**Angular Notes:**

***Libraries and Youtube videos:***

**Fake Filler Extension**

**Sweet alerts library**

**Ngx Skeleton loader.......**

* npm i ngx-skeleton-loader
* import { NgxSkeletonLoaderModule } from 'ngx-skeleton-loader';

**Toastr alerts library**

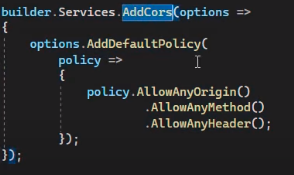
https://youtu.be/nr0DC1FIG4M?si=PDYh3tD7a3jRz0mI

<https://youtu.be/AM1nQgP7b4w?si=yflOFYsQvvOTmwTw>

<https://youtu.be/2Yq3za3qIp0?si=mIF8UJeayx7xHJX8>

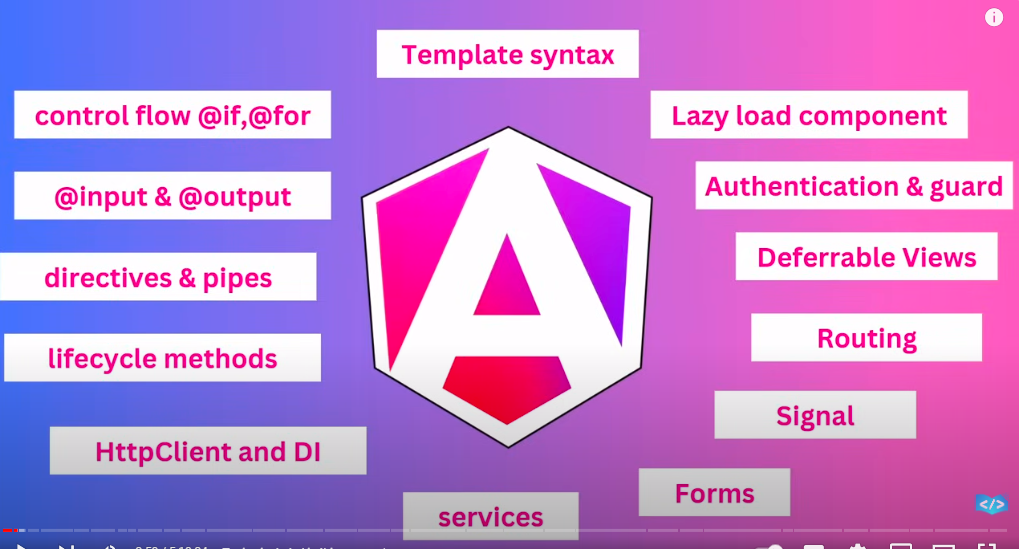
<https://youtu.be/RSH3UM1xpHg?si=boqYTDrFjexUpAPW>

**.Net Cors Policy code.**

****

app.use()

***Topics:***



**Interpolation vs property binding**

Interpolation does not work directly on boolean variables,while property binding work on Boolean variable,so we use property binding in attributes.

**Directives:**

Two types:

* Structural directive

Change the structure of the html code:

\*ngFor,\*ngIf,\*ngSwitch

* Attribute directive

Change the behaviour or appearance of an element

* Components directive

A component directive in Angular is a directive with its own view (template) and encapsulated behavior, defined using the `@Component` decorator.

**Parent to Child data pass:**

When we pass the data from parent to child then we use input decorator and get the result.

**Child to Parent data pass:**

When we pass the data from child to parent then we use output decorator and pass the data to the parent as an event and parent will listen that event.

Quick Analytics

**Pipes:**

To show data in different styling like uppercases and lowercases.

**Custom Pipe:**

ng g p abc

**Custom directive:**

ng g d directive

**Dependency injection:**

 constructor(private el:ElementRef) {

  }

Dependency Injection is a design pattern where an object's dependencies are provided externally rather than the object creating them itself.

**LifeCycle methods:**

Methods that are angular itself called.

Constructor()

ngOnChanges()

ngOnInit()

ngOnDestroy()

**viewChild():**Get the html template

**Services and http call in angular:**

Ng g s service

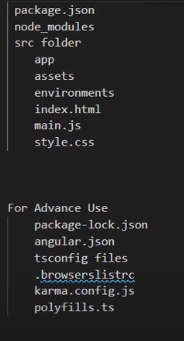
**Maintaining global state with services like shared data which access in every component:**

**Models in angular:**

[**https://dummyjson.com/products/1**](https://dummyjson.com/products/1)

***Angular concepts revision:***

1. **Files and Folder Structure:**



**Package.json:**All project detail

**Node Modules:**All libraries when we use npm it rebuild so we should not make any file here.

**Package\_lock.json:** More detail about the packages that the package needs which new packages.

**Angular.json:**Which file load first in angular,Configuration file.

**Tsconfig.json:**Global configuration

**Tsconfig.app.json:**Application configuration

**Tsconfig.spec.json:**Testing configuration

**Interpolation:**

* Show dynamic data on html
* Perform arithmetic operation in it
* Use ternary operator
* Cannot assign value
* Cannot check typeOf
* Cannot use new keyword
* Cannot increment and decrement operator
* We can assign class dynamically by it.

