

# Angular

Dec 2021

**By: Feven Tadesse**  
**Title: Trainer at the coding school**



# About ODC

## Coding School

it is a technology center that offers **free technical and soft skills training on real projects with social impact and proposes entertainment** for the developer community and young entrepreneur

**Target** : students, graduates or not, young entrepreneurs, young in professional reconversion

## Orange Digital Center Club

Subsidiary of the coding school, technological centers are deployed at **universities** offering training, events and projects support for students.

**Target** : students of the university and students clubs

## FabLab solidaire

It is a digital manufacturing space to create and **prototype** with digital equipment : 3D printers, milling machines, laser cutters ...

**Target** : students, young graduates, young entrepreneurs

## Orange Fab

Start-up accelerator preparing for **national and international business partnerships** with the Orange Group and the worldwide Orange Fab network.

**Target** : entrepreneurs

## Orange Venture Africa

Investment fund with € 50 million to **finance innovative start-ups** on the African continent (fintech, e-health, energy, edutech, govtech).

**Target** : entrepreneurs



# **BASELINE SURVEY**

# Rules For Success!

## **INTERACT**

Practice is the best way to learn. So do all the exercises along the way

Ask questions and explore more than what's on the slide

Check out documentations for the technologies covered

# Objective

- ✓ To introduce programmers to the popular front end development framework
- ✓ To discuss the building blocks of Angular
- ✓ To discuss best practices when considering Angular for development
- ✓ To enable learners to become professional Angular framework developers

# Outline



## Getting Started

- ✓ What is Angular?
- ✓ Why Angular?
- ✓ Setup



## Building Blocks Of Angular

- ✓ Components
- ✓ Templates
  - Event and Two Way Binding
  - Loops



## Time To Travel

- ✓ Routing
- ✓ Forms



## Mini Project



# Ice Breaker :)

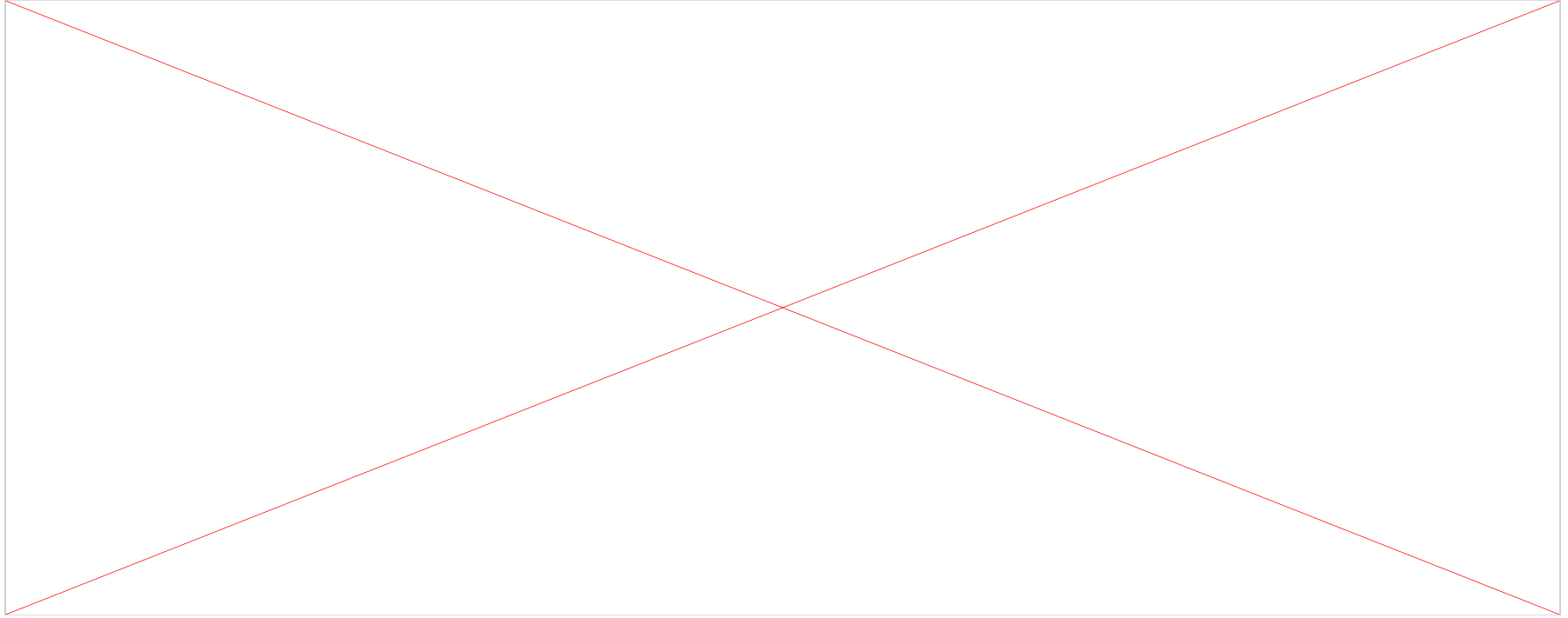
## Question ?

