Curso Angular 6 Avanzado dia 2

NEORIS

Index

- Input - Output
- Lazy vs Eager
- Pipe
- Observable
- Rxjs

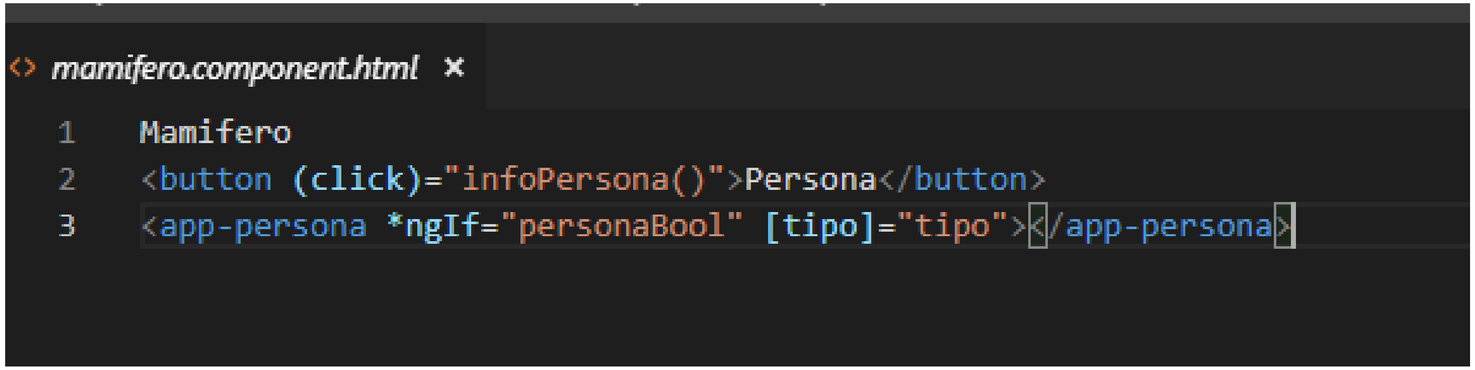
Comunicaicon entre componentes

Es habitual crear un componente por página. Es muy común que esa página se complique. Y la solución a la complejidad es la **división en componentes y reparto de responsabilidades**.
Partiendo de un componente como era el CarComponent vemos que tenía asociadas múltiples responsabilidades:
Obtener los datos desde el *store* (en este caso un simple *array hard-coded*)
Presentar la información realtiva al coche
Responder a los eventos de aceleración y frenado del usuario conductor
Actualizar los indicadores de velocidad y bateria
Las buenas prácticas de arquitectura de software nos obligan a repartir mejor esas responsabilidades. Para empezar vamos a diferenciar los procesos de obtención y manipulación de datos, de los de interacción y presentación de información al usuario.

@Input()

Para que una vista muestre datos tiene que usar directivas como asociada a una propiedad pública de la clase componente. Se supone que dicha clase es la responsable de su valor. Pero también puede **recibirlo desde el exterior**. La novedad es hacer que lo reciba vía *html*.

Empieza por decorar con @Input() la propiedad que quieres usar desde fuera. Por ejemplo



Recuperar la informacion

Para poder recibirlo debe mos anteponer a la varia ble de entrada el prefijo @Input() aquí estará lleg ando la variable que vien e del componente padre

```
export class PersonaComponent implements OnInit {


  @Input() tipo: string;

  constructor() { }

  ngOnInit() {
  }

}
```

Los componentes de nivel inferior no sólo se dedican a presentar datos, también presentan controles. Con ellos el usuario podrá crear, modificar o eliminar los datos que quiera. Aunque no directamente; para hacerlo **comunican el cambio requerido al controlador de nivel superior**.

