Rafiaa Jnaieh
2019

ANGULAR

Introduction

Angular est un framework Javascript côté client qui permet de réaliser des applications de type "Single Page Application". Il est basé sur le concept de l'architecture MVC (Model View Controller) qui permet de séparer les données, les vues et les différentes actions que l'on peut effectuer . il utilise le TypeScript plutôt que JavaScript

Qu'est-ce que le TypeScript ? Pourquoi l'utiliser ?

C'est un sur-ensemble de JavaScript qui est justement transcompilé en JavaScript pour être compréhensible par les navigateurs. Il ajoute des fonctionnalités extrêmement utiles :

- le typage strict
- les fonctions dites lambda ou arrow, permettant un code plus lisible
- les classes et interfaces

1- Préparez l'environnement de développement

Installez les outils

ANGULAR/ CLI "Command Line Interface" d'Angular est l'outil qui vous permet d'exécuter des scripts pour la création, la structuration et la production d'une application Angular.

NODE.JS

NPM NPM est un package manager qui permet l'installation d'énormément d'outils et de libraries dont vous aurez besoin pour tout type de développement

2- Créez votre premier projet

ng new mon-premier-projet

cd mon-premier-projet

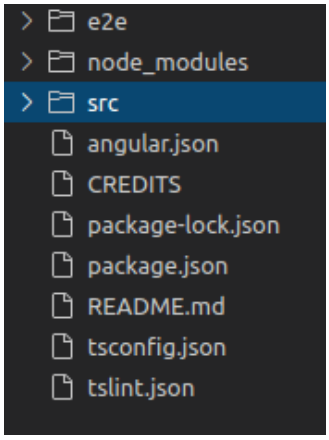
ng serve –open

Découvrez la structure du code

Le dossier **e2e** est généré pour les tests end-to-end,

le dossier **node_modules** contient toutes les dépendances pour votre application : les fichiers source Angular et TypeScript, par exemple.

Le dossier **src** où vous trouverez tous les fichiers sources pour votre application.



Rafiaa Jnaieh – Rafiaajnaieh@hotmail.com

```ts
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.scss']
})
export class AppComponent {
  title = 'my awesome app';
}
```

Créez un component

ng generate component mon-premier

```
installing component
  create src/app/mon-premier/mon-premier.component.scss
  create src/app/mon-premier/mon-premier.component.html
  create src/app/mon-premier/mon-premier.component.spec.ts
  create src/app/mon-premier/mon-premier.component.ts
  update src/app/app.module.ts
```

il a mis à jour le fichier **app.module.ts**

```ts
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';


import { AppComponent } from './app.component';
import { MonPremierComponent } from './mon-premier/mon-premier.component';


@NgModule({
  declarations: [
    AppComponent,
    MonPremierComponent
  ],
  imports: [
    BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

3- Gérez des données dynamiques

String interpolation **{{variable/méthode}}**

```ts
export class AppareilComponent implements OnInit {

  appareilName: string = 'Machine à laver';

  constructor() { }
```

```html
<li class="list-group-item">
  <h4>Appareil : {{ appareilName }}</h4>
</li>
```

Property binding

```
<button class="btn btn-success" [disabled]="!isAuth">Tout allume</button>
```

```
export class AppComponent {
  isAuth = false;

  constructor() {
    setTimeout(
      () => {
        this.isAuth = true;
      }, 4000
    );
  }
}
```

Event binding

```
<button class="btn btn-success"
        [disabled]="!isAuth"
        (click)="onAllumer()">Tout allumer</button>
```

```
onAllumer() {
    console.log('On allume tout !');
}
```

Two-way binding **[(ngModel)]="variable"**

```
<input type="text" class="form-control" [(ngModel)]="appareilName">
```

Propriétés personnalisées @Input

il serait intéressant de faire en sorte que chaque instance d' ComponentXX ait un nom différent qu'on puisse régler depuis l'extérieur du code. Pour ce faire, il faut utiliser le décorateur **@Input**

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

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

  @Input() appareilName: string;
```

```
<app-appareil appareilName="Machine à laver"></app-appareil>
<app-appareil appareilName="Frigo"></app-appareil>
<app-appareil appareilName="Ordinateur"></app-appareil>
```

```
<ul class="list-group">
    <app-appareil [appareilName]="appareilOne"></app-appareil>
    <app-appareil [appareilName]="appareilTwo"></app-appareil>
    <app-appareil [appareilName]="appareilThree"></app-appareil>
</ul>
```

```
export class AppComponent {
  isAuth = false;

  appareilOne = 'Machine à laver';
  appareilTwo = 'Frigo';
  appareilThree = 'Ordinateur';

  constructor() {
```

