UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

WEB TECHNOLOGIES — LECTURE 19

FRONTEND FRAMEWORKS: ANGULAR

Luigi Libero Lucio Starace, PhD

luigiliberolucio.starace@unina.it

https://luistar.github.io

https://www.docenti.unina.it/luigiliberolucio.starace

PREVIOUSLY, ON WEB TECHNOLOGIES

- We've learned about **Single Page Applications**
  - A single web page, with (a good deal of) client-side JavaScript code, that switches between different views by updating the DOM, without fully loading different pages
  - Some complexity involved:
    - Client-side routing
    - View rendering
    - Components and Code Re-use
- Front-end Frameworks are a pre-defined set of software components, tools, and best practices that serve as a **foundation** for developing Single Page Applications

ANGULAR

- One of the most widely-used frontend frameworks
  - Especially in Italy, and for enterprise applications
- Developed by Google
- Version 2.0 released in 2016
- We'll use version 17

ANGULAR: CHARACTERISTICS

- Strong focus on **developer tooling** and **productivity**
- **Batteries included**
  - Angular is an **opinionated** full-fledged framework (e.g.: includes dependency injection mechanisms, routing, …). This helps reduce **decision fatigue** and ensures **consistency across projects**
- Migration support
  - Dedicated tooling to assist developers in migrating to newer versions
- Best practices from the start
  - Enforces TypeScript!

ANGULAR: INSTALLING THE CLI

```
@luigi ➜ D/O/T/W/2/e/20-Angular $ npm install -g @angular/cli

added 227 packages in 20s

@luigi ➜ D/O/T/W/2/e/20-Angular $ ng version
```

```
Angular CLI: 17.0.8
Node: 20.9.0
Package Manager: npm 10.2.2
OS: win32 x64
```

ANGULAR: DEVELOPMENT ENVIRONMENT

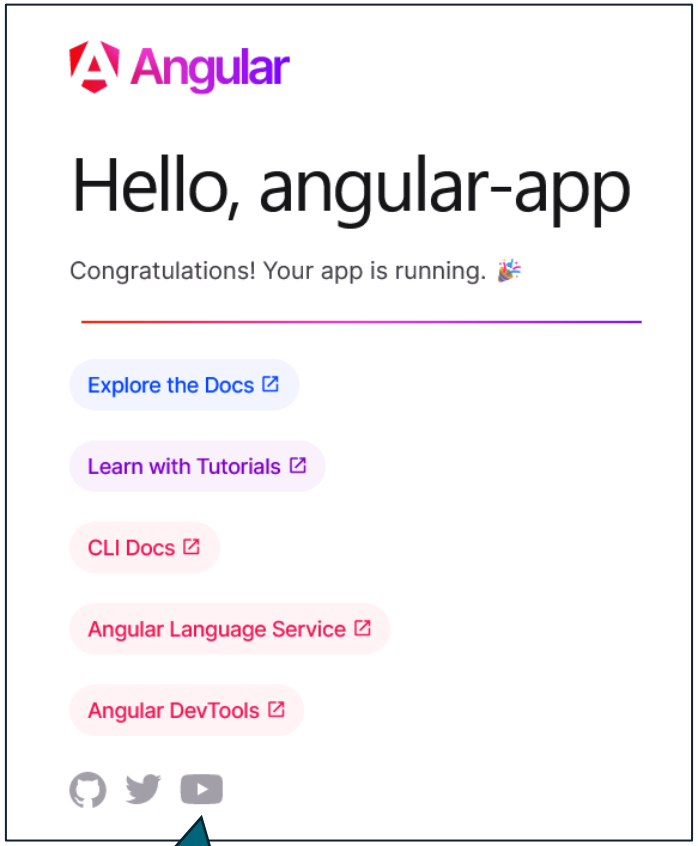
- We'll use Visual Studio Code
- An optional, but recommended step to improve developer experience is to install the Angular Language Service extension

ANGULAR: CREATING A NEW PROJECT

```
@luigi ➜ D/O/T/W/2/e/20-Angular $ ng new angular-app --create-application

? Which stylesheet format would you like to use? SCSS
? Do you want to enable SSR and SSG/Prerendering? No
CREATE angular-app/angular.json (2791 bytes)
CREATE angular-app/package.json (1042 bytes)
CREATE angular-app/README.md (1064 bytes)
...
CREATE angular-app/src/index.html (296 bytes)
CREATE angular-app/src/styles.scss (80 bytes)
CREATE angular-app/src/app/app.component.html (20884 bytes)
CREATE angular-app/src/app/app.component.spec.ts (931 bytes)
CREATE angular-app/src/app/app.component.ts (370 bytes)
CREATE angular-app/src/app/app.component.scss (0 bytes)
CREATE angular-app/src/app/app.config.ts (227 bytes)
CREATE angular-app/src/app/app.routes.ts (77 bytes)
```

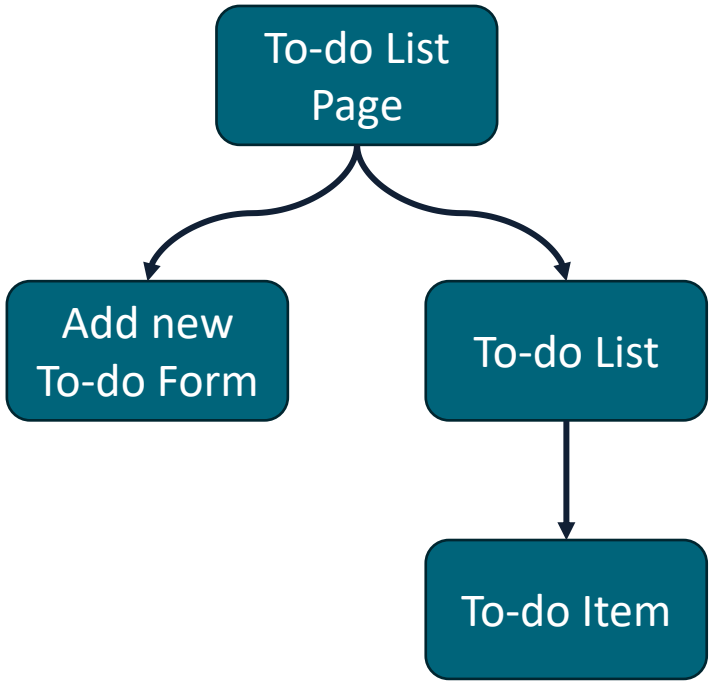
ANGULAR: RUNNING THE PROJECT

```
@luigi ➜ D/O/T/W/2/e/20-Angular $ cd angular-app

@luigi ➜ D/O/T/W/2/e/2/angular-app $ ng serve

Initial Chunk Files | Names         |  Raw Size
polyfills.js        | polyfills     |  82.71 kB |
main.js             | main          |  23.23 kB |
styles.css          | styles        |  96 bytes |

                    | Initial Total | 106.03 kB

Application bundle generation complete. [1.176 seconds]
Watch mode enabled. Watching for file changes...
  ➜  Local:   http://localhost:4200/
```