1. What is your favorite website? What is it made of?
2. What do you like about it? The design / The experience
3. What technologies have you used before? Have you ever used angular?



# Let's see some development test

What does this code do?





# GETTING STARTED

# Our Mini Project

We will develop **our very first Ecommerce site with Angular.**

It will have products and customers pages.

We will work on products page together. Then you will implement it on customers page.

**You will be working on**

- **Customers component**
  - **This component should allow to list down all the lists of customers**
  - **It should have a “View Customer” button to show me customer detail**
  - **It should allow me to add new customer**
  - **It should also allow me to update and delete any customer I want.**



# What is Angular ?

- Open source front end web application framework
- Creating efficient and sophisticated single-page apps
  - 3,600,000+ Angular websites
  - 100,000+ professionals on LinkedIn
- Depends on node and npm for many features
- Last but not least Angular CLI (Command Line Interface)



# Why Angular?

Backed up by **google**

Component-based architecture, cleaner code

High performance and high scalability

Best of all, the Angular ecosystem consists of a diverse group of over

**1.7 million developers, library authors, and content creators.**

Who uses Angular?



# Where do I start?

We're gonna do a small **ECOMMERCE** site

Let's create our first angular project!

**ng new my-ecommerce-app**

(Please say yes to the questions Angular is asking!)

Let's run our project!

**cd my-ecommerce-app** ( To navigate to project folder)

**ng serve --open** (This command will run our angular project)

**Has everyone served their project?**



# Project Structure

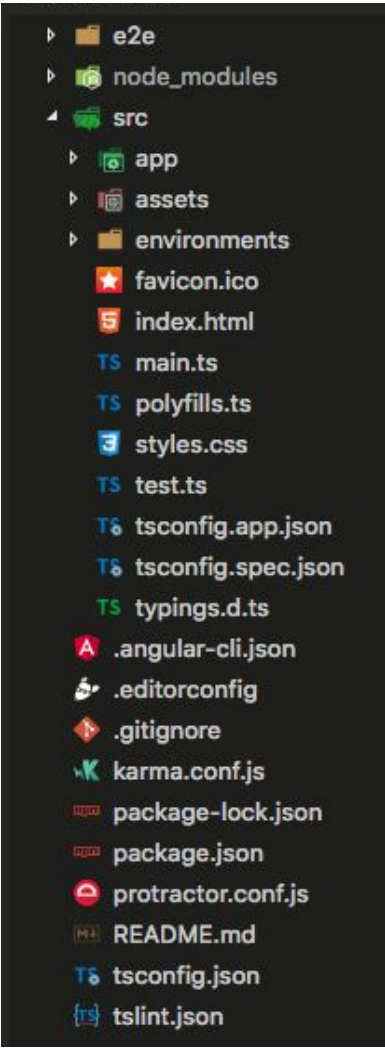
A closer look at project structure:

`node_modules` - where we find all the libraries

`src/app` - where we flesh out our projects

`src/assets` - where we can find all our images and public files

`package.json` - where we find all our dependencies



```

└─ e2e
└─ node_modules
└─ src
  └─ app
  └─ assets
  └─ environments
    └─ favicon.ico
    └─ index.html
    └─ main.ts
    └─ polyfills.ts
    └─ styles.css
    └─ test.ts
    └─ tsconfig.app.json
    └─ tsconfig.spec.json
    └─ typings.d.ts
  └─ .angular-cli.json
  └─ .editorconfig
  └─ .gitignore
  └─ karma.conf.js
  └─ package-lock.json
  └─ package.json
  └─ protractor.conf.js
  └─ README.md
  └─ tsconfig.json
  └─ tslint.json