Rafiaa Jnaieh – Rafiaajnaieh@hotmail.com

4- Structurez le document avec des Directives

Les directives structurelles *ngIf /*ngFor

***ngIf="condition"**

```
<div style="width:20px;height:20px;background-color:red;"
    *ngIf="appareilStatus === 'éteint'"></div>
<h4>Appareil : {{ appareilName }} -- Statut : {{ getStatus() }}</h4>
```

***ngFor="let obj of myArray"**

```
<ul class="list-group">
  <app-appareil  *ngFor="let appareil of appareils"
                 [appareilName]="appareil.name"
                 [appareilStatus]="appareil.status"></app-appareil>
</ul>
```

```
appareils = [
  {
    name: 'Machine à laver',
    status: 'éteint'
  },
  {
    name: 'Frigo',
    status: 'allumé'
  },
  {
    name: 'Ordinateur',
    status: 'éteint'
  }
];
```

Les directives par attribut NgModel/ngStyle/ngClass

ngStyle

```
<h4 [ngStyle]="{color: getColor()}">Appareil : {{ appareilName }} -- Statut : {{ getStatus() }}</h4>
```

```
1 getColor() {
2     if(this.appareilStatus === 'allumé') {
3       return 'green';
4     } else if(this.appareilStatus === 'éteint') {
5       return 'red';
6     }
7 }
```

ngClass prend un objet clé-valeur, mais cette fois avec la classe à appliquer en clé, et la condition en valeur

```
1 <li [ngClass]="{'list-group-item': true,
2                 'list-group-item-success': appareilStatus === 'allumé',
3                 'list-group-item-danger': appareilStatus === 'éteint'}">
4   <div style="width:20px;height:20px;background-color:red;"
5       *ngIf="appareilStatus === 'éteint'"></div>
```

4- Modifiez les données en temps réel avec les Pipes

Les pipes prennent des données en input, les transforment, et puis affichent les données modifiées dans le DOM

Utilisez et paramétrez les Pipes

DatePipe

```
<p>Mis à jour : {{ lastUpdate | date: 'short' }}</p>
```

Mis à jour : 2/23/18, 5:25 PM

```
<p>Mis à jour : {{ lastUpdate | date: 'yMMMMEEEEd' }}</p>
```

Mis à jour : 2018FebruaryFriday23

async

C'est un cas particulier mais extrêmement utile dans les applications Web, car il permet de gérer des données asynchrones

Il faut ajouter AsyncPipe en début de chaîne pour dire à Angular d'attendre l'arrivée des données avant d'exécuter les autres pipes :

```
<p>Mis à jour : {{ lastUpdate | async | date: 'yMMMMEEEEd' | uppercase }}</p>
```

```
lastUpdate = new Promise((resolve, reject) => {
    const date = new Date();
    setTimeout(
      () => {
        resolve(date);
      }, 2000
    );
  });
```

5- Améliorez la structure du code avec les Services

ng generate service User

Injection et instances

un service doit être injecté, et le niveau choisi pour l'injection est très important. Il y a trois niveaux possibles pour cette injection :

- dans **AppModule**: ainsi, la même instance du service sera utilisée par tous les components de l'application et par les autres services ;
- dans **AppComponent** : comme ci-dessus, tous les components auront accès à la même instance du service mais non les autres services ;

•dans un autre component : le component lui-même et tous ses enfants (c'est-à-dire tous les components qu'il englobe) auront accès à la même instance du service, mais le reste de l'application n'y aura pas accès.

```
@NgModule({
  declarations: [
    AppComponent,
    MonPremierComponent,
    AppareilComponent
  ],
  imports: [
    BrowserModule,
    FormsModule
  ],
  providers: [
    AppareilService
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

- injecter ce service dans **XXModule** en l'ajoutant à l'array **providers**
- déclarer comme argument dans le constructeur de **XXComponent**

