Angular TP-05

Observables

Installez le package rxjs-compat dans votre projet.

Dans app-component pour avoir accès aux Observables et aux méthodes que vous allez utiliser, il faut ajouter deux imports :

import { Observable } from 'rxjs/Observable';

import 'rxjs/add/observable/interval';

Le premier import sert à rendre disponible le type Observable, et le deuxième vous donne accès à la méthode interval(), qui crée un Observable qui émet un chiffre croissant à intervalles réguliers et qui prend le nombre de millisecondes souhaité pour l'intervalle comme argument.

Implémentez OnInit et créez l'Observable dans ngOnInit() :

const counter = Observable.interval(1000);

Pour observer vous utiliserez subscribe().

Créez une variable secondes dans AppComponent

secondes: number;

et affichez-la dans le template :

```
<div class="navbar-right">
  <p>Vous êtes connecté depuis {{ secondes }} secondes !</p>
</div>
```

Puis créez les trois fonctions de subscribe dans ngOnInit.

```
    counter.subscribe(
      (value) => {
        this.secondes = value;
      },
      (error) => {
        console.log('An error occured ! : ' + error);
      },
      () => {
        console.log('Observable complete!');
      }
    );
```

Testez !

Subscriptions

Pour rappel, la fonction subscribe() prend comme arguments trois fonctions anonymes :

• la première se déclenche à chaque fois que l'Observable émet de nouvelles données, et reçoit ces données comme argument ;

• la deuxième se déclenche si l'Observable émet une erreur, et reçoit cette erreur comme argument ;

• la troisième se déclenche si l'Observable s'achève, et ne reçoit pas d'argument.

Pour l'instant, cette souscription n'est pas stockée dans une variable : on ne peut donc plus y toucher une fois qu'elle est lancée. Mais on peut la stocker dans un objet Subscription

Importez Subscription depuis ‘rxjs/Subscription’,

Créez une variable de type Subscription :

counterSubscription: Subscription;

Et affectez-lui counter.subscribe.

```
this.counterSubscription = counter.subscribe( …
...);
```

On rajoute la fonction ngOnDestroy(), qui se déclenche quand un component est détruit.

Importez OnDestroy depuis @angular/core, puis implémentez la classe :

export class AppComponent implements OnInit, OnDestroy {

et rajouter la fonction :

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

La fonction unsubscribe() détruit la souscription et empêche les comportements inattendus liés aux Observables infinis, donc n'oubliez pas de unsubscribe !

Testez !

Subjects

Imaginez une variable dans un service, qui peut être modifié depuis plusieurs components ET qui fera réagir tous les components qui y sont liés en même temps.

Pour l'instant, l'array dans studentViewComponent est une référence directe à l'array dans studentService. Il peut être modifié directement depuis n'importe quel endroit du code.

Pour mettre en place le service, dans StudentService :

Rendre l'array des students private :

```
private students = […
…]
```

Créer un Subject dans le service :

Importez-le depuis ‘rxjs/Subject’, puis créez une variable de type Subject :

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

Créer une méthode qui, quand le service reçoit de nouvelles données, fait émettre ces données par le Subject :

```
emitStudentSubject() {
  this.studentsSubject.next(this.students.slice());
}
```

Faites appel à cette méthode, dans les 4 méthodes qui changent les données, après le changement :

```
this.emitstudentSubject();
```

Souscrivez à ce Subject depuis studentViewComponent pour recevoir les données émises, émettez les premières données, et implémentez OnDestroy pour détruire la souscription.

Importez Subsription et OnDestroy, implémentez OnDestroy.

Créez studentSubscription de type Subscription.

Dans ngOnInit ajouter :

```
  this.studentSubscription = this.studentService.studentsSubject.subscribe(
    (students: any[]) => {
      this.students = students;
    }
  );
  this.studentService.emitStudentSubject();
  }
```

Et pensez à unsubscribe() dans ngOnDestroy.

Testez !

Il y a maintenant une abstraction entre le service et les components, où les données sont maintenues à jour grâce au Subject.

Méthode template

Vous créez votre formulaire dans le template, et Angular l'analyse pour comprendre les différents inputs et pour en mettre à disposition le contenu.

Importer FormsModule dans AppModule.

Créez le formulaire

Créer un nouveau component EditStudentComponent qui permettra à l'utilisateur d'enregistrer un nouvel étudiant :

```
<div class="row">
  <div class="col-sm-8 col-sm-offset-2">
    <form>
      <div class="form-group">
        <label for="name">
          Nom de l'étudiant
        </label>
        <input type="text" id="name" class="form-control" name="name" ngModel>
      </div>
      <div class="form-group">
        <label for="status">
          État de l'étudiant
        </label>
        <select id="status" class="form-control" name="status" ngModel>
          <option value="present">present</option>
          <option value="absent">absent</option>
        </select>
      </div>
      <button class="btn btn-primary">Enregistrer</button>
    </form>
  </div>
</div>
```

Angular parcourt votre template et trouve la balise <form>, créant ainsi un objet qui sera utilisable depuis votre code TypeScript.

