Angular

What is Angular?

Opensource **Javascript Framework**

Created and maintained by **Google**

Completely written in **Typescript**

Best for developing **Single Page Applications**

What is Typescript?

Typescript is a **Superset** of Javascript

Enhanced version of Javascript with additional features

Gets compiled to Javascript at the end

```
isAvailable = false;
quantity = 25;
```

Javascript

```
isAvailable: boolean = false;
quantity: number = 25;
```

Typescript

Skillsets needed to learn Angular

You should be familiar with **HTML**

You should be familiar with **Javascript**

You should know to use **CSS**

Why do we need a Framework?

Easy for maintaining Code

Easy to structure our code in modules

Code Reuse

Everything in one place

Why do we need Angular Framework?

Angular is a **framework**

It is **Typescript based**

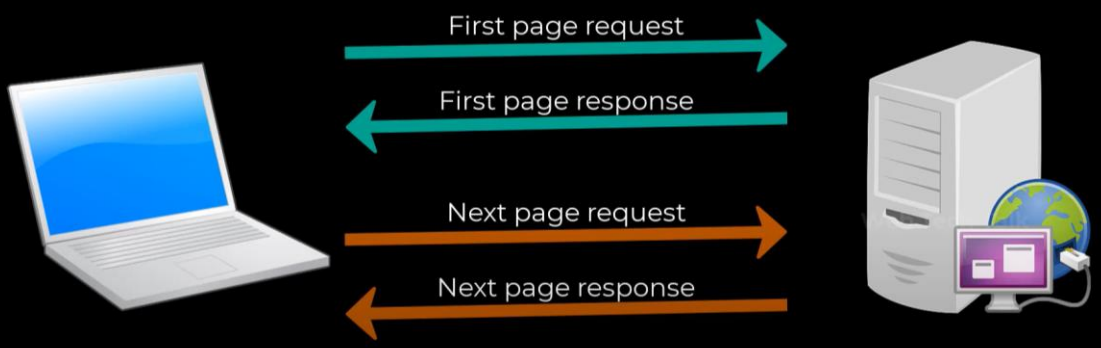
Powerful **Command Line Interface**

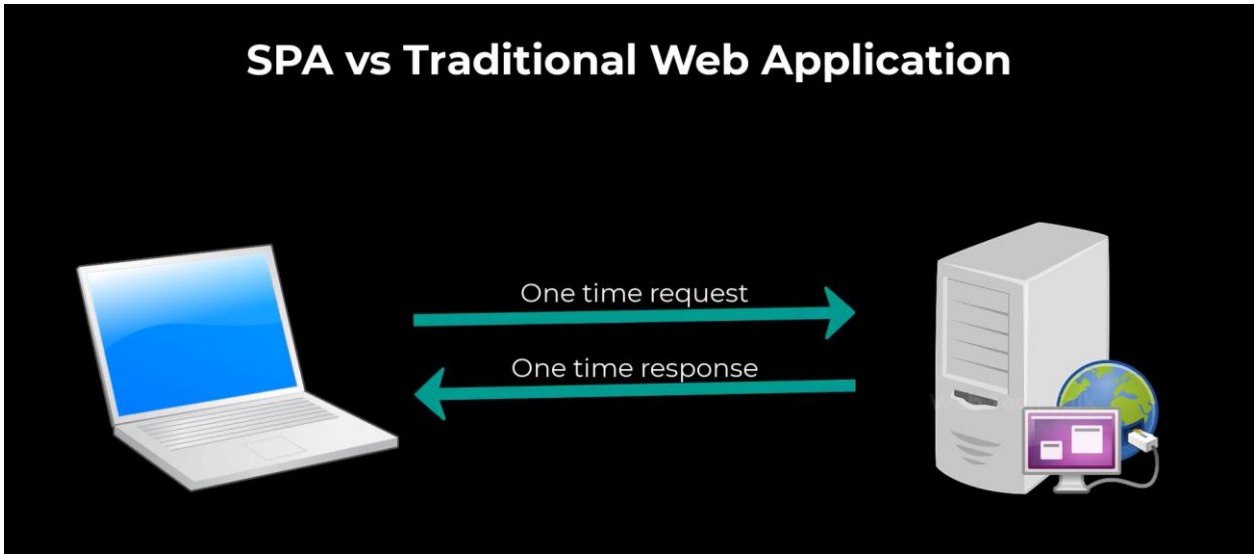
Two way binding

MVC based framework

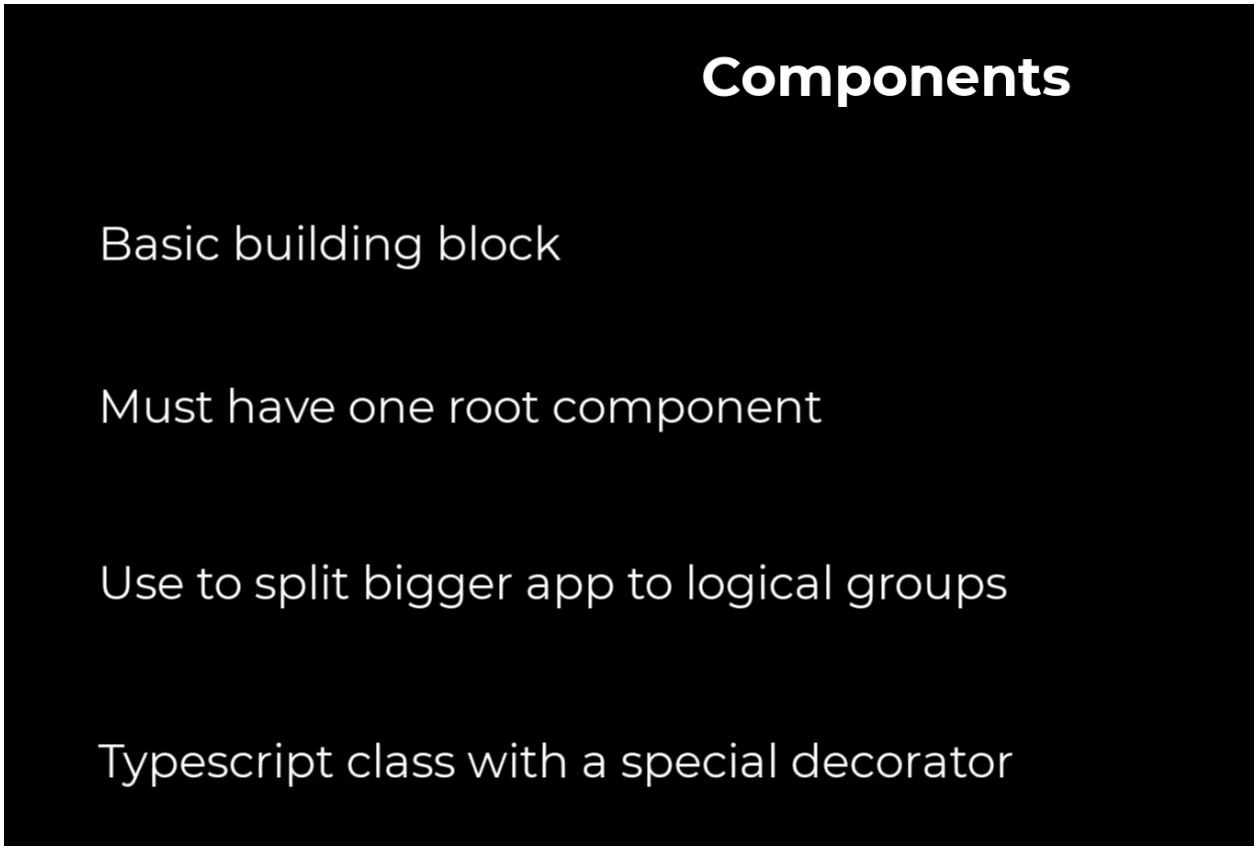
Uses **Dependency Injection**

SPA vs Traditional Web Application



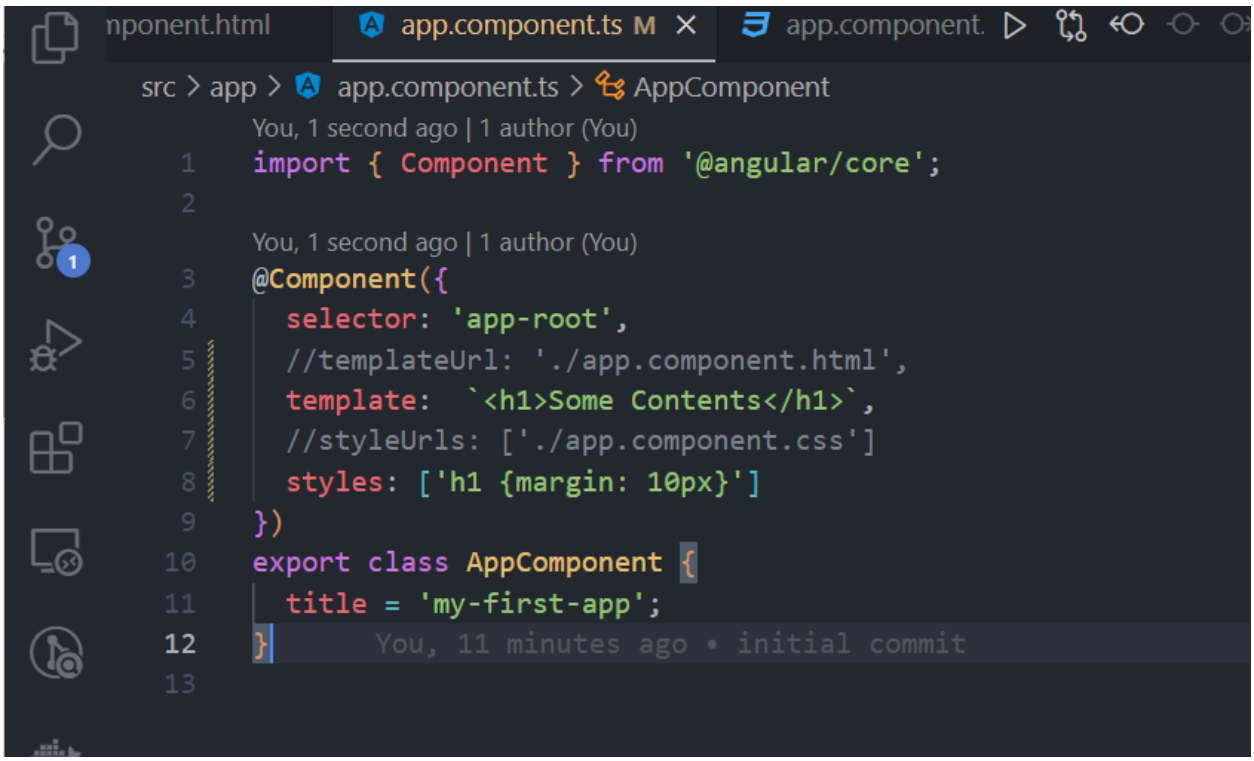

SPA vs Traditional Web Application
One time request
One time response

Components


Components
Basic building block
Must have one root component
Use to split bigger app to logical groups
Typescript class with a special decorator

WHAT IS COMPONENT IN ANGULAR?

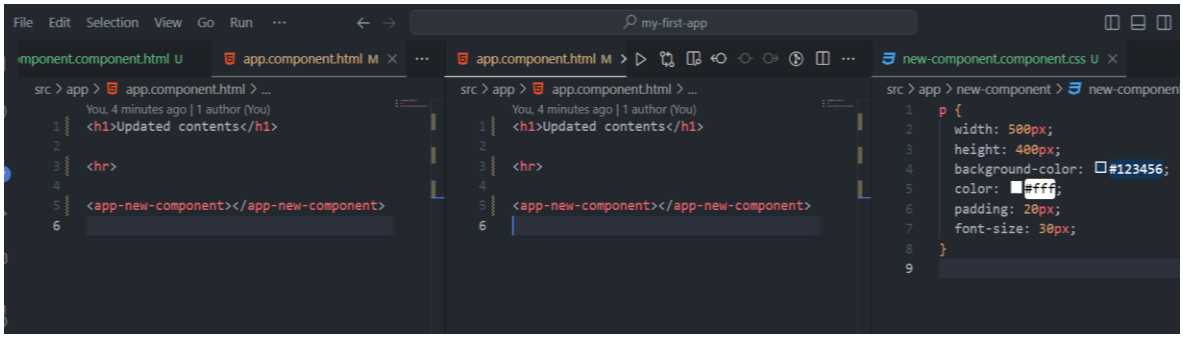
Components are the basic building block of an angular application. Every angular application will have one root component and can have multiple child components. Components help to split our big application into logical groups. A component is ideally a Typescript Class which has a special decorator. The decorator is to instruct Angular to treat this class as a component and in the decorator we can have a selector name. Selector name is similar to CSS selector. Here angular supports element, class and attribute selectors. But it does not support ID selectors. And we need to have a template. Either we can create a separate HTML file and map that in decorator using templateURL or else we can use an inline selector using template. And when we use inline template, we can use backtick to have multiple lines. In general, it will be good to have a separate template file if our template code is going to be more than 3 lines of code. Similarly we can include inline styles using styles array or can map multiple files to styleUrls array. But it is not a mandatory thing to mention styles. Components can be manually created. But it is easy to create it using CLI. If we are manually creating a component, we need to declare that in the root module. Otherwise it will not be available to other components. But, when we create a component using CLI, this step will be taken care by CLI itself.



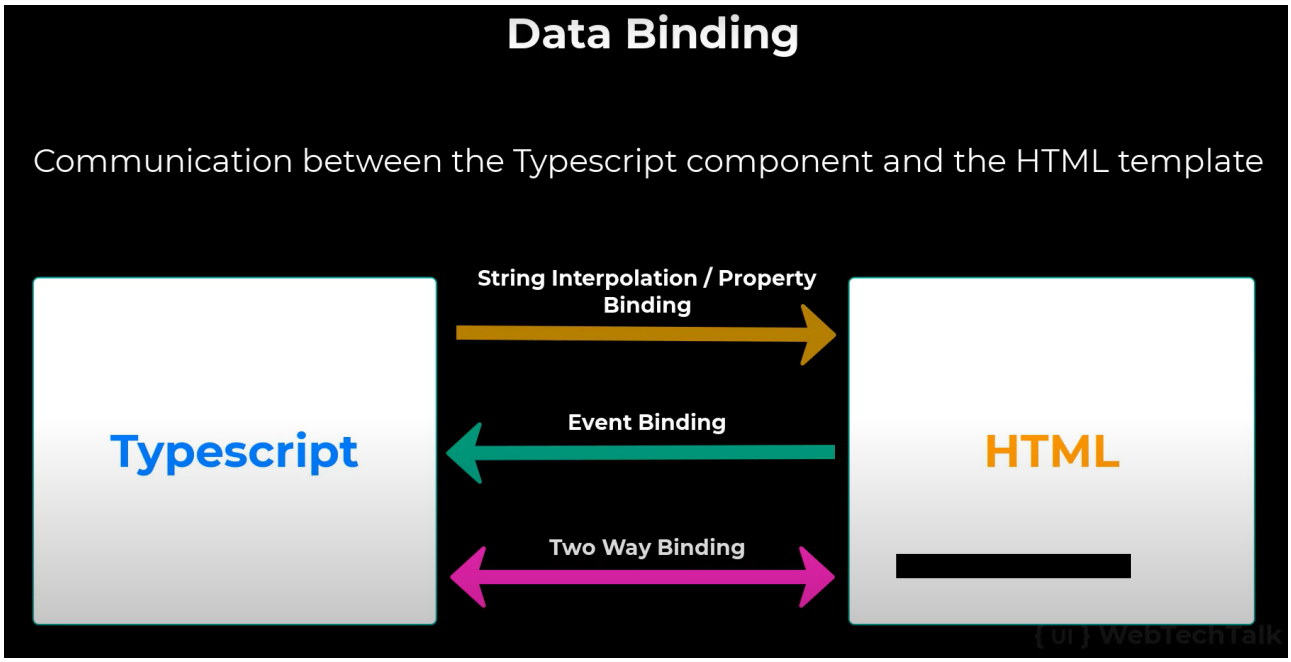
HOW CAN WE CREATE ANGULAR COMPONENT USING CLI?

Components can be manually created. But it is easy to create it using CLI. If we are manually creating a component, we need to declare that in the root module. Otherwise it will not be available to other components. But, when we create a component using CLI, this step will be taken care by CLI itself.

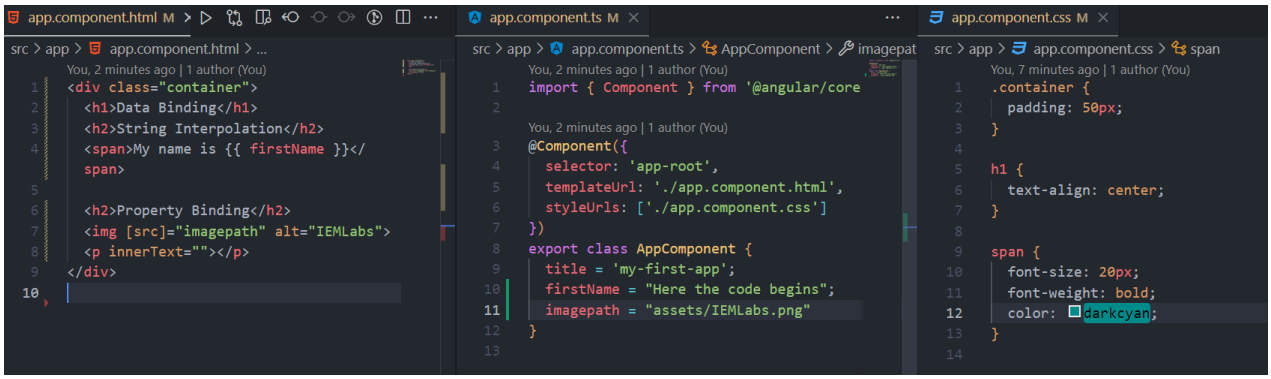
We can create a component using the command ng generate component component-name. Or in short we can give it as ng g c component-name. Now the component is created and also in the root module we can see the component is part of the declaration array. So we can use this component in other components.