```
constructor(private appareilService: AppareilService) {
    setTimeout(
      () => {
        this.isAuth = true;
      }, 4000
    );
  }
```

6- Gérez la navigation avec le Routing

Créez des routes

déclare les routes dans **XXModule**

```
import { Routes } from '@angular/router';

const appRoutes: Routes = [
  { path: 'appareils', component: AppareilViewComponent },
  { path: 'auth', component: AuthComponent },
  { path: '', component: AppareilViewComponent }
];
```

```
imports: [
    BrowserModule,
    FormsModule,
    RouterModule.forRoot(appRoutes)
],
```

dire à Angular où vous souhaitez afficher les components dans le template lorsque l'utilisateur navigue vers la route en question. On utilise la balise **<router-outlet></router-outlet>**

```
<div class="col-xs-12">
  <router-outlet></router-outlet>
</div>
```

Naviguez avec les routerLink

routerLink

```
<ul class="nav navbar-nav">
    <li routerLinkActive="active"><a routerLink="auth">Authentification</a></li>
    <li routerLinkActive="active"><a routerLink="appareils">Appareils</a></li>
</ul>
```

Naviguez avec le Router

router.navigate

```
constructor(private authService: AuthService, private router: Router) { }
```

```
onSignIn() {
    this.authService.signIn().then(
      () => {
        console.log('Sign in successful!');
        this.authStatus = this.authService.isAuth;
        this.router.navigate(['appareils']);
      }
    );
}
```

Paramètres de routes **path/:id**

créer la route dans **AppModule**

```
const appRoutes: Routes = [
  { path: 'appareils', component: AppareilViewComponent },
  { path: 'appareils/:id', component: SingleAppareilComponent },
  { path: 'auth', component: AuthComponent },
  { path: '', component: AppareilViewComponent }
];
```

injecter **ActivatedRoute** dans SingleAppareilComponent, importé depuis @angular/router, afin de récupérer le fragment **id** de l'URL

```
constructor(private appareilService: AppareilService,
            private route: ActivatedRoute) { }
```

```
ngOnInit() {
    const id = this.route.snapshot.params['id'];
    this.name = this.appareilService.getAppareilById(+id).name;
    this.status = this.appareilService.getAppareilById(+id).status;
}
```

Rq :il ne faut pas oublier d'utiliser + avant id dans l'appel pour caster la variable comme nombre.

service

```
getAppareilById(id: number) {
    const appareil = this.appareils.find(
      (s) => {
        return s.id === id;
      }
    );
    return appareil;
}
```

```
<ul class="list-group">
  <app-appareil  *ngFor="let appareil of appareils; let i = index"
                 [appareilName]="appareil.name"
                 [appareilStatus]="appareil.status"
                 [index]="i"
                 [id]="appareil.id"></app-appareil>
</ul>
```

```
<h4 [ngStyle]="{color: getColor()}">Appareil : {{ appareilName }} -- Statut : {{ getStatus() }}</h4>
<a [routerLink]="[id]">Détail</a>
```

Redirection redirectTo

```
const appRoutes: Routes = [
  { path: 'appareils', component: AppareilViewComponent },
  { path: 'appareils/:id', component: SingleAppareilComponent },
  { path: 'auth', component: AuthComponent },
  { path: '', component: AppareilViewComponent },
  { path: 'not-found', component: FourOhFourComponent },
  { path: '**', redirectTo: 'not-found' }
];
```

Guards

C’est un service qu'Angular exécutera au moment où l'utilisateur essaye de naviguer vers la route sélectionnée.

Ce service implémente l'interface canActivate et donc doit contenir une méthode du même nom qui prend les arguments ActivatedRouteSnapshot et RouterStateSnapshot

Sauvegardez ce fichier dans le dossier sous le nom **auth-guard.service.ts**

```typescript
import { ActivatedRouteSnapshot, CanActivate, Router, RouterStateSnapshot } from '@angular/router';
import { Observable } from 'rxjs/Observable';
import { AuthService } from './auth.service';
import { Injectable } from '@angular/core';

