Angular Native Fundamentals

Full Stack Bootcamp (Day 3)

Agenda

- Day 1
  - ES6+
- Day 2
  - React Native
- Day 3
  - Angular
- Day 4
  - Springboot
  - SpringData
- Day 5
  - JSON
  - NoSQL
- Day 6
  - Relational
- Day 7
  - Junit
  - Mockito
- Day 8
  - Docker
- Day 9
  - Kubernetes
- Day 10
  - Images and tips

What’s Angular?

- Angular is a framework developed to create web applications that uses TypeScript
- Open Source and mantained by Google
- Based on MCV Model
- Created for SPA (Single Page Applications)

Angular CLI

- Tool to manage , initializate and mantain Angular applications
- Allow to develop testing tasks
- Redeploy our app

```
$ npm install @angular/cli
```

Remember to **sudo** on Linux

Prerequisites

- Knowledge on:
- JavaScript
- HTML
- CSS 3
- TypeScript (Not necessary, you learn a little bit on Day2)

Seting up the environment

- Install NodeJS (we did on Day2)
- Install AngularCLI

```
charly@worker01:~$ npm -version
8.11.0
```

```
charly@worker01:~$ sudo npm install -g @angular/cli
[sudo] contraseña para charly:
npm WARN config global `--global`, `--local` are deprecated. Use `--location=global`
tead.

added 219 packages, and audited 220 packages in 20s

25 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

```
charly@worker01:~$ ng version
? Would you like to enable autocompletion? This will set up your terminal so pressing
TAB while typing Angular CLI commands will show possible options and autocomplete
arguments. (Enabling autocompletion will modify configuration files in your home
directory.) (Y/n)
```

```
Angular CLI: 14.0.4
Node: 16.15.1
Package Manager: npm
OS: linux x64

Angular: undefined
...

Package                      Version
---------------------------------------------------------
@angular-devkit/architect    0.1400.4 (cli-only)
@angular-devkit/core         14.0.4 (cli-only)
@angular-devkit/schematics   14.0.4 (cli-only)
@schematics/angular          14.0.4 (cli-only)
```

Creating the app

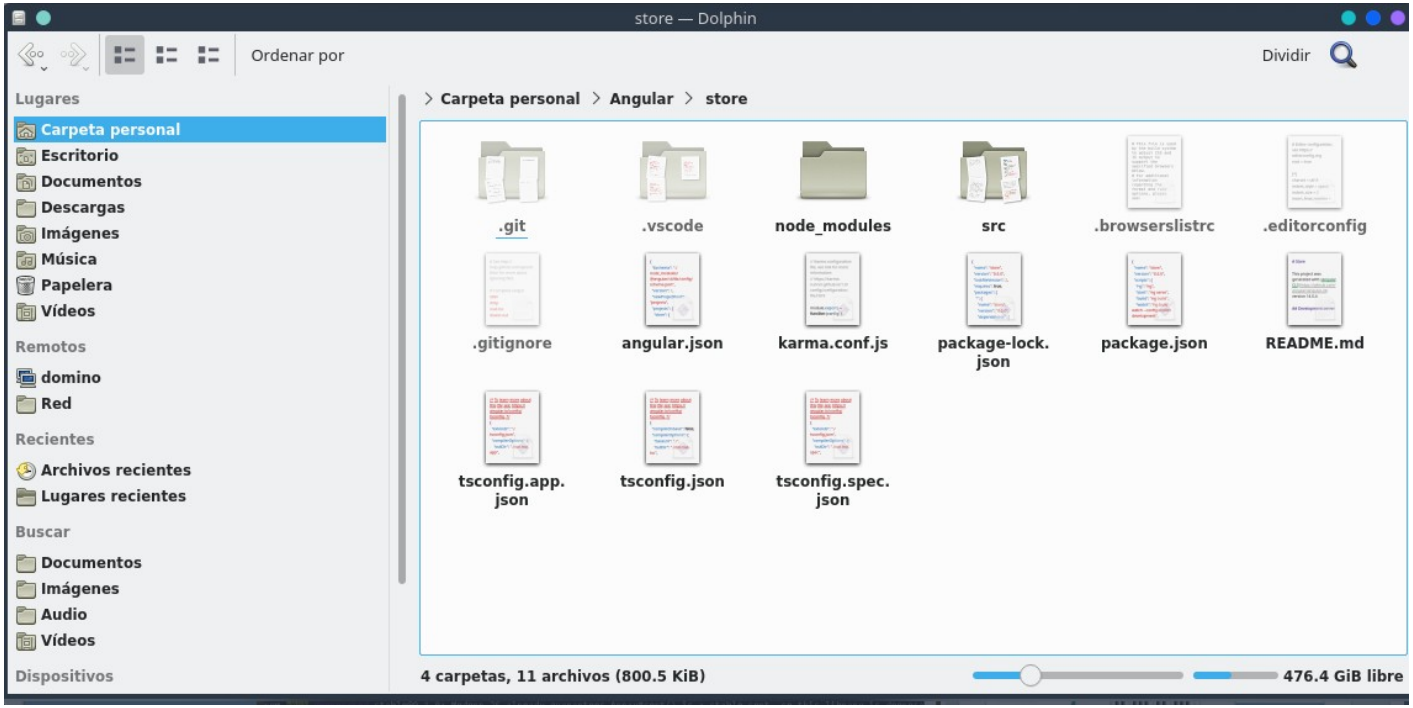
For this day we are going to create one store. To create the project use the ng new command.