COMPONENTS

- Components are the **building blocks** of Angular apps
- Each component is responsible for defining the **function** and **appearance** of an element
  - Its **code** (state, business logic, events),
  - **HTML** layout,
  - **CSS** style
- In Angular, a component may also contain other components
- You can think of an Angular app as a tree of components



ANATOMY OF A COMPONENT

- A component is basically a TypeScript class
- Components are annotated with **metadata** passed as an object to the **@Component** decorator
- Metadata specify, among other things, the HTML layout and CSS (or Sass) styles for the component

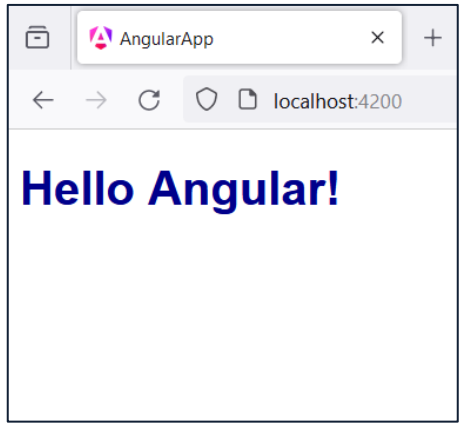
```
@Component({
  selector: 'app-root',
  standalone: true,
  template: "<h1>Hello Angular!</h1>",
  styles: "h1 {font-family: sans-serif; color: darkblue;}",
})
export class AppComponent {}
```

ANATOMY OF A COMPONENT: SELECTOR

- **selector** is used to tell Angular where to render the component
- Angular will create an instance of the component for every matching HTML element
- Below, the AppComponent will render in <app-root> HTML elems

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

@Component({
  selector: 'app-root',
  standalone: true,
  template: "<h1>Hello Angular!</h1>",
  styles: "* {color: darkblue;}",
})
export class AppComponent {}
```

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>AngularApp</title>
</head>
<body>
  <app-root></app-root>
</body>
</html>
```



ANATOMY OF A COMPONENT: TEMPLATES

- Templates and styles can also be specified in a separated file
- Paths are relative to the position of the TypeScript class

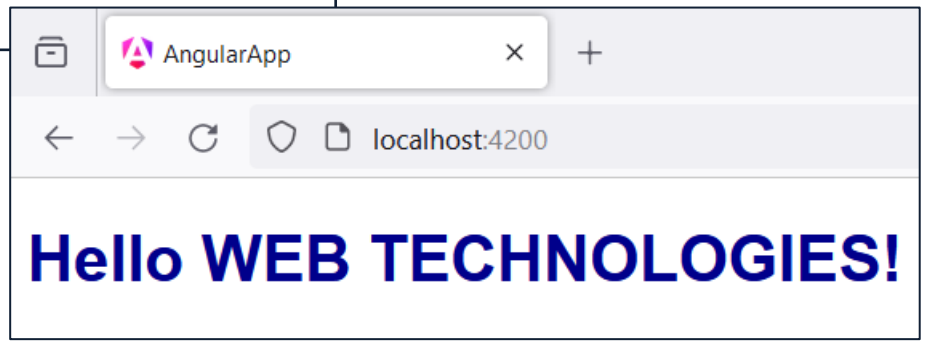
```
import {Component} from'@angular/core';

@Component({
  selector: 'app-root',
  standalone: true,
  templateUrl: './app.component.html',
  styleUrl: './app.component.scss'
})
export class AppComponent {}
```

```
// app.component.html file
<h1>Hello Angular!</h1>
```

```
// app.component.scss file
h1 {
  font-family: sans-serif;
  color: darkblue;
}
```

COMPONENTS: INTRODUCING STATE

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

@Component({
  selector: 'app-root',
  standalone: true,
  template: "<h1>Hello {{name.toUpperCase()}}!</h1>",
  styles: "h1 {font-family: sans-serif; color: darkblue;}",
})
export class AppComponent {
  name = 'Web Technologies';
}
```



Expressions within {{ }} in Angular templates are evaluated, and the result (converted as a string) is rendered in the template

COMPOSING COMPONENTS

- We created our first component, with its template and styles
- What if we want to add a new component to our app?
- Suppose we want to add a Counter component, that includes a button and keeps track of how many time the button was clicked on.
- We can use the Angular CLI to create the boilerplate code for us

```
@luigi ➜ D/O/T/W/2/e/2/angular-app $ ng generate component counter

CREATE src/app/counter/counter.component.html (22 bytes)     // HTML template
CREATE src/app/counter/counter.component.spec.ts (603 bytes) // tests (will see later)
CREATE src/app/counter/counter.component.ts (239 bytes)      // TS class
CREATE src/app/counter/counter.component.scss (0 bytes)      // Styles file
```

GENERATED BOILERPLATE CODE

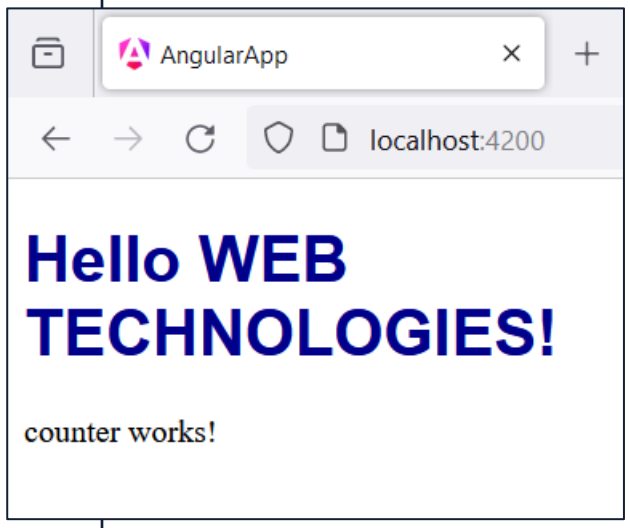
```ts
import { Component } from '@angular/core';

