Web Programming

4th Activity - Services and Observables

2019/2020

1. Using Services

Special class that allows centralize some logic to be used in other components. This logic can be injected into components.

a) Clean the files

app.component.ts

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

@Component({
      selector: 'app-root',
      templateUrl:
'./app.component.html',
      styleUrls: [
'./app.component.css' ]
})
export class AppComponent {}
```

app.component.html

```
<app-header></app-header>
<main>
  <app-post-create></app-post-create>
  <app-post-list></app-post-list>
</main>
```

posts-list.component.ts

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

import { Post } from '../post.model';

@Component({
      selector: 'app-post-list',
      templateUrl: './post-list.component.html',
      styleUrls: [ './post-list.component.css' ]
})
export class PostListComponent {
      posts: Post[] = [];
```

```
}
```

post-create.component.ts

```
import { Component } from '@angular/core';
import { NgForm } from '@angular/forms';
import { Post } from '../post.model';

@Component({
	selector: 'app-post-create',
	templateUrl: './post-create.component.html',
	styleUrls: [ './post-create.component.css' ]
})
export class PostCreateComponent {

	onAddPost(form: NgForm) {
		if (form.invalid) {
			return;
		}
		const post: Post = {
			title: form.value.title,
			content: form.value.content
		};
	}
}
```

Right now, our system doesn't send data between components anymore.

b) Create a service

- Create a file named post.service.ts inside posts folder

posts.service.ts

```
import { Post } from './post.model';
import { Injectable } from '@angular/core';

@Injectable({ providedIn: 'root' })

export class PostsService {
      private posts: Post[] = [];

      //gets a copy of posts
      getPosts() {
            return [ ...this.posts ];
      }

      //insert in our posts
      addPost(title: string, content: string) {
            const post: Post = {
                  title: title,
                  content: content
                  };
            this.posts.push(post);
      }
}
```

In this service, by now we'll have a function that gives us our posts and a function that adds posts.

Two ways to inform angular that this file is a service:

1) We could import service in our **app.module**,
2) but we can also import within the file by using this line of code.

```
@Injectable({ providedIn: 'root' })
```

This line will tell angular we only want one instance of the service.

c) Adjust post-create to use that service

post-create.component.ts

```
import { PostsService } from './../posts.service';
import { Component } from '@angular/core';
import { NgForm } from '@angular/forms';
import { Post } from '../post.model';

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

export class PostCreateComponent {
```

```
    constructor(public postsService: PostsService) {}

    onAddPost(form: NgForm) {
          if (form.invalid) {
                return;
          }

     const post: Post = {
    title: form.value.title,
    content: form.value.content,
  };

    this.postsService.addPost(post.title,post.content);
    }
}
```

To use a service we need to inject it in the component constructor. The constructor is a special method that is read when angular builds a component.

The constructor receives our service, but we also need to have a variable in our code to access the service. So we could do something like:

```
postsServices: PostsService;

  constructor( postService: PostsService) {
        this.postService = postService;
  }
```

But angular has a less verbose way to do the same:

```
constructor( public postService: PostsService) {}
```

Now we can call it the same way we are used to and onAddPost will call the addPost function from our service.

d) Angular lifecycle hooks

In this file we need to fetch data just after the component is constructed, and thus we need to use one life cycle hook (LCH), that angular provides us.

Lifecycle hooks

A component has a lifecycle managed by Angular. Angular creates and renders components along with their children, checks when their data-bound properties change, and destroys them before removing them from the DOM. Angular offers lifecycle hooks that provide visibility into these key life moments and the ability to act when they occur.

ngOnInit()

Initialize the directive/component after Angular first displays the data-bound properties and sets the directive/component's input properties.
Called once, after the first ngOnChanges().

ngOnChanges() Respond when Angular (re)sets data-bound input properties. The method receives a SimpleChanges object of current and previous property values.
Called before ngOnInit() and whenever one or more data-bound input properties change.

ngOnDestroy()

Cleanup just before Angular destroys the directive/component. Unsubscribe Observables and detach event handlers to avoid memory leaks.
Called just before Angular destroys the directive/component.