Data Binding



- **String Interpolation & Property Binding**

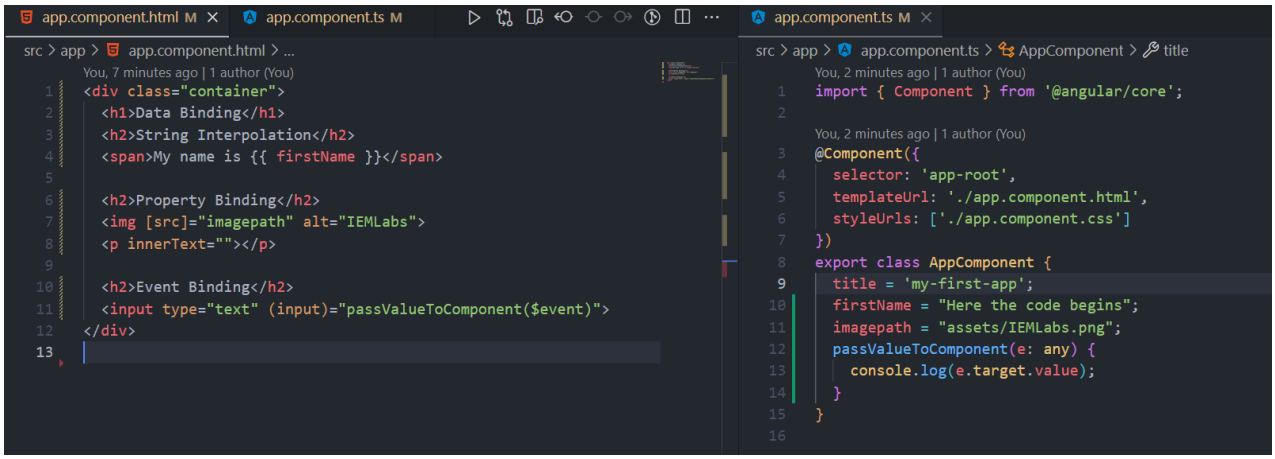


```
<!-- app.component.html -->
<div class="container">
  <h1>Data Binding</h1>
  <h2>String Interpolation</h2>
  <span>My name is {{ firstName }}</
  span>

  <h2>Property Binding</h2>
  <img [src]="imagepath" alt="IEMLabs">
  <p innerText=""></p>
</div>
```

```
// app.component.ts
import { Component } from '@angular/core

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'my-first-app';
  firstName = "Here the code begins";
  imagepath = "assets/IEMLabs.png"
}
```

```
/* app.component.css */
.container {
  padding: 50px;
}

h1 {
  text-align: center;
}

span {
  font-size: 20px;
  font-weight: bold;
  color: darkcyan;
}
```

- **Event Binding**

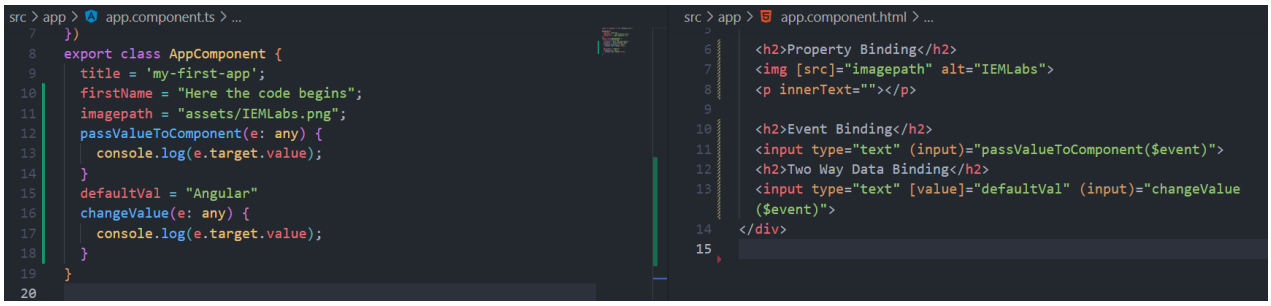


```
<!-- app.component.html -->
<div class="container">
  <h1>Data Binding</h1>
  <h2>String Interpolation</h2>
  <span>My name is {{ firstName }}</span>

  <h2>Property Binding</h2>
  <img [src]="imagepath" alt="IEMLabs">
  <p innerText=""></p>

  <h2>Event Binding</h2>
  <input type="text" (input)="passValueToComponent($event)">
</div>
```

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'my-first-app';
  firstName = "Here the code begins";
  imagepath = "assets/IEMLabs.png";
  passValueToComponent(e: any) {
    console.log(e.target.value);
  }
}
```

- **Two Way Data Binding**

1st method

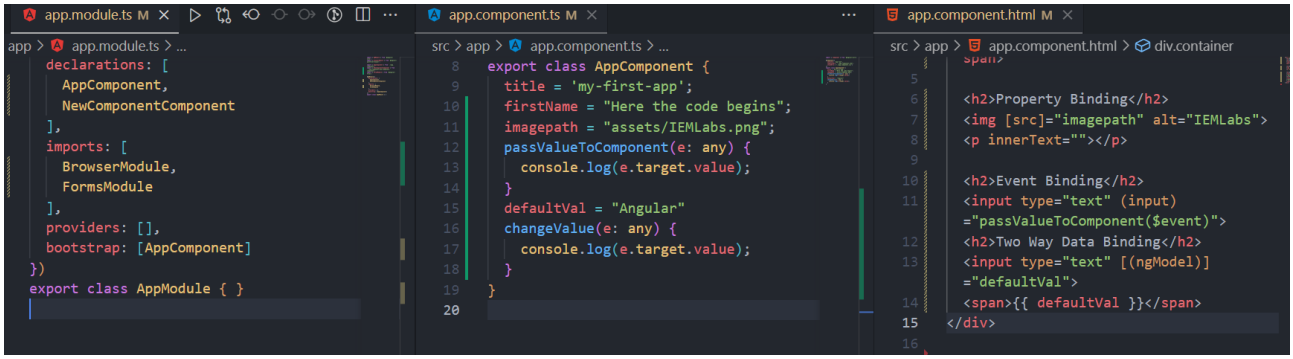


```
// app.component.ts
})
export class AppComponent {
  title = 'my-first-app';
  firstName = "Here the code begins";
  imagepath = "assets/IEMLabs.png";
  passValueToComponent(e: any) {
    console.log(e.target.value);
  }
  defaultVal = "Angular"
  changeValue(e: any) {
    console.log(e.target.value);
  }
}
```

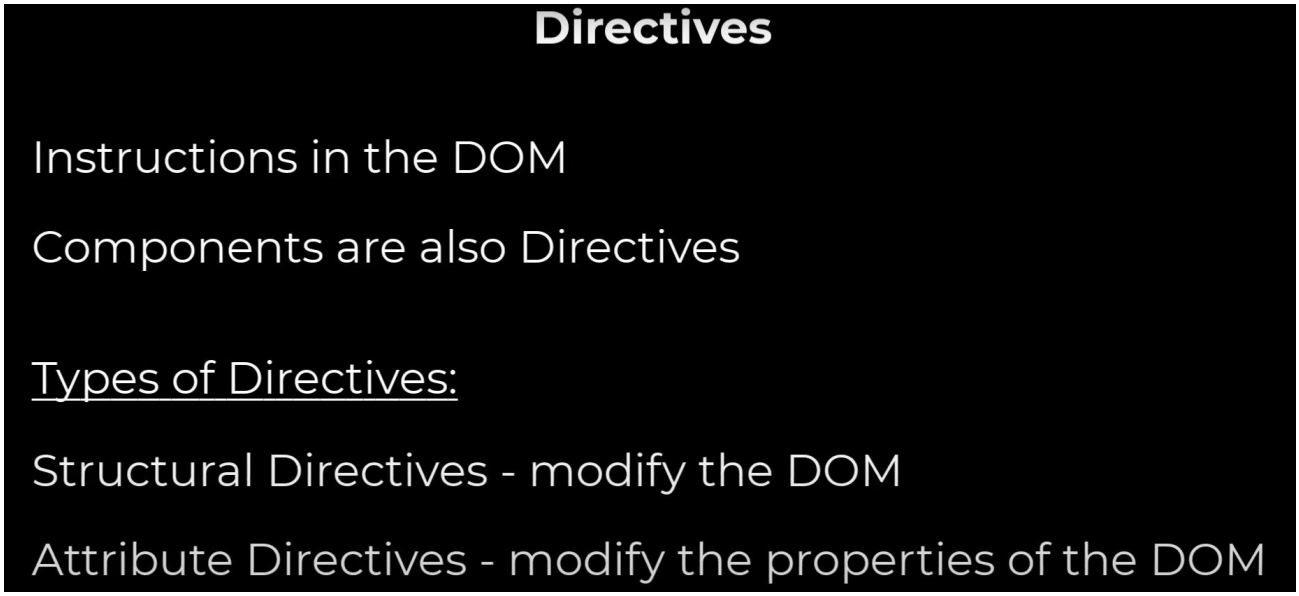
```
<!-- app.component.html -->
  <h2>Property Binding</h2>
  <img [src]="imagepath" alt="IEMLabs">
  <p innerText=""></p>

  <h2>Event Binding</h2>
  <input type="text" (input)="passValueToComponent($event)">
  <h2>Two Way Data Binding</h2>
  <input type="text" [value]="defaultVal" (input)="changeValue
  ($event)">
</div>
```

2nd method

But right now lets see how we can use two way binding. We shall remove the property binding as well as event binding from here. Instead we shall use the combination of both. So, inside this we shall use the special directive ngModel and bind that to a variable. If I save this and open the app, we shall see the app is broken and we can see the error can't bind to ngmodel since it isn't a known property of input. Because, ngModel directive is part of another module in Angular called FormsModule. So we need to import the formsModule in our root module. Only then we can use this directive.

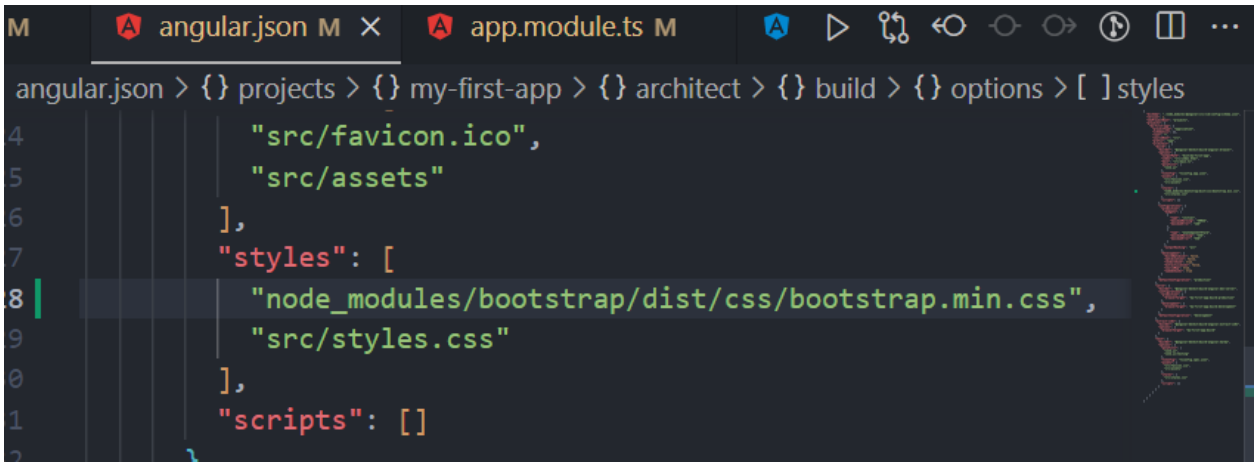


Directives



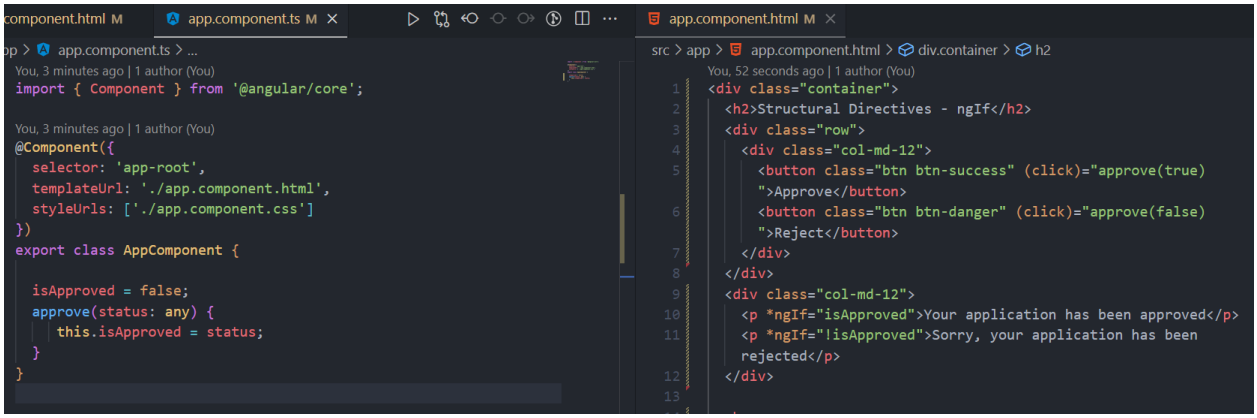
- **Adding bootstrap**

npm install bootstrap

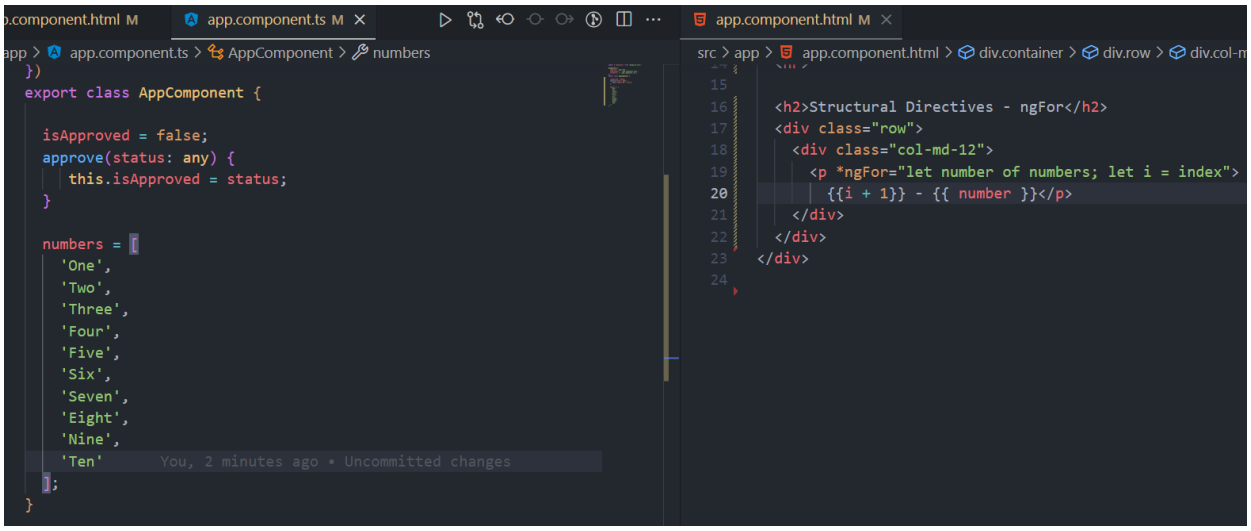


- **Structural Directives**

ngIf



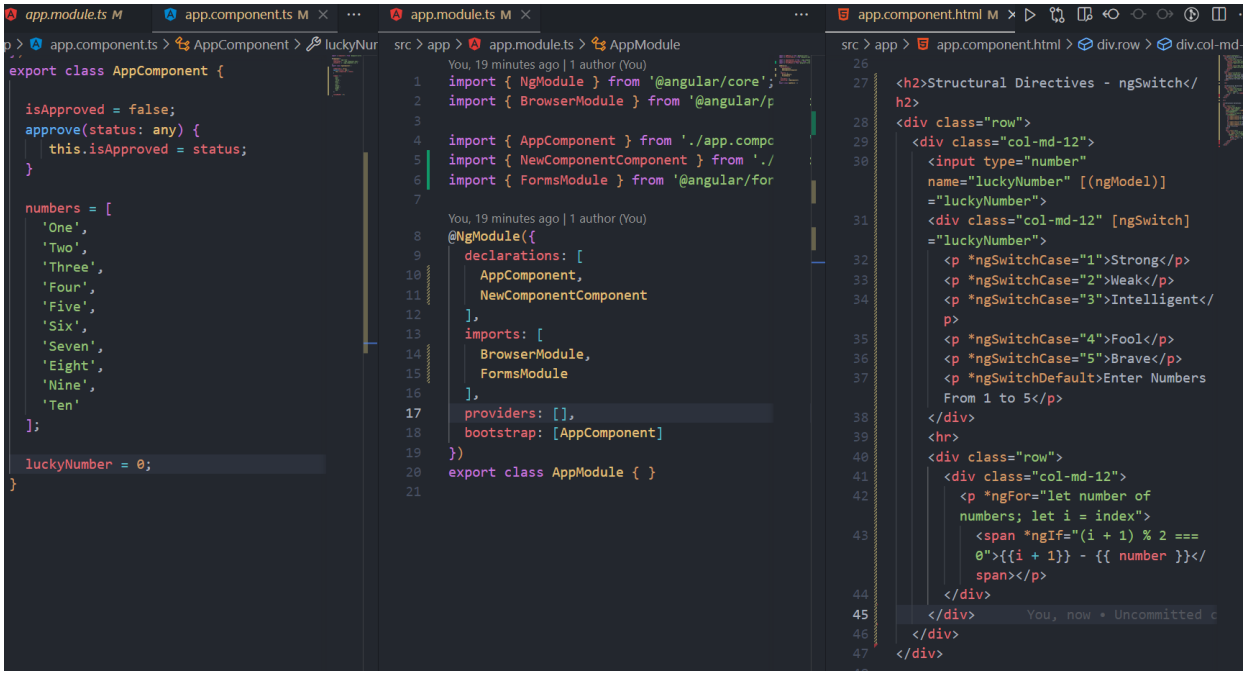
ngFor



