sessions from 7th dec - 10th dec

~~~~~~

-> angular:

- an angular is a platform and javascript framework

- for building the single page applications using Html, Css, Bootstrap and Typescript

- it implements core and optional functionality as a a set of typescript libraries that you import into your applications

-> the architecture of an angular relies on certain fundamental concepts:

terms:

1. Template
2. Directive - (meta data)
3. Component
4. Injector
5. Module

-> component:

- the components are the basic ui building blocks of an angular application.

- angular application contains a tree structure of angular components.

- angular components are a subset of directives always associated with a template.

- unlike other directives, only one ie., component can be instantiated for a given element in a template.

- types of components: 1. parent and 2. child

-> single page applications:

- an angular is a javascript frontend web framework based on bi-directional ui data binding

- is used to design single page applications
~~~~~~

- the applications that load a single Html page and only a part of the page will be updated instead of the entire page - based on the user actions (click, hover, drag, etc.)

-> who are implementing the single page applications:

- gmail, google maps, airbnb, netflix etc.,

-> when angular can use?

- it's called the safe navigation operator which can be used to prevent angular from throwing errors, when trying to access object properties of an object that doesn't exist

-> setup an env., and creating the project:

- install node and npm - 14v
- npm install -g @angular/cli
- ng --version

to create a new project

- ng new yourappname
    1. routing (y)
    2. style - css (select) / stylus / less / scss
- ng serve (or) npm starts

-> to create a responsive:

- bootstrap: npm install --save bootstrap
- in angular.json file: "../node_modules/bootstrap/dist/css/bootstrap.min.css"

-> creating the new custom component

- adding styles to it
- declaring the property(s) in side the class defin., of component
- accessing or interpolation or one way property binding - {{ }}

- to set the component as the landing page
    1. the name of the component should be declared and needs to specify under bootstrap: [] - app.module.ts
    2. in index.html file - needs to specify the given selector of that component

(or)

- creating new component with command
    - ng generate component <compname>

-> property binding:

- displaying the value of property in html
- there are two types:
    1. one way property binding - {{propertyName}}
    2. two way property binding - user can input the values
        1. template driven
        2. reactive form

-> template driven

- ngForm
- ngModel

- (event) = "methodName(parameters)"
- custome method definition

sessions from 13th dec - 17th dec

~~~~~~

-> custom (or) template driven form validation

- modules: FormsModule

- ngForm
- ngModel

- [(ngModel)]=""
- #idName="ngModel"

- submitted, invalid
- errors, errors.invalid
- errors.email
- errors.minlength

- [ngClass]="{'is-invalid': ...}"

-> reactive form validation

- modules: ReactiveFormsModule,
- component level modules: FormBuilder, FormGroup, Validators

- [ngClass]
- *ngIf
- *ngFor

-> let's discuss the few differences between template driven and reactive

- the Template Driven Forms are Asynchronous (Non - Blocking)
~~~~~~

- whereas, the Reactive Forms are Synchronous (Blocking)

- in Template Driven most of the 'logic' is driven from the Template
- in Reactive, the logic resides mainly in the component or typescript code

- the Template Driven approach would be easier or familiar for those who coming from AngularJS v1
- the Reactive approach removes the core validation logic from the Template and hence make the Template code quite clean

-> Creating the Routing in Angular:

- Enabling Routing and Navigation Service
- Routing allows users to Navigate between one page (component) to another page (component) based on the action taken by user,

- static: routerLink = "/home";
- in app-routing.module.ts

  { path: "/home", component: HomeComponent }

- dynamic: [routerLink] = "/page/:variable"

-> Directives in Angular:

- Directives are DOM elements to interact with your applications.
- Directive is a TypeScript Function.
- When this function executes the Angular Compiler checked it inside the DOM element.

- DOM?
  - DOM is used to define a standard for accessing documents.
  - Html DOM model is constructed as a tree of objects.
  - It is a standard object model to access html elements

- we can use DOM model in Angular
    - Navigate the document structure with DOM elements
    - Add Html elements
    - Update elements

- Angular directive begins with ng stands for Angular and extends Html tags with @directive decorator
- Directives enables logic to be included in the Angular Templates.

- Angular directives are categorized into 3
    1. Component Directives
    2. Structual Directives
    3. Attribute Directives

1. Component Directives:

    - Component Directives are used in Main Class.
    - They contain the detail of How the Component should be Processed, Instantiate and used at Runtime.
    - Component can be used as Directives.
    - Every Component has Input and Output option to pass between component and its parent HTML elements

Syntax: <component-selector-name [input-reference]="input-value"> ... </component-selector-name>

Example: <list-item [items]="products"> ... </list-item>

2. Structual Directives:

    - The Structual Directives start with a '*' sign.

- These Directive are used to manipulate and changee the structure of the DOM elements

- Used to add or remove DOM elements in the current Html document

Syntax: <html-selector-name *DirectiveName="itsValue"> ... </html-selector-name>

Example: <p *ngIf="true"> .. </p>

Some List of Structual Directives: *ngIf, *ngFor, *ngSwitch

*ngIf - It allows us to add / remove the DOM element

*ngSwitch - allows us to add / remove the DOM element. It is similar to Switch statement

*ngFor - It is used to repeat a portion of Html element once per each item form an iterable list

3. Attribute Directives:

- It is used to look and behaviour of the DOM elements

- Used to add new directives for thhe existing Html elements to change its look and behaviour

Syntax: <html-selector-name [attriName]="itsValue" >

Example: <p [ngStyle]=""> .. </p>

Some List of Attribute Directives: ngClass, ngStyle

-> Creating the Template or Layout with Custom CSS and Routing:

- css: grid-<>

-> Crud Operations:

- two way property binding with template driven

- created the model and service

- accessing the service into the component

-> Lazy Loading:

- lazy loading is a technique in angular that allows you to load javascript components asynchronously when a specific route is activated.