**Angular Commands:**

* Ng help ->Give all the basic commands
* ng generate **component** [*name*]
* ng generate **class** [*name*]
* ng generate **interface** [*name*] [*type*]
* ng generate **pipe** [*name*]
* ng generate **service** [*name*]

**Angular Components:**

* Building block of the application
* Can change the name of the component selector

**Angular Components with inline style:**

* ng generate **component** user-list –inline-style

No css file

Use where there is style is very less.

* ng generate **component** user-list –inline-template

No html file

Use where there is style is very less.

* ng generate **component** user-list –inline-style –inline-template

no html file and no css file

**Diff Bet Angular 14 and Angular 17:**

In angular 17, components are standalone:true.

Standalone components don't need to be declared within an @NgModule class.

his simplifies development and keeps your code cleaner. Think of it like standalone components being self-contained units, ready to use without extra boxes (modules).

**Events:**

* KeyUp
* keyDown
* mouseover
* input
* mouseleave
* blur

<button (click)=”GetInput(this.value)”>Click Me</button>

**Property Binding:**

**Interpolation vs property binding**

Interpolation does not work directly on boolean variables,while property binding work on Boolean variable,so we use property binding in attributes.

**Style Binding:**

Here we change the style of the html element dynamically.

**Forms in angular:**

There are two types of forms in angular:

* Template driven form(Validations)
* Reactive form

**Template driven form:**

Most work done in form component template.

**Code:**

<p>forms works!</p>

<form #formId="ngForm" (ngSubmit)="getValue(formId.value)">

  <input type="text" [ngModel] name="name" />

  <br /><br />

  <input type="password" [ngModel] name="password" />

  <br /><br />

  <button>Login</button>

</form>

<button (click)="getValue(formId.value)">Login</button>

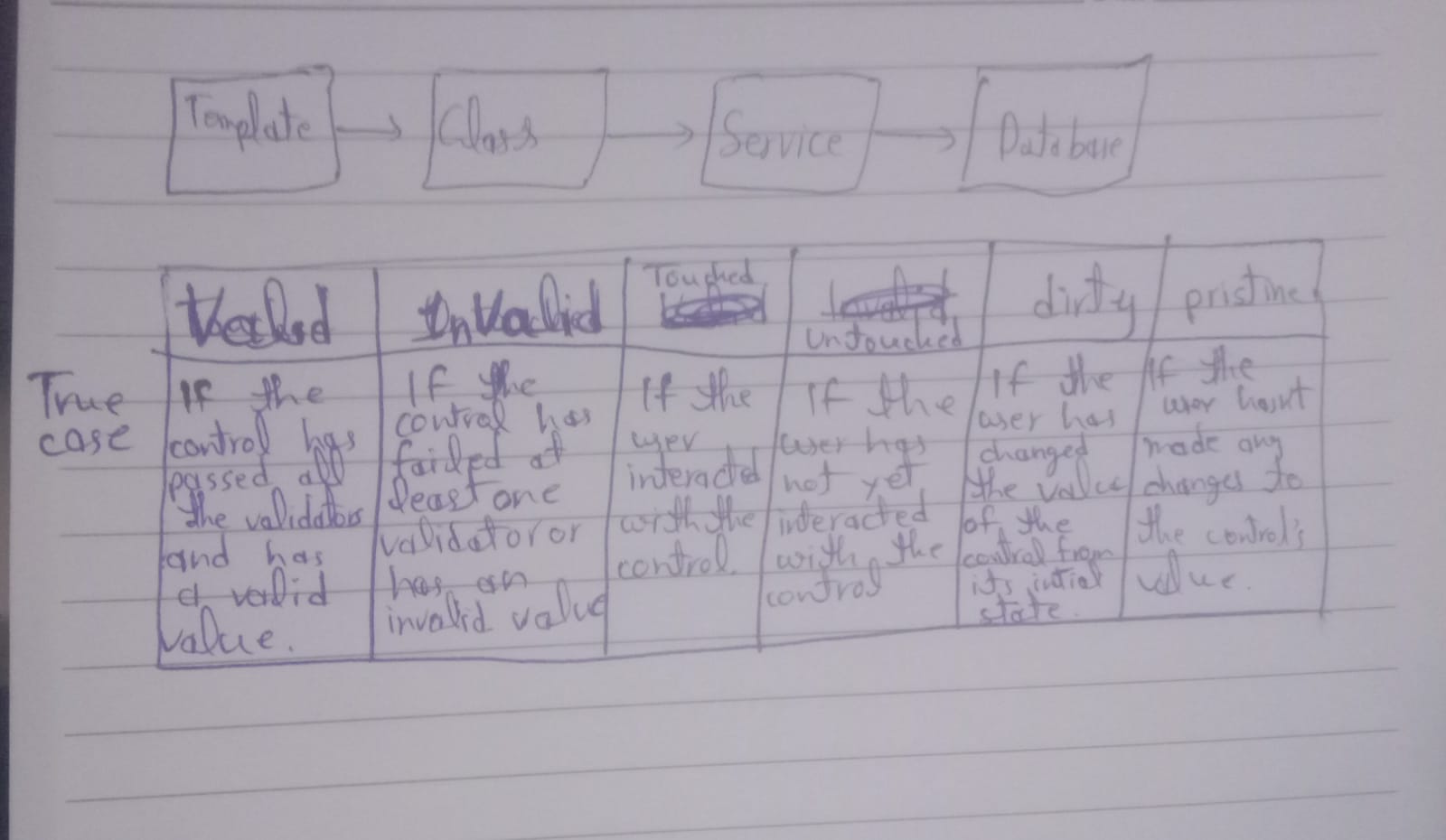
* If we don’t use name in input then it will not get value.

