**Angular** –why Angular?

it makes it very easy for you to interact with your DOM, to change something there at runtime.

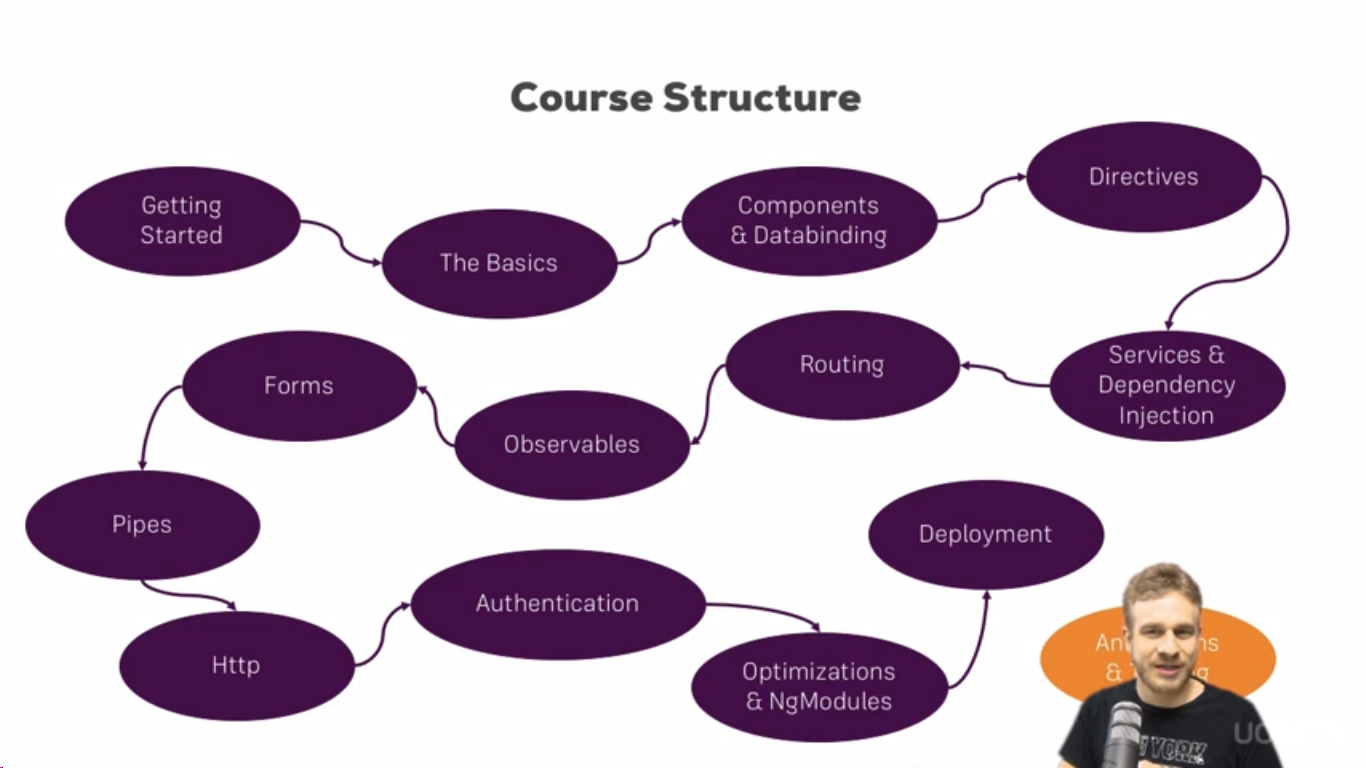
**Installation:--**

1. Window Node.js
2. npm install -g @angular/cli
3. ng new my-app
4. Ng Serve
5. Ng New
6. Team - <https://discord.com/channels/622033978047725582/622034266506657792>
7. Unzip Github file – then run npm install, ng serve .
8. Emmet Link-- <https://docs.emmet.io/cheat-sheet/>
9. Path- node\_modules/bootstrap/dist/css/bootstrap.min.css
10. Command to run - npm install bootstrap --save

**CLI:**

Ng generate component ComponentName --spec false

1. [(ngModel)] =”name” - Import - import {FormsModule} from '@angular/forms'
2. Basic Angular 🡪 Angular Component Binding 🡪 Directive 🡪 Dependency Injection 🡪Routing 🡪Pipes🡪Https



1. What’s is the Types Script: Super set of JavaScript 🡪 which support oops concept.
2. Guide to Unzip the git hub code File and run locallay.