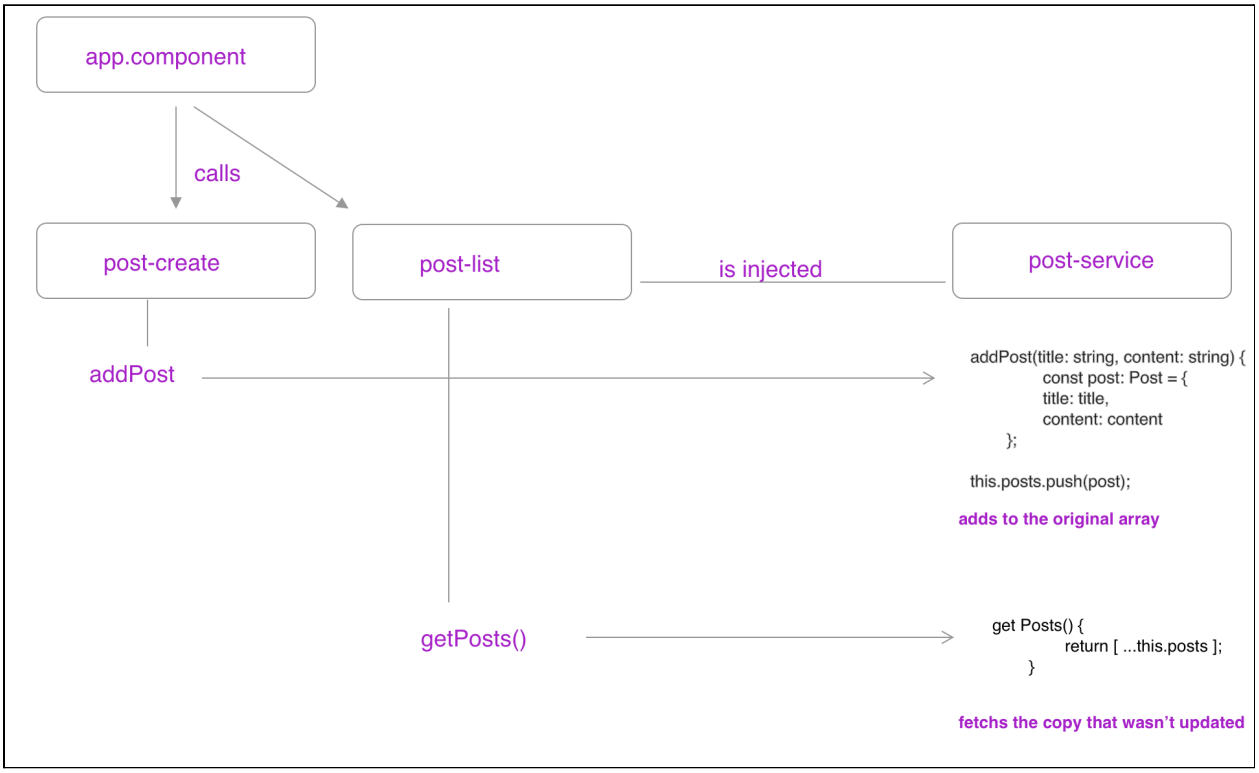
e) Adjust post-list to use that service using the lifecycle OnInit

post-list.component.ts

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

import { Post } from '../post.model';
import { PostsService } from '../posts.service';