```
$ ng new store
```

```
charly@worker01:~/Angular$ ng new store
? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? SCSS   [
https://sass-lang.com/documentation/syntax#scss                ]
CREATE store/README.md (1059 bytes)
CREATE store/.editorconfig (274 bytes)
CREATE store/.gitignore (548 bytes)
CREATE store/angular.json (3091 bytes)
CREATE store/package.json (1036 bytes)
CREATE store/tsconfig.json (863 bytes)
CREATE store/.browserslistrc (600 bytes)
CREATE store/karma.conf.js (1422 bytes)
CREATE store/tsconfig.app.json (287 bytes)
CREATE store/tsconfig.spec.json (333 bytes)
CREATE store/.vscode/extensions.json (130 bytes)
CREATE store/.vscode/launch.json (474 bytes)
CREATE store/.vscode/tasks.json (938 bytes)
CREATE store/src/favicon.ico (948 bytes)
CREATE store/src/index.html (291 bytes)
CREATE store/src/main.ts (372 bytes)
CREATE store/src/polyfills.ts (2338 bytes)
CREATE store/src/styles.scss (80 bytes)
CREATE store/src/test.ts (749 bytes)
```

Understanding files

- Json files on root directory are the one that helps with our project configuration



Package.json

```
1  {
2    "name": "store",
3    "version": "0.0.0",
       ▷ Debug
4    "scripts": {
5      "ng": "ng",
6      "start": "ng serve",
7      "build": "ng build",
8      "watch": "ng build --watch --configuration developmen
9      "test": "ng test"
10   },
11   "private": true,
12   "dependencies": {
13     "@angular/animations": "^14.0.0",
14     "@angular/common": "^14.0.0",
15     "@angular/compiler": "^14.0.0",
16     "@angular/core": "^14.0.0",
17     "@angular/forms": "^14.0.0",
18     "@angular/platform-browser": "^14.0.0",
19     "@angular/platform-browser-dynamic": "^14.0.0",
20     "@angular/router": "^14.0.0",
21     "rxjs": "~7.5.0",
22     "tslib": "^2.3.0",
23     "zone.js": "~0.11.4"
24   },
```

```
25   "devDependencies": {
26     "@angular-devkit/build-angular": "^14.0.4",
27     "@angular/cli": "~14.0.4",
28     "@angular/compiler-cli": "^14.0.0",
29     "@types/jasmine": "~4.0.0",
30     "jasmine-core": "~4.1.0",
31     "karma": "~6.3.0",
32     "karma-chrome-launcher": "~3.1.0",
33     "karma-coverage": "~2.2.0",
34     "karma-jasmine": "~5.0.0",
35     "karma-jasmine-html-reporter": "~1.7.0",
36     "typescript": "~4.7.2"
37   }
38 }
39
```

angular.json

```
13      "root": "",
14      "sourceRoot": "src",
15      "prefix": "app",
16      "architect": {
17        "build": {
18          "builder": "@angular-devkit/build-angular:browser",
19          "options": {
20            "outputPath": "dist/store",
21            "index": "src/index.html",
22            "main": "src/main.ts",
23            "polyfills": "src/polyfills.ts",
24            "tsConfig": "tsconfig.app.json",
25            "inlineStyleLanguage": "scss",
26            "assets": [
27              "src/favicon.ico",
28              "src/assets"
29            ],
30            "styles": [
31              "src/styles.scss"
32            ],
33            "scripts": []
34          },
```

index.html

store > src > index.html > html > head > meta

```html
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Store</title>
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <app-root></app-root>
</body>
</html>
```

app/module.ts

store > src > app > ts app.module.ts