Ejemplo

1) Persona.ts

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

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


  @Input() tipo: string;

  @Output() public emisior: any = new EventEmitter<any>();

  constructor() { }

  ngOnInit() {
  }

  llamoComponetnePadre() {
    this.emisior.emit('sarasa');
  }


}
```

2) Persona.html

persona.component.html | mamifero.component.ts | persona.component.ts

```html
<h1>{{tipo!.nombre}}</h1>
<h1>{{tipo!.raza}}</h1>

<br>
<button (click)="llamoComponetnePadre()">Pretame</button>
```

3) Mamifero.html

persona.component.html | mamifero.component.ts | persona.component.ts | mamifero.component.html

```html
Mamifero
<button (click)="infoPersona()">Persona</button>
<app-persona *ngIf="personaBool" [tipo]="tipo"    (emisior)="prueba($event)" ></app-persona>
```

4) Mamifero.ts

```ts
  prueba(mensaje) {
    console.log(mensaje);
  }
}
```

ngOnChanges: Este evento se ejecuta cada vez que se cambia un valor de un input control dentro de un componente. Se activa primero cuando se cambia el valor de una propiedad vinculada. Siempre recibe un change data map o mapa de datos de cambio, que contiene el valor actual y anterior de la propiedad vinculada envuelta en un SimpleChange

ngOnInit: Se ejecuta una vez que Angular ha desplegado los data-bound properties(variables vinculadas a datos) o cuando el componente ha sido inicializado, una vez que ngOnChanges se haya ejecutado. Este evento es utilizado principalmente para inicializar la data en el componente.

ngDoCheck: Se activa cada vez que se verifican las propiedades de entrada de un componente. Este método nos permite implementar nuestra propia lógica o algoritmo de detección de cambios personalizado para cualquier componente.

ngAfterContentInit: Se ejecuta cuando Angular realiza cualquier muestra de contenido dentro de las vistas de componentes y justo después de ngDoCheck. Actuando una vez que todas las vinculaciones del componente deban verificarse por primera vez. Está vinculado con las inicializaciones del componente hijo.

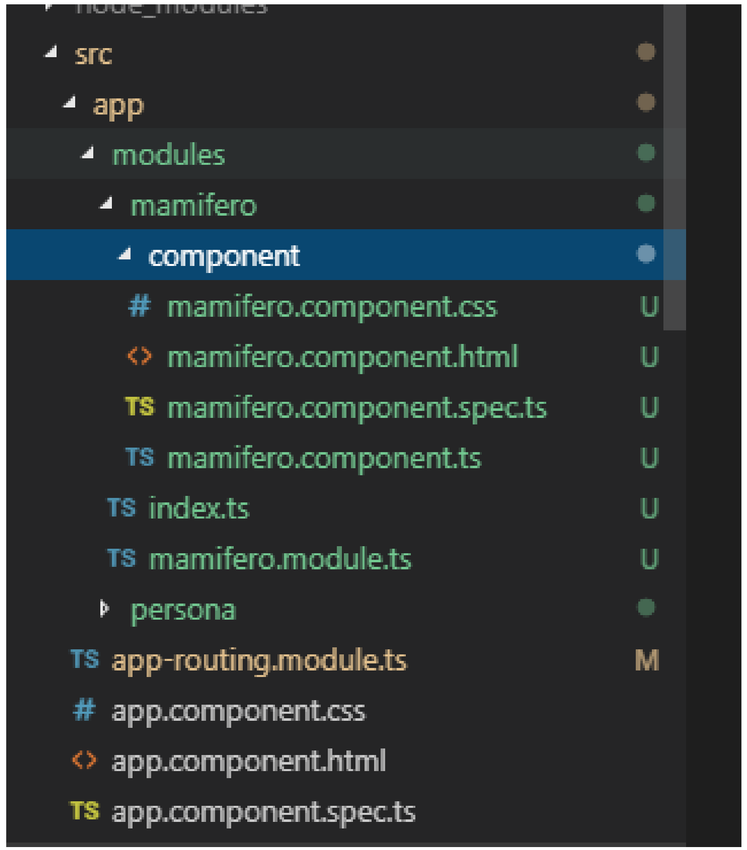
Lazy Loading

¿Qué pasa cuando una aplicación de Angular inicia?
Angular carga todos los componentes que están importados en nuestro módulo principal app.module.ts y puede ser que tarde unos segundos en cargar nuestra aplicación, dependiendo cuantos componentes haya. Para resolver ese problema llegó a nosotros lo que es Lazy Loading.

¿Qué es Lazy Loading?
Es una técnica usada en Angular que nos permite cargar sólo, el o los componentes que necesitemos al inicio de nuestra aplicación, estos componentes no cargan cada vez que entres, sino que solo cargan una sola vez.
Cuando usamos Lazy Loading hacemos llamado de un módulo mediante el sistema de rutas de Angular y este módulo a su vez tiene rutas hijas que se encargan de cargar el componente solicitado por el usuario, más adelante entenderemos esto mejor.

Ejemplo

Una vez creado el componente
creamos una carpeta llamda component
Dentro de ella arrastramos todos los archivos generados.
Posteriormente, creamos en la carpeta r aiz del modulo
un archivo llamado index.ts y su modulo correspondiente



Configuracion del modulo