```ts
export class AppComponent {

  isApproved = false;
  approve(status: any) {
    this.isApproved = status;
  }

  numbers = [
    'One',
    'Two',
    'Three',
    'Four',
    'Five',
    'Six',
    'Seven',
    'Eight',
    'Nine',
    'Ten'
  ];
}
```

```html
<h2>Structural Directives - ngFor</h2>
<div class="row">
  <div class="col-md-12">
    <p *ngFor="let number of numbers; let i = index">
      {{i + 1}} - {{ number }}</p>
  </div>
</div>
```

ngSwitch

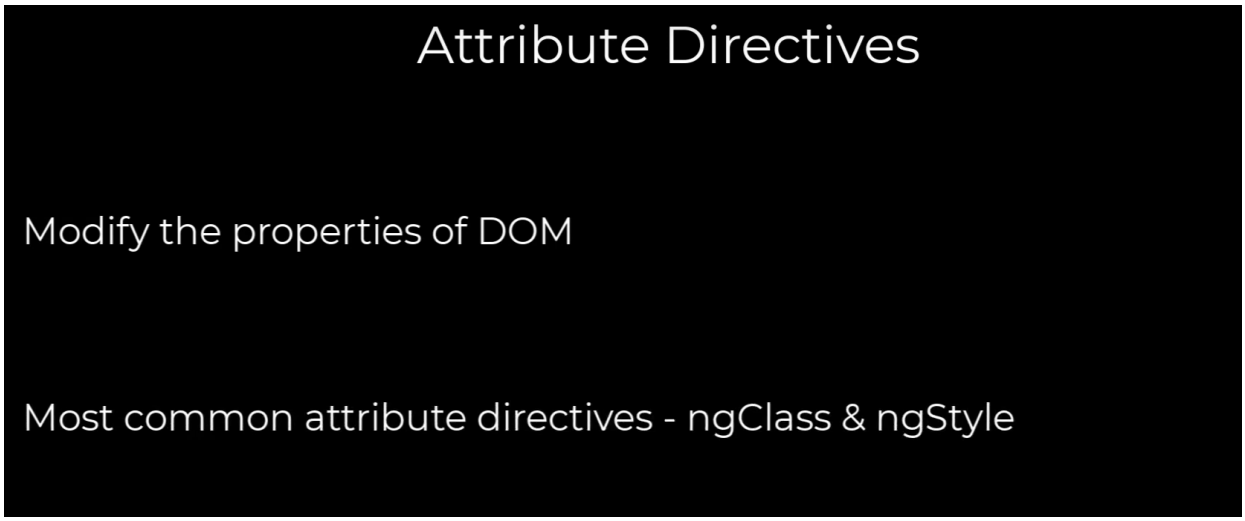


```ts
export class AppComponent {

  isApproved = false;
  approve(status: any) {
    this.isApproved = status;
  }

  numbers = [
    'One',
    'Two',
    'Three',
    'Four',
    'Five',
    'Six',
    'Seven',
    'Eight',
    'Nine',
    'Ten'
  ];

  luckyNumber = 0;
}
```

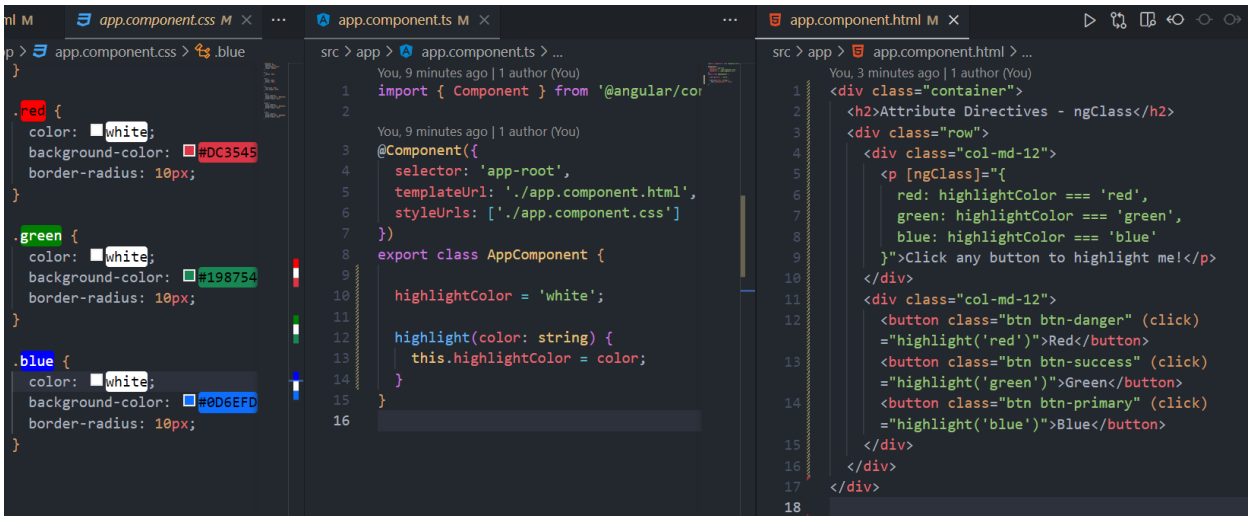
```ts
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/p

import { AppComponent } from './app.compo
import { NewComponentComponent } from './
import { FormsModule } from '@angular/for

@NgModule({
  declarations: [
    AppComponent,
    NewComponentComponent
  ],
  imports: [
    BrowserModule,
    FormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

```html
<h2>Structural Directives - ngSwitch</h2>
<div class="row">
  <div class="col-md-12">
    <input type="number" name="luckyNumber" [(ngModel)]="luckyNumber">
    <div class="col-md-12" [ngSwitch]="luckyNumber">
      <p *ngSwitchCase="1">Strong</p>
      <p *ngSwitchCase="2">Weak</p>
      <p *ngSwitchCase="3">Intelligent</p>
      <p *ngSwitchCase="4">Fool</p>
      <p *ngSwitchCase="5">Brave</p>
      <p *ngSwitchDefault>Enter Numbers From 1 to 5</p>
    </div>
    <hr>
    <div class="row">
      <div class="col-md-12">
        <p *ngFor="let number of numbers; let i = index">
          <span *ngIf="(i + 1) % 2 === 0">{{i + 1}} - {{ number }}</span></p>
      </div>
    </div>
  </div>
</div>
```

- **Attribute Directives**



ngClass

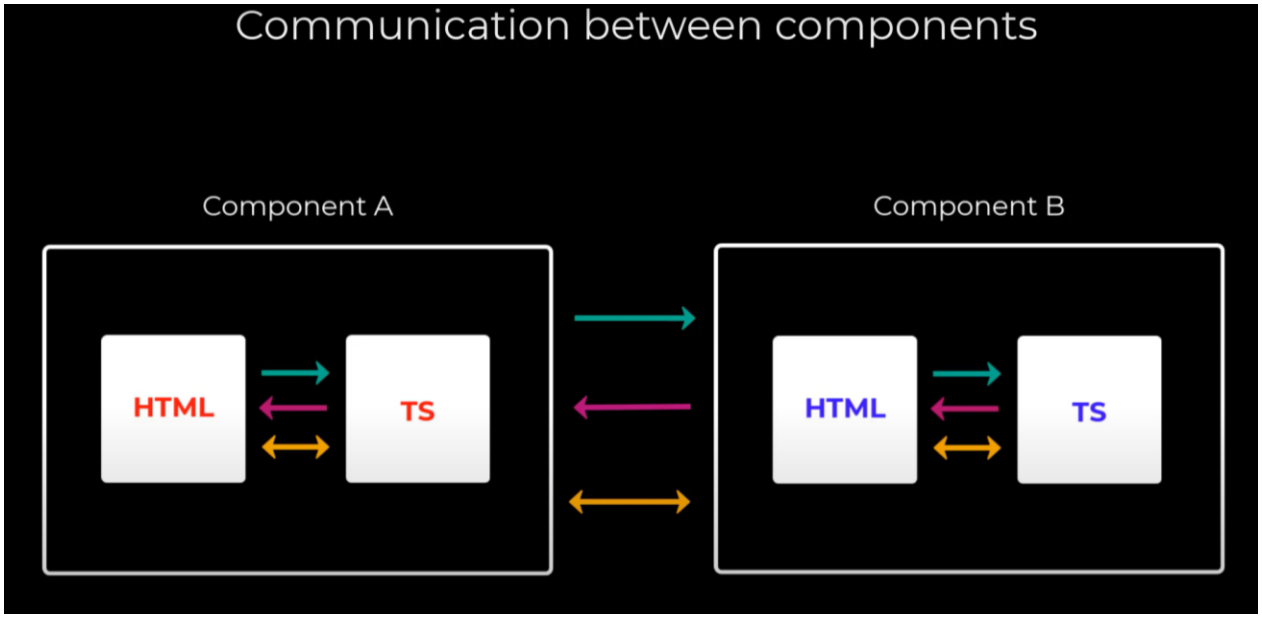


ngStyle

```ts
templateUrl: './app.component.html',
styleUrls: ['./app.component.css']
})
export class AppComponent {

highlightColor = 'white';
isStyleApplied = false;

highlight(color: string) {
  this.highlightColor = color;
}

toggleStyle() {
  this.isStyleApplied = !this.isStyleApplied;
}
}
```

```html
<div>
  <h2>Attributr Directives - ngStyle</h2>
  <div class="row">
    <div class="col-md-12">
      <p [ngStyle]="{
        color: isStyleApplied === true ? 'white' : 'black',
        backgroundColor: isStyleApplied ? 'green' : 'white'
      }">Click the button to toggle the style!</p>
    </div>
    <div class="col-md-12">
      <button class="btn btn-primary" (click)="toggleStyle()
      ">Toggle Style</button>
    </div>
  </div>
</div>
</div>
```

Communication between components

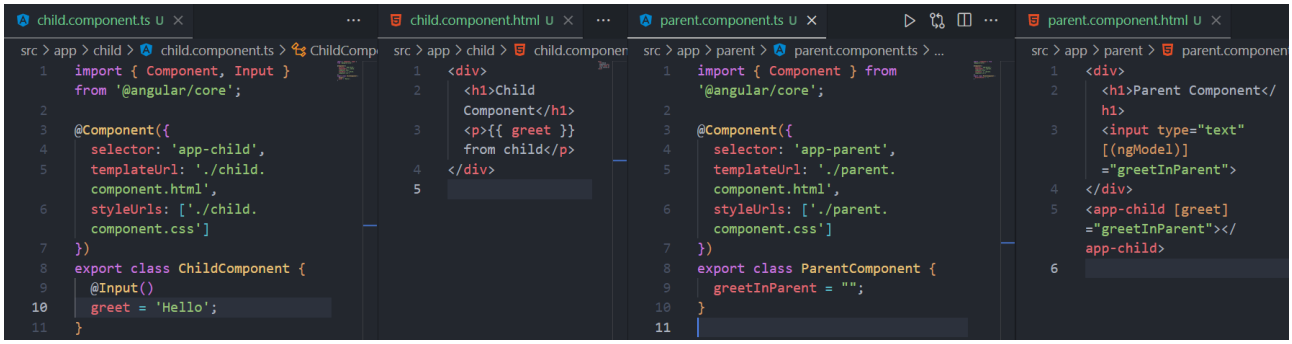


Creating two components – ng g c parent and ng g c child

Our task- Instead of hardcoding this value in the child component, we need to pass it from parent component when user types something in the input box.

@Input

We have to pass data from child component



child.component.ts U

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

@Component({
  selector: 'app-child',
  templateUrl: './child.
  component.html',
  styleUrls: ['./child.
  component.css']
})
export class ChildComponent {
  @Input()
  greet = 'Hello';
}
```

child.component.html U

```
<div>
  <h1>Child
  Component</h1>
  <p>{{ greet }}
  from child</p>
</div>
```

parent.component.ts U

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

@Component({
  selector: 'app-parent',
  templateUrl: './parent.
  component.html',
  styleUrls: ['./parent.
  component.css']
})
export class ParentComponent {
  greetInParent = "";
}
```

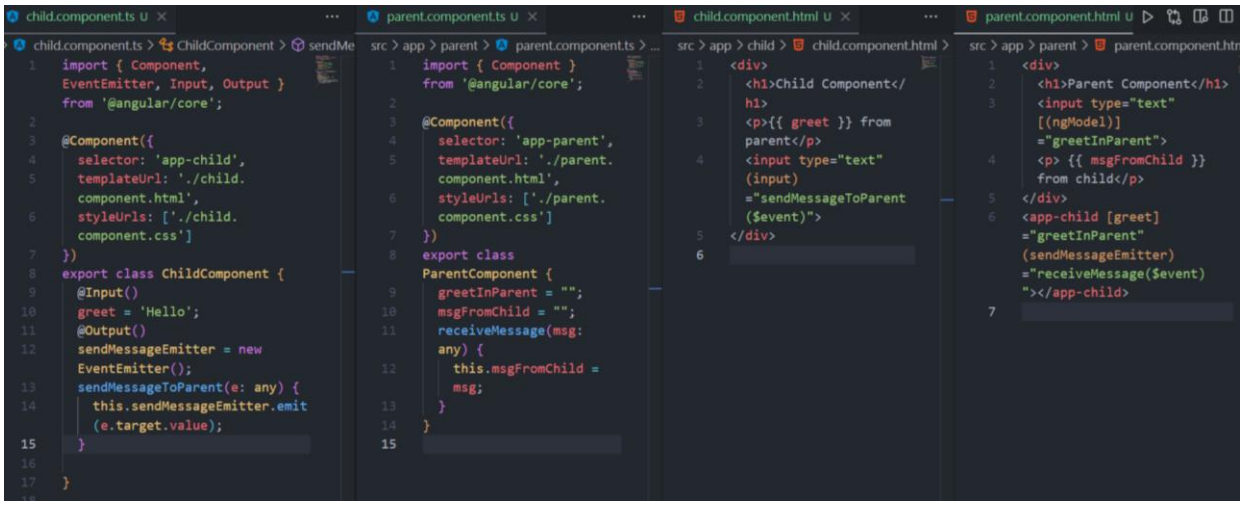
parent.component.html U

```
<div>
  <h1>Parent Component</
  h1>
  <input type="text"
  [(ngModel)]
  ="greetInParent">
</div>
<app-child [greet]
="greetInParent"></
app-child>
```

Now we need to pass data from child component to parent component

We can see, in parent component we have access to child component and so we bind the data using property binding. But if we see in child component we don't have access to parent component. So we cannot follow the property binding approach in this case. We need to use an event binding and an event emitter approach.