Il faut ajouter onSubmit dans form et faire le lien avec le bouton :

```
<form (ngSubmit)="onSubmit(formStudent)" #formStudent="ngForm">

<button class="btn btn-primary" type="submit">Enregistrer</button>
```

Quand on clique sur le bouton de type submit, la méthode que vous attribuez à (ngSubmit) est exécutée, et grâce à la référence locale #formStudent="ngForm", vous pouvez passer l'objet à la méthode.

On crée la méthode onSubmit() dans le TS.

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

Pour avoir accès au formulaire, créez une nouvelle route dans app-routage :

{ path: 'edit', canActivate: [AuthGuard], component: EditStudentComponent },

et un routerLink correspondant dans la barre de menu :

<li routerLinkActive="active"><a routerLink="edit">Nouvel étudiant

Testez !

Validez les données

Modifiez le HTML pour forcer le champs nom :

<input type="text" id="name" class="form-control" name="name" ngModel required>

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

Mettez la valeur absent par default pour le status de l'étudiant

Créez une variable defaultStatus

defaultStatus = 'absent';

Liez là au HTML :

<select id="status" class="form-control" name="status" [ngModel]="defaultStatus">

Testez !

Utiliser les données

Modifiez la méthode onSubmit :

```
onSubmit(form: NgForm) {
   const name = form.value['name'];
   const status = form.value['status'];
}
```

Créez maintenant la méthode dans StudentService qui générera le nouvel étudiant :

```
addStudent(name: string, status: string) {
   const studentObject = {
     id: 0,
     name: '',
     status: ''
   };
   studentObject.name = name;
   studentObject.status = status;
```

```
    studentObject.id = this.students[(this.students.length - 1)].id + 1;
    this.students.push(studentObject);
    this.emitStudentSubject();
}
```

Intégrez la fonction dans EditStudentComponent :

Importez Router depuis @angular/router.

Et modifier onSubmit pour appeler addStudent avec name et status, puis navigez avec router vers le component student :

```
onSubmit(form: NgForm) {
    const name = form.value['name'];
    const status = form.value['status'];
    appel de addStudent()
    Navigation vers student
}
```

Testez !

La méthode réactive

On commence par créez un nouveau dossier models, et dedans un fichier user.model.ts :

```
export class User {
  constructor(
    public firstName: string,
    public lastName: string,
    public email: string,
    public diploma: string,
    public options: any[]
  ) {}
}
```

Ce modèle pourra donc être utilisé dans le reste de l'application en l'important dans les components.

Créez un UserService pour stocker la liste des objets User :

Importez User depuis User.model.

Importez Subject depuis rxjs/Subject'

Créez une variable privé users de Type User[]

Créez un userSubject :

```
userSubject = new Subject<User[]>();
```

Puis ajouter une fonction pour ajouter les users :

```
  emitUsers() {
    this.userSubject.next(this.users.slice());
  }
```

Et une pour ajouter un nouvel user :

```
  addUser(user: User) {
    this.users.push(user);
    this.emitUsers();
  }
```

Maintenant on crée UserListComponent :

Importez OnDestroy, OnInit, User from model, Subscription et UserService.

On implémente la class avec OnInit et OnDestroy.

Créez une variable users de type User[].

et une userSubscription de type Subscription.

Créez ngOnInit :

```
    this.userSubscription = this.userService.userSubject.subscribe(
      (users: User[]) => {
        this.users = users;
      }
    );
    this.userService.emitUsers();
```

Créez une fonction ngOnDestroy pour unsubscribe à userSubscription.

Maintenant créez le HTML :

```
<ul class="list-group">
  <li class="list-group-item" *ngFor="let user of users">
    <h3>{{ user.firstName }} {{ user.lastName }}</h3>
    <p>{{ user.email }}</p>
    <p>Cet étudiant est en {{ user.diploma }}</p>
    <p *ngIf="user.options && user.options.length > 0">
      Il a pris les options suivantes :
      <span *ngFor="let option of user.options">{{ option }} - </span>
    </p>
  </li>
</ul>
```

Ajoutez une route users dans App-routing :

```
{ path: 'users', component: UserListComponent },
```

et créez un routerLink dans la nav-bar :

```
<li routerLinkActive="active"><a routerLink="users">Utilisateurs</a></li>
```

Pour les tests on va ajouter un User en dur dans UserService

```
private users: User[] = [
    new User('Charles', 'Quint', 'charles.quint@lesrois.fr', 'License 3 informatique', ['web application', 'baby-foot'])
];
```

Puis ajoutez ReactiveFormsModule, importé depuis @angular/forms, à l'array imports de votre AppModule.

