
   title: '',
   year: ''
  });
 }

 startSearch(){
  console.log("recherche lancée");
 }

 ngOnInit() {
 }
}
```

searchform.component.html

```
<div class="container">

 <form (ngSubmit)="startSearch()" [formGroup]="searchForm" class="form-inline well well-sm" >

  <div class="form-group col-md-5">
   <label>Veuillez entrer un titre de film</label>
   <input formControlName="title" type="texte"  class="form-control" placeholder="Titre du film">
  </div>

  <div class="form-group col-md-5">
   <label>Veuillez choisir une année</label>
   <input formControlName="year" type="number" min="1900" max="2017" step="1" value="2016" class="form-control">
  </div>

  <button type="submit" class="btn btn-primary">Rechercher</button>
 </form>

</div>
```

11 – Tester si la fonction **startSearch** est enclenchée lors du clic sur le bouton rechercher.

Partie VII : Création d'un service émettant une requête http

0 – Créer un répertoire **services** dans le répertoires **app\site\films**

1 – Dans la console taper : **ng g service site\films\services\searchmovie**

- Ce service est une classe qui permettra de faire différentes requêtes pour la recherche de films.

2 – Ajouter le service au module : **films.module.ts**

3 – Ajouter le module HttpModule au module **films.module.ts**

- Rappel : **import { HttpModule } from '@angular/http';**

films.module.ts

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { RouterModule } from '@angular/router';
import { HttpModule } from '@angular/http';
import { FormsModule, FormBuilder, ReactiveFormsModule } from '@angular/forms';
import { SearchmovieService } from './services/searchmovie.service';

import { FilmsComponent } from './films.component';
import { SearchformComponent } from './searchform/searchform.component';

@NgModule({
 imports: [
  CommonModule,
  FormsModule,
  ReactiveFormsModule,
  HttpModule,
  RouterModule.forChild([
   { path: 'films', component: FilmsComponent }
  ])
 ],
 declarations: [
  FilmsComponent,
  SearchformComponent
 ],
 providers: [
  FormBuilder,
  SearchmovieService
 ]
})
export class FilmsModule { }
```

4 – Ajouter une méthode **search(action: (data: Object) , title: string, year: number = 0): void** à **SearchmovieService**
Cette méthode doit faire une requête de type **get** en utilisant le service http fournit par Angular.
L'URL est la suivant : **`http://www.omdbapi.com/?t=${title}&y=${year}&plot=full`**

- Cette méthode récupérera les titres de film ainsi que l'année du film
- La valeur par défaut de l'année est 0, si aucune année est fournie. Dans ce cas, l'URL sera : **`http://www.omdbapi.com/?t=${title}&plot=full`**
- **action: (data: Object)** indique qu'une fonction pouvant recevoir 1 paramètre de type Object est attendu

searchmovie.service.ts

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

@Injectable()
export class SearchmovieService {
 constructor(private http: Http) {

 }

 search(action: (data: Object) , title: string, year: number = 0): void {
  let results = {};
  let y = year ? `&y=${year}` : '';