```

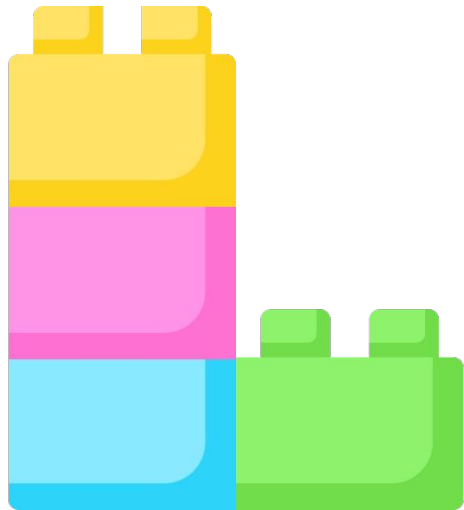
# **BUILDING BLOCKS -COMPONENTS-**

# Components

**Components** - Building blocks of angular apps

Each component has html, typescript and css

- An **HTML template** that declares what renders on the page
- A **Typescript class** that defines behavior
- A **CSS selector** that defines how the component is used in a template
- A **testing specification file** that is used for testing the components





# Cont ...

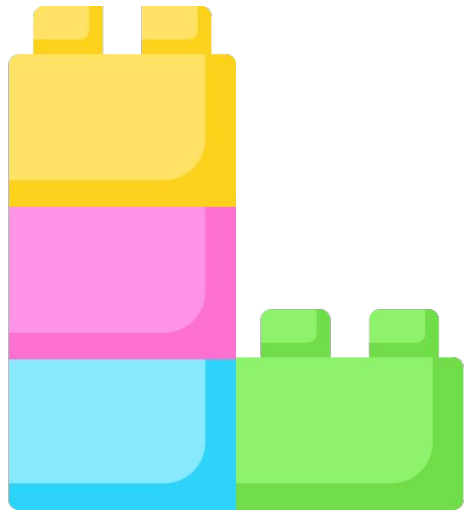
- 1) On terminal navigate to 'app' of your project
- 2) Run **ng generate component <component-name>**

Let's create **Products Component**

Three properties generated: **selector, templateUrl, styleUrls**

**How do we render the products components?**

Let's add **<app-products></app-products>** and **<router-outlet></router-outlet>** to our app component and see what happens!



# Component Lifecycle

What else do you see in the product component?

What is the use of `ngOnInit()` method?

`ngOnInit()` is called just when the component is initiated

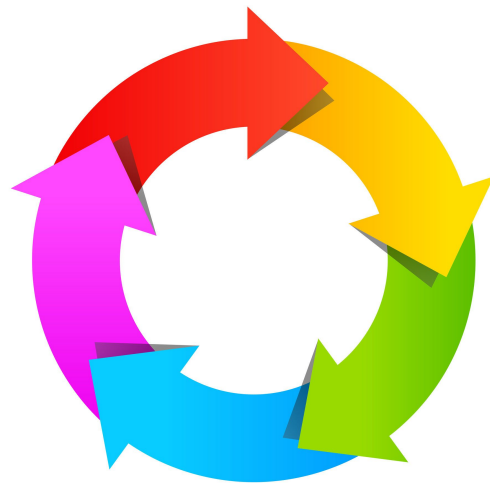
`ngOnDestroy()` is called when the components is destroyed

Let's test out `ngOnInit()` method:

```
ngOnInit() {console.log ('Method called')}
```

There are many other component lifecycle methods which you can find here:

<https://angular.io/guide/lifecycle-hooks>



# **BUILDING BLOCKS -TEMPLATES-**

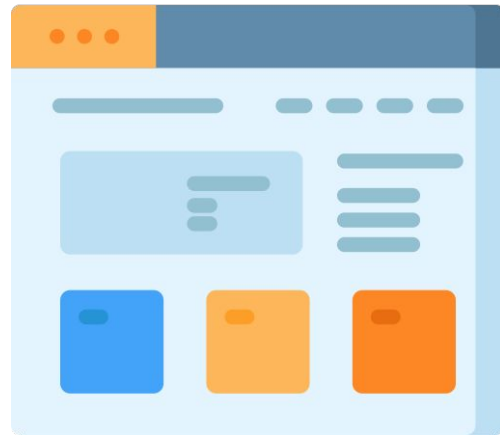
# Let the UI shine !