```
1  import { NgModule } from '@angular/core';
2  import { BrowserModule } from '@angular/platform-browser';
3
4  import { AppRoutingModule } from './app-routing.module';
5  import { AppComponent } from './app.component';
6
7  @NgModule({
8    declarations: [
9      AppComponent
10   ],
11   imports: [
12     BrowserModule,
13     AppRoutingModule
14   ],
15   providers: [],
16   bootstrap: [AppComponent]
17 })
18 export class AppModule { }
19
```

app.component.ts

This will be one of the main files that we will use along the course

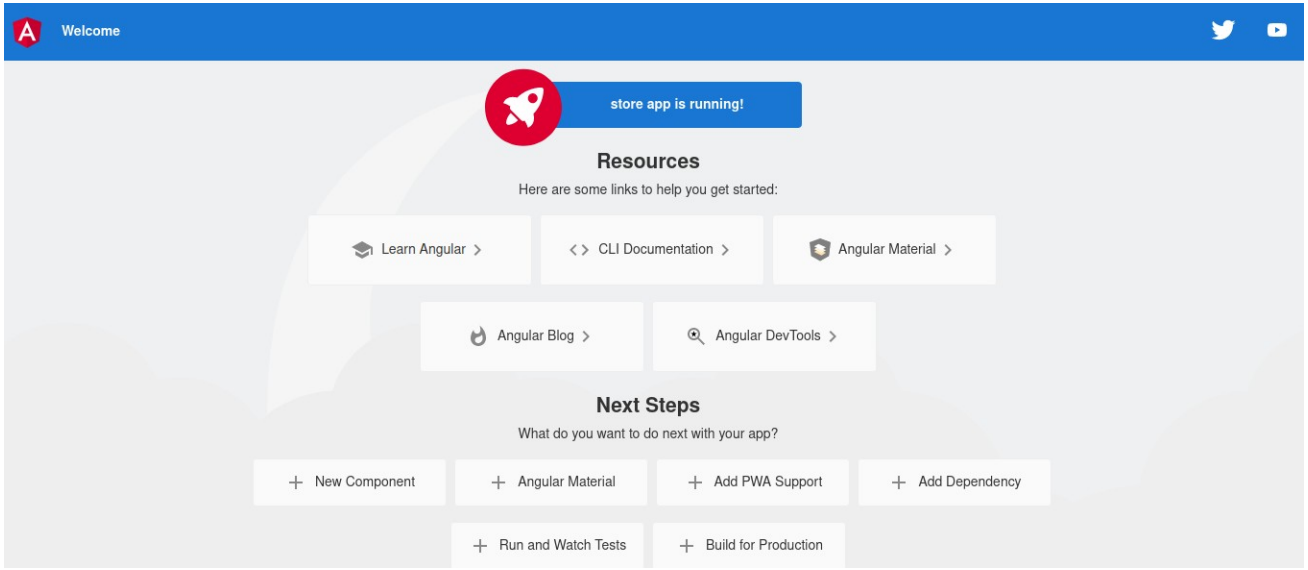
```
1   import { Component } from '@angular/core';
2
3   @Component({
4     selector: 'app-root',
5     templateUrl: './app.component.html',
6     styleUrls: ['./app.component.scss']
7   })
8   export class AppComponent {
9     title = 'store';
10  }
11
```

An Anbgular component is a small part of your application that can be reused when ever you want, it usually has asociated an HTML file where the displaying attributes of the component are set and a css file that helps to create a better presentation for the HTML.
We will talk about the component life cycle ahead.

app.component.html

This file is the one that will render the component

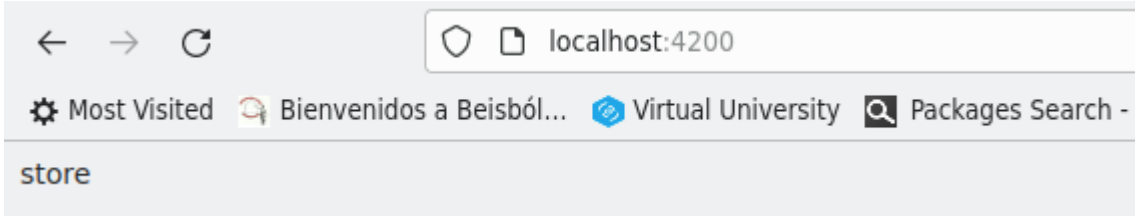
Use ng serve or ng start commands in order to render the main component. Take a few moments to understand what happened and see the HTML on the app.component.html against the HTML displayed



String interpolation

Let’s erase the app.component.html and get the property title from our component

```
1  {{title}}
```



When you call the value of the component variable and put it on the HTML you are doing a string interpolation.

Be aware that your app will automatically update after you modify your component file.

When you do this it’s call one way binding, but when you want to modify the values you may need a two way databinding

One way and two way data binding

In order to create a one way binding you just have to “call” the variable in the right syntax, while a two way databind need a couple of extra steps.

- Import the **`FormsModule`** on your **`app.module.ts`**