@Component({
  selector: 'app-counter',
  standalone: true,
  imports: [],
  templateUrl: './counter.component.html',
  styleUrl: './counter.component.scss'
})
export class CounterComponent {}
```

```html
// counter.component.html file
<p>counter works!</p>
```

```scss
// counter.component.scss file
// This file is empty
```

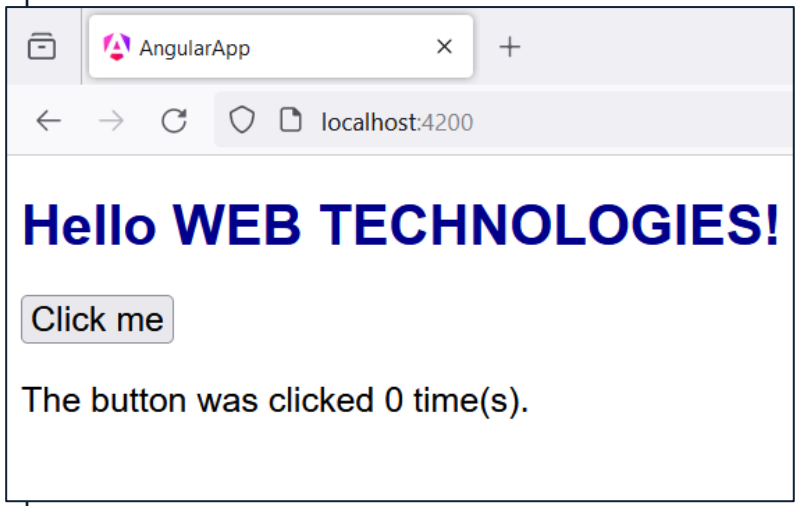
COMPOSING COMPONENTS

```
import { Component } from '@angular/core’;
import { CounterComponent } from './counter/counter.component';

@Component({
  selector: 'app-root',
  standalone: true,
  imports: [CounterComponent],
  template: `
    <h1>Hello {{name.toUpperCase()}}!</h1>
    <app-counter />
  `,
  styles: "h1 {font-family: sans-serif; color: darkblue;}",
})
export class AppComponent {
  name = 'Web Technologies';
}
```



THE COUNTER COMPONENT

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

@Component({
  selector: 'app-counter',
  standalone: true,
  template: `
    <button>Click me</button>
    <p>The button was clicked {{counter}} time(s).</p>
  `,
  styles: `p, button {
    font-size: 1.25em;
    font-family: sans-serif; }
  `
})
export class CounterComponent {
  counter: number = 0;
}
```



HANDLING EVENTS

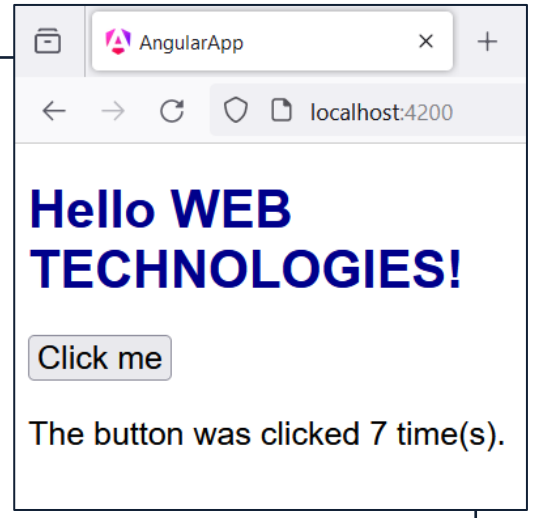
- In Angular, events are bound to handlers using the parentheses syntax
- For example, to assign an handler to the **click** event on a button, we can write in the template

```
<button (click)="increment()">Click me</button>
```

- Where **increment()** is, for example, a method of the component

THE COUNTER COMPONENT

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

@Component({
  selector: 'app-counter',
  standalone: true,
  template: `
    <button (click)="increment()">Click me</button>
    <p>The button was clicked {{counter}} time(s).</p>
  `,
  styles: 'p, button { font-size: 1.25em; font-family: sans-serif; }'
})
export class CounterComponent {
  counter: number = 0;

  increment() {
    this.counter = this.counter + 1;
  }
}
```



STANDALONE COMPONENTS

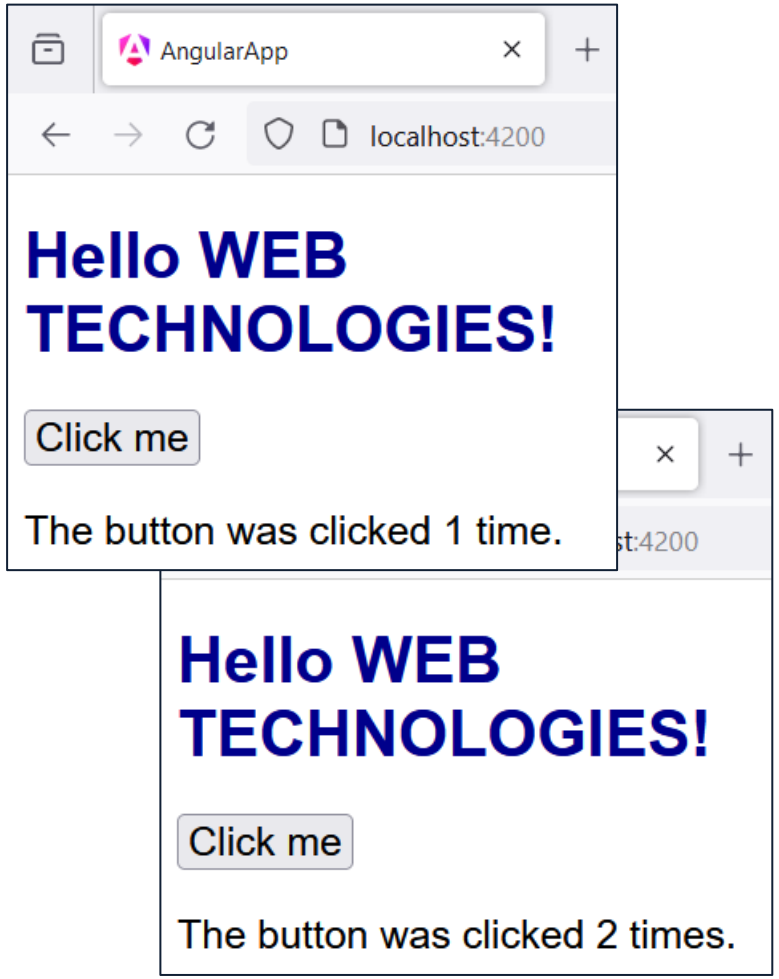
- Standalone components set «`standalone: true`» in their metadata
- These components can directly import other components, directives and pipes used in their templates
- Older Angular code (before version 14) used a different mechanism for importing and using other components, called NgModule
- In the Web Technologies course, we will only use standalone components

ANGULAR TEMPLATES: CONTROL FLOW

- We’ve seen that Control flow constructs are very useful in templates
- Often we need to conditionally render some part of the template, or to iterate over multiple elements and render the same portion of template over and over
- Angular templates support these scenarios with the familiar **@if/@else** and **@for** template syntax
- Before Angular 17, **ngIf** and **ngFor** directives were used

ANGULAR TEMPLATES: @IF/@ELSE