**ngModel:**

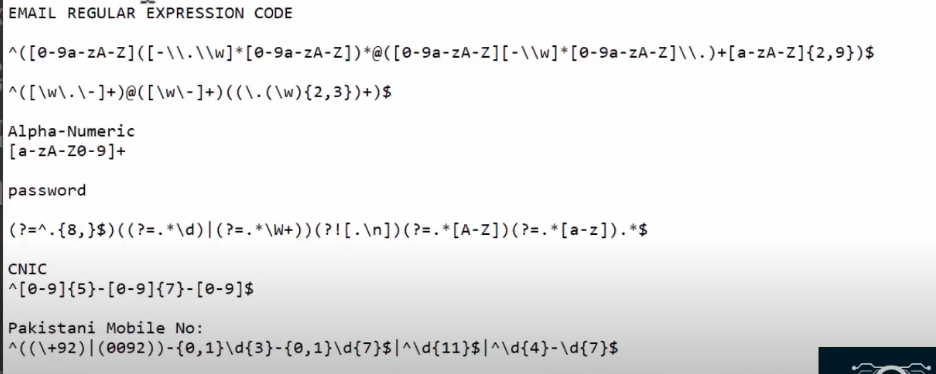
In Angular, ngModel is a directive that creates a two-way data binding between an input field and a property in your component class. Simply put, it keeps the displayed value in the input field and the corresponding property in your component in sync.

**Template driven form validations:**

* Validations in template driven form are provided by the template directives.
* Form Module has built in validators.
* We can create our custom validators.
* Use novalidate attribute in form element to disable the browser validation.
* Implement validations in forms using properties touched,untouched,valid,invalid,dirty,pristine. These are the Boolean properties, store true or false in it.These are called state of the form control.



* We use different classes like is-invalid it make textbox border red in invalid case.
* Some Regex



**Template reference variable:**

In Angular, a template reference variable acts like a nickname for an element, component, or directive in your template. This nickname lets you easily refer to that part of your template in your component's code.

**NgModule:**

An NgModule in Angular is a fundamental building block that organizes components, directives, pipes, and services into cohesive units for your application.

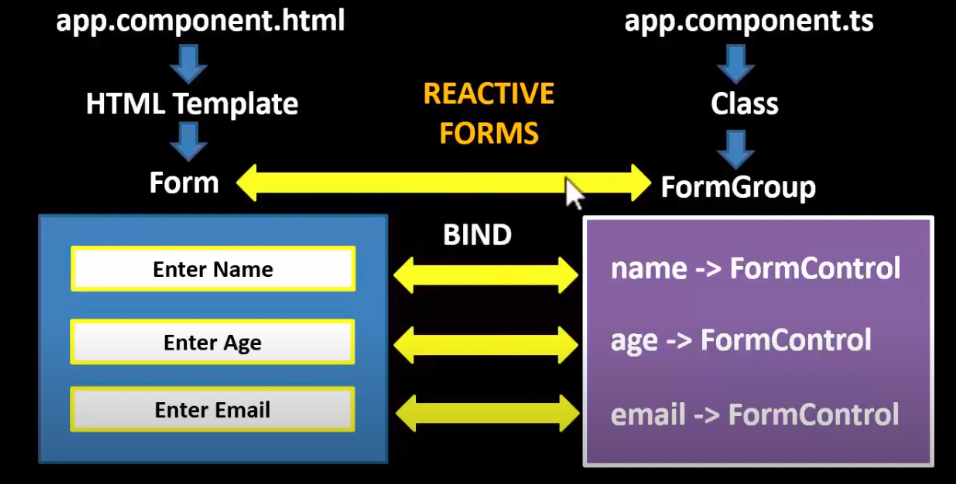
**Reactive form:(Model Driven Form)**

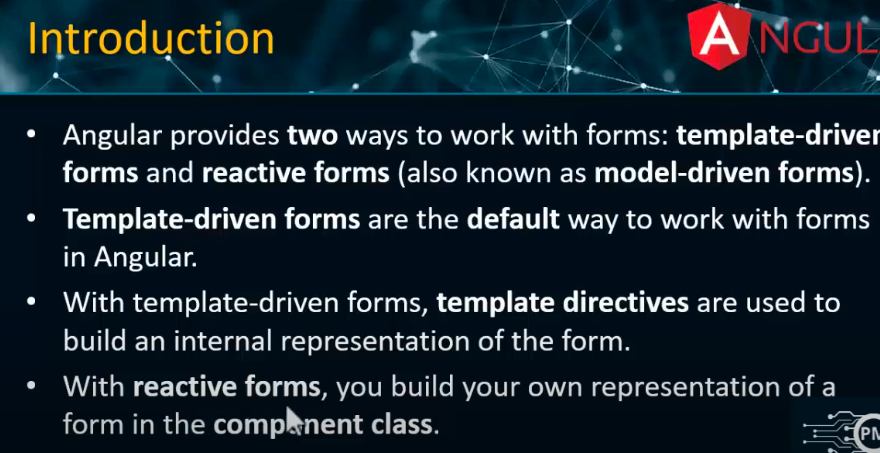
Most work done in form component class.