```
1  import { NgModule } from '@angular/core';
2  import { FormsModule } from '@angular/forms';
3  import { BrowserModule } from '@angular/platform-browser';
```

```
12  imports: [
13    BrowserModule,
14    AppRoutingModule,
15    FormsModule
16  ],
```

- Create the binding on the right element to create the binding

```
1  {{title}}
2  <input type="text" [(ngModel)] = "title">
```

Exercise 1

Input text and labels

Creating components from CLI

From the command line you can create a new component issuing

```
$ ng generate component shared/component/header
```

It also can be write using CLI alias:

```
$ ng g c shared/header
```

```
charly@worker01:~/Angular/store$ ng g c  shared/component/header
CREATE src/app/shared/component/header/header.component.scss (0 bytes)
CREATE src/app/shared/component/header/header.component.html (21 bytes)
CREATE src/app/shared/component/header/header.component.spec.ts (599 bytes)
CREATE src/app/shared/component/header/header.component.ts (276 bytes)
UPDATE src/app/app.module.ts (555 bytes)
```

Component pieces

```
charly@worker01:~/Angular/store$ ng g c  shared/component/header
CREATE src/app/shared/component/header/header.component.scss (0 bytes)
CREATE src/app/shared/component/header/header.component.html (21 bytes)
CREATE src/app/shared/component/header/header.component.spec.ts (599 bytes)
CREATE src/app/shared/component/header/header.component.ts (276 bytes)
UPDATE src/app/app.module.ts (555 bytes)
```

The component ts:

```
1   import { Component, OnInit } from '@angular/core';
2
3   @Component({
4     selector: 'app-header',
5     templateUrl: './header.component.html',
6     styleUrls: ['./header.component.scss']
7   })
8   export class HeaderComponent implements OnInit {
9
10    constructor() { }
11
12    ngOnInit(): void {
13    }
14
15  }
16
```

Decorator Component

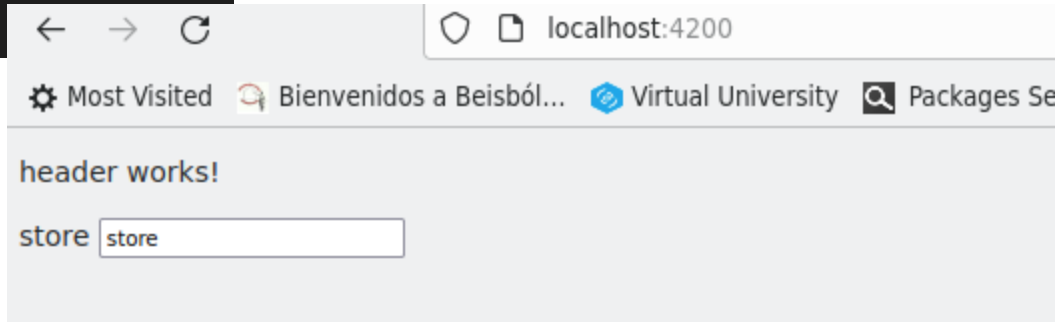
Name of the tag

HTML for the component

Including the component

To include the new component inside our application just use the tag you define on the ts file (in this case app-header).

```
Go to component
1 <app-header></app-header>
2
3 {{title}}
4 <input type="text" [(ngModel)] = "title"/>
```



Creating fake data

To create our store we'll need some fake data, to do that we will install the "json server" using the npm command.

```
$ npm  install -g json-server
```

On your project create a folder called "server" and a file called db.json.
In the file put the data provided. Put the json-server to work in the package.json file.

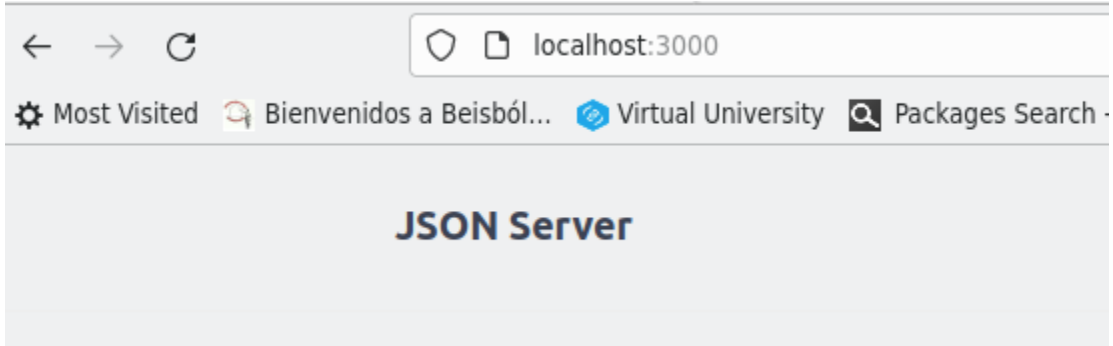
```
▷ Debug
"scripts": {
  "ng": "ng",
  "start": "ng serve",
  "build": "ng build",
  "watch": "ng build --watch --configuration development",
  "test": "ng test",
  "serverAPI": "json-server --watch server/db.json --port 3000"
},
```

To run the fake data server use:

```
$ npm run serverAPI
```

Testing the data

Just use the URL the json-server is running on to see the data that is on the file.



Exercise 2

Puting the data in place

Angular Material

It’s a set of components with high quality design created to have that a versatile way of use and a good set of elements to be used on our applications.
For more information drop by https://material.angular.io/ .
To install angular material use npm, these are called schematics.

```
charly@worker01:~/Angular/store$ ng add @angular/material
 i Using package manager: npm
 ✔ Found compatible package version: @angular/material@14.0.4.
 ✔ Package information loaded.

The package @angular/material@14.0.4 will be installed and executed.
Would you like to proceed? Yes
 ✔ Packages successfully installed.
 ? Choose a prebuilt theme name, or "custom" for a custom theme:
   Indigo/Pink        [ Preview: https://material.angular.io?theme=indigo-pink ]
   Deep Purple/Amber  [ Preview: https://material.angular.io?theme=deeppurple-amber ]
 ❯ Pink/Blue Grey     [ Preview: https://material.angular.io?theme=pink-bluegrey ]
   Purple/Green       [ Preview: https://material.angular.io?theme=purple-green ]
   Custom
```

```
 ? Set up global Angular Material typography styles? (y/N)
```

```
 ? Include the Angular animations module? (Use arrow keys)
 ❯ Include and enable animations
   Include, but disable animations
   Do not include
```

Create a module manually

Hen creating a module we define their porperties inside a description file that usually is left on the **app** directory

```
1  import { NgModule } from "@angular/core";
2  @NgModule({
3
4      exports:[]
5  })
6
7  export class MaterialModule{}
```

You also need to add it to the modules section of your app.module.json

```
imports: [
  BrowserModule,
  AppRoutingModule,
  FormsModule,
  BrowserAnimationsModule,
  MaterialModule
],
```

Using Material components

On the module you created define all the material components you will use, in this case we are going to add one by one according the need of our project. Starting with the material toolbar.

src > app > ts Material.module.ts > ...

```
1  import { NgModule } from "@angular/core";
2  import {MatToolbarModule} from "@angular/mat
3
4  @NgModule({
5
6      exports:[MatToolbarModule]
7  })
8
9  export class MaterialModule{}
```

Modify the MaterialModule to incorporate the ToolbarModule

src > app > shared > component > header > <> header.component.html

Go to component

```
1  <mat-toolbar color="primary">
2      <span>My Store</span>
3    </mat-toolbar>
4
```

Change the header component to create the toolbar for our project

Template inline

If you dont’ want to have a lot of files per module or component you can use the templete inline, that is to use the properties template instead templeteUrls and Style instead StyleUrls

```
src > app > shared > component > header > ts header.component.ts >
1   import { Component } from '@angular/core';
2
3   @Component({
4     selector: 'app-header',
5     template: `
6     <mat-toolbar color="">
7           <span>My Store</span>
8         </mat-toolbar>
9         `,
10    styleUrls: ['./header.component.scss']
11  })
12  export class HeaderComponent {}
13
```

When using template inline you will not use apostrophes but “accents” characters (``), in this example you can erase the header. component.html file since will be ignored.

Exercise 3

Add Angular Material

Directives and routes

Directives are a way to handle attributes among one DOM element that can change their appearance or behaivour

- Structural
- Attributes
- Custom Directives
- Components (Directives with templates)

Routes are the specification on what should be displayed if a URL path is equal to the specified for the route.

Creating a route and a component manually

On certain applications we want to put the components exactly on a specific position on the screen. To do that you may need a component, and then create the entry on the **app-routing.module.ts** file and put the routing component on the **app.component.html**

src > app > pages > TS About.component.ts > AboutComponent

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

@Component({
    selector:"app-about",
    templateUrl:"./About.component.html",
    styleUrls:[]
})

export class AboutComponent implements OnInit{
    constructor(){}
    ngOnInit(): void {

    }
}
```

1

src > app > pages > About.component.html > p

Go to component

```
<p>Hello World!</p>
```

2

src > app > TS app-routing.module.ts > routes > component

```
import { AboutComponent } from './pages/About.comp
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/rou

const routes: Routes = [
  {path:"about",
   component: AboutComponent}
];

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

3

src > app > app.component.html > router-outlet

Go to component