@Injectable()
export class AuthGuard implements CanActivate {

  constructor(private authService: AuthService,
              private router: Router) { }

  canActivate(
    route: ActivatedRouteSnapshot,
    state: RouterStateSnapshot): Observable<boolean> | Promise<boolean> | boolean {
    if(this.authService.isAuth) {
      return true;
    } else {
      this.router.navigate(['/auth']);
    }
  }
}
```

Dans la méthode canActivate , vérifier l'état de l'authentification dans AuthService. Si l'utilisateur est authentifié, la méthode renverra true, permettant l'accès à la route protégée. Sinon, retourner false, ou rediriger l'utilisateur vers la page d'authentification

Maintenant, si vous essayez d'accéder à /appareils sans être authentifié, vous êtes automatiquement redirigé vers /auth.

7- Observez les données avec RxJS

Observables

un Observable est un objet qui émet des informations auxquelles on souhaite réagir.

À cet Observable, on associe un Observer — un bloc de code qui sera exécuté à chaque fois que l'Observable émet une information. L'Observable émet trois types d'information : des données, une erreur, ou un message complete.

On a un Observable, il faut l'observer ! Pour cela, on utilisera la fonction **Subscribe()**

```typescript
import { Observable } from 'rxjs/Rx'
import 'rxjs/add/observable/interval';
```

```
ngOnInit() {
    const counter = Observable.interval(1000);
    counter.subscribe(
      (value) => {
        this.secondes = value;
      },
      (error) => {
        console.log('Uh-oh, an error occurred! : ' + error);
      },
      () => {
        console.log('Observable complete!');
      }
    );
}
```

Subscriptions

quand vous utilisez des Observables personnalisés, il est vivement conseillé de stocker la souscription dans un objet **Subscription**

```
export class AppComponent implements OnInit, OnDestroy {

  secondes: number;
  counterSubscription: Subscription;

  ngOnInit() {
    const counter = Observable.interval(1000);
    this.counterSubscription = counter.subscribe(
      (value) => {
        this.secondes = value;
      },
      (error) => {
        console.log('Uh-oh, an error occurred! : ' + error);
      },
      () => {
        console.log('Observable complete!');
      }
    );
  }

  ngOnDestroy() {
    this.counterSubscription.unsubscribe();
  }
}
```

Subjects

l'utilisation d'un Subject pour gérer la MAJ des appareils permettra de mettre en place un niveau d'abstraction afin d'éviter des bugs potentiels avec la manipulation de données.

```
import { Subject } from 'rxjs/Subject';

export class AppareilService {

  appareilsSubject = new Subject<any[]>();
```

```
emitAppareilSubject() {
    this.appareilsSubject.next(this.appareils.slice());
  }

switchOnAll() {
    for(let appareil of this.appareils) {
      appareil.status = 'allumé';
    }
    this.emitAppareilSubject();
}

switchOffAll() {
    for(let appareil of this.appareils) {
      appareil.status = 'éteint';
      this.emitAppareilSubject();
    }
}
```

```
ngOnInit() {
  this.appareilSubscription = this.appareilService.appareilsSubject.subscribe(
    (appareils: any[]) => {
      this.appareils = appareils;
    }
  );
  this.appareilService.emitAppareilSubject();
}
```

Pourquoi est-ce important ?

Si les données sont mises à jour par une autre partie de l'application, il faut que l'utilisateur voie ce changement sans avoir à recharger la page. Il va de même dans l'autre sens : un changement au niveau du view doit pouvoir être reflété par le reste de l'application sans rechargement.

Opérateurs

Un opérateur est une fonction qui se place entre l'Observable et l'Observer (la Subscription, par exemple), et qui peut filtrer et/ou modifier les données reçues avant même qu'elles n'arrivent à la Subscription.

Rafiaa Jnaieh – Rafiaajnaieh@hotmail.com

```
const nums = of(1, 2, 3 , 4 , 5 , 6);
// Create a function that accepts an Observable.
const squareOddVals = pipe(
  filter((n: number) => n % 2 !== 0),
  map(n => n * n)
);
// Create an Observable that will run the filter
//and map functions
const squareOdd = squareOddVals(nums);
// Subscribe to run the combined functions
squareOdd.subscribe(x => console.log(x));
```