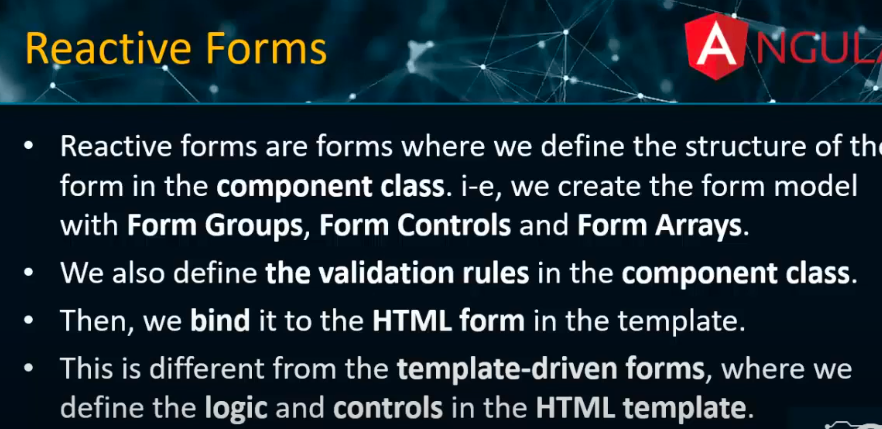
[disabled]=”loginForm.disabled”

