



PARTIE 3



JEREMY PARIS

Développeur AngularJS & Angular dans la Web Mobile
Squad

Organisateur de COP Angular Michelin †

Organisateur de COP JS CGI

Formateur et apprenant en Angular



LES FORMULAIRES

LES FORMULAIRES

- Aspect crucial dans une application web
- Des formulaires peuvent être complexe : la gestion des données, les changements qui peuvent affecter une autre page, la validation des données, l'affichage des erreurs

NGFORM

- Angular fournit un outillage pour gérer cela : *NgForm*
- Directive *@angular/forms*
- Dès l'import de FormsModule, les *form* par défaut deviennent des formulaires Angular

Remarque : Vous pouvez utiliser le `ng-no-form` pour éviter que le formulaire ne soit considéré comme un `ngform` mais plutôt comme un formulaire HTML5

NGSUBMIT

- On peut écouter la directive sur l'évènement *ngSubmit* qui notifie de la soumission du formulaire
- On place le handler d'évènement *ngSubmit* sur l'élément *form*

EXAMPLE

```
<form (ngSubmit)="addCharacter()">
  <div>
    <label>Name</label>
    <input name="name" [(ngModel)]="characterToAdd.name">
  </div>
  <div>
    <label>Is a female ?</label>
    <input type="checkbox" name="isFemale" [(ngModel)]="characterToAdd.isFemale">
  </div>
  <div *ngIf="state3" style="display: flex">
    <label>House</label>
    <div *ngIf="state3.houses">
      <select class="form-control" required [(ngModel)]="characterToAdd.house">
        <option *ngFor="let house of state3.houses" [value]="house">{{house.name}}</option>
      </select>
    </div>
  </div>
  <button type="submit">ADD</button>
</form>
```

EXAMPLE (TYPESCRIPT)

```
addCharacter() {  
    console.log('add a character');  
    console.log(this.characterToAdd);  
    this.characters.push(this.characterToAdd);  
}
```


REACTIVE FORMS

ReactiveFormsModule :

- Favorise la gestion explicite des données
- Permet de faire des contrôles plus fin
- Autre directive que ng..., commence par form comme la directive *formGroup* pour le formulaire et *formControlName* pour les inputs
- On passe une valeur et pas une expression
- Reactive forms sont synchrones

REACTIVE FORMS

- Permet de construire les formulaires programmatiquement
- Facilement testable, car ce n'est que du code
- *Input* ou *Select* représentés par un FormControl
- *FormControl* est la brique élémentaire d'un formulaire, qui encapsule l'état du champ et sa valeur

REACTIVE FORMS

Exemple :

```
createForm() {  
  const form = new FormGroup({  
    name: new FormControl(),  
    isFemale: new FormControl()  
  });  
}
```

Utilisation de la classe *FormBuilder* qui possède des méthodes utilitaires pour créer des contrôles et des groupes

REACTIVE FORMS

Example :

```
<h2>Sign up</h2>
<form (ngSubmit)="register()" [formGroup]="userForm">
  <div>
    <label>Username</label>
    <input formControlName="username">
  </div>
  <div>
    <label>Password</label>
    <input type="password" formControlName="password">
  </div>
  <button type="submit">Register</button>
</form>
```

LES VARIABLES DE TEMPLATE

- Une variable de template est une référence à un élément du DOM dans un template
- Il peut également faire référence à un composant ou une directive Angular
- Utilisation du symbole *#* ou du prefixe *ref-* pour déclarer une variable de référence (*ref-fax* ou *#fax*)

LES VARIABLES DE TEMPLATE

- On peut y faire référence dans le template
- La portée d'une variable de template est le template
- Il ne faut pas nommer deux variables de template avec le même nom

```
<form (ngSubmit)="addCharacter()" #characterForm="ngForm">
```

VALIDATION

- Champs obligatoires, respect de format
- Deux types de validation :
 - Validation côté modèle : *Validator*
 - Validation côté template :

- *required*

```
<input name="name" [(ngModel)]="characterToAdd.name" required>
```

- *minlength* et *maxlength*, *email*

```
<input name="name" [(ngModel)]="characterToAdd.name" required minlength="3">
```

ERREURS ET SOUMISSION

- Si on rencontre une erreur de validation :
 - *disabled* est lié à la propriété *valid* du formulaire :


```
<button type="submit" [disabled]="!characterForm.form.valid">ADD</button>
```

- *hasError* permet par exemple d'afficher un message si le mot de passe est vide