```typescript
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { Routes, RouterModule } from '@angular/router';
import { MamiferoComponent } from './component/mamifero.component';

const router: Routes =[{
  path: '', component: MamiferoComponent
}]
@NgModule({
  declarations: [MamiferoComponent],
  imports: [
    CommonModule,
    RouterModule.forChild(router)
  ]
})
export class MamiferoModule { }
```

IMPORTANTE: NO OLVIDAR CREAR EL ARCHIVO I NDEX.TS

app-routing.module.ts

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';

const routes: Routes = [{
path: 'mamifero', pathMatch: 'prefix', loadChildren: './modules/mamifero/mamifero.module#MamiferoModule‘ }];

@NgModule({
imports: [RouterModule.forRoot(routes)],
exports: [RouterModule]
})
export class AppRoutingModule { }
```

Modulo Mamifero

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { Routes, RouterModule } from '@angular/router';
import { MamiferoComponent } from './component/mamifero.component';

const router: Routes =[{
path: '', component: MamiferoComponent
}]
@NgModule({
declarations: [MamiferoComponent],
imports: [
CommonModule,
RouterModule.forChild(router)
]
})
export class MamiferoModule { }
```

Pipe

Los pipe nos pueden ayudar para dar cierto formato como tambien para generar filtros customizados sobre la variabl e que estemos aplicando vea mos un ejemplo corto

Mamifero.ts