```
// template for the Counter component
<button (click)="increment()">Click me</button>
<p>
  The button was clicked {{counter}}
  @if(counter===1){time} @else{times}.
</p>
```



ANGULAR TEMPLATES: @FOR

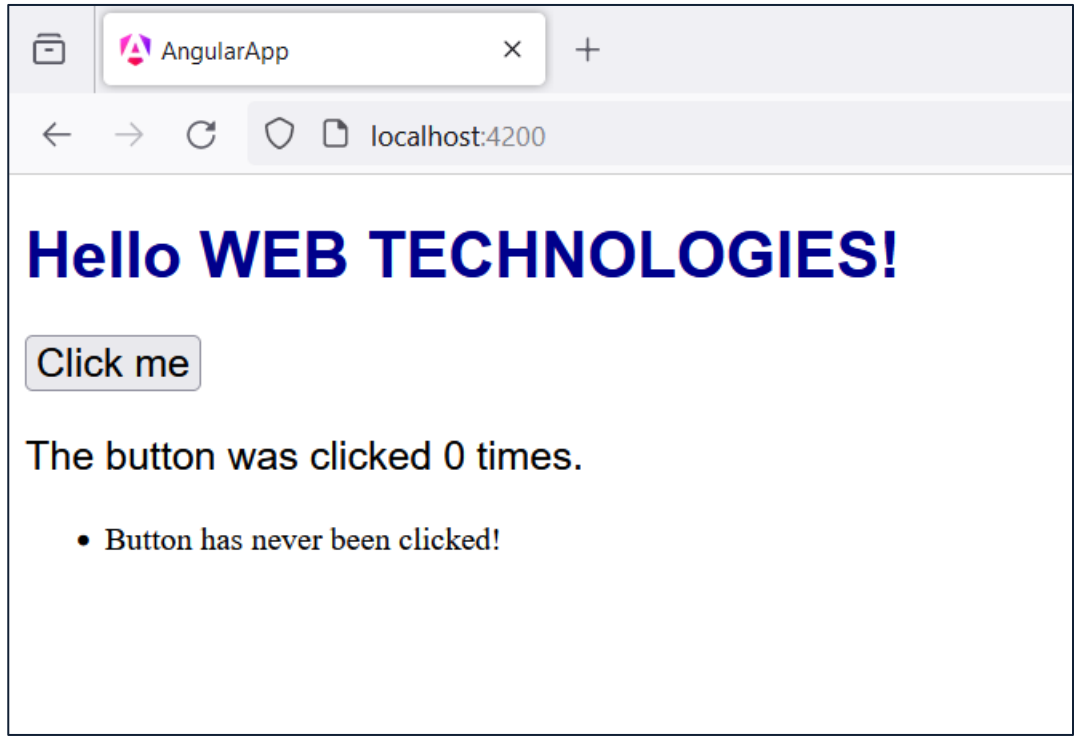
- Let's introduce a new feature to our Counter: it will keep track of the times the button was clicked

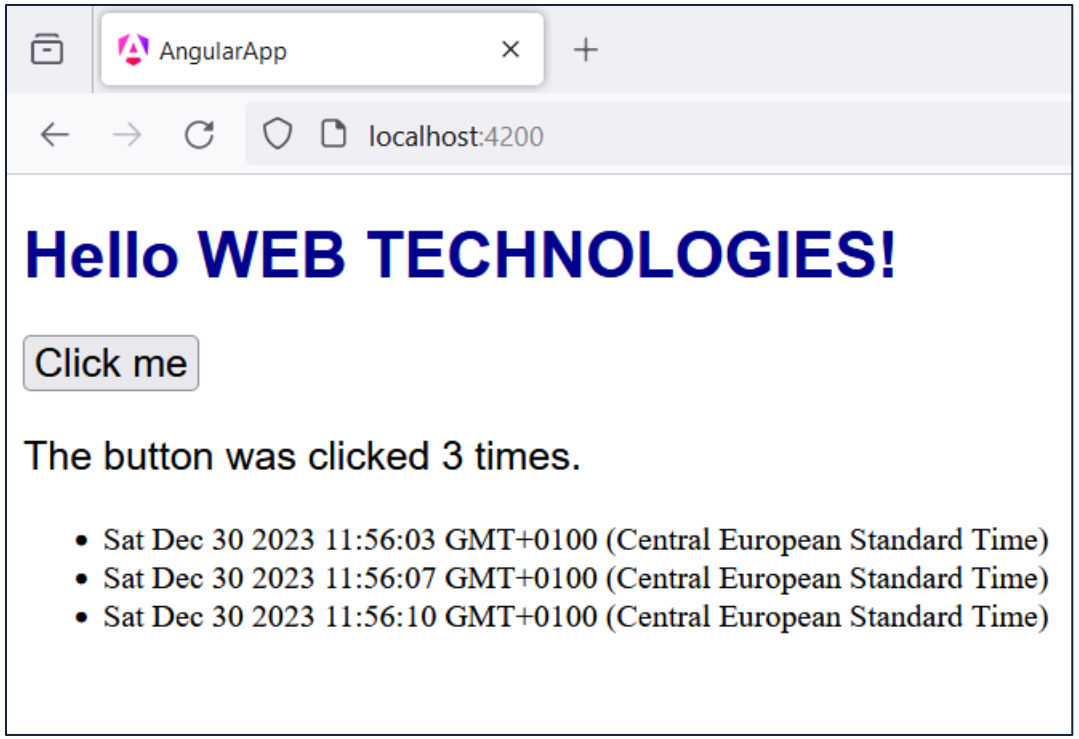
```
export class CounterComponent {
  counter: number = 0;
  clickTimestamps: Date[] = []

  increment() {
    this.counter = this.counter + 1;
    this.clickTimestamps.push(new Date());
  }
}
```