## Small Exercise

Let's give one of our product a name "Nike" and a price of \$100  
And add a button below with a label "view Detail" with an orange color

## Do I Need Bootstrap for my UI?

(<https://getbootstrap.com/docs/5.0/getting-started/introduction/>)



**npm install bootstrap**

**OR**

**Include the css link to the index.html file**

# Cont ...



## Nike

Some quick example text to build on the card title and make up the bulk of the card's content.

[View Detail](#)

## Puma

Some quick example text to build on the card title and make up the bulk of the card's content.

[View Detail](#)

## Sketchers

Some quick example text to build on the card title and make up the bulk of the card's content.

[View Detail](#)

# Interpolation

So far everything displayed is coded in the `products.component.html`.

But where should this information come from?

The `products.component.ts`

Let's try this simple example:

- 1) Let's create a variable called "productName" in the `product.component.ts`
- 2) Let's give it a value of "Nike"
- 3) Now let's head to `product.component.html` and use this variable

**`{{productName}}`**

This is how we use it. This is called `interpolation`.

# Interface

But a real product has more than just a name. So let's create an **class**

Create a file in the src/app/product folder with a name 'product.object.ts'. Give the product an id, name, price and description properties

```
export class Product{
  id: number;
  name: string;
  price: number;
  imageUrl: string;
  description: string;
}

constructor(){this.id = null;
               this.name = '';
               this.price = null;
               this.imageUrl = '';
               this.description = '';
```

Now let's create a variable "product" of type Product in our Products Component. And give it a value  
**How do we do that?**

# Cont ...

**This is how my product.component.ts looks like:**

// Option 1

```
product = new Product();  
product.id = 1;  
product.name = 'Nike';  
product.price = 2000;  
product.imageUrl = '../assets/img.png';  
product.description = 'you can read more ';
```

//Option 2

```
product: Product = {  
  id: 1,  
  name: 'Nike',  
  price: 2000,  
  imageUrl: '../assets/img.png',  
  description: 'you can read more'  
}
```



# Cont ...

Now if we're able to create our product variable successfully, we can then render this variable in our html. **Guess how?**

This is how my **product.component.html** looks like:

```
<div class="card" style="width: 18rem;">
  
  <div class="card-body">
    <h5 class="card-title">{{product.name}}</h5>
    <p class="card-text">{{product.description}}</p>
    <a href="#" class="btn btn-primary">View Details</a>
  </div>
</div>
```

# Cont ...

What we saw above is one option.

Let's see option 2!

Let's refactor our `<img>` tag as below:

```
<img [src]="product.imageUrl" class="card-img-top" alt="productImg">
```

What we just did is called **property binding**

Property binding in Angular helps you set values for properties of HTML elements or directives

# Pipes

Shall we format the name of our product with **UpperCasePipe**? What is Pipe?

**Pipes** are a good way to format strings, currency amounts, dates and other display data

Angular ships with several built-in pipes and you can also create your own

```
{{product.name | uppercase}}
```

You can find all the pipes of Angular and usage examples here:

<https://angular.io/guide/pipes>



# It's getting hot !

Our application is looking great so far! But our **View Detail** button is doing nothing. Let's give it some work.

What is **Event Binding**?

Event binding lets you listen for and respond to user actions such as keystrokes, mouse movements, clicks, and touches.

Listen to an event (click) and then do something (call **onViewDetail** function)

```
<button class="btn btn-primary" (click)="onViewDetail(product.id)">View Detail</button>
```

Where is this **onViewDetail** function defined?

# Cont ...

The onViewDetail function is defined inside the product.component.ts.

What do you think this function should do?

For now let's tell this function to log out the id of the product which is the parameter it is taking.

```
onViewDetail(id: number){  
    console.log(id);  
    // when we learn routing we will route(move) to a different component  
    // (product detail component) with this id  
}
```

# Cont ...

Don't you want to edit your product name?