@Output



child.component.ts U

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

@Component({
  selector: 'app-child',
  templateUrl: './child.
  component.html',
  styleUrls: ['./child.
  component.css']
})
export class ChildComponent {
  @Input()
  greet = 'Hello';
  @Output()
  sendMessageEmitter = new
  EventEmitter();
  sendMessageToParent(e: any) {
    this.sendMessageEmitter.emit
    (e.target.value);
  }

}
```

parent.component.ts U

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

@Component({
  selector: 'app-parent',
  templateUrl: './parent.
  component.html',
  styleUrls: ['./parent.
  component.css']
})
export class
ParentComponent {
  greetInParent = "";
  msgFromChild = "";
  receiveMessage(msg:
  any) {
    this.msgFromChild =
    msg;
  }
}
```

child.component.html U

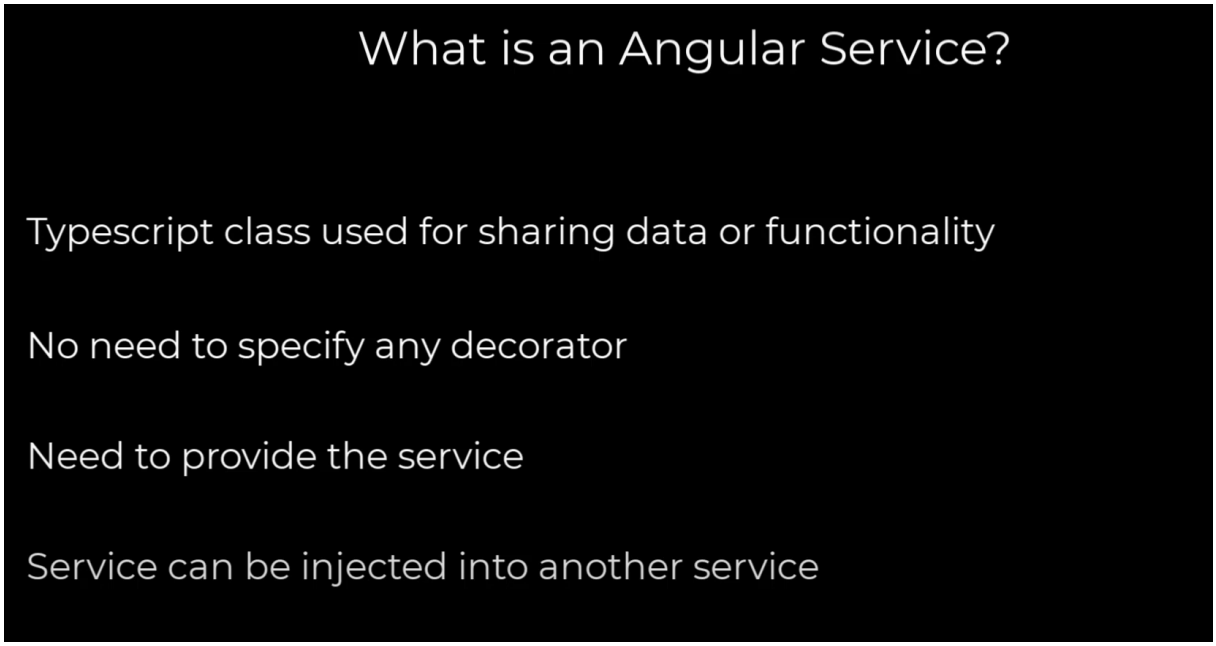
```
<div>
  <h1>Child Component</
  h1>
  <p>{{ greet }} from
  parent</p>
  <input type="text"
  (input)
  ="sendMessageToParent
  ($event)">
</div>
```

parent.component.html U

```
<div>
  <h1>Parent Component</h1>
  <input type="text"
  [(ngModel)]
  ="greetInParent">
  <p> {{ msgFromChild }}
  from child</p>
</div>
<app-child [greet]
="greetInParent"
(sendMessageEmitter)
="receiveMessage($event)
"></app-child>
```

Services & Dependency Injection

Angular Service is just a typescript class which is mainly used for sharing data or functionality throughout the application. We have seen Components and Directives are also typescript classes but we added some decorators like @Component and @Directive and informed Angular that we need to treat them specially. But we don't want to specify any decorators for Services. But, we should provide the service either in the component or in the module. When we provide the service in the component then that component and it's child component will get the same instance of that service. When we provide the service in module, then all the components inside that module will get the same instance of that service. When we provide the service in root module, then the entire application will have only one instance of that service. A service can be injected into another service. In that case, it is necessary to add @Injectable decorator to the service which is receiving the other service. For the remaining services, decorator is not mandatory. But newer version of Angular suggests to add this @Injectable decorator to all services.

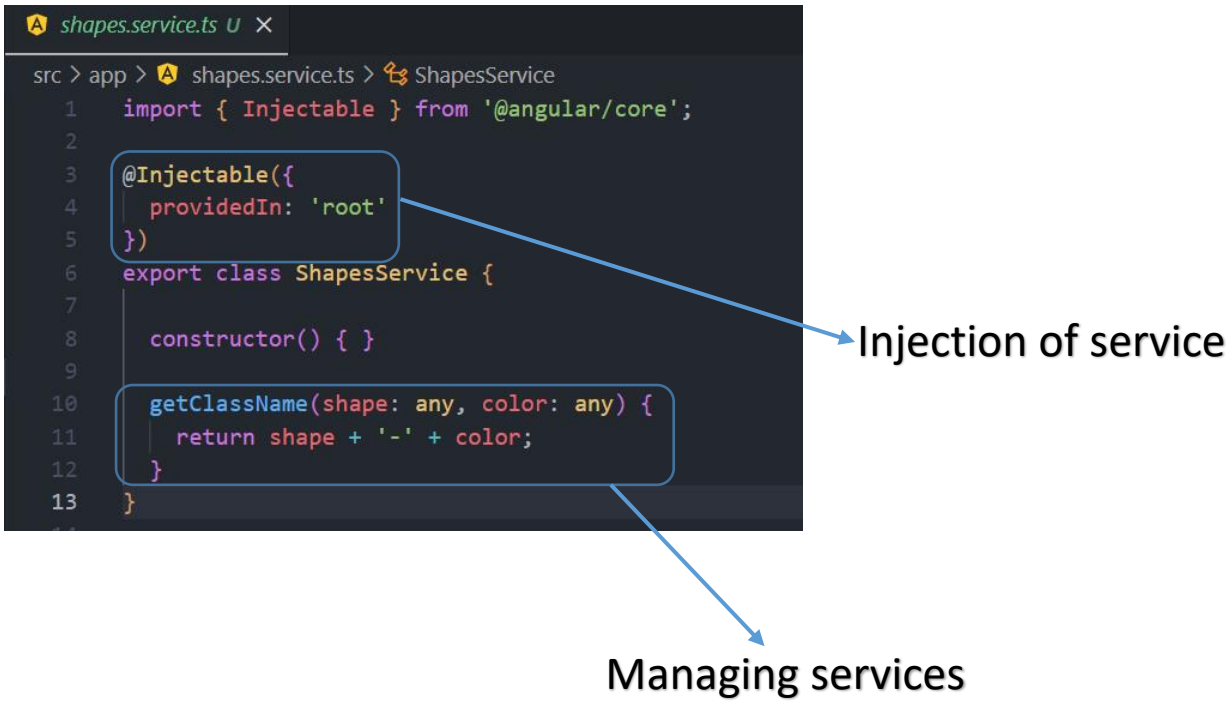


Uses of Service

Code Reuse

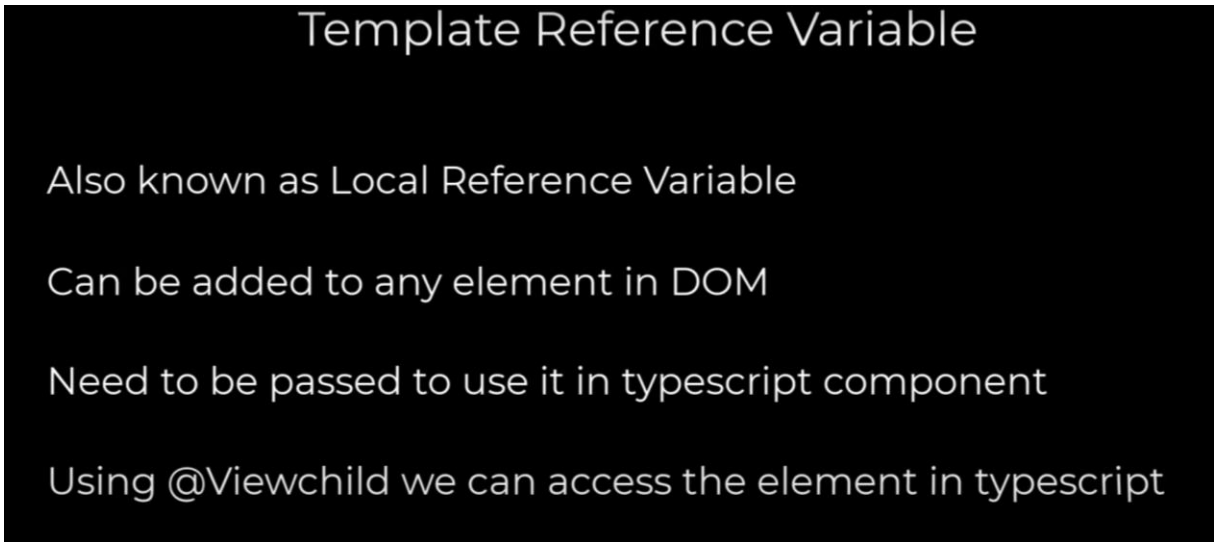
Cross component Communication

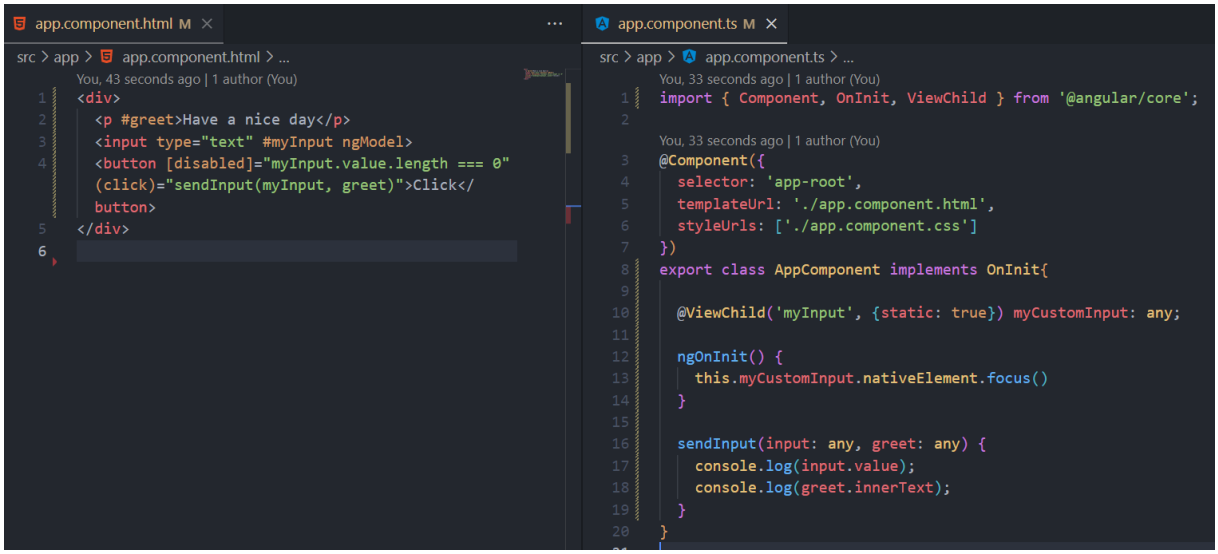
Creating a service – ng g s service-name [ng generate service service-name]



Template Reference Variable & @Viewchild

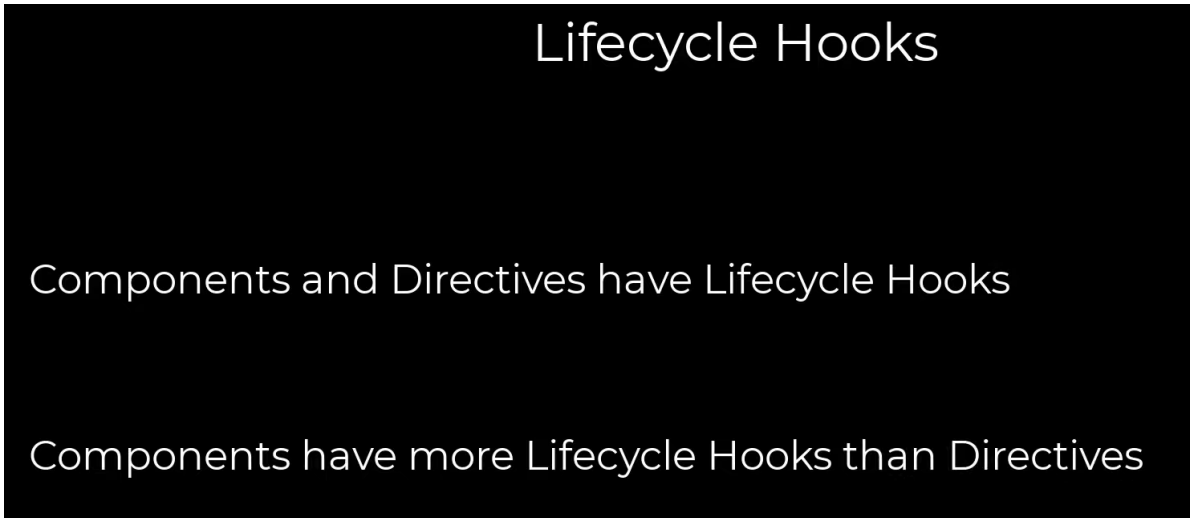
A template reference variable or a local reference variable is a variable which we add in any html element using # as the first character. Using template reference variable we can access the dom elements and their properties easily. But, template reference can be used only in the HTML template. In order to access that in the typescript component we need to explicitly pass the reference to Typescript through an event in the html template. But sometimes we need to get access to the DOM before we trigger an event in HTML. For that type of usecase, we can use @ViewChild.

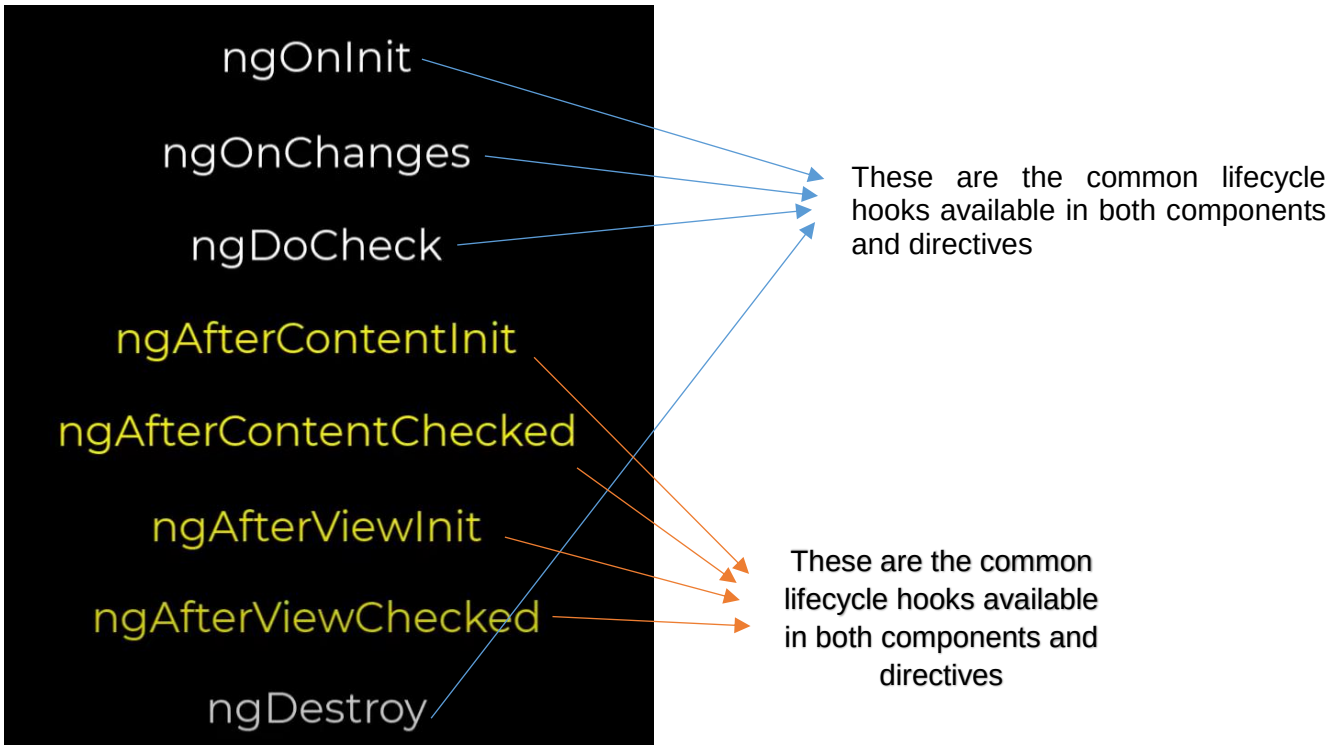




Life Cycle Hooks

All components and directives in Angular have a lifecycle as Angular creates, updates, and destroys them. We can implement the lifecycle hook interfaces and use the respective method to tap into any lifecycle moments. Components have more lifecycle hooks than directives.