  this.http.get(`http://www.omdbapi.com/?t=${title}${y}&plot=full`).subscribe(
   (response) => {
    action( response.json() );
   });
 }
}
```

Fonction de rappel. Cette fonction doit pouvoir recevoir un objet.

Voir opérateur ternaire :
si year = !0 alors y = `&y=${year}` sinon y = ''

Requête get qui renvoit un **Observable**

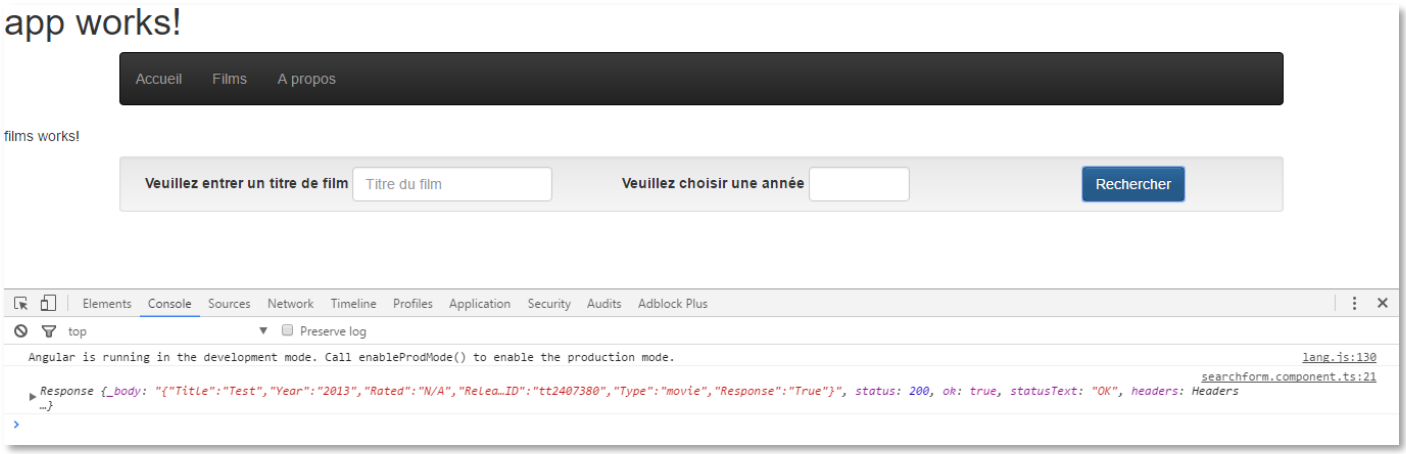
La fonction de rappel sera exécutée dès que le serveur aura répondu.
Celle-ci recevra les informations en paramètre sous forme d'un objet.

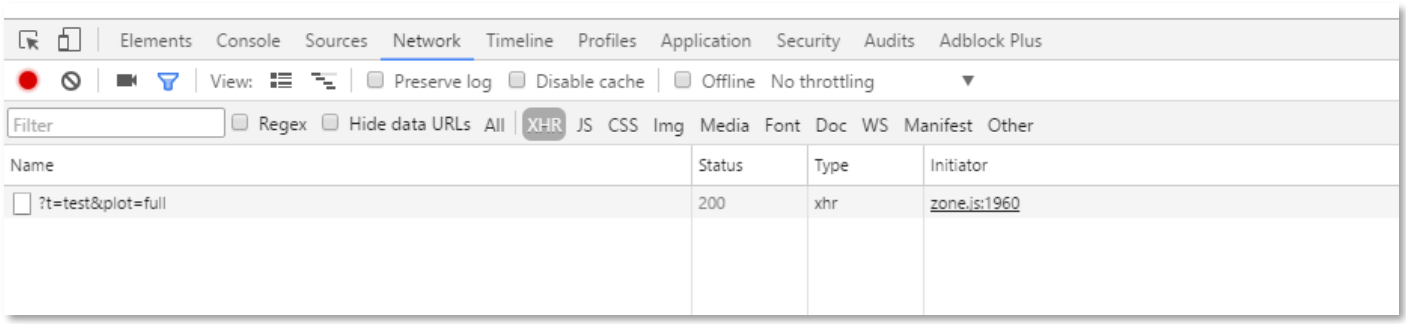
5 – Injecter le service à **searchform.component.ts**
Lancer la recherche en appelant la méthode **search** de ce service lors de la soumission du formulaire, pour cela utiliser la méthode **startSearch** de **searchform.component.ts**

searchform.component.ts

```
import { Component, OnInit } from '@angular/core';
import { FormBuilder, FormGroup } from '@angular/forms';
import { SearchmovieService } from '../services/searchmovie.service';
@Component({
 selector: 'app-searchform',
 templateUrl: './searchform.component.html',
 styleUrls: ['./searchform.component.css']
})
export class SearchformComponent implements OnInit {
 searchForm: FormGroup;

 constructor(private searchService: SearchmovieService, fb: FormBuilder) {
  this.searchForm = fb.group({
   title: '',
   year: ''
  });
 }

 startSearch() {
  let action = (data:Object) =>{
   console.log(data);
  } ;
  this.searchService.search(action,"test");
 }
ngOnInit() { } }
```

6 – Tester si la requête se fait bien en lançant une recherche.
Regarder dans la console du navigateur si une réponse apparaît.





7 – Fournir les valeurs entrées par l'utilisateur dans le formulaire au service.
Utiliser pour cela l'attribut **searchForm: FormGroup**

searchform.component.ts méthode startSearch()

