GHSV:

Directives, routing, attributes, observables, lazy loading

Angular is Google's own product with various versions available in the market and it is one of the growing Javascript frameworks.
It is used for building Single Page Applications, also known as SPA.
It supports two-way data binding.
It is used for the development of mobile/desktop/web applications.
It is a component-based web development framework. It uses various components for the development of web applications.

CLI - Command Line Interface
NPM - Node Package Manager
Install Node.js
npm install -g @angular/cli
node -v
ng --version
ng serve -> to run the application on port # 4200
ng serve --port = 4300 -> to change the port if already a applicaiton is running on above port#
ng new <newprojectname> -> Create new project
npx kill-port 4200 - KILL the process which runs on port number 4200

=> Install Bootstrap Framework in Angular app

1. Before this remove -> content from -> src -> clear css from app.component.html
2. ng i bootstrap
3. bootstrap version you can check in package.json like this - "bootstrap": "^4.5.0"
4. Add this line into '\oneConversion\src\styles.scss -> @import '~bootstrap/dist/css/bootstrap.min.css'; this bootstrap.min.css available in where our node_modules are installed
5. You can test whether bootstrap installed correctly or not , you can add simple code in app.component.html page - <h1 class="bg-primary">Welcome to One Conversion Project</h1>

=> Default Folder Structure and Boot Process

1. E2E
    - end to end test scripts will be residing
    - Protractor framework is used to run end to end test scripts
    - app.po.ts -> Protractor file which will have "po" file
    - app.e2e.spec.ts file

- e2e it tells that this script is end to end functional test scripts
- spec - whenever you added spec that means it is a test script

2. node_modules
   - this is the folder where all modules and libraries reside
   - add/remove modules/packages
   - we will not touch this folder for development purpose
3. src
   - This is the main workarea / app code resides in this folder
   - This is the basic structure
   - app folder is the place where you will see components, modules, services, directives, pipes, etc...
   - app.component.ts [where you see component it means that file corresponds to a component]
   - app.module.ts [it means that this is a module]
   - app.service.ts [it means these are service files]
   - app.component.spec.ts [It means these are UNIT test scripts]
   - assets - where you can keep you styles, images, Icons
   - environments
     - here we will configure variables or pipelines for dev, test, stage and prod
   - polyfills - if the user browser is outdated one - polyfills will add ES6 functionality for backward compatibility
   - styles.css - Keep Gloabal Stylesheet, Don't make it too heavy, use limited typography, fonts, generic classes. If it is specific to application then keep them in app.component.css
4. angular.json - This is the file for all the configuration of our angular project
5. karma.conf.js - karma runner for running our unit test scripts, configure our test scripts
6. tsconfig.json - Build and compilation related to our angular application
7. tslint.json - linting and code standards

Boot Process:

1. main.ts
   - Booting the angular application
   - This is the first file that angular will check to load how to start an angular application
   - We need ATLEAST one module to be present in out main.ts, default one is AppModule
   - bootstrapModule is the one which loads and starts the application
   - Above method is - NOT RELATED TO BOOTSTRAP FRAMEWORK AT ALL
   - Yes, we can change default AppModule to our own customer module

2. test.ts
   - test script for booting process
   - All the code written in main.ts is tested here in this file

****

****

Angular 9 Tutorial For Beginners #6 - Package.json and Package-lock.json

- You will scripts commands - which are very useful in your build pipelines
- to understand any existing angular project , always refer the package.json to see the scripts
- It will have the all dependencies and devdepencies
- package-lock.json will resolve all the required sub modules which are in package.json file (ex - platform-browser)