Let's talk about **two-way data binding**

For now let's allow users to be able to edit the product name in an `<input>` textbox

Data flows from the **component class out to the input tag and from input value to the component's variable.**

Let's see how we do this in code

# Cont ...

Let's add this new line to our products.component.html:

```
<h5 class="card-title">{{product.name}}</h5>  
<input [(ngModel)]="product.name" placeholder="name"/>
```

[(ngModel)] is Angular's two-way data binding syntax

Did your app crash?

Let's import **FormsModule** in our app.module.ts

Have you noticed **declarations** array in our app.module.ts?

# A Little Deeper :)

Don't you think one product is a little too small for an ecommerce site ! Let's add more products

Let's create a list of products in our product.object.ts file:

```
export const PRODUCTS: Product[] = [  
  { id: 1, name: 'Nike', imageUrl: '../assets.nike.jpg'},  
  { id: 2, name: 'Puma', imageUrl: '../assets.puma.jpg'},  
  { id: 3, name: 'Skechers', imageUrl: '../assets.skechers.jpg'}  
];
```

Let's create an instance of this array in our products component:

```
productList = PRODUCTS;
```





# Cont ...

Let's list products in html with `*ngFor`

```
<div class="card" *ngFor="let product of productList" style="width: 18rem;">
```

If price is less than 10 then we're on discount so why don't we change the color of the price to **red** if it is less than 10.

Let's do the following refactor:

```
<p class="card-text" *ngIf="product.price < 10">$ {{product.price}} (*On Discount)</p>
```

Now what if I want to see all the properties of a product and update them?

Let's talk about our “View Detail” button

# Mini Project

For the customers page you will:

- Create customers component to view lists of customers
- Create a customer object that contains all fields necessary for a customer
- Create list of customers inside your customer.object.ts file
- Your html file should loop through this list of customers
- If our customer has been with us for more than 2 years change the color of the name to orange

# TIME TO TRAVEL

# Routing

We want to travel from one view (product list view) to another (product detail view)

To handle the navigation from one view to the next, you use the Angular **Router**

So let's use angular router to route to edit product component when “editProduct” function is called:

Let's add this line of code:

```
constructor(private router: Router){ }
```

And refactor our “editProduct” function:

```
editProduct(id: number){  
    this.router.navigateByUrl('/products/edit/' + id); // 'id' is called parameter  
}
```



# Cont ...

So let's go ahead and create our edit product component.

**How can we create this component?**

But did we tell our program to navigate to edit product component when it finds `‘/products/edit/id’` route?

Let's go to `app.routing.module.ts` and take a look at `Routes` array. What do you think happens here?

Let's go ahead and define some routes:

```
const routes: Routes = [  
  {  
    path: products/edit/:id',  
    component: EditProductComponent  
  },  
];
```

# Cont ...

Finally we can successfully route to our edit product component!

Here is what we want to do in the edit product component:

1. We want to get the id from the active (current) route
2. We want to loop through the list products which is found inside (PRODUCTS) array to find the specific product
3. We want to display the product details in the `editProduct.component.html`
4. We also need to have `Submit button` so that I can edit the product (We will deal with this a little later)

# Cont ...

1. So first let's get the id from the current api

## Getting Route Information

To get information from a route we use an interface called **ActivatedRoute**

So how do we inject this interface in our product component?

```
constructor(private route: ActivatedRoute){ }
```

# Cont ...

Now let's use our `ActivatedRoute` to get our id from the route.  
Where should we write this logic?

Let's refactor our edit product component:

(For now let's just log out our id)

```
ngOnInit() {  
    this.route.params.subscribe(  
        param => {  
            Console.log (param.id);  
        }  
    );  
}
```



# Cont ...

## 2. Let's loop through the list products which is found inside (PRODUCTS) array to find the specific product

Here we need to initialize two variables:

- `products = PRODUCTS; // List of products`
- `selectedProduct = new Product(); // the selected product`

Now what we want to do is:

- Look for that product inside the list which is now in the “products” variable
- Assign that product to a new variable which is now “selectedProduct” variable

# Cont ...

Let's refactor our edit product component:

```
ngOnInit() {  
    this.route.params.subscribe(  
        param => {  
            for (let product of this.products){  
                if (product.id == param.id){ this.selectedProduct = product; }  
            }  
        }  
    );  
}
```

# Cont ...

Do you see any problem here?