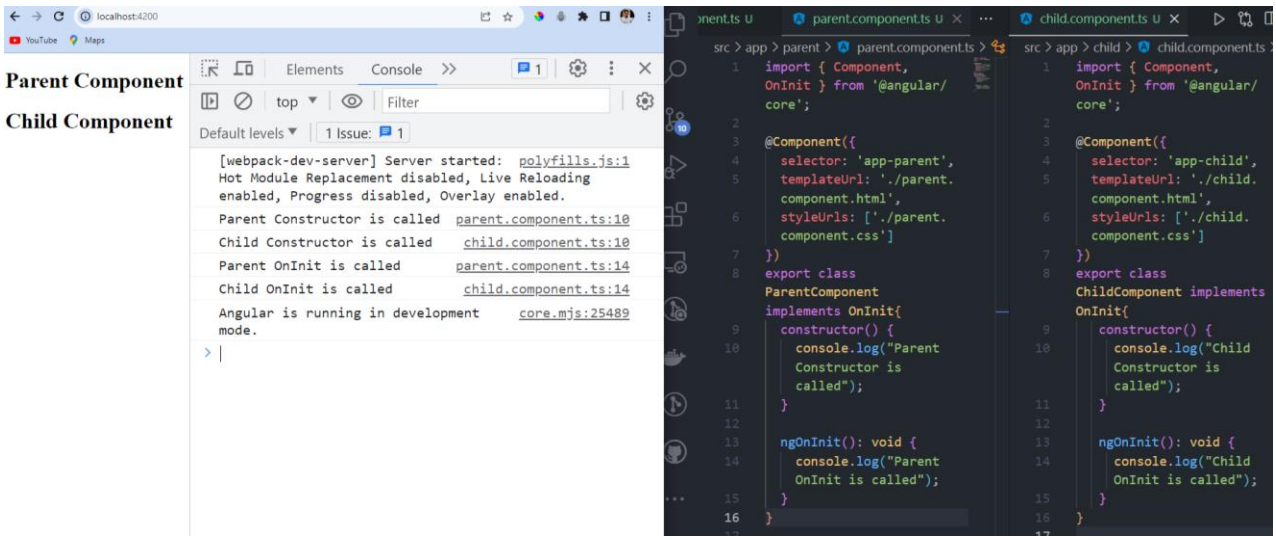


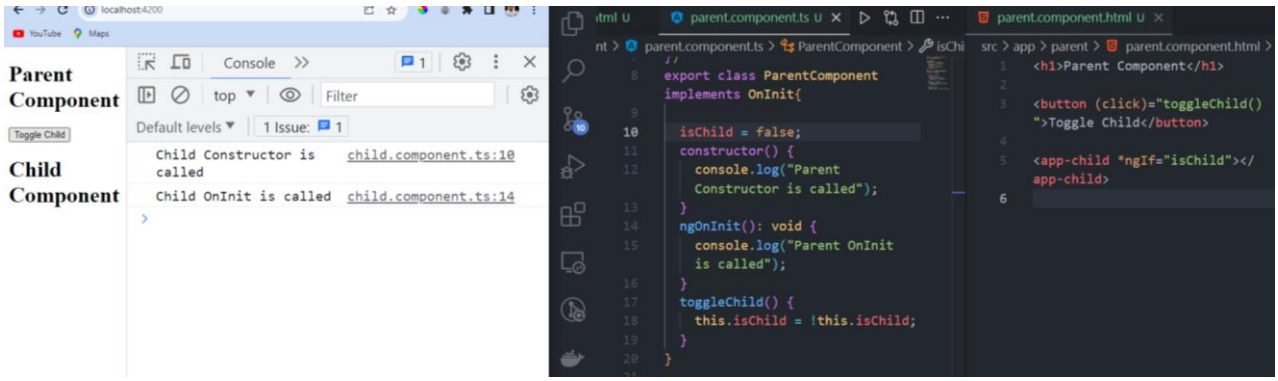


Constructor is not a life cycle hook. It is a typescript feature.

- **ngOnInit**

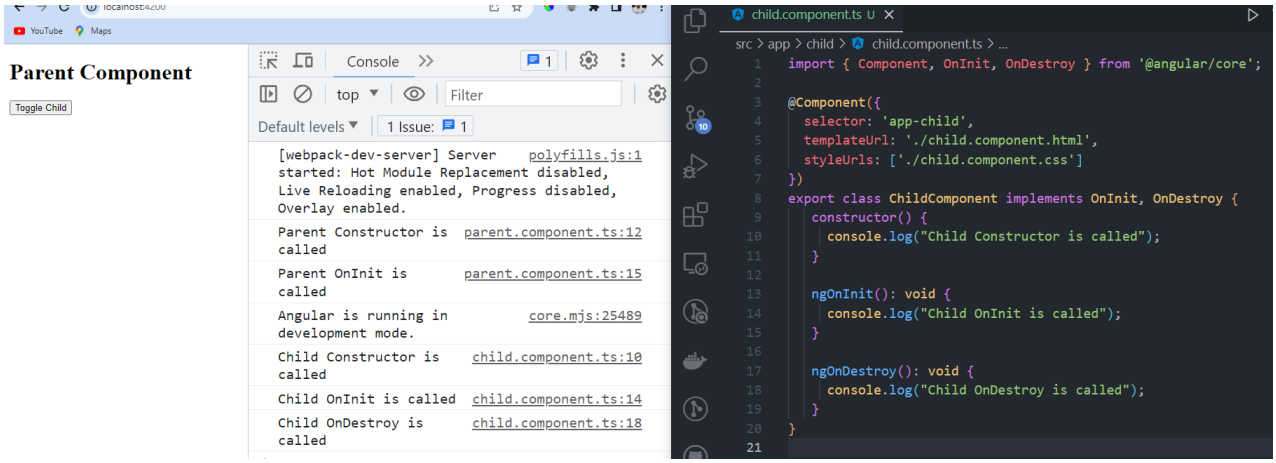
ngOnInit will be invoked when the component has initialized. This hook will be called only once after the first ngOnChanges. We can implement the OnInit interface to use this hook. Mostly, we use ngOnInit for doing initializations. When we create a component using Angular CLI, we can see the ngOnInit hook has been implemented already.



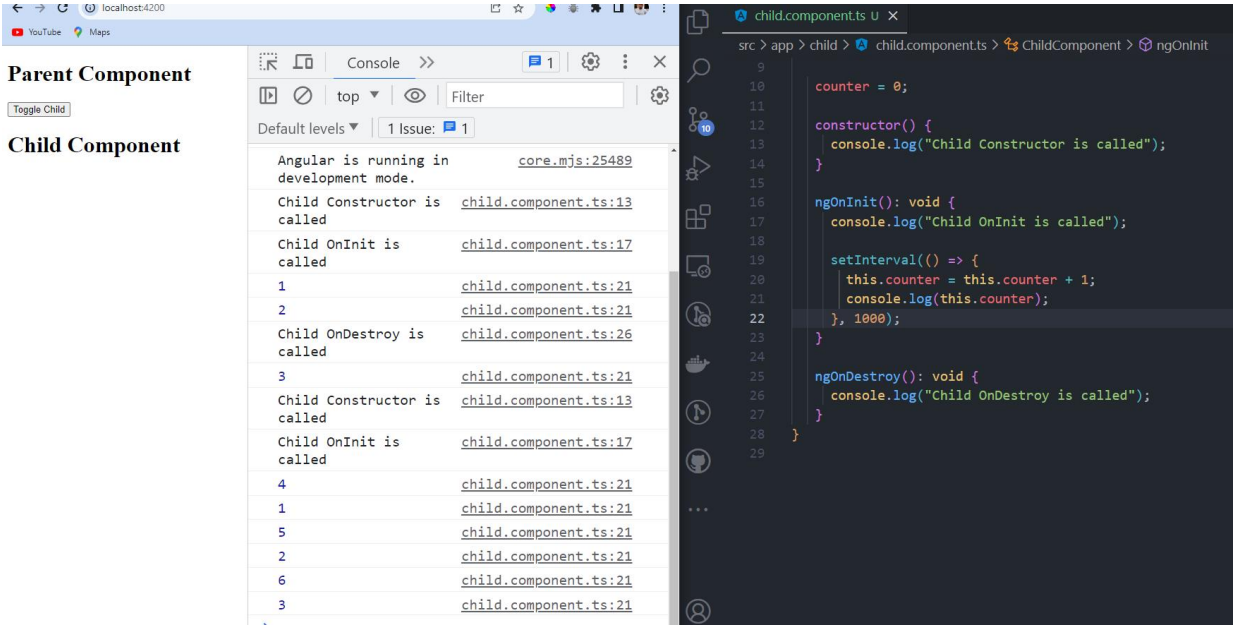


- **ngOnDestroy**

ngOnDestroy will be called once just before the component is removed from the DOM, in other words when the component is destroyed from the DOM.

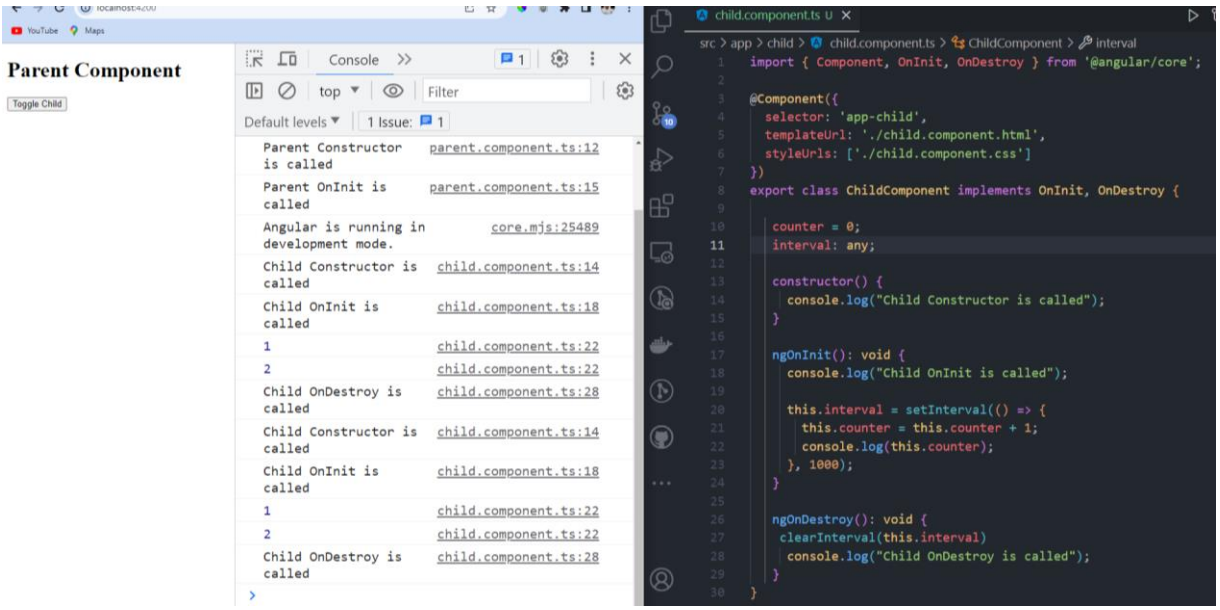


When we click the button, we can see constructor is called once and then OnInit is called. If I click again, OnDestroy is called and so component is removed. If I click again, component is constructed again so constructor is called again and then OnInit is called. Clicking again will destroy it. We can understand, OnDestroy will be called once when the component is removed from the DOM. So, if we need any cleaning activities we can do it in OnDestroy.



Once We initialize the component we can see the counter getting incremented. But, when we destroy the component, we can see the component is destroyed but the interval we created is still running, so if we toggle the button again we can see two intervals are running and even after destroying they are still running. So, this is a memory leak. We should avoid these type of memory leak.

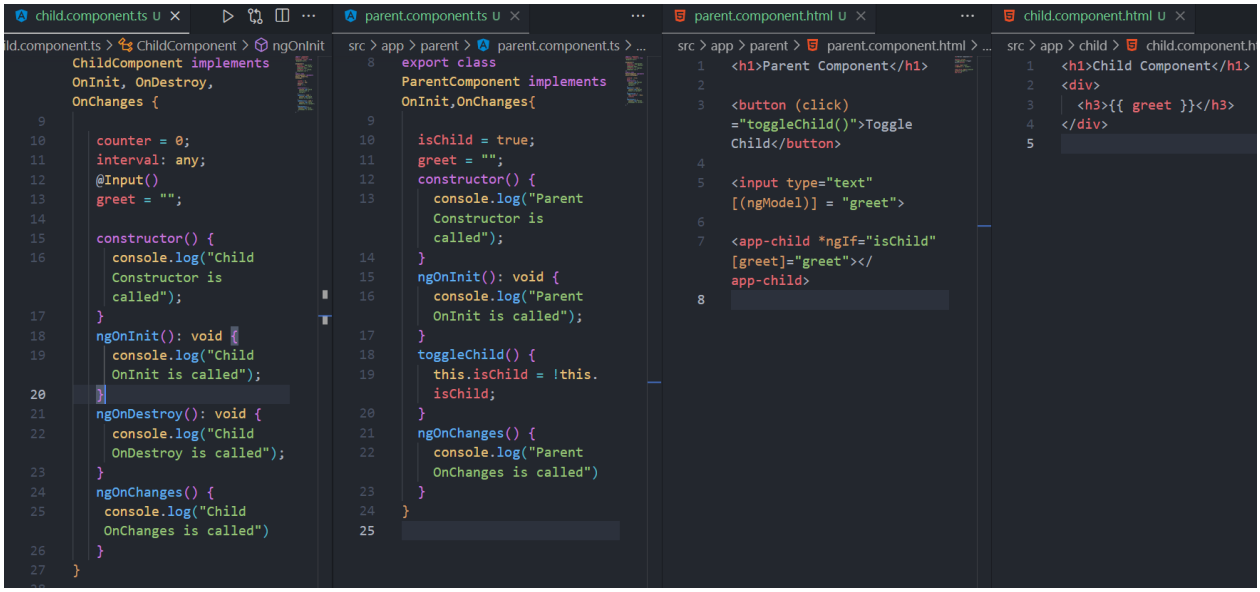
So, we can use onDestroy to clear these intervals, we can unsubscribe if we subscribe any observables, etc. So, in child component get the interval and clear that in onDestroy hook.

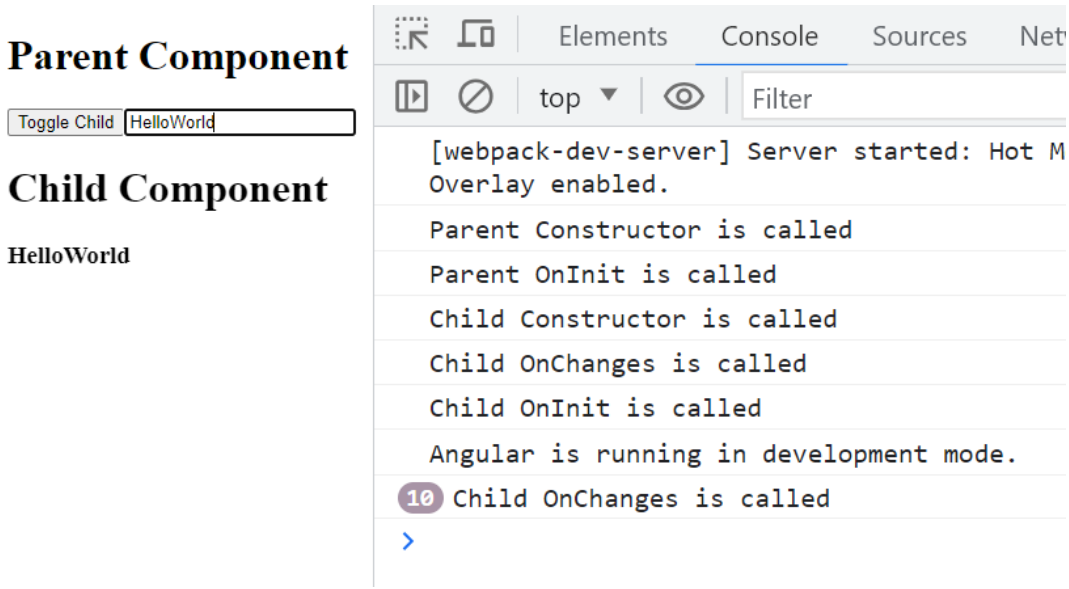


So, when we click once, we can see the counter has started and when we destroy the component, we can see the counter is also cleared. If we click again a new counter is starting and now it is stopped. So, we have fixed the memory leak.

- **ngOnChanges**

OnChanges will be called once before OnInit and will be called whenever the data-bound input properties changes. When we say data-bound input, it is not the normal input in the same component. It should be something coming outside of that component through @Input.

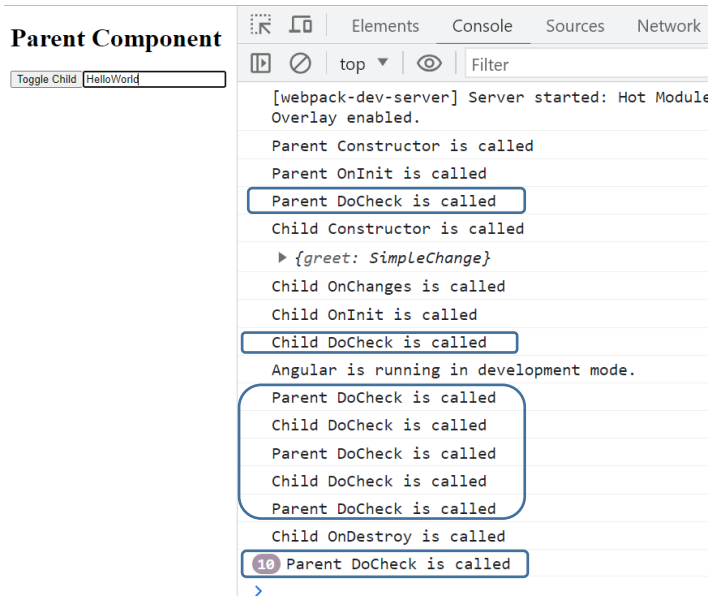




When we load the page, we can see parent constructor is called and then parent oninit is called after that child constructor is called and before child oninit we can see child onchanges is called once. When we change this input value in parent, the onchanges in child is called for every keystroke. so, it is not a good practice to do any expensive operations like API call inside onChanges. Also, ngOnChanges accepts a SimpleChanges object and using that we can make use of previous and current values.

- **ngDoCheck**

DoCheck will be invoked when the change detector of the given component is invoked. It is called immediately after ngOnChanges() on every change detection run, and immediately after ngOnInit() on the first run.