****

Angular 9 Tutorial For Beginners #7 - Angular CLI

- ng new <projectname>
- ng --version
- ng serve -> used for transpiling/compling of our angular application with default port#4200
- ng serve --port = 4300 -> change the port from #4200 to #4300
- production - Ahead Of Time (AOT) is enabled defaultly for production
- ng test -> will run all our unit tests, tests are run using karma runner, unit tests are written in jasmine framework, code coverage, we can disable some tests,
- ng e2e -> it will run all our end to end tests, e2e-spec.ts are end to end test scripts, protractor
- ng update -> if you running on the version which is less than current version , it will update to latest CLI
- ng build -> build and generate the output of your application , compiled javascript code, this is mainly for promoting code to next environment
- ng lint -> code syntax linting, , set rules for code should be written, coding standards, it follows the coding standards.
- ng generate [component],[module],[pipe],[service]
- ng generate component <component name> OR short cut -> ng g c <component name>

```
PS E:\Angular9Ex\oneConversion> ng generate component login <ENTER>
    CREATE src/app/login/login.component.html (20 bytes)
    CREATE src/app/login/login.component.spec.ts (621 bytes)
    CREATE src/app/login/login.component.ts (272 bytes)
    CREATE src/app/login/login.component.scss (0 bytes)
    UPDATE src/app/app.module.ts (471 bytes)
```

- ng generate service <service name>

PS E:\Angular9Ex\oneConversion> ng generate service authentication (ENTER)

CREATE src/app/authentication.service.spec.ts (397 bytes)
CREATE src/app/authentication.service.ts (143 bytes)

****
****
Angular 9 Tutorial For Beginners #8 - Modules

1. Modules in Angular are logical functionality
2. For ex: Users module which will have components like - login, authenication, forget, signup,services
3. Modules can have components, pipes, directivies, services etc... (all are related to a particular functionality)
4. Plug and Play features
5. It allows our application to be modulear, easy to debug, reduces the application foot print, easy to maintain
6. Default module called - AppModule (every angular application should have atleast one module)
7. We need to import required core packages - ex - BrowserModule, ngModule
8. every module needs to be defined by @NgModule
9. We can declare components as we need
10. We can also import other modules inside a module
11. which is the default component to load -> using bootstrap: [AppComponent]
12. Providers -> we can add required services
13. using -> ng generate module <modulename>

****
****

Angular 9 Tutorial For Beginners #9 - Decorators

Decorators in Angular:
All decorators are in angular are represented with the symbol @, Ex: @NgModule, @Component, @HostListener, @Inject, @Pipe

1. Typescript feature used for passing meta data, ex: app.component.ts contains information for - selector, templateurl, styleurl
    - Some of meta data is optonal like - you can remove templateurl replate with template: '<h1>Welcome</h1>'
2. Decorators are functions that will return functions
3. Decorators are invoke at run time
4. Decorators allows us to invoke functions

Types of Decorators:

1. Class Decorators - @NgModule, @Component
2. Property Decorators - @input, @output
3. Method Decorators - @HostListener
4. Parameter Decorators - @Inject

Decorators are the way to pass meta data to angular application

****

Angular 9 Tutorial For Beginners #10 - Components

1. Component is a smaller functionality which can be reused mutliple times in the application
2. Smaller feature inside a bigger functionality, Example: Authentication (Module) is bigger functionality which has smaller components like Login, Forgot password, Register, authenticate. This allows maintenance and faster development
3. Plug and play whereever we need
4. Tree hierarchy starts with AppComponent (like child1,child2, child3)
5. Appcomponent is the single most important component
    - it is default one
    - In the index.html file - we see selector name <app-root>
6. Each component has 3 important things
    - selector
    - templateurl or template
    - stylesUrl
7. Better use templateUrl instead template
8. you can have components inside components
    - To call sub component , we will use selector name and call the element name in parent component

Use keyword exports in @Ngmodule secion which we want to export the component

todo.module.ts

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';

import { CompletedtasksComponent } from './completedtasks/completedtasks.component';

@NgModule({
  declarations: [CompletedtasksComponent],
```

```
  exports: [CompletedtasksComponent],
  imports: [
    CommonModule
  ]
})
```

include todo.module.ts in app.module.ts to call the other module (todo module) component.

app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { LoginComponent } from './login/login.component';
import { TodoModule } from './todo/todo.module';
import { HoverEffectPipe } from './hover-effect.pipe';
import { TasksComponent } from './tasks/tasks.component';

@NgModule({
  declarations: [
    AppComponent,
    LoginComponent,
    HoverEffectPipe,
    TasksComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    TodoModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

app.component.html