```
<app-header></app-header>
<router-outlet></router-outlet>
```

4

Routes for non existing paths

Just ad a route with a “**” value for the path property. You can also use the redirect property to set a landing page (for example the home page)

```
const routes Routes=[
        {path:”about”, component:AboutComponent}
        {path:”**”, redirectTo:””, pathMatch:”full”}
]
```

Exercise 4

Add routing

Modules from CLI

To create a module from the CLI just use the `ng g m` or `ng generate module` command

```
$ ng g m pages/products -m=app –-route products
```

Interfaces

Example

src > app > pages > products > interfaces > TS product.interface.ts > Product > stock

```
1   import { AppRoutingModule } from "src/app/app-routing.module";
2
3   export interface Product{
4       id: number;
5       name: string;
6       price: number;
7       description: string;
8       categoryId: number;
9       stock: number;
10  }
```

Services

Services are Angular services are objects that get instantiated just once during the lifetime of an application. They contain methods that maintain data throughout the life of an application, i.e., data is available all the time

```
$ ng g s pages/products/services/products
```

HttpClient

The HttpClient on Angular provides the facility to create request to API servers, in order to use it declare it on your app.module.ts and use it in your code

```
10  import { HttpClientModule} from "@angular/common/http"
11
12  @NgModule({
13    declarations: [
14      AppComponent,
15      HeaderComponent
16    ],
17    imports: [
18      BrowserModule,
19      AppRoutingModule,
20      FormsModule,
21      BrowserAnimationsModule,
22      MaterialModule,
23      HttpClientModule
24    ],
```

src > app > pages > products > services > ts products.service.ts > ProductsService

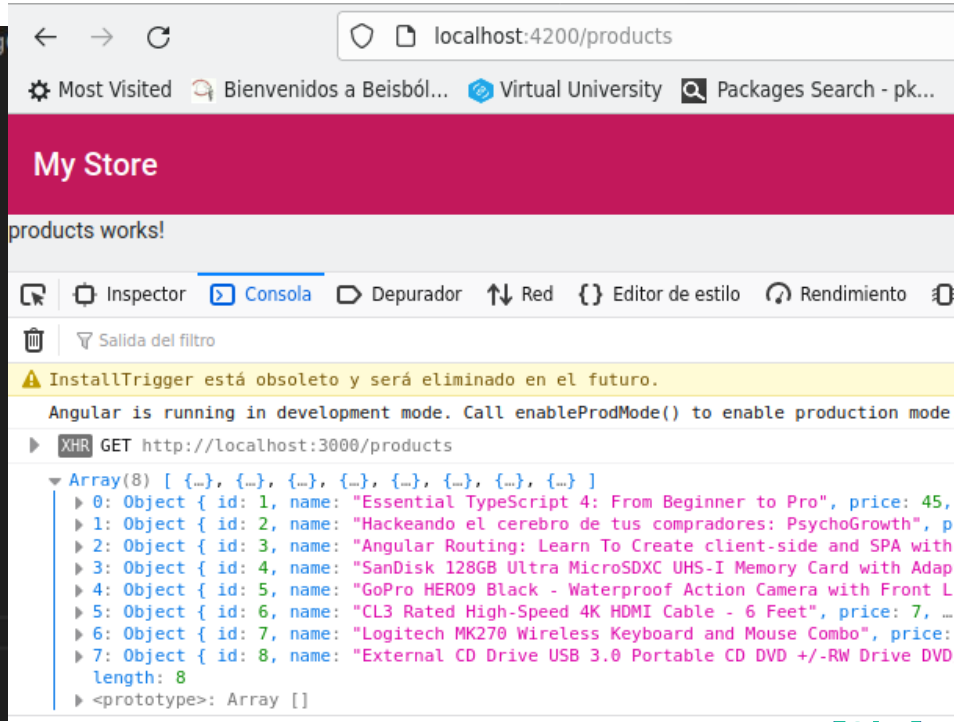
```
1   import { Injectable } from '@angular/core';
2   import { HttpClient } from '@angular/common/http';
3   import { Observable } from 'rxjs';
4   import { Product } from '../interfaces/product.interface';
5
6   @Injectable({
7     providedIn: 'root'
8   })
9   export class ProductsService {
10    private jsonServerURL="http://localhost:3000/products";
11    constructor(private httpClient: HttpClient) { }
12
13      getProducts(): Observable<Product[]> {
14        return this.httpClient.get<Product[]>(this.jsonServerURL);
15      }
16  }
```

Using a service

In the component where you are going to use the service start with the import, then call the method you need. Example:

src > app > pages > products > TS products.component.ts > ProductsComponent > ng