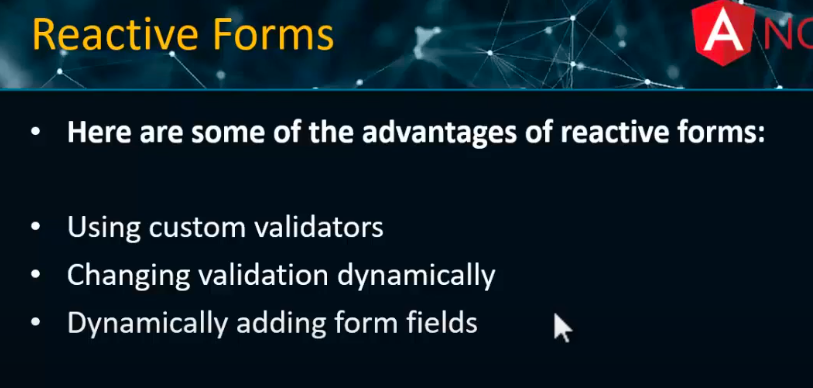
A screenshot of a computer

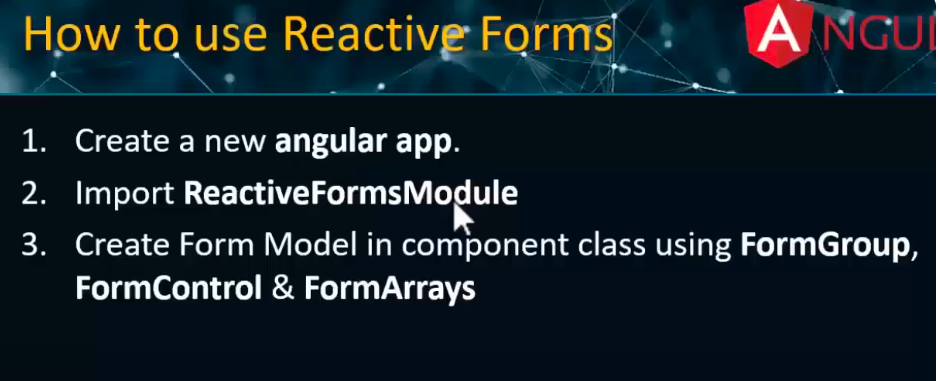
Description automatically generated











**Reactive form validations:**

 newFormData=new FormGroup({

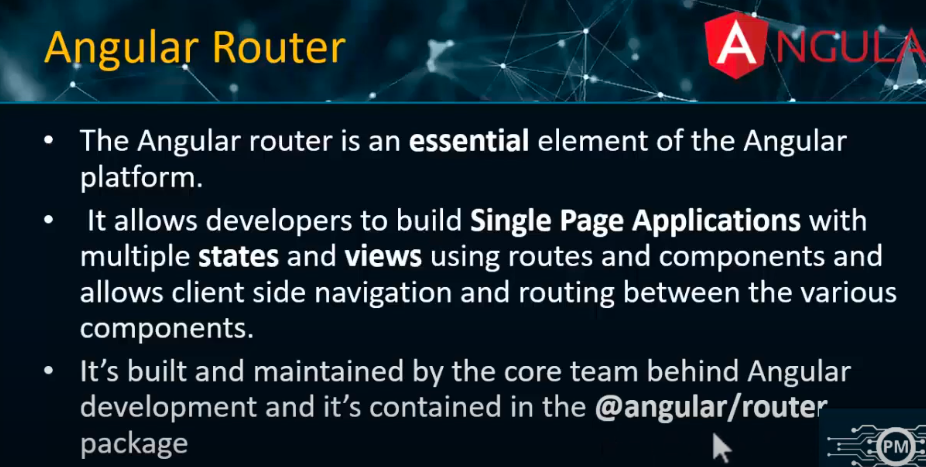
      name:new FormControl('',[Validators.required, Validators.minLength(3),Validators.maxLength(15)]),

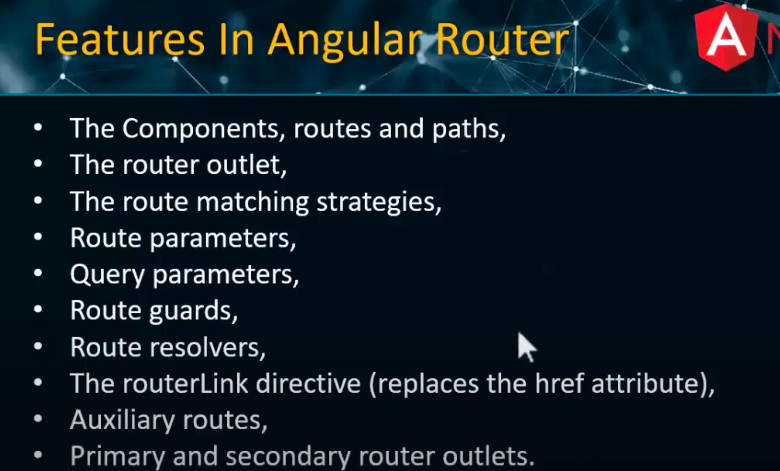
      age:new FormControl('',[Validators.required,Validators.min(10)]),

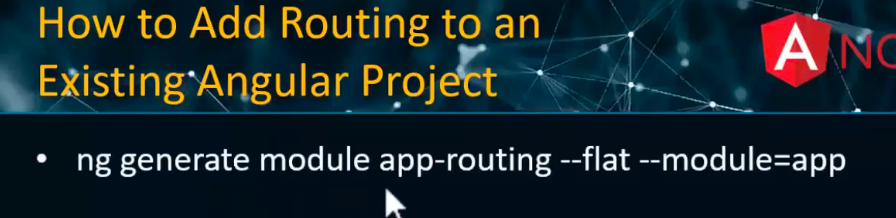
      email:new FormControl('',[Validators.required])

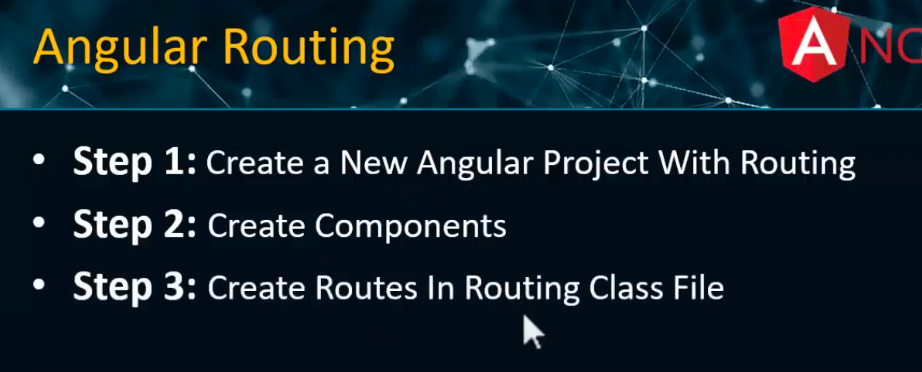
    })