```
// template for the Counter component
<button (click)="increment()">
  Click me
</button>
<p>
  The button was clicked {{counter}}
  @if(counter==1){time} @else{times}.
</p>
<ul>
  @for(ts of clickTimestamps; track ts){
    <li>{{ts}}</li>
  } @empty {
    <li>Button has never been clicked!</li>
  }
</ul>
```

ANGULAR TEMPLATES: @FOR

- Let’s introduce a new feature to our Counter: it will keep track of the times the button was clicked





COMPONENT COMMUNICATION

- Sometimes, components need to **share** data with other components
- This can happen when a parent component passes information to child components (e.g.: to configure their behaviour, or to pass data to visualize)
- Or when a child component needs to communicate to its parent that some event has occurred
- These communication patterns can be implemented in Angular using the **@Input()** and **@Output()** decorators

ANGULAR: @INPUT() DECORATOR

- In the child component, properties that may be passed by the parent are decorated using **@Input()**. There might be a default value, used when the parent does not pass any specific value.
- In our example, suppose that the text inside the button can be customized by the parent component

```
import {Component,Input} from '@angular/core';
export class CounterComponent {
  @Input() buttonText = "Click me";
  counter: number = 0;
  clickTimestamps : Date[] = []
  increment() {
    this.counter = this.counter + 1;
    this.clickTimestamps.push(new Date());
  }
}
```

```
<button (click)="increment()">
  {{buttonText}}
</button>
<p>
  The button was clicked {{counter}}
  @if(counter==1){time}@else{times}.
</p>
// rest of the template omitted
```

ANGULAR: @INPUT() DECORATOR

- The parent component can now pass data to the child component by setting a property on the HTML element

```
@Component({
  selector: 'app-root', standalone: true,
  imports: [CounterComponent],
  template: `
    <h1>Hello {{name.toUpperCase()}}!</h1>
    <app-counter buttonText="CLICK HERE"/>
  `,
  styleUrl: "./app.component.scss"
})
export class AppComponent {}
```

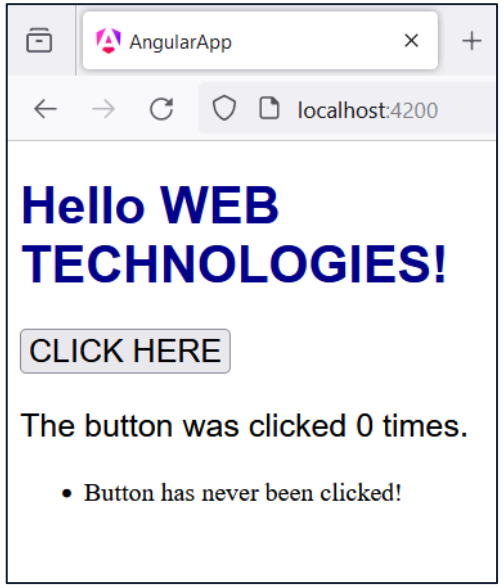
```
export class CounterComponent {
  @Input() buttonText = "Click me";
  counter: number = 0;
  clickTimestamps : Date[] = [];

  increment() {
    this.counter = this.counter + 1;
    this.clickTimestamps.push(new Date());
  }
}
```

```
<button (click)="increment()">
  {{buttonText}}
</button>
// rest of the Counter template omitted
```

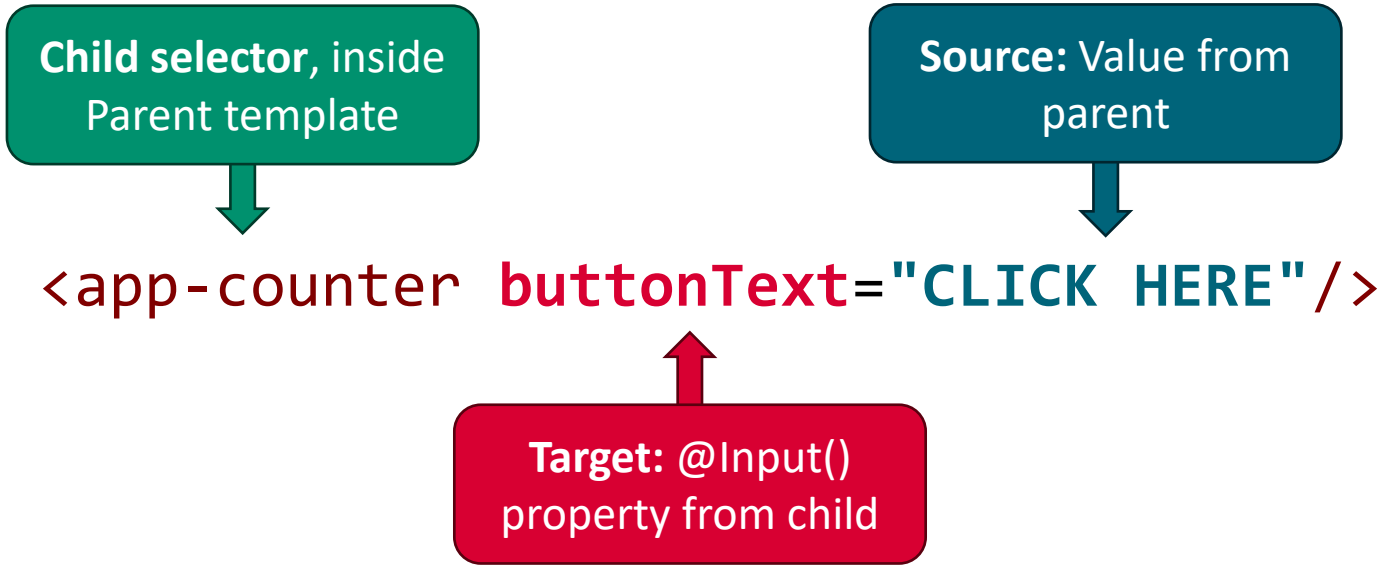
ANGULAR: @INPUT() DECORATOR

- The parent component can now pass data to the child component by setting a property on the HTML element
- If we removed `buttonText="CLICK HERE"` from `<app-counter />` the button would display the default value («Click me»)