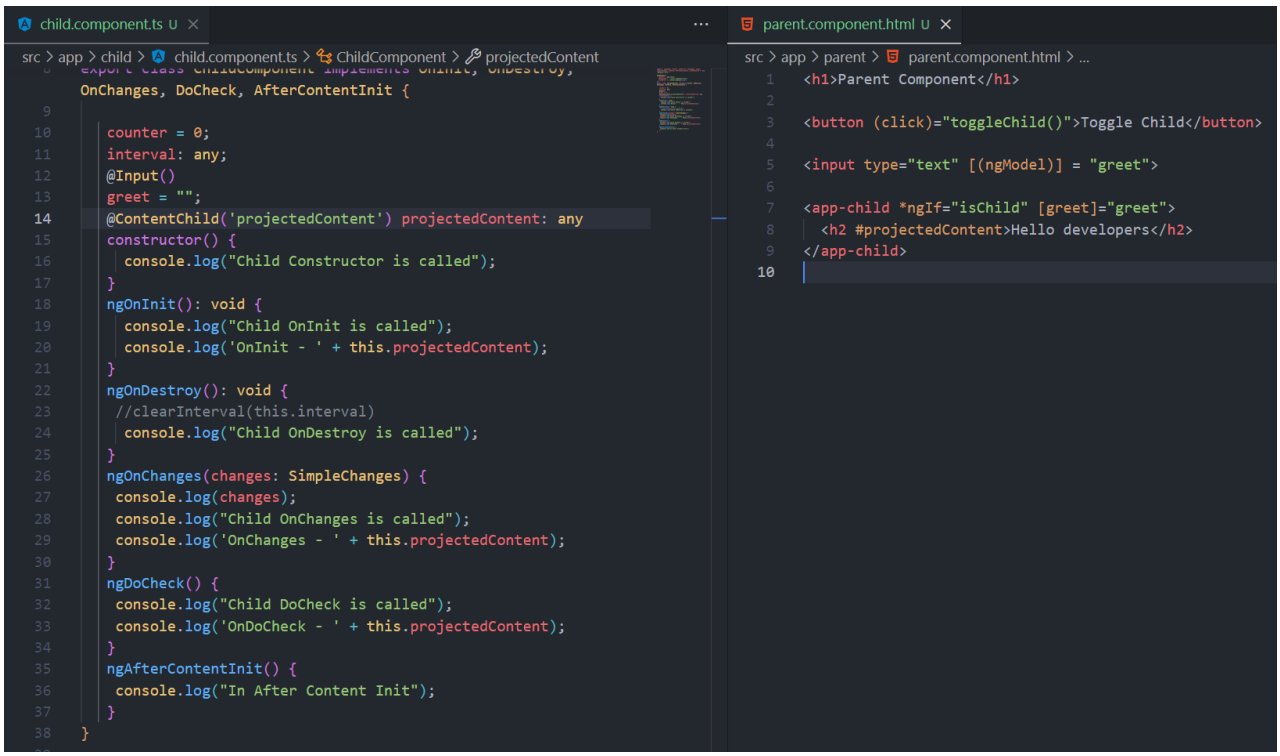


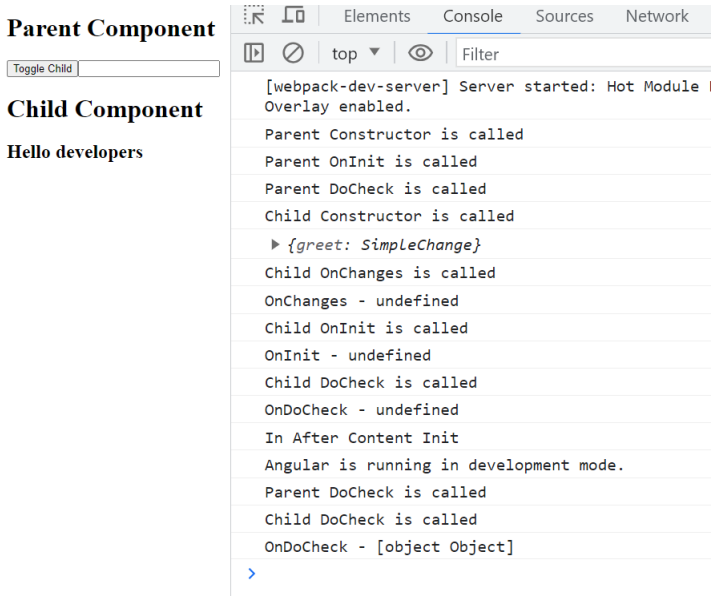
We can see parent docheck is called just after parent oninit and child do check is called just after child oninit. And, in development mode change detection run twice and so we can see Docheck is called one more time in each component. And, when we click the button in parent, we can see the parent do check is called.Also, when we

type, docheck is called again. even when I click outside after typing, it is called. Similarly, when I type something in parent, it triggers the onchanges in child because it is an inbound property. But, we can also see the child docheck is called after child onchanges to detect changes. So, we should avoid triggering any expensive operations inside ngDoCheck. Also, we should avoid using ngOnChanges and ngDoCheck in the same component.

- **ngAfterContentInit**

ngAfterContentInit will be called only once after the first docheck. We need to do a content projection.

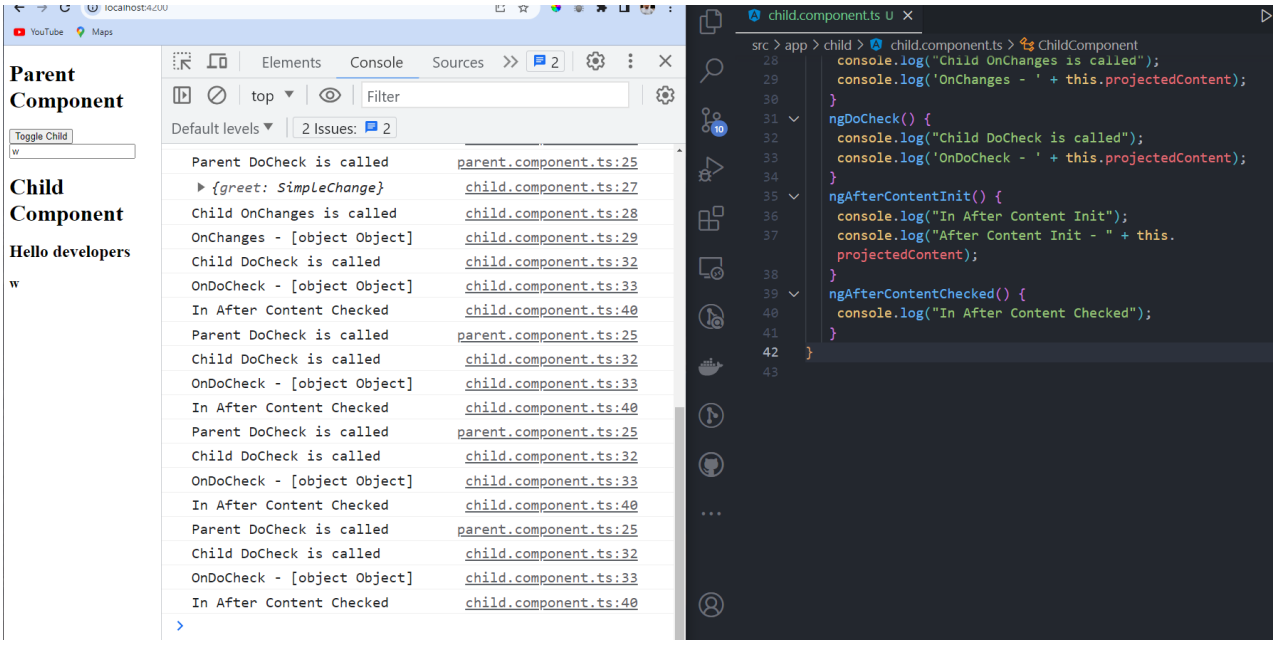




We can see AfterContentInit is called after Child docheck and also, we cannot access this variable till the content is initialized. That means, in all other hooks this is coming as undefined and only in AfterContentInit

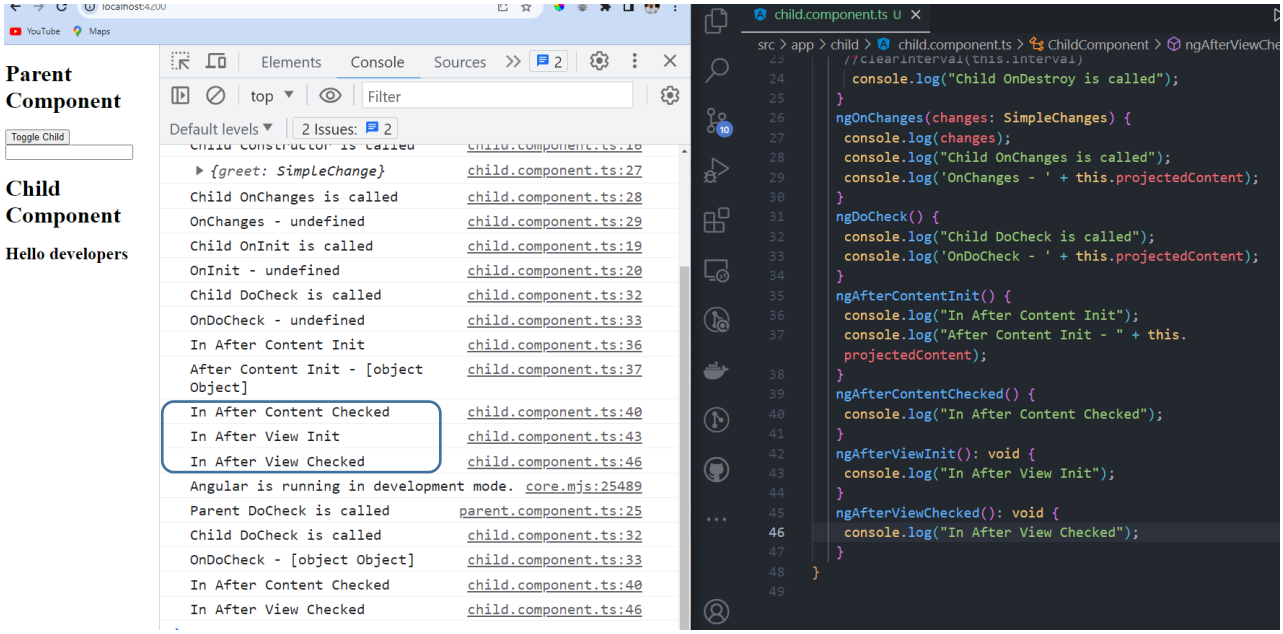
- **ngAfterContentInit**

ngAfterContentChecked will be called after ngAfterContentInit() and every subsequent ngDoCheck().



- **ngAfterViewInit & ngAfterViewChecked**

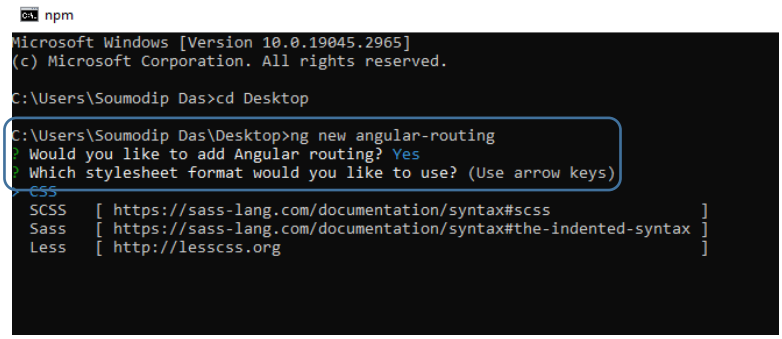
ngAfterViewInit will be called once after the first ngAfterContentChecked and ngAfterViewChecked will be called after ngAfterViewInit and after every subsequent ngAfterContentChecked.

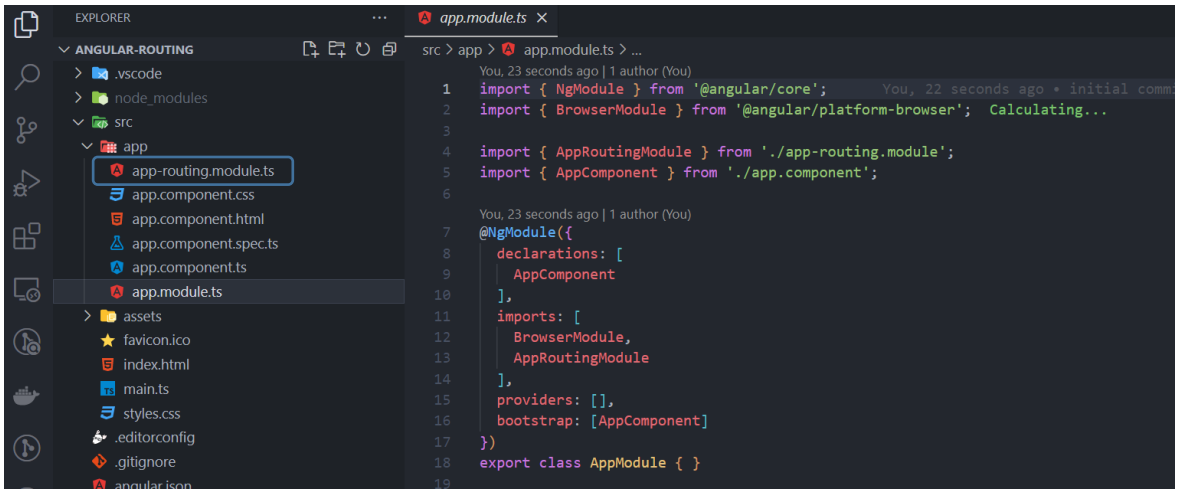


Routing

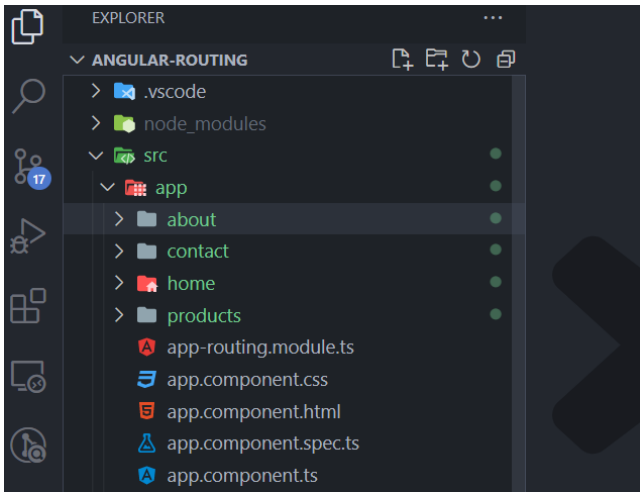
Routing means Navigating between pages. Navigation is one of the important aspect in a web application. Even though a single page application (SPA) does not have multiple page concept, it does moves from one view to another view. So, Routing is an important concept in Angular and the Angular router is a core part of the Angular platform. It enables developers to build Single Page Applications with multiple views and allow navigation between these views.

We will create an app with routing

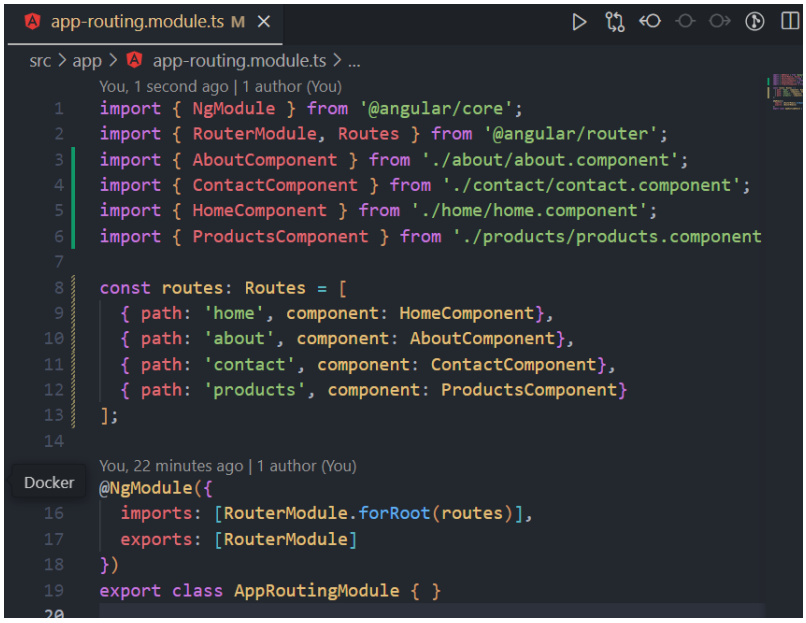




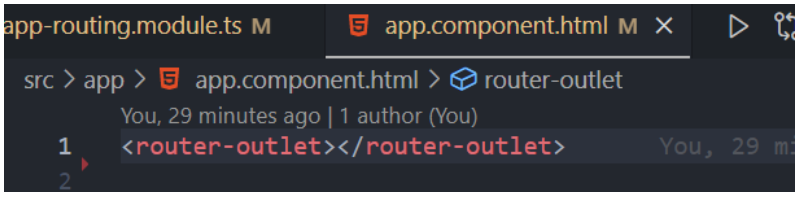
We created 4 new components



In Routing module, we can see a constant of type routes from angular/router. Through this routes, we need to inform angular about which compoment should be loaded when we hit a path in URL. Routes is an array in which we can pass a javascript object. It accepts many optional keys, one of them is path. So, now we need to inform what should angular do when it meets the path home. So, for that we can use another key, component and mention HomeComponent there. One thing to note here is path value is a string and Home Component is a component and so we need to import that and we will create paths for other components.



Angular knows which component it needs to load when a path is matched. But we haven't informed angular yet where the component has to loaded. We can inform that using router-outlet directive. So, in App component we will create a router outlet directive.