**Routing in angular:**

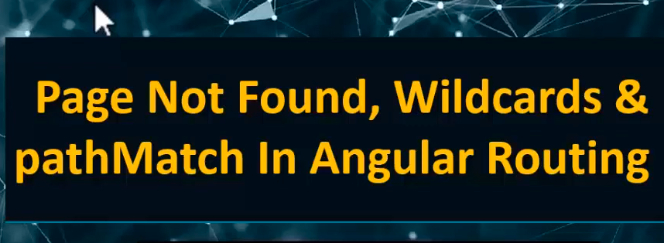








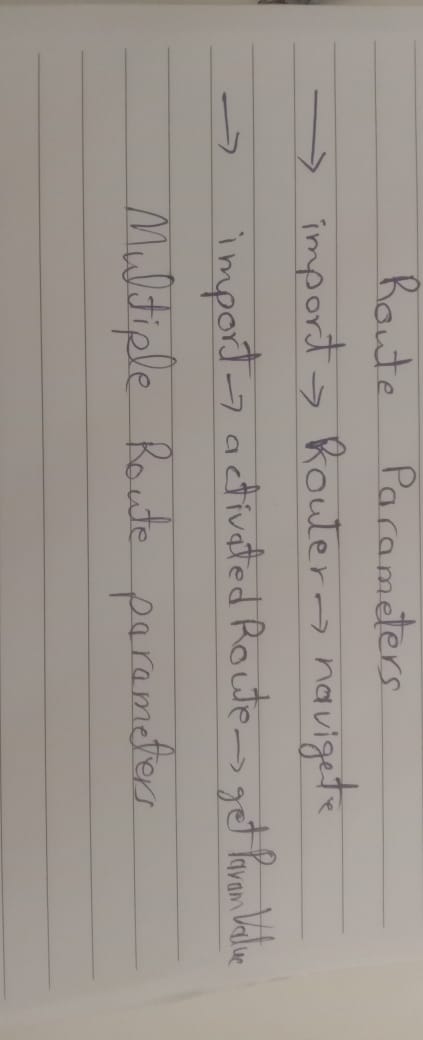




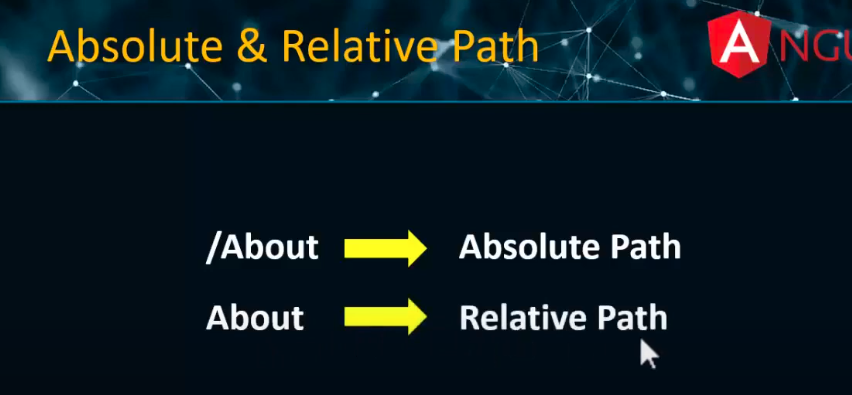
pathMatch=”full” means if all the path is empty.

pathMatch=”prefix” means if start is empty.