Why are we still having our `products.component` rendering?

How can we avoid this?

We can create our very own **navbar and navigate through the components**.

Where should we put our navbar html code?

We can take our navbar from the bootstrap documentation

# Cont ...

Code snippet from app.component.html looks like this:

```
<ul class="navbar-nav me-auto mb-2 mb-lg-0">
  <li class="nav-item">
    <a class="nav-link active" aria-current="page" [routerLink]="['products']">Products</a>
  </li>
  <li class="nav-item">
    <a class="nav-link" [routerLink]="['customers']">Customers</a>
  </li>
</ul>
```

Please make sure to also add this `<router-outlet></router-outlet>` at the end of your html file.

What is the use of this tag?

# Cont ...

We're left with one thing to be all set.

We didn't tell Angular where to go when 'customers' and 'products' route is hit.

So go ahead and define the paths. Where and how do you do that?

Here is a little code hint:

```
{  
  path: 'customers',  
  component: CustomersComponent  
},
```

# Let's Fill Some Forms !

## 4. Why don't we try to display the values in a form and let the user update the product

Why don't we try to display the values in a form and let the user update the product

We can choose any form we like from our [bootstrap documentation](https://getbootstrap.com/docs/5.0) (https://getbootstrap.com/docs/5.0)

And of course one submit button to submit our form. Mine looks like this:

Name

Nike

Price

10

Submit

# Cont ...

## Small Challenge

- 1) Work on the editProduct.component.html file to create your own version of this form.
- 2) When Submit button is clicked “updateProduct” method should be called

# Cont ...

A snippet of the code for my form looks like this:

```
<form (ngSubmit)="updateProduct()">
  <div class="mb-3 row">
    <label for="staticEmail" class="col-sm-2 col-form-label">Name</label>
    <div class="col-sm-10">
      <input type="text" class="form-control" id="inputPassword" name="name"
        [(ngModel)]="newProduct.name" [placeholder]="selectedProduct.name">
    </div>
  </div>
</form>
```



# Cont ...

So to submit our form this is how we refactor our `<form>` tag

```
<form (ngSubmit)="updateProduct()" #f=ngForm>
```

When our `updateProduct` function is called we want to update the data and route to the products list page

## How can we do that?

```
updateProduct(){  
    this.selectedProduct.name = this.newProduct.name;  
    this.selectedProduct.price = this.newProduct.price;  
    this.newProduct = new Product();  
    this.route.navigateByUrl('products');  
}
```

# Validate !

Last but not least - **Form Validation**

Let's update the submit button tag a bit:

```
<button type="submit" class="btn btn-primary" [disabled]="!f.valid">Submit</button>
```

What does this line do?

So now we should add the validation to all the input tags.

# Cont ...

So this are our validation rules:

- 1) our name and price field are required
- 2) name should have minimum length of 3

The submit button should be disabled if this rules are not met which we already did

And we need the error displayed under the input fields

In order to do that we need to do two things:

- 1) `<input type="text" class="form-control" name="name" #name="ngModel">`
- 2) Let's add the following code section under the input tag and see what happens

# Cont ...

```
<div *ngIf="name.invalid && (name.dirty || name.touched)" class="alert alert-danger">
  <div *ngIf="name.errors?.['required']">
    Name is required.
  </div>
  <div *ngIf="name.errors?.['minlength']">
    Name must be at least 4 characters long.
  </div>
</div>
```

# Mini Project

For the customers page you will:

- Create edit customers page component to edit the customer
- When edit button is clicked route to edit customers component
- The detail of the customer should be searched from the list of customers and viewed on html file
- Allow the user to update the customer detail
- Add validation for you forms validation

# Cont ...

Two things are remaining to complete the product component.

Adding new product to the list

Deleting products from the list

## Adding new product to the list

- We want to have a button in the products component to “Add New Product”.
- We should have a new variable called “**newProduct**” of type **product** in products.component.ts to hold the new product we will create.
- We need a form to add new product.
- When the form is submitted “AddProduct” method will be called.
- This method will add the new product to the list.