Remarque : Une directive custom permet d'abstraire l'affichage des erreurs

ERREURS ET SOUMISSION

- Angular ajoute automatiquement des classes CSS sur les champs et le formulaire :
 - *ng-invalid* et *ng-valid*, *ng-dirty*, *ng-touched*, *ng-untouched*, *ng-pristine*

```
input.ng-invalid {  
  border: 3px  red solid;  
}
```

- Exemple :

CRÉER SON VALIDATEUR



CRÉER SON VALIDATEUR

- Valideur spécifique : par exemple vérifier si le personnage a plus de 18 ans
- En utilisant le FormBuilder, depuis le code

```
isOldEnough = (control: FormControl) => {  
  // control is a date input, so we can build the Date from the value  
  const birthDatePlus18 = new Date(control.value);  
  birthDatePlus18.setFullYear(birthDatePlus18.getFullYear() + 18);  
  return birthDatePlus18 < new Date() ? null : { tooYoung: true };  
};
```

Exemple : Valideur asynchrone pour vérifier auprès du backend si un nom d'utilisateur est disponible par exemple

EXERCICE



EXERCICE



LES SERVICES

LES SERVICES

- Des classes que l'on peut injecter dans une autre
- Singleton => instance unique injectée partout
- Deux services fournis de base : title et meta
- Possibilité de créer son propre service

```
export class CharacterService {  
  
  list() {  
    return [{ name: 'Jean Neige' }];  
  }  
}
```

- Exemple :

INJECTION DE DÉPENDANCES



INJECTION DE DÉPENDANCES

- Design pattern
- Faire appel à des fonctionnalités définies ailleurs
- Angular crée l'instance du service lui-même, pas le composant qui a besoin du service => inversion de contrôle
- Pour utiliser une dépendance :
 - Rendre disponible l'injection
 - Déclarer la dépendance

INJECTION DE DÉPENDANCES

- Rendre disponible l'injection :
 - Export du service `export class CharacterService {`
 - *@Injectable* : permet de déclarer que ce service a lui-même des dépendances

INJECTION DE DÉPENDANCES

- Déclaration du service dans le provider : déclaration raccourci qui fait un lien entre le token et le service, l'injecteur maintient un dictionnaire de token et injecte les services quand ils sont réclamés)

INJECTION DE DÉPENDANCES

```
@NgModule({
  declarations: [
    AppComponent,
    CharactersGridComponent,
    CharacterComponent,
    CharactersGrid2Component,
    PanelWithListComponent
  ],
  imports: [
    MatTabsModule,
    HttpClientModule,
    MatProgressBarModule,
    BrowserAnimationsModule,
    MatCardModule,
    MatGridListModule,
    MatButtonModule,
    MatInputModule,
    MatRadioModule,
    MatExpansionModule,
    BrowserModule,
    FormsModule
  ],
  providers: [HttpClientModule, CharacterService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

INJECTION DE DÉPENDANCES

On pourrait changer la classe du token pour bouchonner, uniquement en dev par exemple.

```
providers: [HttpClientModule, { provide: CharacterService, useClass: FakeCharacterService }],
```

Injecteur hiérarchique par composant, hérite de l'injecteur de son parent

INJECTION DE DÉPENDANCES

Déclarer la dépendance avant de l'utiliser

```
/** Import de la classe services */  
import { CharacterService } from './services/character.service';
```

EXERCICE

- Implémenter un service qui renvoie des personnages en dur
- Déclarer votre service
- Injecter votre service dans le composant liste des personnages



HTTP

- Utiliser les classes du module *HttpClientModule* dans le package angular/common/http
- *HttpClientModule* est une réécriture de *HttpModule*
- Permet de bouchonner ton serveur, et de retourner des réponses données
- Utilise le paradigme de la programmation réactive
=> façon de construire une application avec des événements, et d'y réagir

HTTP

- Le module HTTP propose le service *HttpClient*
- Permet de réaliser des requêtes AJAX avec *XMLHttpRequest*
- Propose des méthodes : *get, post, put, delete, patch, head* et *jsonp*
- Ces méthodes retournent un objet Observable. On peut s'y abonner pour obtenir la réponse

HTTP

- Ces méthodes retournent un objet Observable. On peut s'y abonner pour obtenir la réponse
- Accès à la réponse HTTP avec le statut via *HttpResponse* ou *HttpErrorResponse*
- Il existe des intercepteurs à travers *HttpInterceptor*

Remarque : une bonne pratique est de mettre dans un service dédié, et de s'abonner à la méthode de son service à partir d'un composant

HTTP

Exemple d'appel à une api :