- it improves the speed of the application loading time by splitting the application into several bundles.

- when the user navigates through the application the bundles are loaded as required.

- ViewContainerRef:

  - ViewContainerRef represents a container where one or more views can be attached.

  - the first thing to mention here is that any DOM element can be used as a View Container

  - Angular doesn't insert views inside the element but appends then after the element bound to ViewContainer

- ComponentFactoryResolver:

  - a simple registry that maps Components to generated ComponentFactory classes that can be used to Create Instance of Components.

  - use to obtain the factory for a given component type, then the factory's create() method to create a component.

- ViewChild:

  - a ViewChild is a Component, Directive or Element as a part of a Template.

  - If we want to access to a Child Component, Directive, or Element inside the Parent Component,

    we use @ViewChild()

  - since the Child Component can be located inside the Parent Component it can be accesse as @ViewChild

- ElementRef:

    - the ElementRef is a wrapper around a Native Element inside of a View.

    - It's simply a Class that wraps Native DOM elements inside the browser and allows you to work with the DOM by providing the nativeElement object which exposes all the Methods and Properties of the Native Elements

-> Custom Directives:

- creating the custom directive which executes your program and drives through the output

These are the modules that we are using in creating the Custom Directive

- Directive
- ElementRef
- Input
- Renderer2

```
<app-root>
[appInputBox]

class .. {


}
```

-> Pipes:

- pipes are a simple way to transform values in an angular template
- have some builtin pipes: uppercase, lowercase, date, currency, json etc.,

-> Custom Pipes:

- writing the custom code (or) custom definition for the name of the pipe which we have given to apply to our value(s)

-> Life Cycle Hooks:

- the life cycle hooks are methods that angular invokes on directives and components
- as it creates, destroys
- they can fine tune behavior of components during creation, updation, destruction

- when an angular application starts, it creates and renders the root component
- then it create its children and renders it
- for each loaded component, it checks and when its data bound properties change and update them
- it destroys the component and removes from the dom when no longer required

- there are different types of life cycle hooks:
    1. ngOnChanges
    2. ngOnInit
    3. ngDoCheck
    4. ngAfterContentInit
    5. ngAfterContentChecked
    6. ngAfterViewInit
    7. ngAfterViewChecked
    8. ngOnDestroy

- OnInit: it is to 'initialize the directive / component after angular first displays the data - bound properties and sets the directive / component \'s input properties'. it is called once after the first ngOnChanges()

- AfterContentChecked: it is called after the default change detector has completed checking all content of a directive

- AfterContentInit: it calls after the component's content has been fully initialized and injected into components view. it also updates the properties decorated with the concentchild and contentchildren before raising this hook.

- AfterViewChecked: is called after the default change detector has completed checking a components view for changes

- AfterViewInit: it is called when the components view has been attached. the angular compiles all views into js files, not html - the framework manages a template in code and has a rendering engne to interact with the DOM

- DoCheck: that invokes a custom change-detection function for a directive. in addition to the check performed by the default change-detector

- OnChanges: it is used to respond when angular (re)sets data - bound input properties. this method receives a simplechanges object of a current and previous properties values. it is called before ngOnInit() and whenever one or more data bound input properties change.

- OnDestroy: it can hooked into on components and directives, by defining this specific method, named as OnDestroy on our class, we are informing the angular runtime, that it should call our method at the appropriate time.

- SimpleChanges: simplechanges in an angular feature, it is used to see the changes and a few more details of the declared property names in a component. and also it needs to be used in angular ngOnChanges() to see the values changes and to do relevant things

-> dependency injection:

- service, dependency injection, @input, @output and Event Emitter
- service:
    - service is a broad category encompassing any value, function, or feature that an application needs.
    - a service is typically a class with a norrow, well-defined purpose.
    - it should do something specific and do it well

- angular distinguishes components from services to increase modularity and reusability

- dependency injection:
    - dependency injection (or) di, it is a design pattern in which a class requests dependencies from external sources rather than creating them
    - the di framework provides dependencies to a class upon instantiation
    - use di to increase the flexibility, modularity in your application
- @input():
    - it is a common pattern in angular is sharing data between a parent component and one or more child components
    - @input() let's a parent component update data in the child component
- @output():
    - @output() binds a property of a component to send data from one component to the calling component
    - @output() binds a property of the type of angular EventEmitter class.
    - to transfer the data from child to parent component
- @input() and @output():
    - allow angular to share data between the parent context and child directives or components.
    - an @input property is writable while an @output() property is observable
- event emitter:
    - eventemitter is an angular abstraction and its only purpose is to emit events in components.
    - eventemitter is really an angular abstraction and should be used only for emitting custom events in components
    - otherwise, just we use 'rxjs' as if it was any other library
    - use in components with the @output directive to emit custom event sync (or) async., and register handlers for those events by subscribing to an instance