```
export class MamiferoComponent implements OnInit {

public fecha: Date = new Date(2019, 2, 7);
constructor() { }

ngOnInit() {
}

}
```

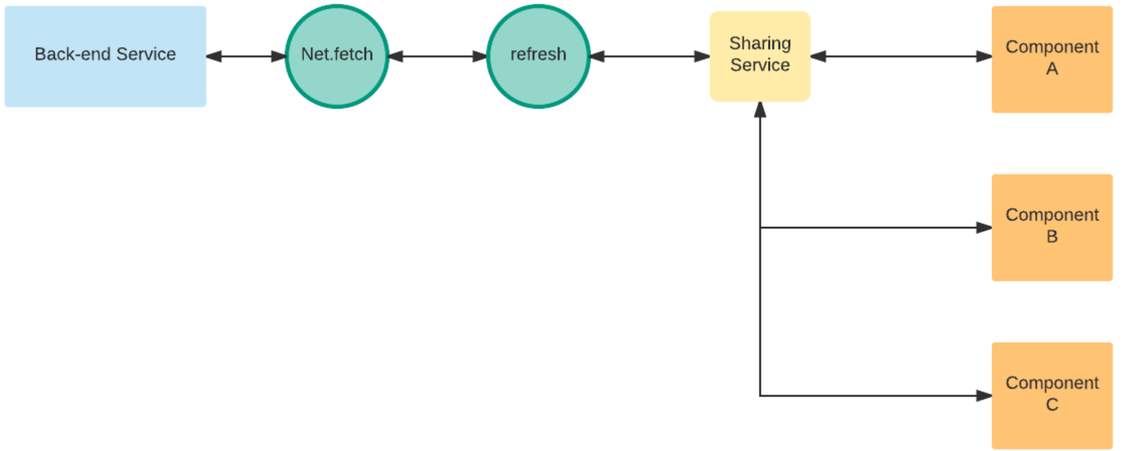
Mamifero.html

```
<h1>dia del curso {{fecha | date:"MM/dd/yy" }}</h1>
```

https://angular.io/guide/pipes

Observable y RXJS

Una práctica común en las aplicaciones de angular para almacenar los datos para después ser utilizados en los diferentes componentes es guardarlo en sus services pues estos son inyectados como singlenton, hasta aquí vamos bien pues de cierta forma los datos no se repiten y tenemos un punto común de acceso a los datos desde todos los componentes. ¿Dónde nos llegan los dolores de cabeza? Es a la hora de propagar el cambio de estos datos a través de todos los componentes, normalmente lo hacemos utilizando los events pero conlleva a un esfuerzo bastante grande y de cierta forma la aplicación queda muy acoplada y frágil ante los cambios futuros ya sea por bugs o nuevos requerimientos. En este articulo intentaremos darle solución a esta problemática utilizando la **reactive programming** en particular la especificación para javascript rxjs.



Para el ejemplo utilizaremos : http://myjson.com/

Configurando el servicio

Nos creamos un servicio y agregamos el siguiente codigo

```
import { Injectable } from '@angular/core';
import { HttpClient, HttpHeaders, HttpResponse } from '@angular/common/http';
import { Observable, of, Observer, BehaviorSubject } from 'rxjs';
import { catchError, map, tap } from 'rxjs/operators';

@Injectable({
providedIn: 'root'
})
export class UrlserviceService {

constructor(public http: HttpClient) { }


public getServicio(): Observable<any>{

const httpOptions = {
headers: new HttpHeaders({
})
};

return this.http.get('https://api.myjson.com/bins/ykaps', httpOptions)
.pipe(
catchError(err => {
return of([]);
}));
}
}
```

Actualizamos el modulo HTTP de ser necesario

```
},
"private": true,
"dependencies": {
  "@angular/animations": "~7.2.0",
  "@angular/common": "~7.2.0",
  "@angular/compiler": "~7.2.0",
  "@angular/core": "~7.2.0",
  "@angular/forms": "~7.2.0",
  "@angular/platform-browser": "~7.2.0",
  "@angular/platform-browser-dynamic": "~7.2.0",
  "@angular/router": "~7.2.0",
  "@angular/http": "^7.1.1",
  "core-js": "^2.5.4",
  "rxjs": "~6.3.3",
  "tslib": "^1.9.0",
  "zone.js": "~0.8.26"
},
"devDependencies": {
```

Dentro del package.json

En el modulo donde trabajaremos con nuestro servicio incorporamos el httpClientModule

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { Routes, RouterModule } from '@angular/router';
import { MamiferoComponent } from './component/mamifero.component';
import { UrlserviceService } from 'src/app/urlservice.service';
import { HttpClientModule } from '@angular/common/http';

const router: Routes =[{
path: '', component: MamiferoComponent
}]
@NgModule({
declarations: [MamiferoComponent],
imports: [
CommonModule,
HttpClientModule,

RouterModule.forChild(router)
],
providers:[UrlserviceService]
})
export class MamiferoModule { }
```

Invocamos a nuestro servicio desde el componente

```
import { Component, OnInit } from '@angular/core';
import { UrlserviceService } from '../../../urlservice.service';

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

public fecha: Date = new Date(2019, 2, 7);
constructor(private service: UrlserviceService) { }

ngOnInit() {

this.service.getServicio().subscribe(data => {
console.log(data.nombre);
});
}



}
```

Dudas y consultas

dante.panella@neoris.com

www.neoris.com

Dante Panella
Master
Dante.panella@neoris.com

NEORIS