@Component({
	selector: 'app-post-list',
	templateUrl: './post-list.component.html',
	styleUrls: [ './post-list.component.css' ]
})
export class PostListComponent implements OnInit {
	posts: Post[] = [];

	constructor(public postService: PostsService) {}

	ngOnInit() {
		this.posts = this.postService.getPosts();
	}

}
```

We call our service, just after the component constructor but do we get our posts?

The answer is **No**. And this is because when we create a post, we add it to the post array in **post.service.ts**. The **getPosts** from the post-list is not updated with this insertion because we are fetching a copy of the array, so it will list the empty array. We could solve this using the original post, but it's not the cleanest way.



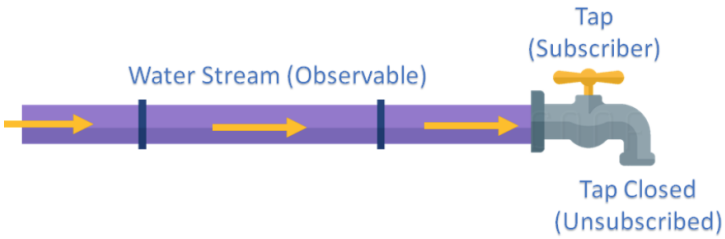
f) Observables

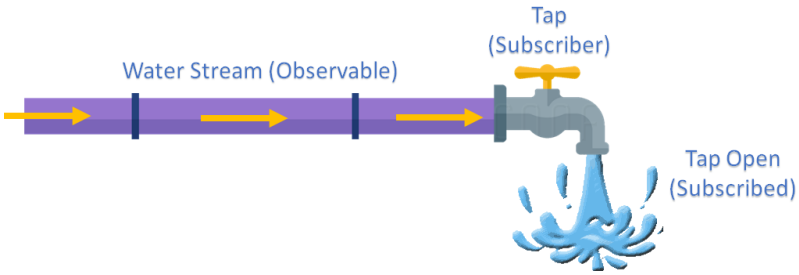
We want to use an event driven approach, where we actively call for new data if something occurs, and we can do this with **Observables.** For this we'll use an **external package** rxjs.

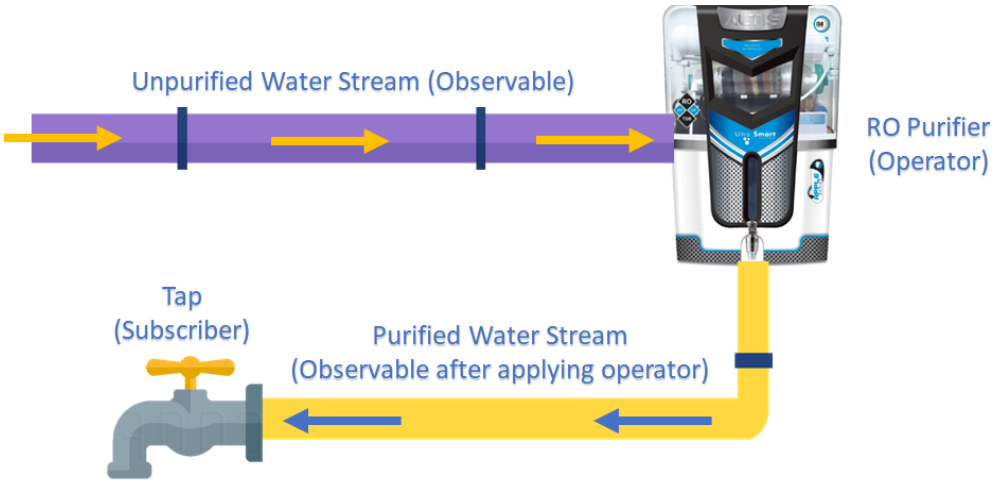
The essential concepts in RxJS which solve async event management are:

- **Observable**: represents the idea of an invokable collection of future values or events.
- **Observer**: is a collection of callbacks that knows how to listen to values delivered by the Observable.
- **Subscription**: represents the execution of an Observable, is primarily useful for cancelling the execution.

- **Operators**: are pure functions that enable a functional programming style of dealing with collections with operations like map, filter, concat, reduce, etc.
- **Subject**: is the equivalent to an EventEmitter, and the only way of multicasting a value or event to multiple Observers.







By now, we only need to understand these concepts to understand our code and logic.

So, know how observers work, we need to implement the following:

1) a subject to be observed
2) this observable will be updated whenever a post is added
3) we'll create another function to send this new data

posts.service.ts