```
import { of } from 'rxjs';
import { scan } from 'rxjs/operators';
/*
appliquer une fonction à chaque élément émis par un observable,
séquentiellement, et émettre chaque valeur successive
*/
const source = of(1, 2, 3 , 4);
// basic scan example, sum over time starting with zero
const example = source.pipe(scan((acc, curr) => acc + curr, 0));
const subscribe = example.subscribe(val => console.log(val));
// output: 1,3,6,10
```

```
import { Subject } from 'rxjs';
import { scan } from 'rxjs/operators';

const subject = new Subject();
//scan example building an object over time
const example = subject.pipe(
  scan((acc, curr) => Object.assign({}, acc, curr), {})
);
//log accumulated values
const subscribe = example.subscribe(val =>
  console.log('Accumulated object:', val)
);
//next values into subject, adding properties to object
// {name: 'Joe'}
subject.next({ name: 'Joe' });
// {name: 'Joe', age: 30}
subject.next({ age: 30 });
```

8- les Forms - méthode template

importer FormsModule dans AppModule

```
<form (ngSubmit)="onSubmit(f)" #f="ngForm">
  <input type="text" id="name" class="form-control" name="name" ngModel>
  <button type="submit">Enregistrer</button>
</form>
```

Créez le formulaire

```
onSubmit(form: NgForm) {
    console.log(form.value);
}
```

ngModel sans parenthèses ni crochets. Pour signaler à Angular que vous souhaitez enregistrer ce contrôle

(ngSubmit) , vous recevez cette soumission et exécutez la méthode onSubmit

#f est une référence locale. Vous donnez simplement un nom à l'objet sur lequel vous ajoutez cet attribut .

En générale, on ne donne pas de valeur à une référence locale : on écrit simplement **#f** ou #**my-app**

Validez les données

```
<button class="btn btn-primary" type="submit" [disabled]="f.invalid">Enregistrer</button>
```

Exploitez les données

```
onSubmit(form: NgForm) {
  const name = form.value['name'];
  const status = form.value['status'];
  this.appareilService.addAppareil(name, status);
  this.router.navigate(['/appareils']);
}
```

Construisez un formulaire réactif avec FormBuilder

```
<form [formGroup]="userForm" (ngSubmit)="onSubmitForm()">
  <div class="form-group">
    <label for="firstName">Prénom</label>
    <input type="text" id="firstName" class="form-control" formControlName="firstName">
  </div>
  <div class="form-group">
    <label for="lastName">Nom</label>
    <input type="text" id="lastName" class="form-control" formControlName="lastName">
  </div>
```

```
userForm: FormGroup;

constructor(private formBuilder: FormBuilder,
            private userService: UserService,
            private router: Router) { }

ngOnInit() {
  this.initForm();
}

initForm() {
    this.userForm = this.formBuilder.group({
      firstName: ['', Validators.required],
      lastName: ['', Validators.required],
      email: ['', [Validators.required, Validators.email]],
      drinkPreference: ['', Validators.required]
    });
}
```

Ajoutez dynamiquement des FormControl

Un `FormArray` est un array de plusieurs `FormControl`, et permet, par exemple, d'ajouter des nouveaux `controls` à un formulaire. Vous allez utiliser cette méthode pour permettre à l'utilisateur d'ajouter ses hobbies.

```
initForm() {
    this.userForm = this.formBuilder.group({
      firstName: ['', Validators.required],
      lastName: ['', Validators.required],
      email: ['', [Validators.required, Validators.email]],
      drinkPreference: ['', Validators.required],
      hobbies: this.formBuilder.array([])
    });
}
```

Modifiez ensuite `onSubmitForm()` pour récupérer les valeurs, si elles existent (sinon, retournez un array vide) :

```
onSubmitForm() {
    const formValue = this.userForm.value;
    const newUser = new User(
      formValue['firstName'],
      formValue['lastName'],
      formValue['email'],
      formValue['drinkPreference'],
      formValue['hobbies'] ? formValue['hobbies'] : []
    );
    this.userService.addUser(newUser);
    this.router.navigate(['/users']);
}
```

Afin d'avoir accès aux `controls` à l'intérieur de l'array, pour des raisons de typage strict liées à TypeScript, il faut créer une méthode qui retourne `hobbies` par la méthode `get()` sous forme de `FormArray` (`FormArray` s'importe depuis `@angular/forms`) :