<https://www.udemy.com/course/the-complete-guide-to-angular-2/learn/lecture/6709112#questions/8079942>.

12. **Angular Flow**-

So Angular gets started, this **main.ts file** gets started, there we bootstrap angular

application and we pass this **app** module as an argument. In this module, we tell Angular

hey there is this **app component** which you know when you try to start yourself and Angular now analyzes the **app component**, reads the set up we pass here and therefore knows this selector, **app-root** and now Angular is able to handle **app-root** in the **index.html** file and it knows all right this is the selector I know, you told me that I should know it because it was listed in this.

13. **Angular 8 Components**

**Component** defines a class that contains **application data** and **logic**, and is associated with an HTML template which helps to build the web pages.

1. A Decorator is a special kind of declaration that can be attached to a [class declaration](https://www.typescriptlang.org/docs/handbook/decorators.html#class-decorators), [**method**](https://www.typescriptlang.org/docs/handbook/decorators.html#method-decorators), [**accessor**](https://www.typescriptlang.org/docs/handbook/decorators.html#accessor-decorators), [**property**](https://www.typescriptlang.org/docs/handbook/decorators.html#property-decorators), or [**parameter**](https://www.typescriptlang.org/docs/handbook/decorators.html#parameter-decorators). Decorators use the form @expression, where expression must evaluate to a function that will be called at runtime with information about the decorated declaration..

In Simple words: decorator is a JavaScript function expression which execute the object which pass under decorator at runtime.

In angular, we have to Import decorator using - **@Component**. @component decorator is known to TypeScript, so when it parses this file and compiles it to JavaScript.

1. an app module? Angular uses components to build web pages and uses modules to basically bundle different pieces,
2. String Interpolation – {{name}}, name:string=”Ravi”.
3. Property Binding : [(ngModel)]=””—Two way binding.
4. \*ngIf="IfStatement; else noServer

