Hajek Valentin TP6 Angular info2

1. Pourquoi des services : (bon moyen pour partager des infos entre les classes qui ne se connaissent pas)

On va créer un HeroServices où toutes les classes d’application peuvent utiliser pour obtenir des héros. On utiliser l’injection dans le constructeur HerosComponent

On va créer un MessageService : (dans HeroService qui utilise le service pour envoyer un message – dans MessagesComponent ce message)

1. a) Création

cd src/app

"ng generate service hero"

La classe hero.service est généré

b)Services Injectable ()

Injectable est importer dans la classe hero.service.ts permettant de dire à angular que ce service pourrait avoir des dépendance injectées(futur).

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

c) Obtenir des données de Héros / Fournir le HeroService

Permet d’obtenir toujours des héros :

import { Hero } from './hero';

import { HEROES } from './mock-heroes';

On ajoute une méthode getHeroes permettant de renvoyer les faux héros

getHeroes(): Hero[] {

return HEROES;

}

d) Mettre à jour Heroes Component

On supprime la ligne d’importation du heros on la remplace par l’importation de HeroService :

import { HEROES } from '../mock-heroes';

on remplace par:

Import { HeroService } from '../hero.service';

Puis on doit redéfinir le héros donc on remplace :

heroes = HEROES;

par :

heroes: Hero[];

e) Injecter le HeroService

On crée le constructeur : dans heroes.component.ts :

constructor(private heroService: HeroService) { }

f) Ajouter getHeroes() : Permettant de récupérer les héros du service

getHeroes(): Hero[] {

return HEROES;

}

Puis on l’appelle dans ngOnInit()

ngOnInit() {

this.getHeroes;

}

Adaptés :



3. Donnée observables :

On récupérait de manière synchrome grâce à : this.heroes = this.heroService.getHeroes ();

L’utilisation de serveur nécessite une manière assynchrone, c’est-à-dire qu’il doit attendre l’envoie du serveur pour retourner les héros (fonction de rappelle, retourner une promesse, retourner un Observable)

Ici observable car HttpClient.get() renvoie un observable

a)

import { Observable, of } from 'rxjs';

remplacement de getHeroes par:

getHeroes(): Observable<Hero[]> {

return of(HEROES);

}

b) Abonnez-vous à HeroesComponent

Ancienne :

getHeroes(): Hero[] {

return HEROES;

}

Remplacé par:

getHeroes(): void {

this.heroService.getHeroes()

.subscribe(heroes => this.heroes = heroes);

}

Car permet d’au lieu d’avoir un tableau de héros à la prop. Heroes composant (synchrone) l’application n’aurait pas attendu la réponse du serveur ; Dans la nouvelle version on attend que Observable émette le tableau des héros, subscribe passe le tableau reçut en sortie qui définir la propriété heroes du composant.

Maintenant il y à donc une possibilité asynchrone lorsque HeroeService demandera des héros au serveur

4. Afficher les messages

On va : Ajouter un composant MessageComponent ; crée un MessagerieService ; injecter un MessageService->HeroService ; Afficher un message lorsque les héros sont récupéré par HeroService

1. Création d’un messageComponent

Dans src/app

ng generate component [messages](https://angular.io/api/service-worker/SwPush#messages)

Modification du template AppComponent:

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

<app-heroes></app-heroes>

<app-messages></app-messages>

1. Création du service Messagerie

Dans src/app

ng generate service [message](https://angular.io/api/common/http/HttpErrorResponse#message)

Puis on remplace dans MessageService : composé de deux méthodes add pour add un message au cache, et clear pour effacer le cache

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

@Injectable({

providedIn: 'root',

})

export class MessageService {

messages: string[] = [];

add(message: string) {

this.messages.push(message);

}

clear() {

this.messages = [];

}

}

1. Injecter MessageService dans HeroService

On importe le MessageService :

import { MessageService } from './message.service';

On ajouter au constructeur un messageService : MessageService :

constructor(private messageService: MessageService) { }

1. Suite à la déclaration du constructeur la méthode getHeroes ne fonctionne plus il faut donc la changer par :

getHeroes(): Observable<Hero[]> {

// TODO: send the message \_after\_ fetching the heroes

this.messageService.add('HeroService: fetched heroes');

return of(HEROES);

1. Afficher le message de HeroService

On importe le MessageService : On le met en public car on va le lier et l’utiliser dans le template

import { MessageService } from '../message.service';

Puis on modifie le constructeur pour permettre a Angulart d’injecter le singleton MessageService dans cette propriété quand il crée le HeroService

constructor(public messageService: MessageService) {}

1. Lien avec le MessageService

On remplace le template de MessageCOmponent dans messages.component.html par :

<div \*ngIf="messageService.messages.length"> //Affiche zone des messages

<h2>Messages</h2>

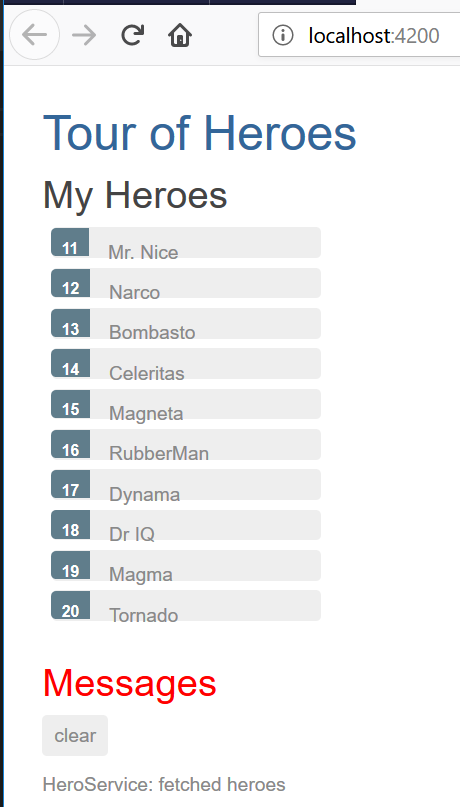
<button class="clear" //L’évenemtn click du bouton lier à MessageService.clear()

(click)="messageService.clear()">clear</button>

<div \*ngFor='let message of messageService.messages'> {{message}} </div>

// ng for présente la liste des messages dans les éléments <div>

</div>



En résumé :

- Nous avons remanié (refactor) l'accès aux données de la classe HeroService.

- Nous avons fourni le HeroService dans le module racine AppModule afin qu'il puisse être injecté n'importe où.

- Nous avons utilisé l'injection de dépendance d'angular pour injecter dans un composant.

- Nous avons donné à la méthode HeroService get data une signature asynchrone.

- Nous avons découvert Observable et la bibliothèque RxJS Observable.

- Nous avons utilisé of() de RxJS pour retourner un Observable de héros (Observable <Hero []>).

- Le hook de cycle de vie ngOnInit du composant appelle la méthode de HeroService, pas le constructeur.

- Vous avez créé un MessageService pour une communication faiblement couplée entre les classes.

- Le HeroService injecté dans un composant est créé avec un autre service injecté, MessageService.