Tout est prêt pour utiliser le formulaire !

Créez NewUserComponent qui contiendra votre formulaire réactif.

FormBuilder

Importez FormGroup depuis @angular/forms dans NewUserComponent.

Créez une variable userForm de type FormGroup,

Dans le constructeur injectez FormBuilder, importé depuis @angular/forms,

puis modifiez ngOnInit(),

```
  ngOnInit() {
    this.initForm();
  }
```

Et créez la méthode initForm()

```
initForm() {
      this.userForm = this.formBuilder.group({
      firstName: '',
      lastName: '',
      email: '',
      diploma: ''
    });
}
```

Les contrôles correspondants aux options seront ajoutés par la suite avec une autre méthode.

Maintenant créez le HTML :

```
<div class="col-sm-8 col-sm-offset-2">
  <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>
  <div class="form-group">
    <label for="email">Adresse e-mail</label>
    <input type="text" id="email" class="form-control" formControlName="email">
  </div>
  <div class="form-group">
    <label for="diploma">Diplome</label>
    <input id="diploma" class="form-control" formControlName="diploma">
  </div>
  <button type="submit" class="btn btn-primary">Valider</button>
 </form>
</div>
```

Analysez le template :

• Sur la balise <form>, vous utilisez le property binding pour lier l'objet userForm à l'attribut formGroup du formulaire, créant la liaison pour Angular entre le template et le TypeScript.

• Également dans la balise <form>, vous avez toujours une méthode onSubmitForm() liée à ngSubmit, mais vous n'avez plus besoin de passer le formulaire comme argument puisque vous y avez déjà accès par l'objet userForm que vous avez créé.

• Sur chaque <input> qui correspond à un control du formulaire, vous ajoutez l'attribut formControlName où vous passez un string correspondant au nom du control dans l'objet TypeScript.

• Le bouton de type submit déclenche l'événement ngSubmit, déclenchant ainsi la méthode onSubmitForm(), que vous allez créer dans votre TypeScript.

Pour tout mettre ensemble, injectez UserService et Router (sans oublier de les importer) dans le constructeur du component, et créez la méthode onSubmitForm() :

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

Il faut rajouter un lien dans UserListComponent pour accéder à NewUserComponent.

```
<a routerLink="/new-user">Nouvel utilisateur</a>
```

Pensez à ajouter la route new-user dans app-routing.

Validators

Comme pour la méthode template, il existe un outil pour la validation de données dans la méthode réactive : les Validators.

Il faut importer Validators.

Modifiez initForm :

```
this.userForm = this.formBuilder.group({
  firstName: ['', Validators.required],
  lastName: ['', Validators.required],
  email: ['', [Validators.required, Validators.email]],
  diploma: ['', Validators.required]
```

Modifiez le bouton submit

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

Ajoutez dynamiquement des FormControl

Importez FormArray depuis @angular/form,

Modifiez le initForm() et ajouter :

```
options: this.formBuilder.array([])
```

Modifiez ensuite onSubmitForm() pour récupérer les valeurs :

```
formValue['options'] ? formValue['options'] : []
```

Il faut maintenant une fonction get qui retourne les options

```
getOptions(): FormArray {
  return this.userForm.get('options') as FormArray;
}
```

Et une méthode pour ajouter une option en plus dans le template ;

```
onAddOption() {
  const newOptionControl = this.formBuilder.control(null, Validators.required);
  this.getOptions().push(newOptionControl);
}
```

Enfin, il faut ajouter une section au template qui permet d'ajouter des options en ajoutant des <input> :

```
<div formArrayName="options">
    <h3>Vos options</h3>
    <div class="form-group" *ngFor="let optionControl of getOptions().controls;
    let i = index">
```

```
        <input type="text" class="form-control" [formControlName]="i">
      </div>
      <button type="button" class="btn btn-success" (click)="onAddOption()">
      Ajouter une option</button>
</div>
<button type="submit" class="btn btn-primary"
[disabled]="userForm.invalid">Valider</button>
```

Testez !

Félicitations ! Maintenant, vous savez créer des formulaires par deux méthodes différentes, et comment récupérer les données saisies par l'utilisateur.

Il est encore temps d'améliorer le look de votre application avec :

Css : https://www.w3schools.com/w3css/default.asp
Bootstrap : https://www.w3schools.com/bootstrap/default.asp