But if we see when the server is started, the URL will be just localhost:4200. There will not be any path added. So, we can also inform angular what we need to do when there is no path. So, we will add an empty path route.

```
const routes: Routes = [
  { path: '', component: HomeComponent},
  { path: 'about', component: AboutComponent},
  { path: 'contact', component: ContactComponent},
  { path: 'products', component: ProductsComponent}
];
```

Now we will create a header component and all the route works But this is wrong, because we are refreshing the page everytime. We can see that in the console. Everytime when I hit an url, the application is loading again. You can also see that in the browser tab. So, how can we fix that. Angular provided another directive to resolve this. We should not use href. Instead, we can use RouterLink directive.

```
const routes: Routes = [
  { path: '', redirectTo: 'home', pathMatch: 'full'},
  { path: 'home', component: HomeComponent},
  { path: 'about', component: AboutComponent},
  { path: 'contact', component: ContactComponent},
  { path: 'products', component: ProductsComponent},
  { path: 'product/:id', component: ProductsComponent},
  { path: '**', redirectTo: 'home'}
];
```

Redirect to home for entering a wrong path

app-routing.module.ts M | header.component.html U

src > app > header > header.component.html > ...

```
<header>
  <nav>
    <a routerLink="/home" routerLinkActive="active">Home</a>
    <a routerLink="/about" routerLinkActive="active">About</a>
    <a routerLink="/contact" routerLinkActive="active">Contact</
    a>
    <a routerLink="/products"
    routerLinkActive="active">Products</a>
  </nav>
</header>
```

Assume we need to do something before moving to the new page. Example: Assume, we need to update in Database and after successful update, we need to go to the new page. There is another way of routing. It is called routing programmatically.

home.component.html U

src > app > home > home.component.html > ...

```
<div class="main-container">
  <h1>Home page</h1>
  <div>
    <button (click)="goToProducts
    ()">Go To Products</button>
  </div>
</div>
```

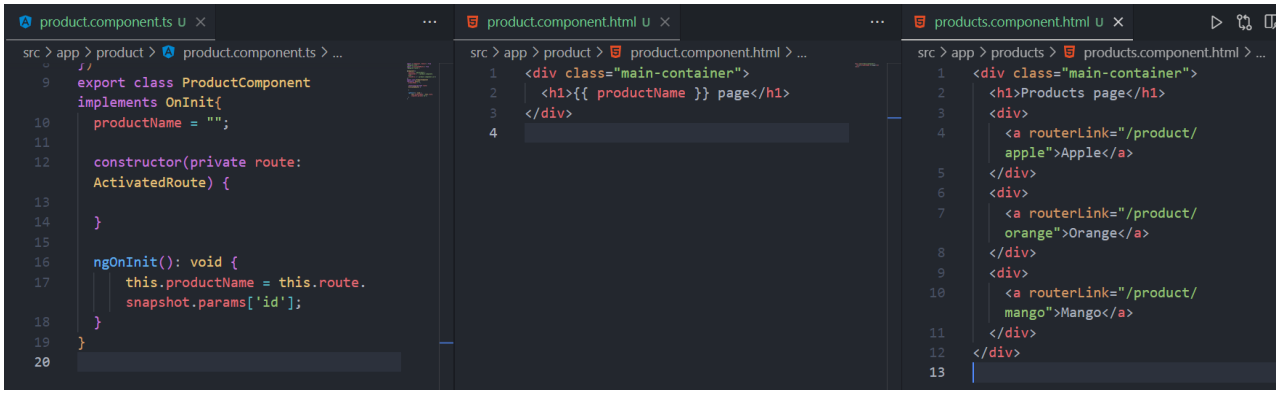
home.component.ts U

src > app > home > home.component.ts > HomeComponent

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

@Component({
  selector: 'app-home',
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.css']
})
export class HomeComponent {
  constructor(private router: Router) {

  }
  goToProducts() {
    console.log("Trying to update database...");
    setTimeout(() => {
      console.log("Database is updated");
      this.router.navigate(['products']);
    },2000);
  }
}
```

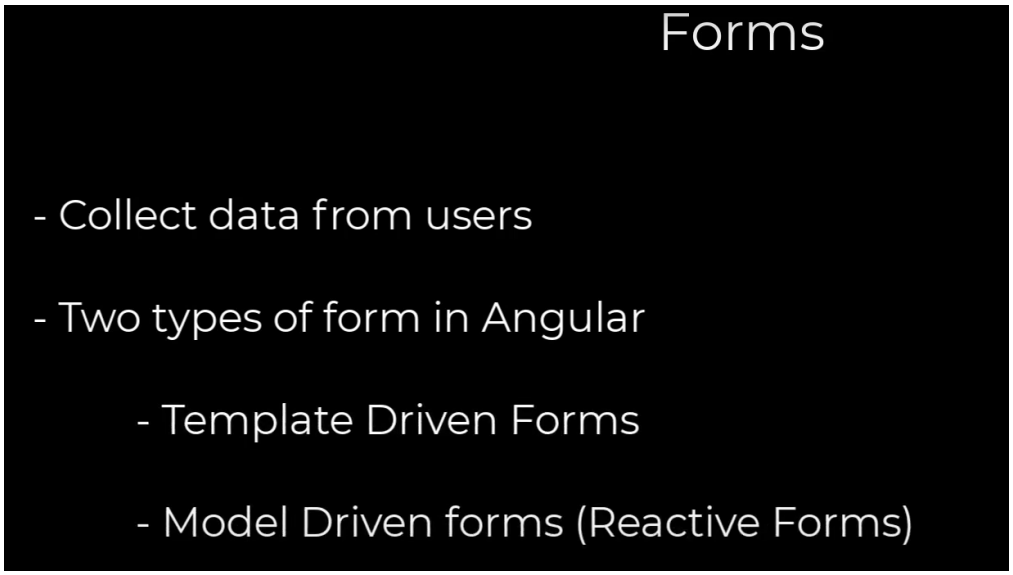
How we can pass a parameter through route?

To achieve this, let me create a product page and configure the routes.



Forms

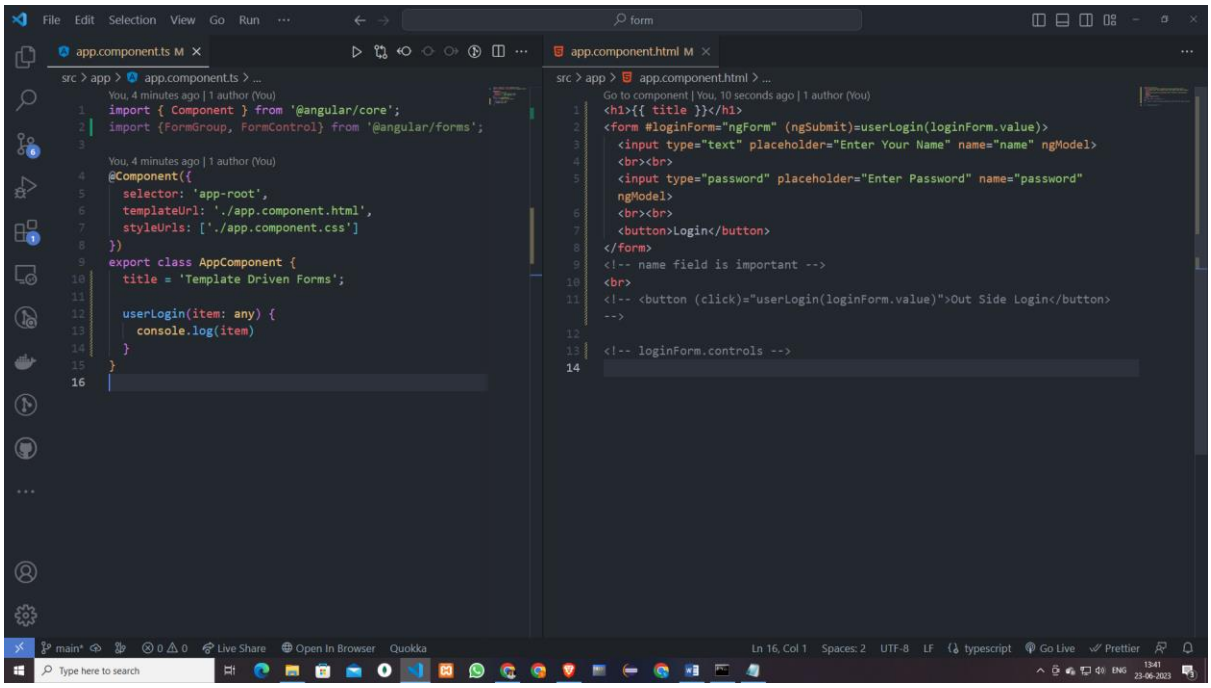
Forms in general are used to collect data from users. Some applications use forms to enable users to log in, to register, to update a profile and more. Angular supports two types of forms. They are Template driven forms and Model driven forms(Reactive forms).



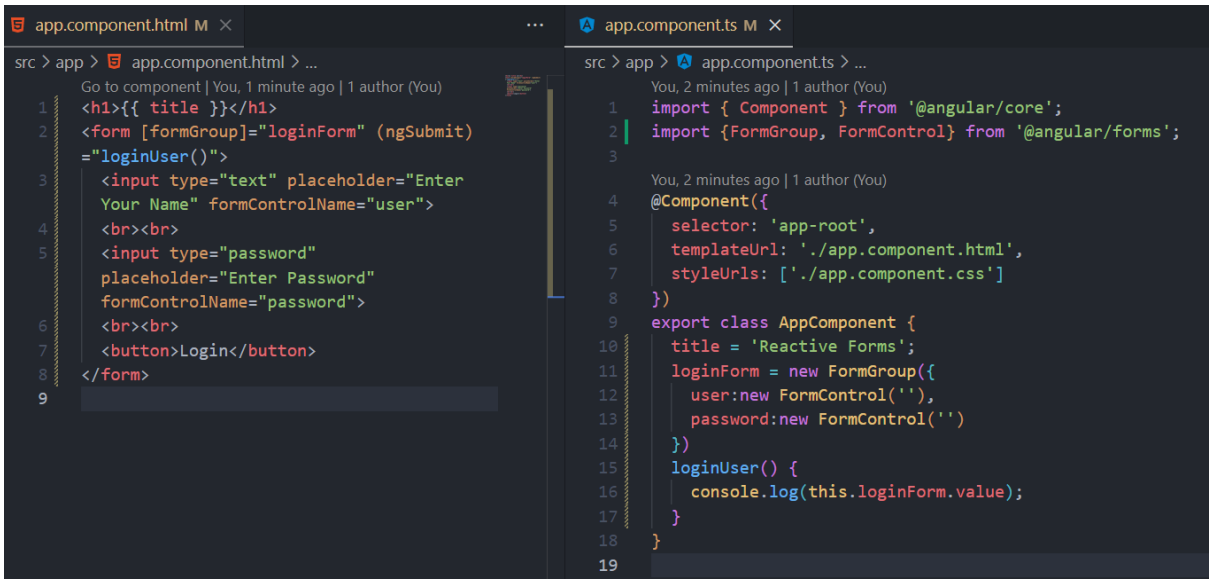
Template-driven approach is the easiest way to build the Angular forms. The logic of the form is placed in the template. But in Reactive forms, the logic of the form is defined in the component as an object. Compared to template-driven forms, they are more robust: they're more scalable, reusable, and testable So, Template-driven forms are suitable for small or simple forms, while reactive forms are more scalable and suitable for complex forms.

Template Driven	Reactive
Easy to create	
Logic in Template	Logic in Component
	Robust, Scalable, Reusable & Testable
Suitable for simple forms	Suitable for complex forms

- **Template Driven Form**



- **Reactive Form**



HTTP

In a typical web application, there may be static as well as dynamic contents. When we say static contents, those are hardcoded in the html code itself. And, dynamic contents may come from a database or any other backend.

- **API(Application Programming Interface)**

API is a software intermediary or a messenger which helps to talk to different application or database. And, Most front-end applications need to communicate with a server over the HTTP protocol. And, Angular provides us a HTTP Client module which we can use for that purpose.

https://api.github.com – API

https://api.github.com/users - API Response

We have to import the HttpClient Module in app

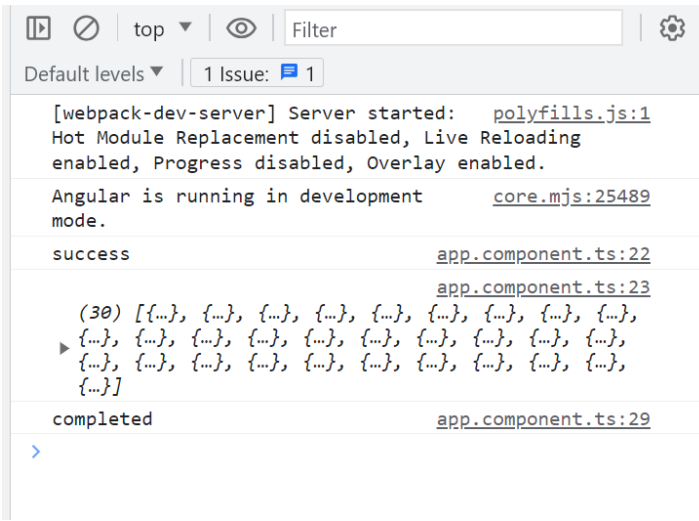
We can see the data is coming but in Observable. How we can get the data from Observable

```
users = [1, 2, 3]
url = "https://api.github.com/users";
constructor(private http: HttpClient) {