```
startSearch() {
  let action = (data:Object) =>{
    console.log(data);
  };

  this.searchService.search(action, this.searchForm.value.title, this.searchForm.value.year );
 }
```

Partie VIII : Ajouter des validateurs à un formulaire

1 – Dans le fichier **searchform.component.ts**, importer **FormControl** et **Validators** du package **@angular/forms** et mettre des validations aux champs de recherche :

- Le **titre** est requis, aura **30 caractères maximum** et ne devra avoir que les caractères suivant : **a-z, A-Z, 0-9, espace, virgule et point**
- La **date** sera un entier **optionnel** compris **entre 1900** et **2017**
- Modifier l'initialisation de **this.searchForm = fb.group(…)** dans le constructeur pour que cela soit pris en compte
- Créer un **validateur personnalisé** pour valider les dates entre 1900 et 2017, celui-ci se trouvera dans un nouveau fichier **form.validators.ts** placé dans le **répertoire searchform**. Il faudra l'importer dans **searchform.component.ts**
- https://regex101.com/ permet de tester les expressions régulières (regex)

form.validators.ts

```
import { AbstractControl, ValidatorFn } from '@angular/forms';

export class FormValidators {

  static integerBetween(min: number, max: number): ValidatorFn {
    return (c: AbstractControl) => {

      if (!Number.isInteger(c.value)) {
        // si la valeur n'est pas un entier
        return {
          integer: {
            valid: false
          }
        };
      } else if ((c.value < min) || (c.value > max)) {
        // si la valeur est en dehors des limites
        return {
          limit: {
            valid: false
          }
        };
      }
      // si tout va bien
      return null;
    }
  }
}
```

static integerBetween(min: number, max: number): ValidatorFn

Ceci est un validateur paramétrable.

Il reçoit la valeur minimum et maximum

Il retourne un validateur qui lui-même recevra le FormControler.

Si tout se passe bien, le validateur **doit** retourner **null**.

Si la valeur n'est pas valide, il retourne un objet avec l'erreur définie.

Une méthode **static** indique qu'elle est accessible directement sans instancier la classe.

Nous pouvons mettre plusieurs validateurs dans la classe **FormValidators**

searchform.component.ts

```
import { Component, OnInit } from '@angular/core';
import { FormBuilder, FormGroup, FormControl, Validators } from '@angular/forms';
import { SearchmovieService } from '../services/searchmovie.service';
import { FormValidators } from './form.validators';

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

export class SearchformComponent implements OnInit {
 searchForm: FormGroup;
 title: FormControl;
 year: FormControl;

 constructor(private searchService: SearchmovieService, fb: FormBuilder) {
  let titlePattern = '[a-zA-Z0-9,\. ]+';
  let yearPattern = '[0-9]{4}';

  this.title = fb.control('', [Validators.required, Validators.maxLength(30), Validators.pattern(titlePattern)]);
  this.year = fb.control('', [Validators.pattern(yearPattern), FormValidators.integerBetween(1900, 2017)]);

  this.searchForm = fb.group({
   title: this.title,
   year: this.year
  });
 }

 startSearch() {
  let title = this.title.valid ? this.title.value : null;
  let year = this.year.valid ? this.year.value : null;
  let action = (data: Object) => {
   console.log(data);
  };

  if(title) {
   this.searchService.search(action, title, year);
  }
 }

 ngOnInit() {
 }
}
```

Partie IX : Création d'une directive pour l'affichage des résultats

1 – Créer une directive : **ng g directive site\films\directives\list**

- Cette directive servira à afficher les résultats
- Angular CLI a ajouté cette directive à : **films.module.ts**

2 – Retoucher **searchform.component.ts** pour qu'il enregistre les résultats dans un **attribut privé : results** et les erreurs dans un **attribut privé errors**

searchform.component.ts

```
import { Component, OnInit } from '@angular/core';
import { FormBuilder, FormGroup, FormControl, Validators } from '@angular/forms';
import { SearchmovieService } from '../services/searchmovie.service';
import { FormValidators } from './form.validators';