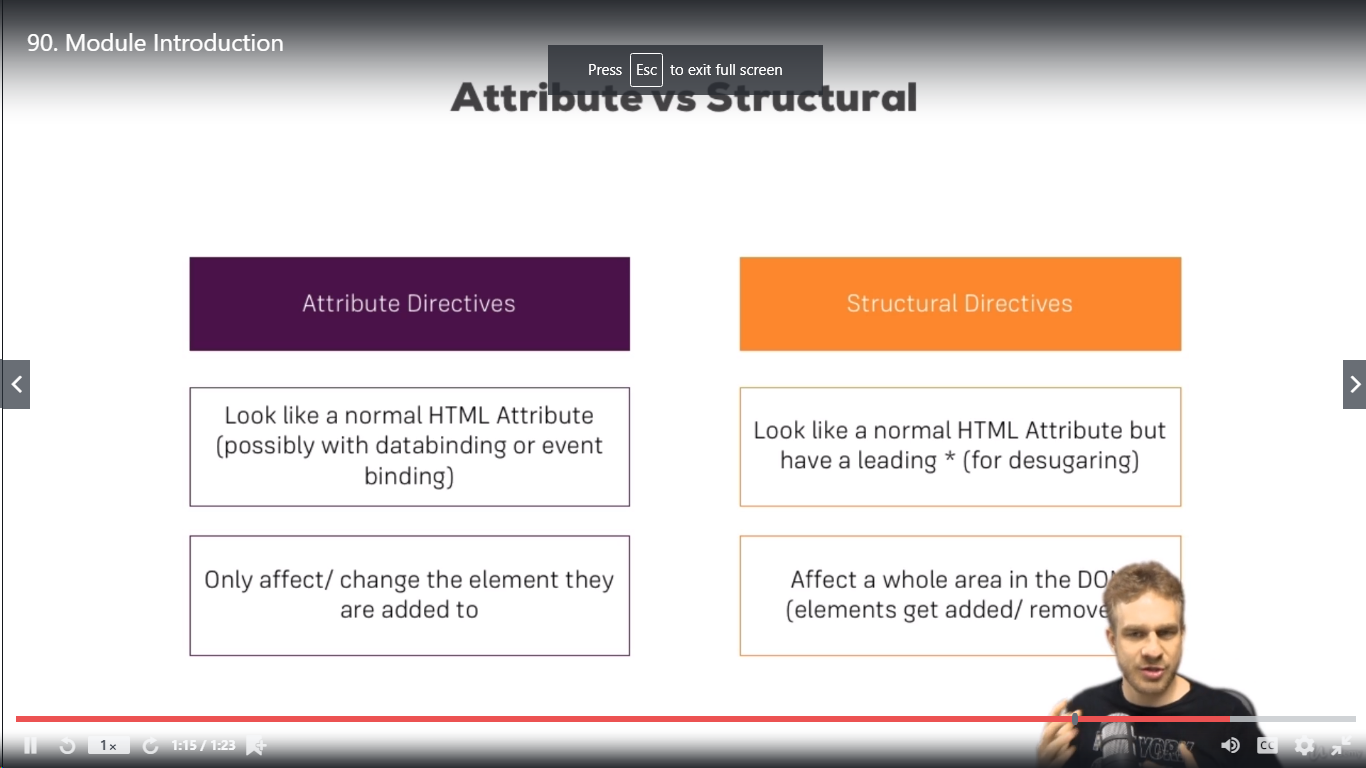
**18,** this.serverStatus1=Math.random()> 0.5 ?'online':'offline';

**19.**

**Directive:**

**What’s: Directive are instruction in the DOM**, components are kind of such instructions in the DOM. Once we place the selector of our component somewhere in our templates,

at this point of time we're **instructing Angular** to add the content of our component template and the business logic in our TypeScript code in this place where we use the selector.