```
1   import { Component, OnInit } from '@angular/core';
2   import { ProductsService } from './services/products.service';
3   import { tap } from 'rxjs';
4
5   @Component({
6     selector: 'app-products',
7     templateUrl: './products.component.html',
8     styleUrls: ['./products.component.scss']
9   })
10  export class ProductsComponent implements OnInit {
11
12    constructor(private queryService: ProductsService) { }
13
14    ngOnInit(): void {
15      this.queryService.getProducts().pipe(
16        tap(res => console.log(res))
17      ).subscribe();
18    }
19  }
```



Exercise 5

- Services, components http

Using the data in the component

- The variable can be used within the component that has declared it using the String Interpolation
- If the variable is an array some looping directives can be used such as the *ngfor
- Angular Material components can be used to render data with improved presentation

rc > app > pages > products > TS products.component.ts > ...

```
1  import { Component, OnInit } from '@angular/core';
2  import { ProductsService } from './services/products.service';
3  import { tap } from 'rxjs';
4  import { Product } from './interfaces/product.interface';
5
6  @Component({
7    selector: 'app-products',
8    templateUrl: './products.component.html',
9    styleUrls: ['./products.component.scss']
10 })
11 export class ProductsComponent implements OnInit {
12   productList !: Product[];
13
14   constructor(private queryService: ProductsService) { }
15
16   ngOnInit(): void {
17     this.queryService.getProducts().pipe(
18       tap((actualProducts : Product[]) => this.productList=actualProducts)
19     ).subscribe();
20   }
21 }
```

src > app > pages > products > products.component.html > section.products > mat-card.card > r

Go to component

```
1  <section class="products">
2      <mat-card class="card" *ngFor="let product of productList">
3          <mat-card-header>
4              {{product.name}}
5          </mat-card-header>
6          <mat-card-content>
7              {{product.description}}<br>
8              $ {{product.price}}
9          </mat-card-content>
10         <mat-card-actions>
11             <button mat-flat-button color="primary">Add to the cart</button>
12         </mat-card-actions>
13     </mat-card>
14 </section>
```

Pipes

Pipes are a way to transform data or format the data inside the components

```
6    <mat-card-content>
7        {{product.description}}<br>
8        {{product.price | currency}}
9    </mat-card-content>
```

- **CurrencyPipe**
- **DatePipe**
- **DecimalPipe**
- **JsonPipe**
- **LowerCasePipe**
- **UpperCasePipe**
- **PercentPipe**
- **SlicePipe**
- **AsyncPipe**

Communication between components

Let's define a communication parent/child when the "parent" document calls the "child" to render it. When that happens then the way to pass the data between the components is using @Input decorator.
When the "child" components ask a complete new component to render (parent) and the parent needs the child data, then the @Output decorator is used.

```
14  export class ProductComponent implements OnInit
15
16    @Input() product !: Product;
17    constructor() { }
```

src > app > pages > products > product > product.component.html > mat-card.card >

Go to component

```
1   <mat-card class="card">
2       <mat-card-header>
3           {{product.name}}
4       </mat-card-header>
5       <mat-card-content>
6           {{product.description}}<br>
7           {{product.price | currency}}
8       </mat-card-content>
9       <mat-card-actions>
10          <button mat-flat-button color="primary">Add to the cart
11              <mat-icon>shopping-basket</mat-icon>
12          </button>
13      </mat-card-actions>
14  </mat-card>
```

src > app > pages > products > products.component.html > section.products

```
1   <section class="products">
2           <app-product [product]="product" *ngFor="let product of productList "></app-product>
3   </section>
```

@Output

For the @Output decoration there will be a similar type of connection

```
14  export class ProductComponent implements OnInit {
15
16    @Input() product !: Product;
17    @Output() addToCartClick = new EventEmitter<Product>();
18    constructor() { }
19
20    ngOnInit(): void {
21    }
22
23    onClick():void{
24      this.addToCartClick.emit(this.product);
25
26    }
```

1

src > app > pages > products > products.component.html > section.products >

```
1  <section class="products">
2          <app-product
3          (addToCartClick)="addToCart($event)"
4          [product]="product"
5          *ngFor="let product of productList "></app-product>
6  </section>
```

1
2

@Output (cont...)

With a Service you can use the values and update them in the target component (in this case the header)

src > app > shared > services > ts cart.service.ts > CartService > cartTotal

3