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

export class SearchformComponent implements OnInit {
 private searchForm: FormGroup;
 private title: FormControl;
 private year: FormControl;
 private results: any;
 private errors: string;

 constructor(private searchService: SearchmovieService, fb: FormBuilder) {
  let titlePattern = '[a-zA-Z0-9,\. ]+';
  let yearPattern = '[0-9]{4}';

  this.title = fb.control('', [Validators.required, Validators.maxLength(30), Validators.pattern(titlePattern)]);
  this.year = fb.control('', [Validators.pattern(yearPattern), FormValidators.integerBetween(1900, 2017)]);

  this.searchForm = fb.group({
   title: this.title,
   year: this.year
  });
 }

 startSearch() {
  let title = this.title.valid ? this.title.value : null;
  let year = this.year.valid ? this.year.value : null;
  let that = this;  // référence vers l'objet courant : voir portée des variables et closures(fermetures)

  let action = (data: Object) => {
   if(data['Error']) {
    that.errors = data['Error'];
    that.results = '';
   } else {
    that.errors = '';
    that.results = data;
   }
  };

  if(title) {
   this.searchService.search(action, title, year);
  }
 }

 ngOnInit() {
 }
}
```

3 – Renommer le **sélecteur** de la **directive** en **movieList** (attention à la casse).
Ajouter une **entrée** à la directive pour qu'elle puisse recevoir des données sous forme d'une liste d'objets.
L'entrée se nommera aussi **movieList** et enregistrera la valeur dans **un attribut privé**, nommé **_list**

list.directive.ts

```
import { Directive, Input } from '@angular/core';

@Directive({
 selector: '[movieList]'
})
export class ListDirective {
 private _list: Object;

 constructor() {
 }

 @Input()
 set movieList(value) {
  console.log("valeur recu par la directive" + value);
 }
}
```

- Il est maintenant possible de recevoir des valeurs dans la directive

4 – Appeler la directive depuis le template **searchform.component.html**

- L'appel se fera sur un élément
- La **directive recevra** la valeur de l'attribut **results** du composant **SearchformComponent**
- Nous exécuterons la directive que s'il n'y a pas d'erreurs et que des résultats sont présents

searchform.component.html

```
<div class="container">

 <form (ngSubmit)="startSearch()" [formGroup]="searchForm" class="form-inline well well-sm">

  <div class="form-group col-md-5">
   <label>Veuillez entrer un titre de film</label>
   <input formControlName="title" type="texte" class="form-control" placeholder="Titre du film">
  </div>

  <div class="form-group col-md-5">
   <label>Veuillez choisir une année</label>
   <input formControlName="year" type="number" min="1900" max="2017" step="1" value="" class="form-control">
  </div>

  <button type="submit" class="btn btn-primary">Rechercher</button>
 </form>

</div>

<div class="container">
 <div *ngIf="errors" class="alert alert-danger">
  <strong>Aucun résultat !</strong> {{errors}}.
 </div>

 <div *ngIf="results" class="list-group" [movieList]="results">
 </div>
</div>
```

Un bloc d'erreur va s'afficher si l'attribut errors n'est pas vide.

La directive ne s'exécutera seulement si errors est vide

5 – Modifier la directive pour qu'elle affiche les données

- **ElementRef** contient un attribut **nativeElement** qui est l'élément **DOM** de base.

list.directive.ts

```
import { Directive, Input, ElementRef } from '@angular/core';

@Directive({
 selector: '[movieList]'
})
export class ListDirective {
 private _list: Object;
 private _el: any;


 constructor(el: ElementRef) {
  this._el = el.nativeElement;
 }

 @Input()
 set movieList(movie) {
  let temp = `<a class="list-group-item list-group-item-action">`;
  temp += `Titre: ${movie.Title} Année:${movie.Year} Réalisateur: ${movie.Director}` ;
  temp += '</a>';
  this._el.innerHTML = temp;
 }
}
```

Pour l'exercice nous utilisons cette façon de faire pour rester simple mais pour aller plus loin, il vaudra mieux à l'avenir suivre ce fonctionnement :
https://angular.io/docs/ts/latest/guide/structural-directives.html

6 – Tester le résultat