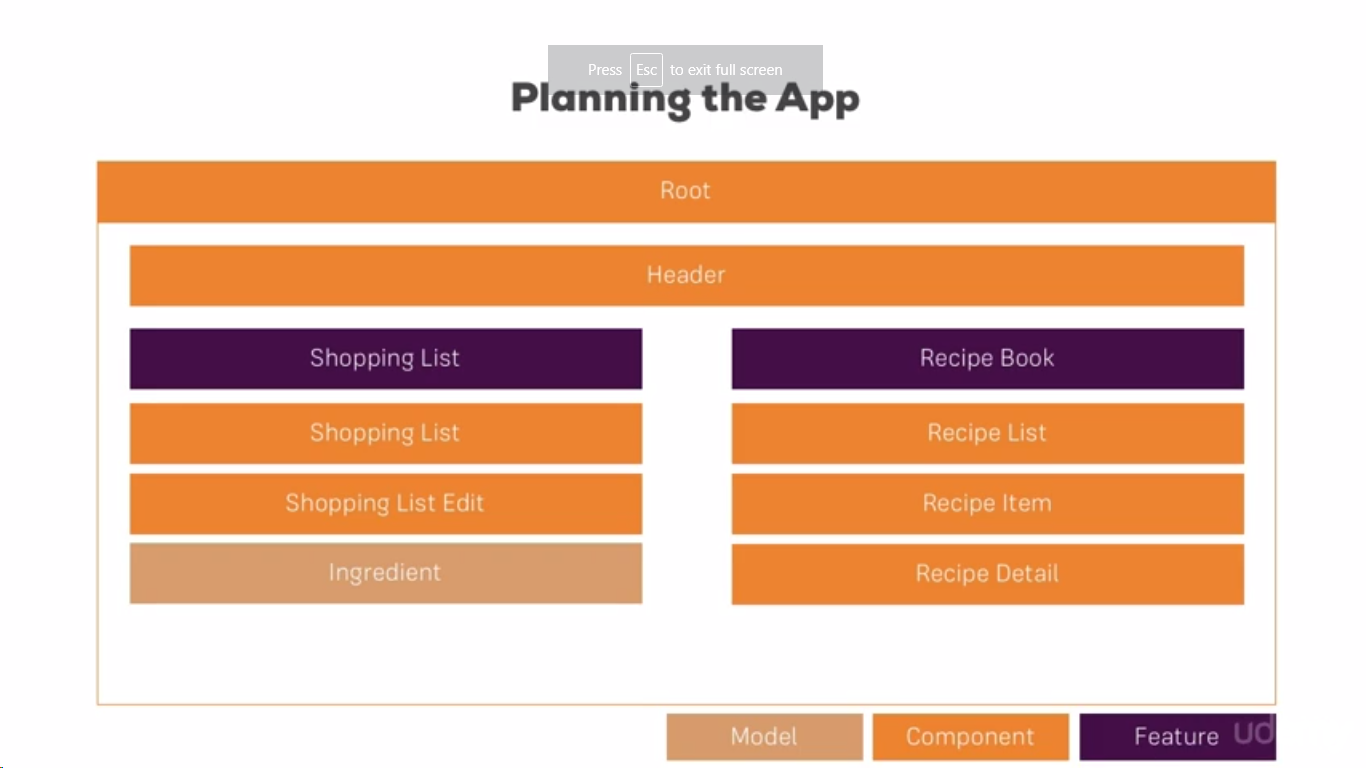


This was our instruction, Angular111111111

**How :** We typically add directives with an attributes selector.

**Where** :

Project Structure:



How to debug Code::

**Debug code using** -

1.Debug code in the browser using Source Map.---deep me Jana Hai to- web pack ke Andar jao.

2.Angular Augury

**View Encapsulation:**

encapsulation:ViewEncapsulation.emulated -- // default meant , CSS restrict component specific.

//none -- Globally can be shared.

//Native

**Local reference: which just** hold the reference of element.

#localReferenceVariableName.

This can be used inside only template.

**ViewChild:** This is ElementRef type which allow us to pass value from element to TS.

Any directive, component, and element which **is** part **of** component template **is** accessed as **ViewChild**. Whereas, any element or component which **is** projected inside <ng-content> **is** accessed as ContentChild

**#NativeElement.**

@ViewChild('serverContentInput',{static:false}) serverContendInput:ElementRef;

this.serverContendInput.nativeElement.value

**Ng Content:**

**ng**-**content** is used to project content into Angular components.

Used Inside – App=element-server only --<ng-content></ng-content>

 <app-element-server

                       \*ngFor="let serverElement of serverElements"

                        [element]="serverElement">

                        <p>

                          <strong

                          \*ngIf="serverElement.type === 'server'"

                                    style="color: red">

                                    {{ serverElement.content }}

                          </strong>

                          <em

                          \*ngIf="serverElement.type === 'blueprint'">

                          {{ serverElement.content }}</em>

                        </p>

                    </app-element-server>

**Angular Life Cycle:**

So once a new component is instantiated, Angular goes through a couple of different phases in this creation.