```
@Component({
  selector: 'app-root', standalone: true,
  imports: [CounterComponent],
  template: `
    <h1>Hello {{name.toUpperCase()}}!</h1>
    <app-counter buttonText="CLICK HERE"/>
  `,
  styleUrl: "./app.component.scss"
})
export class AppComponent {}
```



ANGULAR: @INPUT() DECORATOR RECAP



PARENT TO CHILD COMMUNICATION

- This communication pattern is widely-used in front-end frameworks
- In other frameworks (e.g.: React, Vue), this mechanism is implemented using Props.

ANGULAR: PROPERTY BINDING

- Angular allows to dynamically set values for attributes of HTML elements and components in templates, binding these values to properties of the current component
- The name of the attribute to bind is specified between square brackets «[]», and the **value** is the **name** of the property to bind

```
<h1>Hello {{name.toUpperCase()}}!</h1>
<app-counter [buttonText]="btnMsg"/>
<img [src]="imageUrl"/>
```

```
export class AppComponent {
  name = 'Web Technologies';
  btnMsg = "CLICK HERE";
  imageUrl = "https://picsum.photos/300/150"
}
```

PROPERTY BINDING VS INTERPOLATION

- You might wonder if there's any difference between using **property binding** and **template interpolation**

```
<img [src]="imageUrl"/>
```

```
<img src={{imageUrl}}/>
```

- In the above example, the outcome is the same
- But that's not always the case!
- Template interpolation evaluates the provided expression and **converts the result to a string**.
- To pass an object or an array as an element property, you need to use property binding!

ANGULAR: @OUTPUT DECORATOR

- To create a communication channel between a child and its parent, you can use the **@Output** decorator on a class property, and assign it a value of type **EventEmitter**
- To send an event on the channel, the child component invokes the **emit()** method on the @Output property

ANGULAR: @OUTPUT DECORATOR

```
export class CounterComponent {
  @Input() buttonText = "Click me";
  @Output() counterClickedTenTimes = new EventEmitter<number>();
  counter: number = 0;
  clickTimestamps : Date[] = []

  increment() {
    this.counter++;
    this.clickTimestamps.push(new Date());
    if(this.counter%10 === 0)
      this.counterClickedTenTimes.emit(this.counter);
  }
}
```

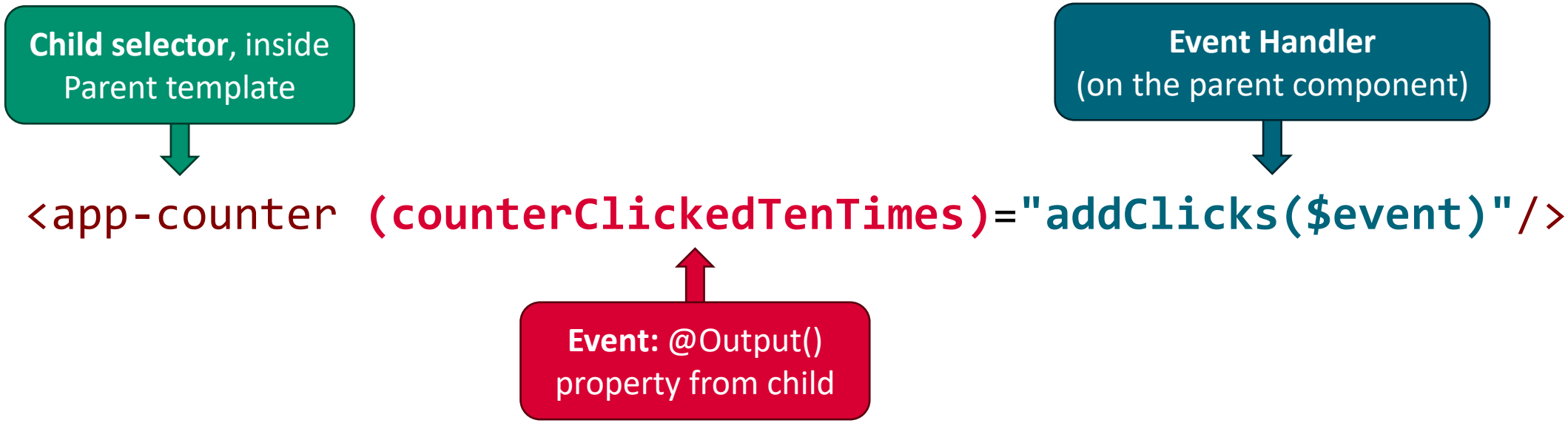
ANGULAR: @OUTPUT DECORATOR

- The parent component can now react to events on the communication channel by defining an event handler, as usual

```
<h1>Hello {{name.toUpperCase()}}!</h1>
<app-counter [buttonText]="btnMsg" (counterClickedTenTimes)=addClicks($event) />
<ul>@for(click of clicksList; track click){
  <li>Button was clicked on {{click}} times.</li>
}</ul>
```

```
export class AppComponent {
  name = 'Web Technologies'; btnMsg = "CLICK HERE";
  clicksList : number[] = [];

  addClicks(nClicks: number){
    this.clicksList.push(nClicks)
  }
}
```

ANGULAR: @OUTPUT() DECORATOR RECAP



ANGULAR: ROUTING

- Single Page Applications need to deal with **client-side routing**
- Angular includes a **Router** to support devs in this task
- The **Router** enables **navigation** between different **views**
  - It does so by interpreting the URL as an instruction to change the view
- Angular apps created using the **ng new** command already have Routing configured and ready-to go
- We'll go over the steps to configure Routing in the next slides

ANGULAR ROUTER SETUP: BOOTSTRAP

- In **./src/main.ts**, the entry point of the Angular app, we take care of loading the root component (AppComponent)
- In doing so, we provide a configuration object, defined in **./app/app.config.ts**
- The app.config object specifies that the application uses Routing