```
import { Post } from "./post.model";
import { Injectable } from "@angular/core";
import { Subject } from "rxjs";

@Injectable({ providedIn: "root" })
export class PostsService {
 private posts: Post[] = [];
 private postsUpdated = new Subject<Post[]>();

 getPosts() {
   return [...this.posts];
 }

 getPostUpdateListener() {
   return this.postsUpdated.asObservable();
 }

 addPost(title: string, content: string) {
   const post: Post = {
     title: title,
     content: content
   };
   this.posts.push(post);
   this.postsUpdated.next([...this.posts]);  }
}
```

Now in our post-list component we also need some changes in order to use these new types of data.

post-list.component.ts

```
import { Component, OnInit, OnDestroy } from '@angular/core';
import { Subscription } from 'rxjs';
import { Post } from '../post.model';
import { PostsService } from '../posts.service';

@Component({
   selector: 'app-post-list',
   templateUrl: './post-list.component.html',
   styleUrls: [ './post-list.component.css' ]
})
export class PostListComponent implements OnInit, OnDestroy {
   posts: Post[] = [];
   private postsSub: Subscription;
   constructor(public postService: PostsService) {}

   ngOnInit() {
       this.posts = this.postService.getPosts();
       this.postsSub = this.postService.getPostUpdateListener()
       .subscribe((posts: Post[]) => {
           this.posts = posts;
        });
       }

    ngOnDestroy() {
              this.postsSub.unsubscribe();
```

```
        }
}
```

Adjusting the form

post-create.component.html

```
<mat-card>
  <form  #postForm="ngForm" (submit)="onAddPost(postForm)">
    <mat-form-field>
      <input matInput name="title" type="text" ngModel required
   minlength="3" #title="ngModel" placeholder=" Post Title">
<mat-error *ngIf="title.invalid">Please enter a valid post title</mat-error>
    </mat-form-field>
    <mat-form-field>
      <textarea matInput name="content" rows="6" ngModel required
   placeholder="Post Content"> </textarea>
    </mat-form-field>
    <button mat-raised-button type="submit" color="accent">Save
   Post</button>
  </form>
</mat-card>
```

post-create.component.ts

```
import { PostsService } from './../posts.service';
import { Component } from '@angular/core';
import { NgForm } from '@angular/forms';
import { Post } from '../post.model';

@Component({
   selector: 'app-post-create',
```

```
    templateUrl: './post-create.component.html',
    styleUrls: [ './post-create.component.css' ]
})
export class PostCreateComponent {
    constructor(public postsService: PostsService) {}

    onAddPost(form: NgForm) {
           if (form.invalid) {
                  return;
           }


           this.postsService.addPost(form.value.title, form.value.content);
           form.resetForm();
    }
}
```

post-list.component.html

```
<mat-accordion multi="true" *ngIf="posts.length > 0">
 <mat-expansion-panel *ngFor="let post of posts">
   <mat-expansion-panel-header>
     {{ post.title }}
   </mat-expansion-panel-header>
   <p>{{ post.content }}</p>
   <mat-action-row>
     <button mat-button color="primary">EDIT</button>
     <button mat-button color="warn">DELETE</button>
   </mat-action-row>
 </mat-expansion-panel>
</mat-accordion>
<p class="info-text mat-body-1" *ngIf="posts.length <= 0">
 No posts added yet!
</p>
```

References

- https://itnext.io/understanding-angular-life-cycle-hooks-91616f8946e3
- https://medium.com/bb-tutorials-and-thoughts/angular-understanding-angular-lifecycle-hooks-with-a-sample-project-375a61882478
- https://luukgruijs.medium.com/understanding-creating-and-subscribing-to-observables-in-angular-426dbf0b04a3