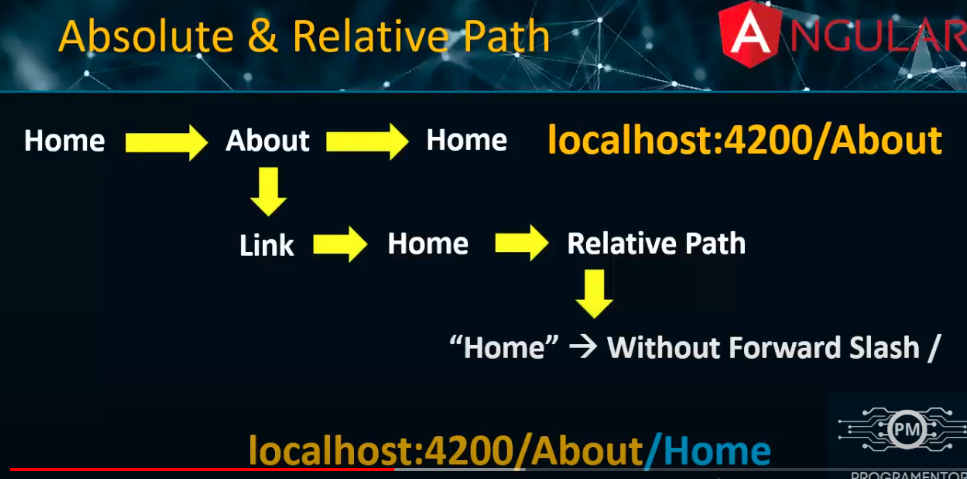
**Router parameters:**

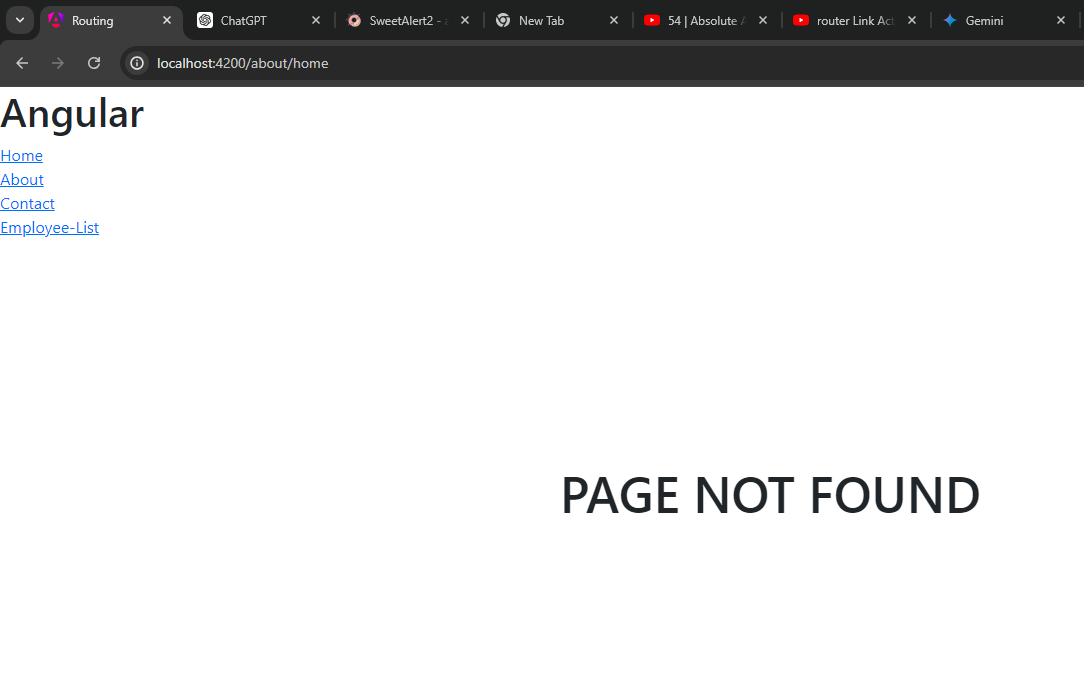
****

**Relative path and absolute path:**



**Problem in relative path**





**Three types of parameters:**

* **Required parameters**
* **Query parameters**
* **Optional parameters**

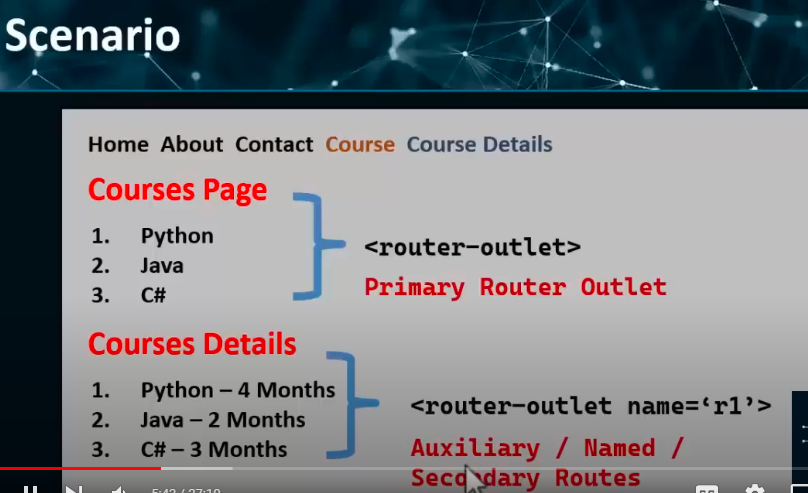
**Optional route parameters:**

**ParamMap observable**

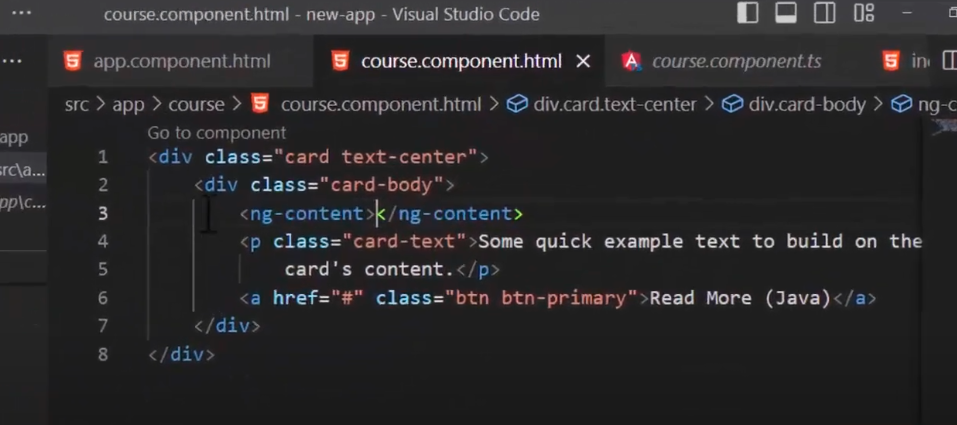
**Child Routes**

**Query Parameter**

**Auxiliary router|Multiple router outlet**







**Deploy your angular website on github:**



**­­­­**

**Lazy Loading:**

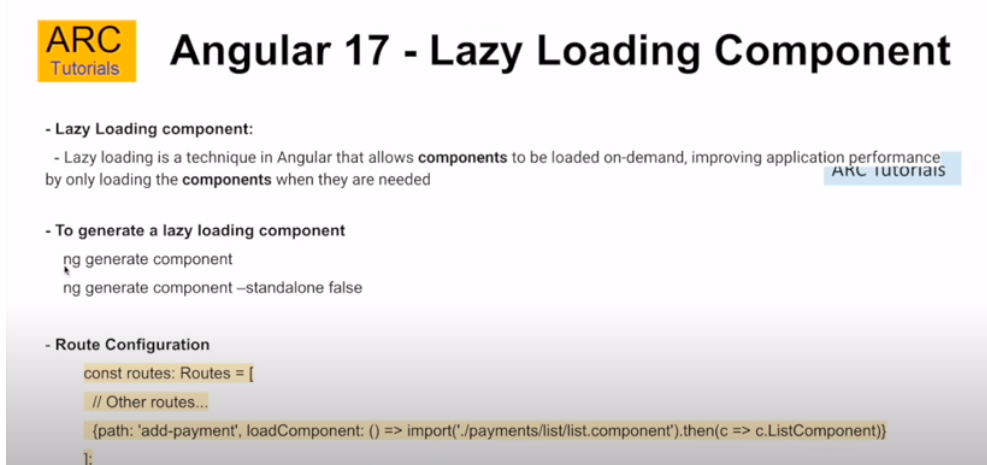
Load component A when we load the component B.