```
@Injectable()
export class CharacterService {

    /** Constructor of the service */
    constructor(private http: HttpClient) {

    }

    /** Function to get all characters from the api */
    pullCharacters() {
        let options: any;
        let url = 'https://anapioficeandfire.com/api/characters/';
        return this.http.get(url);
    }
}
```

HTTP

Exemple d'appel à une api :

```
constructor(public characterService: CharacterService) {  
    charactersData.splice(0, 1900);  
  
    //this.characters = charactersData.map(props => new Character(props));  
  
    /** use my service, and change the data binding from the api */  
    characterService.pullCharacters()  
        .subscribe((characters: Array<Character>) => {  
        |   this.characters = characters;  
    });  
});
```

EXERCICE

- Appeler l'api get des personnages
- Vérifier sur votre page de liste des personnages le branchement



RXJS



- Bibliothèque qui permet d'implémenter le concept de programmation réactive visant à conserver une cohérence d'ensemble en propageant les modifications d'une source réactive
- Déclenchement d'évènements, on peut y réagir ou non. Ces évènements peuvent être combinés, transformés, filtrés... en utilisant des méthodes comme map

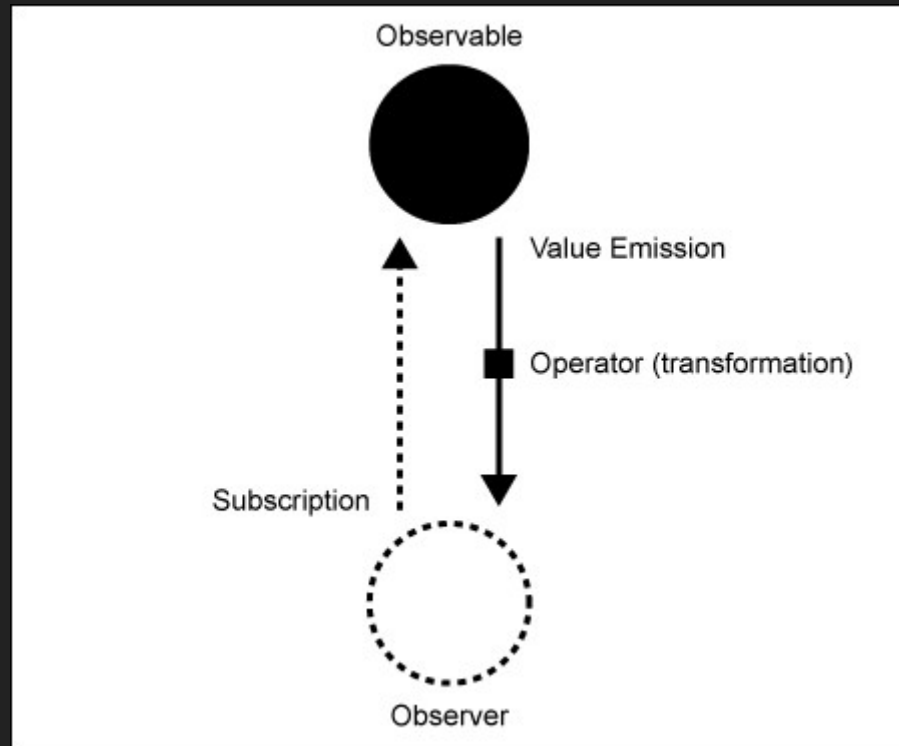
RXJS

- Toute donnée entrante sera dans un flux. Ces flux peuvent être écoutés, évidemment modifiés (filtrés, fusionnés, ...), et même devenir un nouveau flux que l'on pourra aussi écouter. Cette technique permet d'obtenir des programmes faiblement couplés? tu te contentes de déclencher un événement, et toutes les parties de l'application intéressées réagiront en conséquence
- Dans Angular => répondre à une requête HTTP, lever un événement spécifique dans un composant

RXJS

- Ces flux peuvent être subscribe. Un listener de flux est un observer
- Observer différent promise car il continue d'écouter jusqu'à un événement de terminaison
- Observable est une collection de données dont les valeurs sont reportées dans le temps, en asynchrone

RXJS



RXJS

Un observable, peut être transformé via des fonctions comme :

- *map(fn)* va appliquer la fonction fn sur chaque événement et retourner le résultat
- *subscribe(fn)* appliquera la fonction fn à chaque événement qu'elle reçoit

RXJS