```
import { bootstrapApplication } from '@angular/platform-browser';
import { appConfig } from './app/app.config';
import { AppComponent } from './app/app.component';

bootstrapApplication(AppComponent, appConfig)
  .catch((err) => console.error(err));
```

ANGULAR ROUTER SETUP: APP.CONFIG

- The **./src/app.config.ts** exports an appConfig object of type **ApplicationConfig**
- The **provideRouter()** function takes as input a description of the application routes of type **Routes**, defined in **./src/app.routes.ts**, and sets up the necessary providers for the Router module to work

```
import { ApplicationConfig } from '@angular/core';
import { provideRouter } from '@angular/router';

import { routes } from './app.routes';

export const appConfig: ApplicationConfig = {
  providers: [provideRouter(routes)]
};
```

ANGULAR ROUTER SETUP: APP.ROUTES

- The Routes object is defined in **./src/app.routes.ts**

```
import { Routes } from '@angular/router';
import { CounterPageComponent } from './counter-page/counter-page.component';
import { HomePageComponent } from './home-page/home-page.component';

export const routes: Routes = [    //Routes is a Route[]
  {                                //Each Route has many properties:
    path: "",                      //- path is the URL associated with the Route
    title: "Home Page",            //- title is the title of the HTML page
    component: HomePageComponent   //- component is the component associated with
  }, {                             //  the Route
    path: "counter",
    title: "Counter",
    component: CounterPageComponent
  }
];
```

ANGULAR ROUTER SETUP: ROUTER–OUTLET

- With all the setup in place, we can use the **RouterOutlet** component in our templates
- The **RouterOutlet** component analyzes the current URL, and renders the Component associated with the matching path
- The template for the root component may look similar to the one shown below

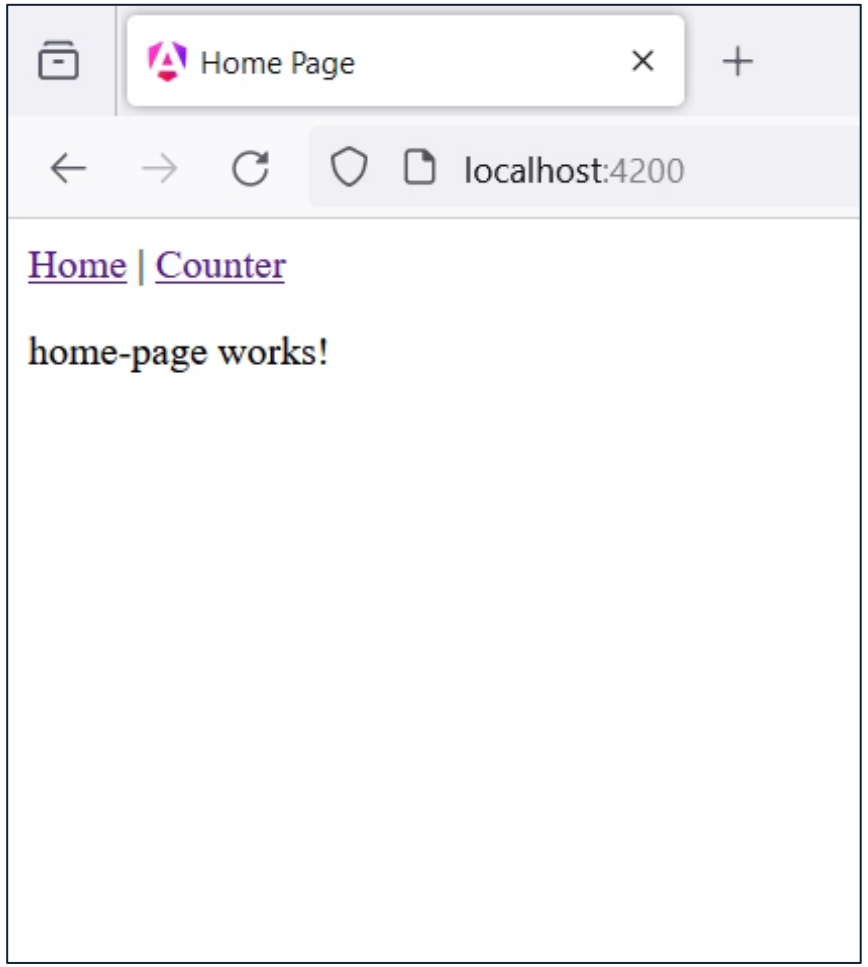
```
<nav>
  <a href="/">Home</a> | <a href="/counter">Counter</a>
</nav>
<router-outlet></router-outlet>
```

ANGULAR ROUTER SETUP: ROUTER-OUTLET

```
import { Component } from '@angular/core';
import { CommonModule } from '@angular/common';
import { RouterOutlet } from '@angular/router';