* Load the lazy loading module: ng g m <module> routing
* Inside lazy loading module we can have

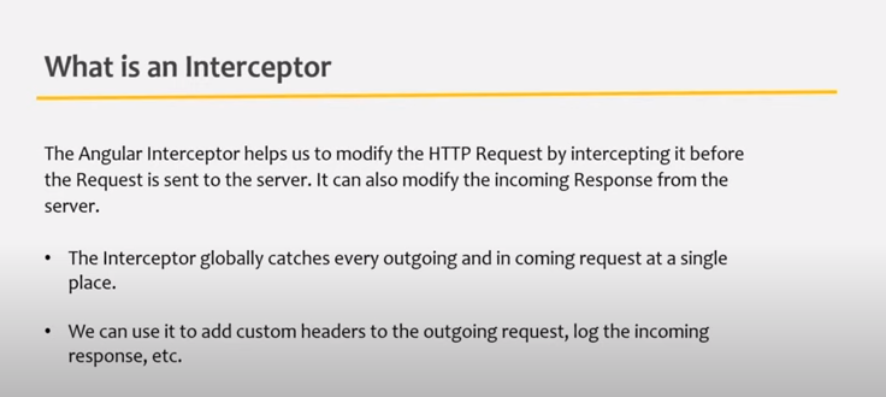
->standalone components

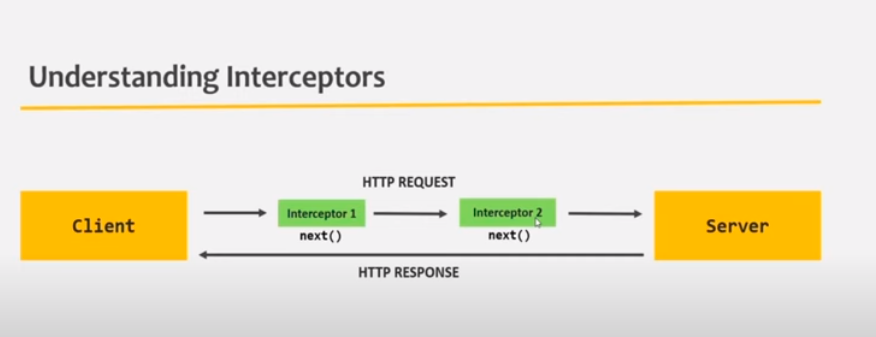
->regular components

* Load the lazy loading component: ng g m <module> routing



**Lazy Loading:**





**Dialog in angular:**

ng add @angular/material

***Apollo client for angular:***

**Packages install:**

npm install apollo-angular apollo-angular-link-http apollo-client apollo-cache-inmemory graphql-tag graphql

ng add apollo-angular

<https://youtu.be/ljlGauovWXk?si=ZoYqgtEgd-DxcaQz>

**Problems with query,watchQuery system:**

 **Outdated Typing**: If your GraphQL schema changes but your TypeScript types don't, the types become incorrect or outdated.

 **Typos**: Manually writing types increases the chance of making spelling mistakes or errors.

 **Partial Typing**: If you don't type every field in your schema, some fields might be missing in your TypeScript types, leading to incomplete or incorrect data handling.

**npx code generator:**

**Material UI MatDialogBox usage:**ng add @angular/material

import {MatToolbarModule} from '@angular/material/toolbar';

it is important to close the server one time when we start to use the matdialogBox

**viewChild in angular:**

manipulate DOM elements

**DetectChanges:**

Cdr->detect changes

`ChangeDetectionRef` is an Angular service that allows you to programmatically control and trigger change detection in your application.

**Injectable:**

@Injectable({ providedIn: 'root' }): Declares that the GetCategoryGQL service can be injected into other components or services at the root level of the application, making it available throughout the entire application.

**Angular popup another method:**

https://www.npmjs.com/package/ng-angular-popup

* npm i ng-angular-popup
* import module on the website
* add in html file <ng-toast></ng-toast>
* inject toast:ngToastService in the constructor

**Interview Questions:**

<https://youtu.be/-jeoyDJDsSM?si=oxBKg-gL4Hlth-Sh>

**Cors Policy:**

The CORS (Cross-Origin Resource Sharing) policy is a security feature implemented by browsers that controls which resources a web application running at one origin (domain) can request from another origin.

**Google Login in angular:**

Website aur app aur desktop

**ngClass:**