```
[1, 2, 3, 4, 5]  
  .map(x => x * 2)  
  .filter(x => x > 5)  
  .forEach(x => console.log(x)); // 6, 8, 10
```

```
Observable.from([1, 2, 3, 4, 5])  
  .map(x => x * 2)  
  .filter(x => x > 5)  
  .subscribe(x => console.log(x)); // 6, 8, 10
```

RXJS

EventEmitter : Angular propose un adaptateur autour de l'objet Observable : *EventEmitter*. Il possède une méthode `subscribe()` pour réagir aux événements

```
@Output() matriculeChange: EventEmitter<string> = new EventEmitter<string>();

@Input() canEdit: boolean;

constructor(private barcodeScanner: BarcodeScanner, private platform: Platform) {

}

/** return true if photo camera available */
photoAvailable(): boolean {
  return !this.platform.is('core') && !this.platform.is('mobileweb'); // computer
}

changeMat(ev) {
  this.matriculeChange.emit(ev);
}
```

RXJS

```
<mat-field *ngIf="is2Wheel" (matriculeChange)="changeMat($event)"  
[canEdit]="canEdit" [matricule]="tempMatricule" [isRequired]="is2Wheel"></mat-field>
```



LES PIPES

NOM D'UNE PIPE

- Pipe ou tube ou tuyau ou |
- Remplace le filter de AngularJS
- Transformer des données brut en entrée que l'on veut afficher (ou filtrer, tronquer, etc)

LES PIPES

- Il présente quelques avantages supplémentaires :
 - Built-in pipes : comme le pipe *async*, *decimal* ou *percent*...
 - Certains pipes sont paramétrables
 - Les pipes sont chainables
 - On peut également créer ses propres pipes
- Il existe les pipes pures et impures
 - Angular exécute une pipe pure quand il détecte un changement de valeur sur un input de type primitif (*String*, *Number*, *Boolean*...) ou une référence d'objet (*Date*, *Array*...)

UTILISATION DANS LE CODE

```
import { JsonPipe } from '@angular/common';

export class CharactersComponent {
  characters: Array<any> = [{ name: 'J

  charactersAsJson: string;

  constructor(jsonPipe: JsonPipe) {
    this.charactersAsJson = jsonPipe.trans
  }
}
</any>
```

LES PIPES UTILES

Exemple de pipes :

- *json* : utile pour le debug => applique

JSON.stringify() `<p>{{ characters | json }}</p>`

- *slice* : permet d'afficher un sous ensemble de la collection (ou string) Couplé à ngFor, permet de sélectionner les éléments à afficher

`<p>{{ characters | slice:0:2 | json }}</p>`

LES PIPES UTILES

Exemple de pipes :

- *uppercase* : permet d'afficher le texte en majuscule

```
{{ character.name | uppercase }}
```

- *lowercase* : permet d'afficher le texte en minuscule

```
{{ character.name | lowercase }}
```

- *number* : permet de formater un nombre (s'appuie sur l'internationalisation du navigateur)

```
<p>{{ 12345 | number }}</p>
```

LES PIPES UTILES

Exemple de pipes :

- *percent* (s'appuie sur l'internationalisation du navigateur) `<p>{{ 0.8 | percent }}</p>`
- *currency* (s'appuie sur l'internationalisation du navigateur) `<p>{{ 10.6 | currency:'EUR' }}</p>`
- *date* (s'appuie sur l'internationalisation du navigateur)

```
<div class="card-subtitle ">{{event.startDate | date:'dd/MM/yyyy'}}</div>
```

LE PIPE ASYNC

- Permet d'afficher des données obtenues de manière asynchrone
- Utilise *PromisePipe* ou *ObservablePipe* selon que tes données viennent d'une *Promise* ou d'un *Observable*
- Retourne une chaîne de caractères vide jusqu'à ce que les données deviennent disponibles

```
<p>{{ characters | async }}</p>
```

LE PIPE CUSTOM


```
/*Pipe to sort collection by team Id */
@Pipe({
  name: 'sortSetByEventPipe',
  pure: false,
})
export class SortSetByEventPipe implements PipeTransform {
  transform(collection: any[], eventId: any): any {

    let transformedData = [];