# Cont ...

## Deleting products from the list

- We want to have a button in the each products of products component to “Delete Product”.
- This button need to listen to a click event and call “**deleteProduct**” method.
- This method should look for the specific product in the list and remove it.

# SERVICES



# Sneak Peak Into MockAPI

Instead of hitting a real server you can mimic server communication using [In-memory Web API](#)

But for our case we will create a mock api on mockapi.io and use that.

Let's head to <https://mockapi.io/> and create a project.

Then you can create a resource.

Then you will click “Generate All” to generate data. On one click it will generate 50 random data

# Get Data From Server

`HttpClient` is Angular's mechanism for communicating with a remote server over HTTP.

Make `HttpClient` available everywhere in the application by importing it in the app module:

Here is a code snippet of the `app.module.ts`:

```
@NgModule({  
  imports: [  
    HttpClientModule,  
  ],  
})
```

# Services

Now where do you think we should write the logic to hit an api and fetch data?

`products.component.ts` or `products.service.ts`

## Why Services?

Components shouldn't fetch or save data directly and they certainly shouldn't knowingly present fake data. They should focus on **presenting data**.

Services are a great way to **share information among components** that don't know each other.

For eg: if the customers component need products list or vice versa

# Cont ...

So let's go ahead and create our very first service:

We have two options to create our products service:

It can be inside the products component or in “services” folder in the app component.

I will create mine in a “services” folder.

Here is the command: `ng generate service products`

So inside my products.service.ts I can `fetch, add, update or delete product`

# Cont ...

So product.service.ts is a service provider so we have to include it inside `providers[]` inside `app.module.ts`

Here is how my `app.module.ts` looks like:

```
imports: [  
    BrowserModule,  
    FormsModule,  
    HttpClientModule,  
    AppRoutingModule  
],  
providers: [ProductsService],  
})
```

# Cont ...

In order to do those functionalities I need to inject `HttpClient` inside my constructor like this:

```
constructor(private httpRequest: HttpClient) { }
```

Here is a code snippet to get all products and to add a new product from API in my `products.service.ts`:

```
getProducts(): Observable<any>{  
    return this.httpRequest.get('https://61cda2a97067f600179c5b5d.mockapi.io/products');  
}
```

Then I can use this function inside my `products.component.ts`

# Cont ...

Now if we're done constructing our service we can go ahead and **inject** our service inside our components and use this functions.

Here is how my products.component.html looks like:

```
constructor(private router: Router, private productService: ProductService){ }
```

Where should I fetch all products?

# Cont ...

Here is a code snippet of my `ngOnInit()` function of my `products.component.ts`:

```
ngOnInit(): void {  
    this.productsService.getProducts().subscribe(  
        data => {this.productList = data}  
    );  
}
```

This will fetch the list of products when the component first loads.



# Cont ...

Now I can go back to my service to define the rest of the functions.

Here is how I can add and update a product inside my service:

```
addProduct(product: Product): Observable<any>{  
    return this.httpRequest.post('https://61cda2a97067f600179c5b5d.mockapi.io/products', product)  
}
```

```
updateProduct(id: number, product: Product): Observable<any>{  
    return this.httpRequest.put('https://61cda2a97067f600179c5b5d.mockapi.io/products/' + id,  
    product)  
}
```

# Cont ...

You can guess how delete function works.

And you're just left with integrating this functions inside the component.

This is how we can integrate APIs on our Angular project :)

# Mini Project

For the customers page you will:

- Allow the user to update the customer detail
- Allow the user to add and delete any customer of their choice
- All the CRUD operations should be connected to our mockapi

**WELL DONE!**

# PRESENTATION

# TakeAways

- ✓ **Angular is a simple and non opinionated framework**
- ✓ **It is important to know the best practices when it comes to Angular development**
- ✓ **Referring to the Angular documentation while working with Angular is a good practice**

# Resources

- ✓ **Angular Documentation**  
<https://angular.io/docs>

- ✓ **Angular Tutorial Links**  
<https://angular.io/guide/router-tutorial>  
<https://angular.io/guide/routing-with-urlmatcher>  
<https://angular.io/guide/router-tutorial-toh>

# SHORT ASSESSMENT



# FINAL SURVEY

# SOCIAL MEDIA BREAK



# Merci Beaucoup



Orange  
Digital  
Center