}
ngOnInit(): void {
    this.getUsers();
}
getUsers() {
  console.log(this.http.get(this.url)) //http client method
}
goToGithub(url: any) {
  window.open(url, '_blank')
}
```

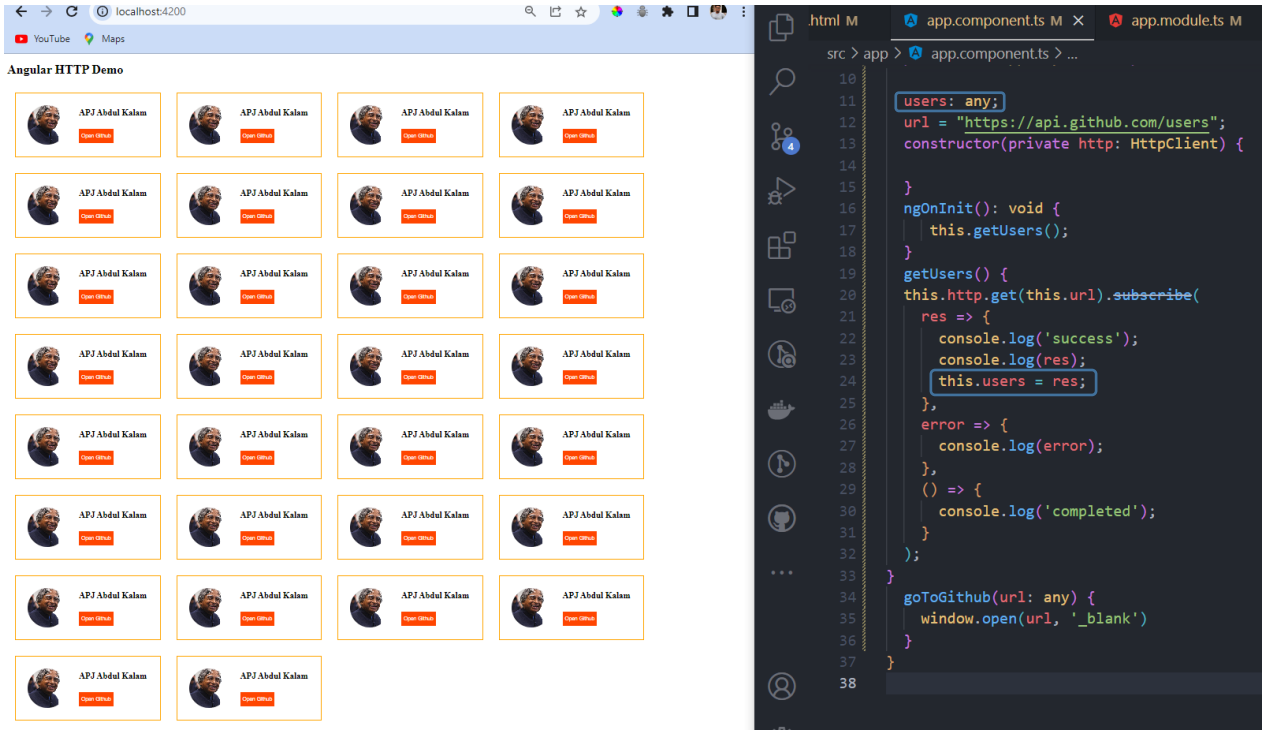
```
                                                    app.component.ts:20
▼ Observable i
  ▶ operator: ƒ (liftedSource)
  ▶ source: Observable {source: Observable, operator: ƒ}
  ▶ [[Prototype]]: Object
Angular is running in development mode.                 core.mjs:25489
>
```

There are three methods in observable.

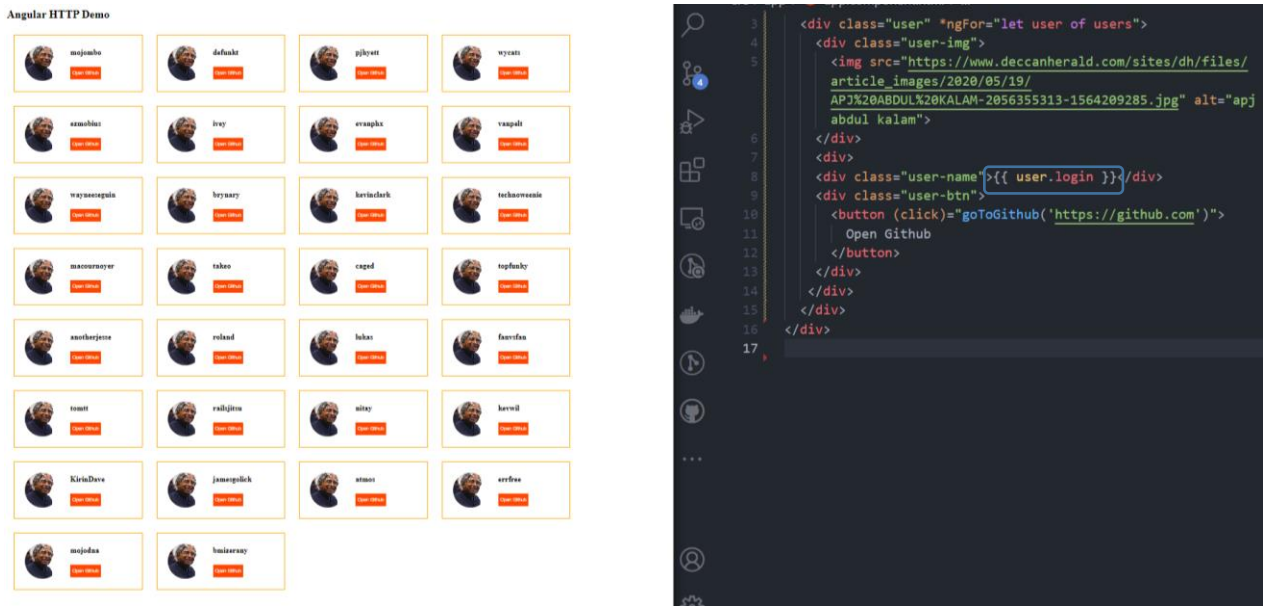
- Success
- Error
- Completion



Now, we got the data. we need to map it correctly to the html template using data binding.

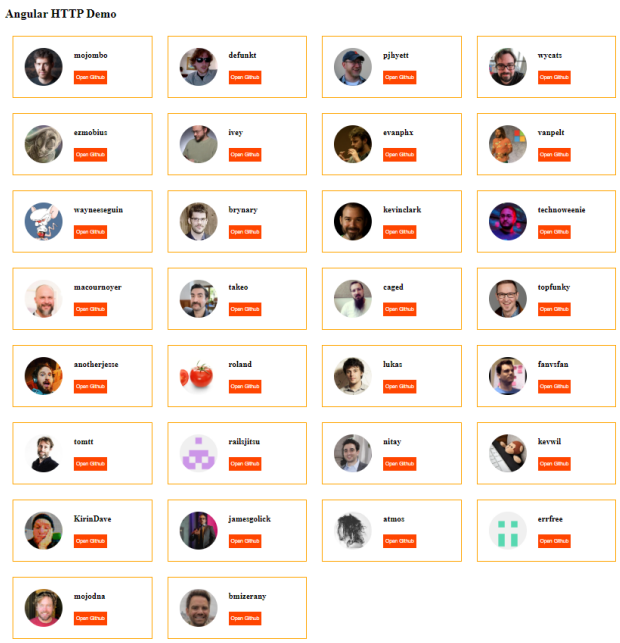


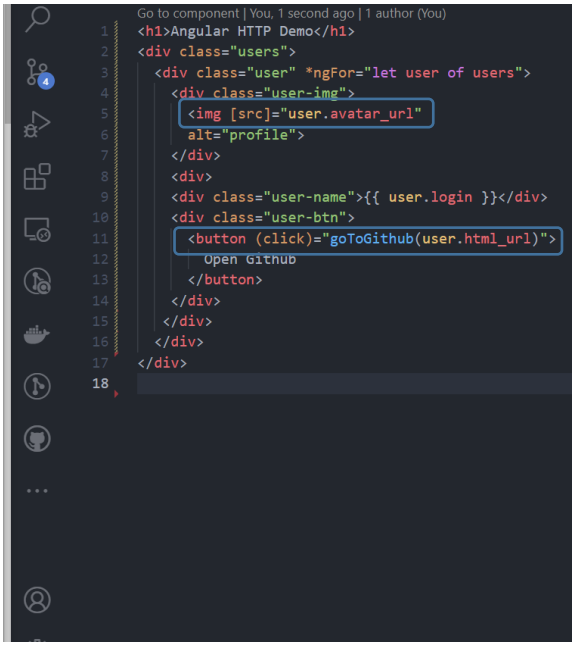
We can use string interpolation to bind it. We can see the names coming now dynamically. For the image, we can use this avatar_url property. And we can use property binding here to bind it to the src property.



We can use see name are coming. For the images we are using avatar_url property and html_url property for the url

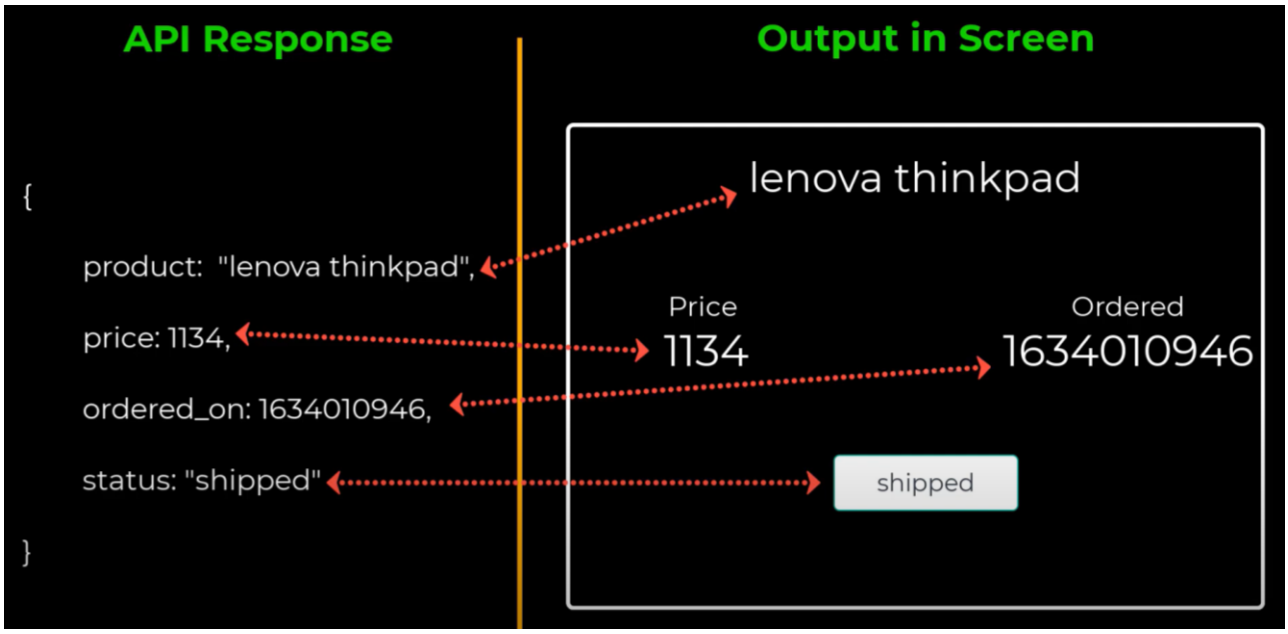
```
▼ Array(30) i
  ▶ 0: {login: 'mojombo', id: 1, node_id: 'MDQ6VXNlcj
  ▶ 1: {login: 'defunkt', id: 2, node_id: 'MDQ6VXNlcj
  ▶ 2: {login: 'pjhyett', id: 3, node_id: 'MDQ6VXNlcj
  ▶ 3: {login: 'wycats', id: 4, node_id: 'MDQ6VXNlcjC
  ▶ 4: {login: 'ezmobius', id: 5, node_id: 'MDQ6VXNlc
  ▼ 5:
      avatar_url: "https://avatars.githubusercontent.
      events_url: "https://api.github.com/users/ivey/
      followers_url: "https://api.github.com/users/iv
      following_url: "https://api.github.com/users/iv
      gists_url: "https://api.github.com/users/ivey/g
      gravatar_id: ""
      html_url: "https://github.com/ivey"
      id: 6
      login: "ivey"
      node_id: "MDQ6VXNlciY="
```

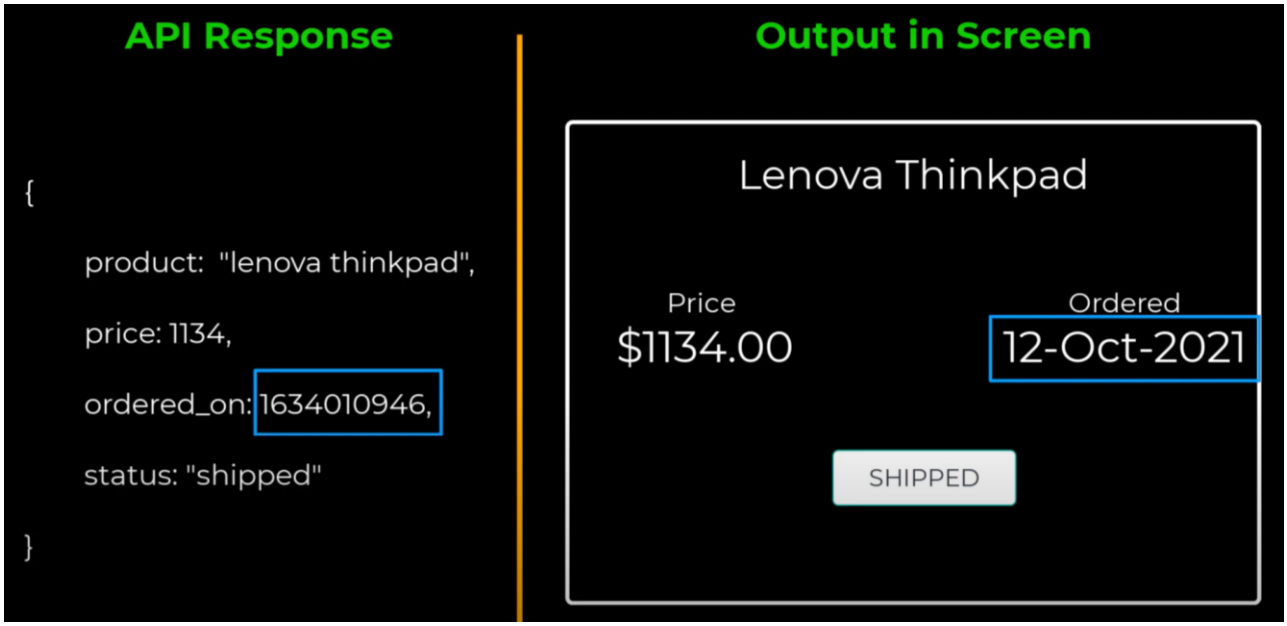


Pipes

We usually gets response from server in JSON format and we display that in the HTML using data binding. Sometimes we may need to transform the data to specific format before displaying it to the end users.

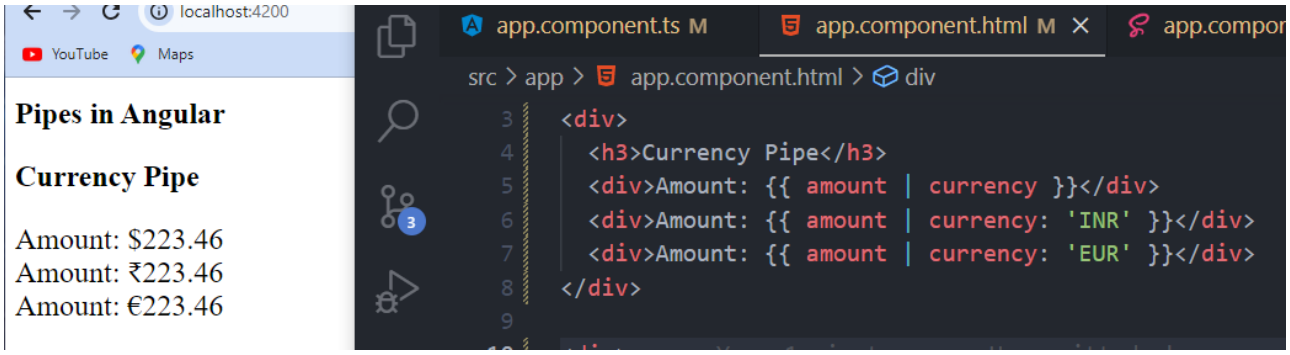


One of the best examples is date format. Date may come from backend in a format like this. So, we may need to show this to user in MM-DD-YYYY format.

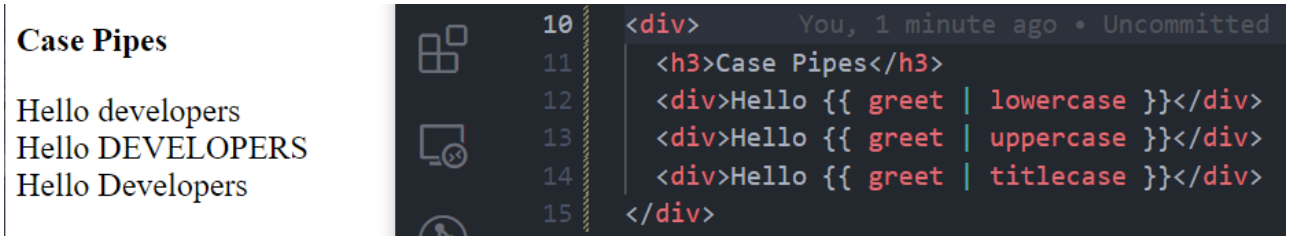


We can use pipes in angular to transform data. Angular provides us several built in pipes. Also, in angular we can create custom pipes based on our requirement.

- **Currency pipes**

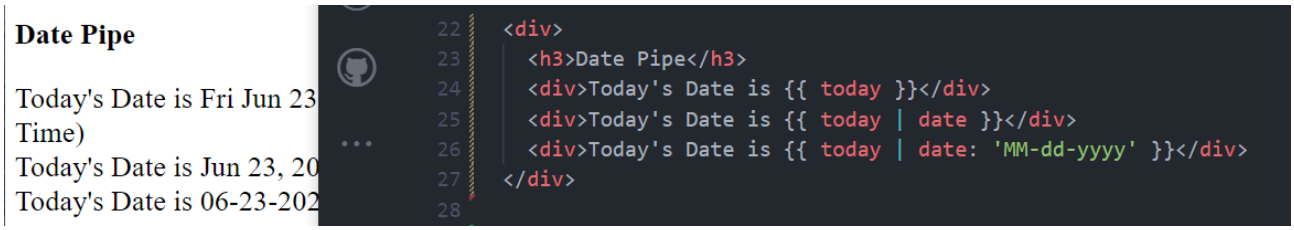


- **Case pipes**

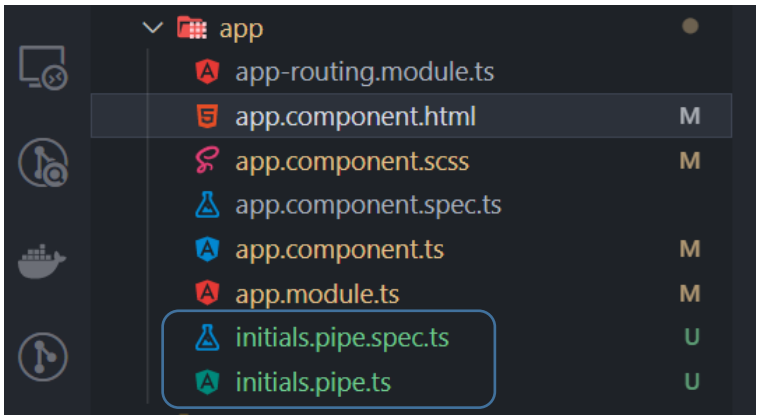


- **Percent pipes**

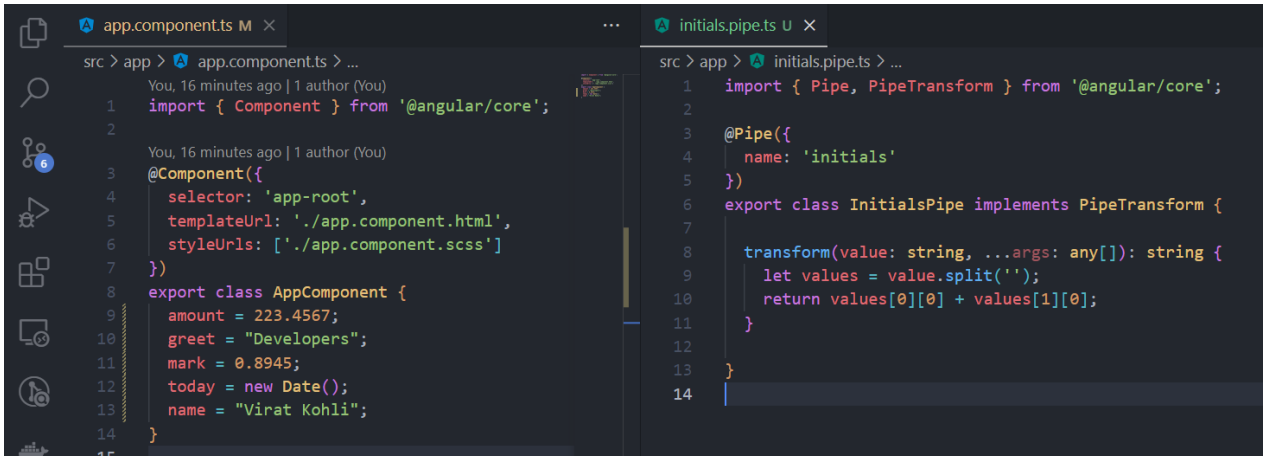
- **Date pipes**



We can make a custom pipe. For creating a custom pipe we need to go to cli and type ng generate pipe pipe-name or ng g p pipe-name.



- **Custom pipes**



Custom Pipe

VK

```
<div>
  <h3>Custom Pipe</h3>
  <div class="initials">{{ name | initials }}</div>
</div>
```