1. **ngOnChanges:**

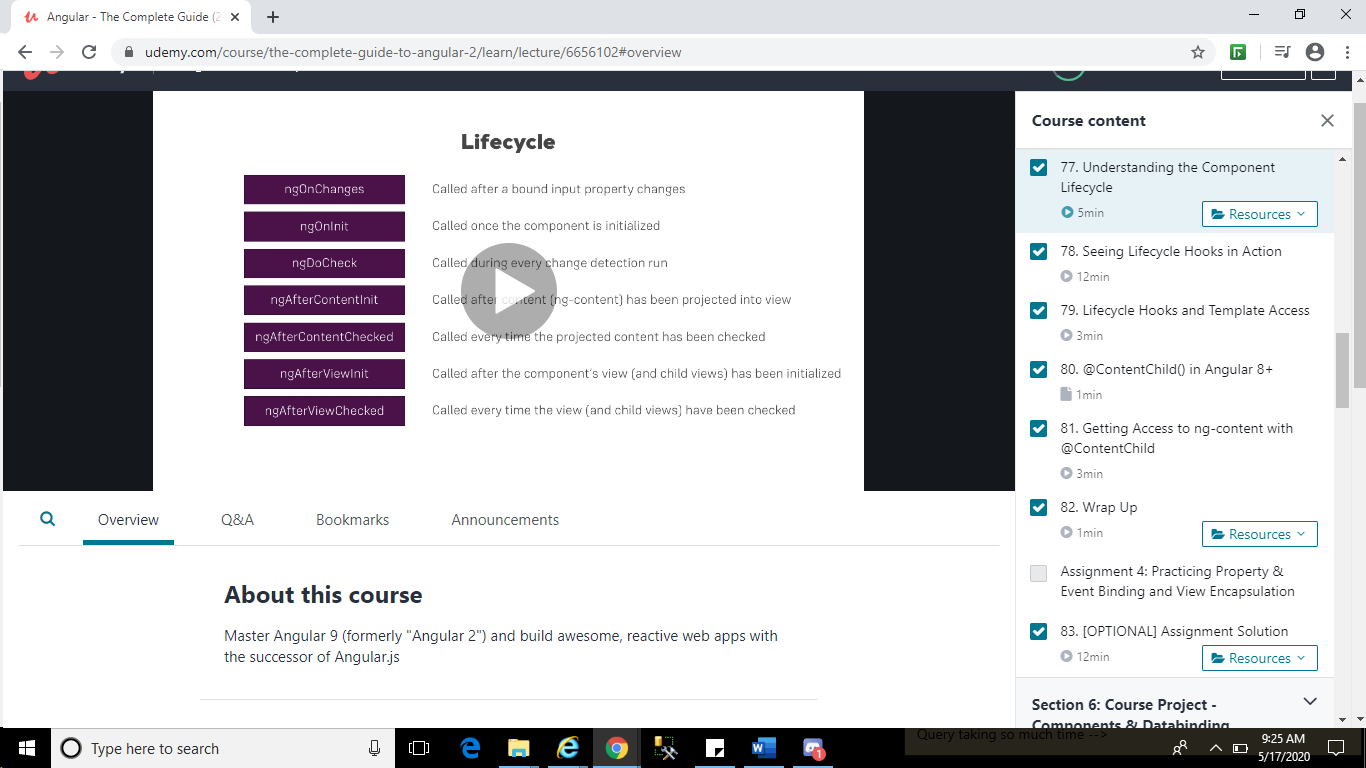
The first phase, the first hook we can hook into is ngOnChanges and this may actually be executed multiple times, it's executed right at the start when a new component is created but thereafter, it's

also always called whenever one of our bound input properties changes.

and with that, I mean properties decorated with @inpu…

so whenever these properties received new values.

1. **ngOnInit:**



**Ng Template:**

**ng**-**template** is an **Angular** element used to render HTML **templates**. We use **ng**-**template** with **angular** \*ngIf directive to display else **template**. ... If you see the output it will display only **ng**-**template** works which is in div element. And have a look at the generated HTML source code

@**HostBinding** and @**HostListener** are two decorators in Angular that can be really useful in custom directives. @**HostBinding** lets you set properties on the element or component that hosts the directive, and @**HostListener** lets you listen for events on the host element or component