```
import { Injectable } from "@angular/core";
import { TitleStrategy } from "@angular/router";
import { Observable, Subject } from "rxjs";
import { Product } from "src/app/pages/products/interfaces/product.interface";
@Injectable({providedIn:'root'})
export class CartService{
    products :  Product[] = [];
    private cartSubject = new Subject<Product[]>();
    private totalSubject = new Subject<number>();
    private quantitySubject = new Subject<number>();
    get cartAction$() : Observable<Product[]>{
        return this.cartSubject.asObservable();
    }
    get totalAction$() : Observable<number>{
        return this.totalSubject.asObservable();
    }
    get quantityAction$() : Observable<number>{
        return this.quantitySubject.asObservable();
    }
    private cartTotal():void{
        const total=this.products.reduce((acc, product)=> acc+=product.price, 0)
        this.totalSubject.next(total);
    }
    private productsQuantity():void{
        const total = this.products.length;
        this.quantitySubject.next(total);
    }
    private addToCart(product:Product):void{
        this.products.push(product);
        this.cartSubject.next(this.products);
    }
    newProductToTheCart(product:Product):void{
```

src > app > shared > component > header > ts header.component.ts > HeaderCor

4

```
import { Component } from '@angular/core';
import { CartService } from '../../services/cart.service';

@Component({
  selector: 'app-header',
  template: `
  <mat-toolbar color="primary">
        <span>My Store</span>
        {{quantity | async | json}}
        {{total | async | currency}}
    </mat-toolbar>
      `,
  styleUrls: ['./header.component.scss']
})
export class HeaderComponent {
  quantity = this.cartService.quantityAction$;
  total =  this.cartService.totalAction$;
  cart = this.cartService.cartAction$;
  constructor (private cartService : CartService){}
}
```

*ngIf

We can use a condition to display an ngcontainer component with the directive *ngIf: A structural directive that conditionally includes a template based on the value of an expression coerced to Boolean. When the expression evaluates to true, Angular renders the template provided in a then clause, and when false or null, Angular renders the template provided in an optional else clause. The default template for the else clause is blank.

Creating cart indicator

Once we have the products a cart service can be used along with the oolbar ro create a component thet will show the number of items in the cart and the total.

src > app > shared > component > header > TS header.component.ts > HeaderCo

```
1   import { Component } from '@angular/core';
2   import { CartService } from '../../services/cart.service';
3
4   @Component({
5     selector: 'app-header',
6     template: `
7     <mat-toolbar color="primary">
8         <span>My Store</span>
9         <span class='spacer'></span>
10        <app-cart></app-cart>
11      </mat-toolbar>
12        `,
13    styleUrls: ['./header.component.scss']
14  })
15  export class HeaderComponent {
16    quantity = this.cartService.quantityAction$;
17    total =  this.cartService.totalAction$;
18    cart = this.cartService.cartAction$;
19    constructor (private cartService : CartService){}
20  }
```

src > app > shared > component > header > header.component.scss

```
1   .spacer{
2       flex: 1 1 auto;
3   }
```

Creating cart indicator (2)

src > app > pages > cart > TS cart.component.ts > CartComponent

```
1   import { NgIf } from "@angular/common";
2   import { Component } from "@angular/core";
3   import { CartService } from "src/app/shared/services/cart.service";
4
5   @Component({
6       selector:'app-cart',
7       template:`
8       <ng-container *ngIf = '{total : total$ | async , quantity : quantity$ |async} as dataCart'>
9           <ng-container *ngIf = 'dataCart.total'>
10              <mat-icon>shoppig_cart</mat-icon>
11                  {{total$ | async | currency}}
12                  {{quantity$ | async | json}}
13          </ng-container>
14      </ng-container>
15      `
16  })
17  export class CartComponent{
18      quantity$  = this.cartService.quantityAction$;
19      total$ = this.cartService.totalAction$;
20      constructor(private cartService : CartService){}
21  }
```

Exercise 7

Creating the start of the cart.

Next Steps

- Create a page similar to the products page to show the cart list
- Make a form for the checkout

You already have the habilites to create the complete store, but some advanced topics could be helpful. Please consider learning about Reactive programming and the RxJS framework.
Review all the components of Angular Material and play with them, the more components you know the best graphical interface you will create
Learn about designing and things like color theory.

Day 3 summary

Angular Introduction – What's Angular, What can you do with Angular.
TypeScript and prerequisites– What's TypeScript and how to work with it
Installation– How to setup our environment.
Components– What's a component and how to create one.
Core components– What are the core components of Angular.
Angular Material– Introduction to Angular Material and their components.
Flexbot– learn how and when to use Flex.
Services – How to have services that interact with the components.
Interfaces– Learn what are andhow to create interfaces.
HttpClient– Learn how to consume REST services
Async variables – Learn how to handle async variables
States– Learn how to set state variables
Next steps– What else to learnt