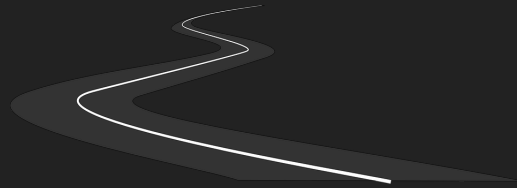
    if (collection != null) {
      collection.forEach((element: any) => {
        if (element.eventId === eventId) {
          transformedData.push(element);
        }
      });
      return transformedData;
    } else {
      return collection;
    }
  }
}
```

EXERCICE



EXERCICE

- Utiliser le pipe async dans notre liste de personnage avec l'appel API
- Créer son propre pipe pour classer les personnages par ordre alphabétique
- L'appeler à travers le code et le template



LES MODULES / LES ROUTES

MODULE

- Gestion de module fondamentale dans Angular
- Tout est défini dans des modules
- Le mot clé *export* permet d'exposer une fonction
- Un composant peut appeler à l'aide de *import*
- Named export : nommer les imports que l'on souhaite utiliser

MODULE

- Utilisation d'un alias

```
import { CharacterService as charService } from './services/character.service';
```

- On peut utiliser le joker *

```
import * as CharacterService from './services/character.service';
```

SUR MA ROUTE

- URL = état de l'application
- Assigner à une URL un composant
- Rendre la Single Page Application plus intelligente

Remarque : Ne regardez pas la doc

DIFFÉRENTS ROUTEURS :

- *ngRoute* : AngularJS, maintenu mais limité pour les grosses applications
- *ui-router* : efficace
- *Router module* : proposé par Angular, complet

ROUTER MODULE

- Commencer par ajouter qu'on utilise le module

routeur

```
/** Import du module de routing */  
import { RouterModule } from '@angular/router';
```

- Configurer vos routes à travers un fichier de configuration

```
import { Routes } from '@angular/router';  
import { CharacterComponent } from '../character/character.component';  
  
export const ROUTES: Routes = [  
  { path: 'characterz', component: CharacterComponent }  
];
```

ROUTER MODULE

- Dans l'app module, importer le module de routing avec le fichier de configuration

```
imports: [  
  MatTabsModule,  
  HttpClientModule,  
  MatProgressBarModule,  
  BrowserAnimationsModule,  
  MatCardModule,  
  MatGridListModule,  
  MatButtonModule,  
  MatInputModule,  
  MatRadioModule,  
  MatExpansionModule,  
  BrowserModule,  
  FormsModule,  
  RouterModule.forRoot(ROUTES)  
],
```

ROUTER MODULE

- On inclus le composant à afficher dans la page à l'aide de

Déclaration de liens de navigation :

```
<a href="" routerLink="/" routerLinkActive="selected-menu">Home</a>
```

- Directive routerLinkActive pour appliquer un style quand une route est sélectionnée
- Côté code, on peut naviguer avec

```
this.router.navigate(['']);
```

ROUTER MODULE

- Passage de paramètres (1 ou plusieurs) :

```
[routerLink]="['/characters', character.id, houses', house..
```

- On peut ensuite récupérer les paramètres facilement :

```
const id = this.route.snapshot.paramMap.get('characterId');
```

ROUTER MODULE

- Redirection

```
{ path: '', pathMatch: 'full', redirectTo: '/breaking' }
```

- Bien penser à l'ordre des routes => la stratégie du routeur consiste à prendre la première route qui correspond
- Utilisation de route fille pour l'insertion des composants enfants à l'aide de children, permet par exemple de naviguer dans des onglets

ROUTER MODULE

Guards : gardes ou gardiens => permet de filtrer les accès à l'application :

- *canActivate* : peut empêcher l'activation de la route ou de rediriger en cas d'erreur, ou d'une page inaccessible
- *CanActivate* et qui retourne un booléen, une promesse ou un observable
- *canActivateChild* : peut empêcher l'activation des enfants de la route indiquée
- *canLoad* : permet de télécharger un module en lazy loading
- *canDeactivate* : empêche de quitter la route actuelle. Permet de demander une confirmation avant de quitter une page

ROUTER MODULE

- Resolver : permet de retourner des données synchrones ou asynchrones
 - par exemple pour naviguer vers une page lorsque les données sont chargées (résolu)
- Évènements du router comme :
 - *NavigationStart*
 - *NavigationEnd*

EXERCICE



EXERCICE

- Créer le fichier de routing
- Créer une route vers la liste des personnages
- Créer un lien vers cette route
- Mettre en place une garde
- *Bonus : passer un paramètre à la route pour afficher un personnage*



RETOUR D'EXPÉRIENCE NGRX

RETOUR D'EXPÉRIENCE NGRX

Voir présentation Powerpoint