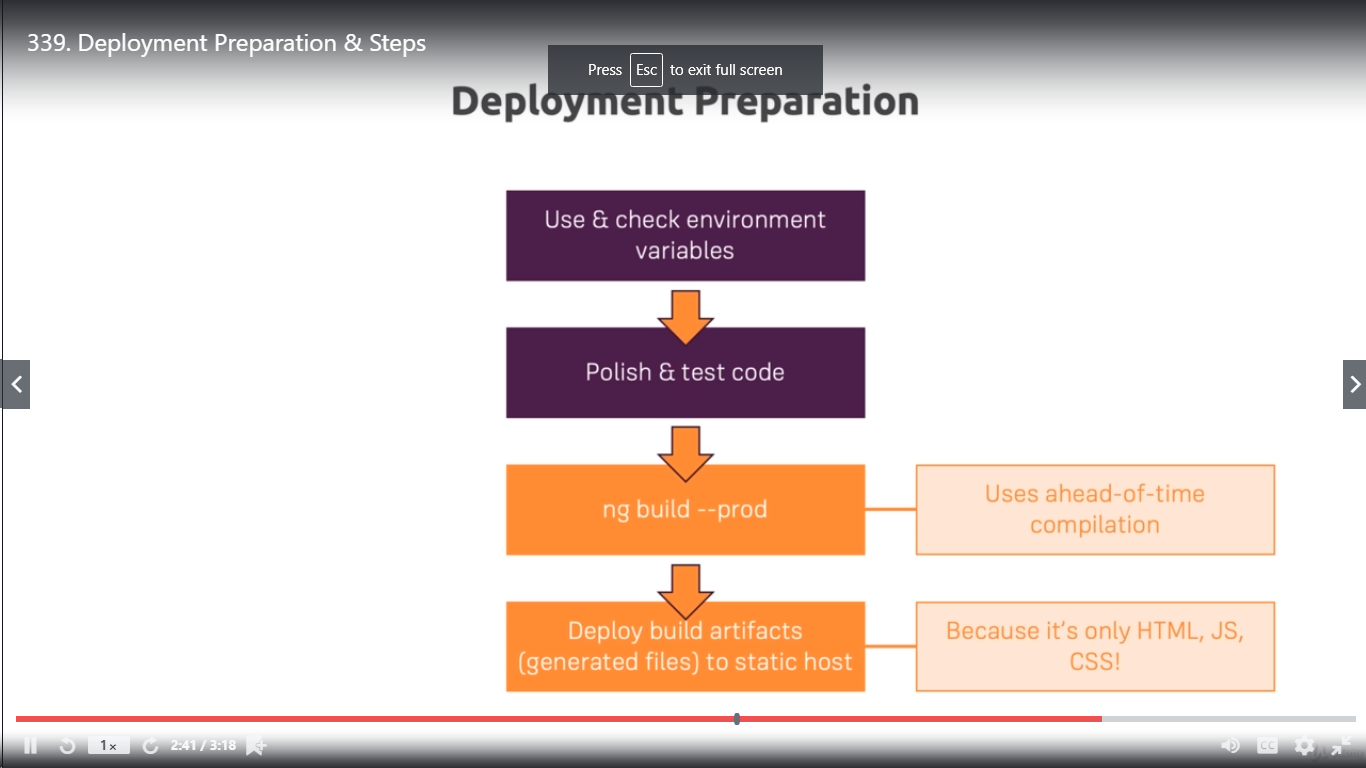
* Official Docs: <https://rxjs-dev.firebaseapp.com/>
* <https://www.learnrxjs.io/learn-rxjs/operators>
* RxJS Series: <https://academind.com/learn/javascript/understanding-rxjs/>
* Updating to RxJS 6: <https://academind.com/learn/javascript/rxjs-6-what-changed/>

--More about Pipes : Angular.io 🡪Doc 🡪 Additional Documentation.

* Firebase Auth REST API Docs: <https://firebase.google.com/docs/reference/rest/auth>
* More on JWT: [https://jwt.io](https://jwt.io/)

Official Docs: <https://angular.io/guide/dynamic-component-loader>

* Official Docs: <https://angular.io/guide/ngmodules>
* NgModules FAQ: <https://angular.io/guide/ngmodule-faq>



Firebase – installation- steps-

* 1. npm install -g firebase-tools
  2. another command type : firebase login

or - firebase login:ci

3.firebase init

Overwrite pe no select karna hai.

4.firebase Deploy.