@Component({
  selector: 'app-root',
  standalone: true,
  imports: [CommonModule, RouterOutlet],
  template: `
    <nav>
      <a href="/">Home</a> | <a href="/counter">Counter</a>
    </nav>
    <router-outlet></router-outlet>
  `,
  styleUrl: "./app.component.scss"
})
export class AppComponent {}
```

ANGULAR ROUTER: LINKS

- If we use traditional links in our templates, with an href attribute, we cause the browser to perform a new HTTP request and re-load the app
- The Router then analyzes the URL, and loads the appropriate component in place of the `<router-outlet/>`
- This is not very sleek, and defeats the very purpose of a Single Page App



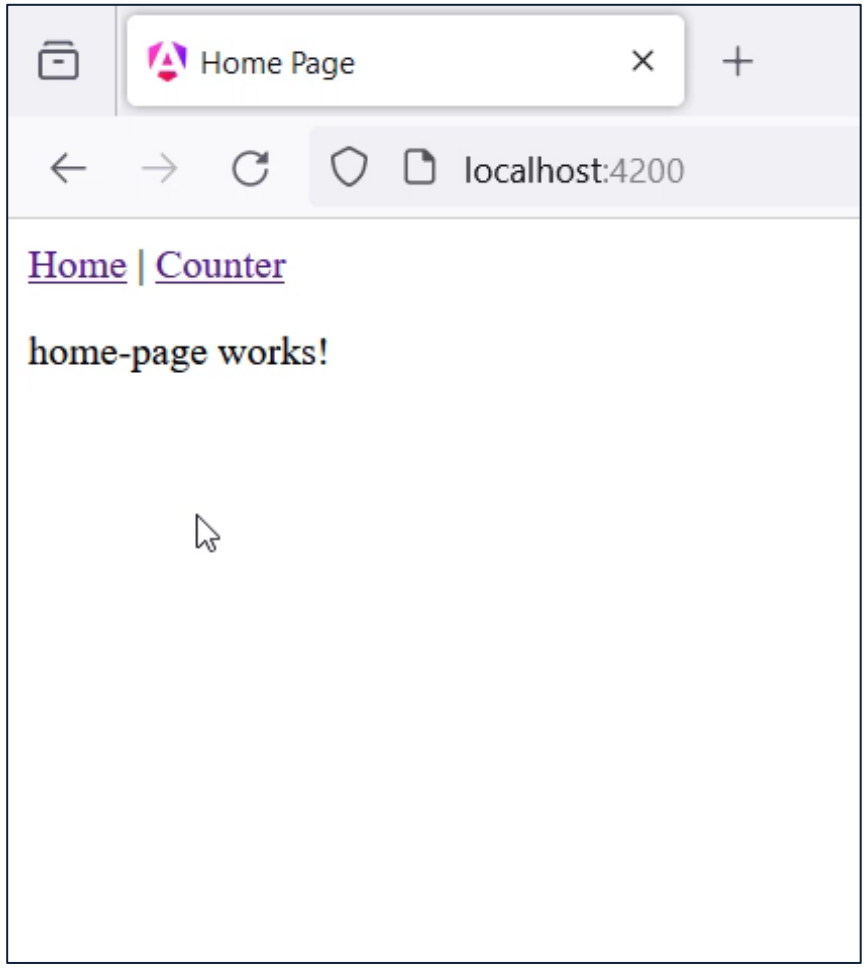
ANGULAR ROUTER: LINKS

To properly navigate between views in our Angular SPAs, we need to use the **RouterLink** directive on elements that trigger navigation

```
import { Component } from '@angular/core';
import { CommonModule } from '@angular/common';
import { RouterLink, RouterOutlet } from '@angular/router';

@Component({
  selector: 'app-root', standalone: true,
  imports: [CommonModule, RouterOutlet, RouterLink],
  template: `
    <nav><a routerLink="/">Home</a> | <a routerLink="/counter">Counter</a></nav>
    <router-outlet></router-outlet>
  `,
})
export class AppComponent {}
```

ANGULAR ROUTER: LINKS

- When we use RouterLink, we can trigger the Router and load different views without reloading the page
- This is how SPAs should handle routing!
- Notice that, if a user manually inserts the **localhost:4200/counter** URL, they would trigger a new page load, and the Router would still properly render the Counter component.



ANGULAR ROUTER: ROUTERLINKACTIVE

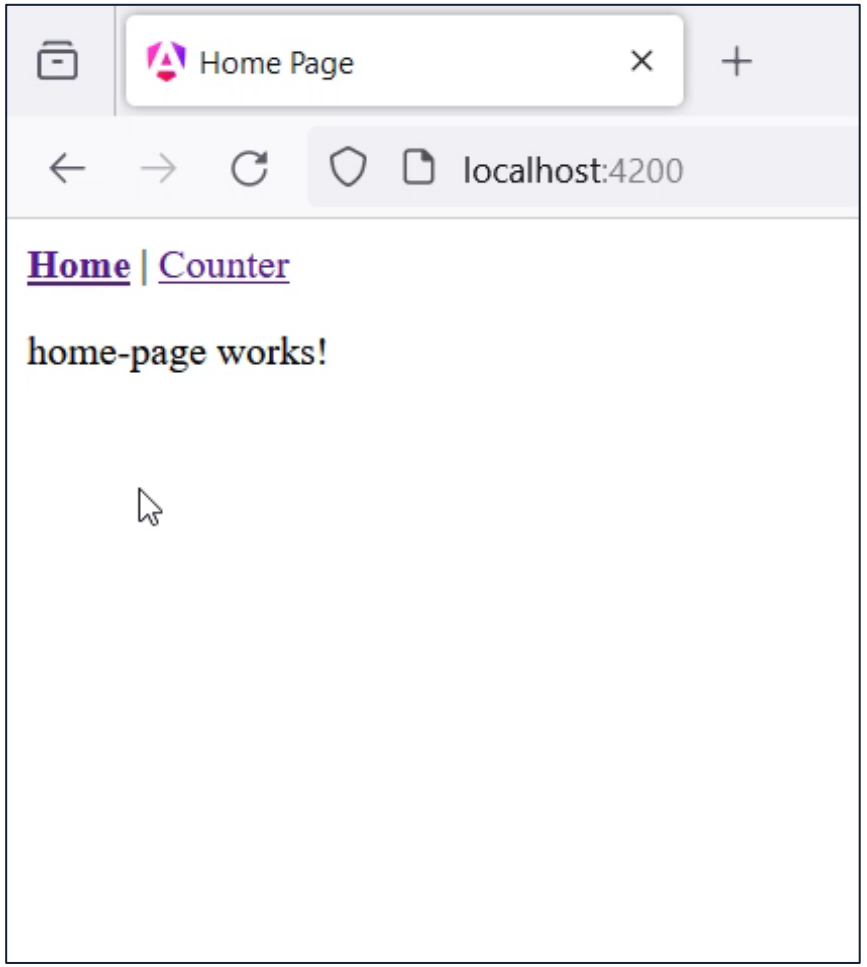
- Sometimes, we want to style the router links that are currently active differently, to remind the users which page they are on
- The **RouterLinkActive** directive allows us to specify a list of CSS classes that should be applied to **RouterLinks** that correspond to the Route the app is currently on
- These directives can be used in a template as follows

```
<nav>
  <a routerLink="/" routerLinkActive="active" [routerLinkActiveOptions]="{exact: true}">
    Home
  </a> |
  <a routerLink="/counter" routerLinkActive="active">Counter</a>
</nav>
<router-outlet></router-outlet>
```

ANGULAR ROUTER: ACTIVE LINKS

- With the template shown in the previous slide and a little bit of CSS (shown below), we can highlight active links in bold

```
nav .active {
  font-weight: bold;
}
```



REFERENCES

- **Angular Docs**
  https://angular.dev/overview
  Relevant parts: Introduction: Essentials; In-depth Guides: Components, Template Syntax, Routing; Developer Tools: Angular CLI