```
<h1 class="bg-primary">Welcome to One Solution</h1>
<app-tasks></app-tasks>
<app-completedtasks></app-completedtasks>
<router-outlet></router-outlet>
```

export class TodoModule { }

9. create component - ng g c <componentname> . below files will be generated
    - component.html
    - component.spec.ts
    - component.ts
    - component.scss
10. Automatically component will be imported into app.module.ts whenever you create a component
    - where you generate the componente is very very important
    - the parent module will be updated accordingly

****

Angular 9 Tutorial For Beginners #11 - Directives

There are 3 types of directives
1. Component Directives
    - Every Angular application must have atleast 1 component
    - Have its own templates
    - Event attached

2. Structural Directives
    - Updates structure of the view
    - ngFor (iteration purpose), nglf (evaluate conditional expression) , ngSwitch (evaluate some expressions , true/false)

app.component.html
<div *ngIf="showDiv">Show Div message using NgIf directive value
app.component.ts
showDiv=true;

3. Attribute Directives
    - ngStyle, ngClass (these are mainly 2 directives)

app.component.html
<div *ngIf="showDiv" [ngStyle] ='{color:colorName}'>Show Div message using NgIf directive value
app.component.ts
colorName='green';

4. Generate custom Direcivies

- adding power and extending our application
- ng generate directive <directive_name>

****

Angular 9 Tutorial For Beginners #12 - NgIf

- is a built-in structural directive
- starts with *ngIf
- returns true or false for a given conditional expression
- add or remove an element dynamically (based on true or false)
-

1. ngIf with else statement
- template reference variables
- always using ng template directive- w

app.component.html

```
<div *ngIf="showDiv; else showElseMsg" >Example of If (True) Else</div>
<ng-template #showElseMsg>Example of If(false) else </ng-template>
```

app.component.ts

```
showDiv=true;
```

3. ngIf , then, else

```
<div *ngIf="showDiv; then showThenMsg; else showElseMsg" ></div>
<ng-template #showthenMsg>Then message </ng-template>
<ng-template #showElseMsg>else block</ng-template>
```

****

Angular 9 Tutorial For Beginners #13 - NgSwitch

- It is a built in directive which starts with *
- Unlike if statement - switch can take multiple value parameters for condition check
- We can also default value for our switch
- Allows element to be shown or hidden based on a condition expression

There are mainly 3 types of elemnts of ngSwitch
- ngSwitch
- ngSwitchCase
- ngSwitchDefault

```
<div [ngSwitch]="switchValue">
<div *ngSwitchCase="1">Test for Value 1 is passed</div>
<div *ngSwitchCase="2">Test for Value 2 is passed</div>
<div *ngSwitchDefault>Test for Default is passed</div>
</div>
```

****

Angular 9 Tutorial For Beginners #14 - NgFor

- it is a built-in structural directive
- Loop the elements to display the array data in the template
- <div *ngFor="let item of collection"/>
- index -> get the index of current element
- First -> returns true if the current element is the first element in the collection
- Last -> returns true if the current element is the last element in the collection
- Even -> returns true if the current element is the even element in the collection
- Odd -> returns true if the current element is the Odd element in the collection

Ex:

app.component.ts

```
  users = [
    {userid: 10, firstname:'james'},
    {userid: 20, firstname:'jack'},
    {userid: 30, firstname:'jhonny'},
    {userid: 40, firstname:'clark'},
    {userid: 50, firstname:'wood'},
    {userid: 60, firstname:'root'}
    ];
```

app.component.html

```
<table class="table">
<tr>
    <th>User Id </th>
    <th>First Name </th>
    <th>Index </th>
    <th>First </th>
    <th>Last </th>
    <th>Even </th>
    <th>Odd</th>
```

```
</tr>
<tr *ngFor="let user of users; index as i;first as f;last as l;even as e; odd as o">
<td>{{user.userid}}</td>
<td>{{user.firstname}}</td>
<td>{{i}}</td>
<td>{{f}}</td>
<td>{{l}}</td>
<td>{{e}}</td>
<td>{{o}}</td>

</tr>
</table>
```

****

Angular 9 Tutorial For Beginners #15 - ngStyle

- ngStyle directive lets set a given DOM elements style properties
- We can set these values via variables

```
<div [ngStyle] = "'background-color':value"></div>
```

applying ngStyle with 2 properites

```
<div *ngIf="showDiv" [ngStyle] ="{'color':colorName, 'background-color':'#ddd'}">Show Div message using NgIf directive value</div>
```

- using ngStyle via variables

```
<div *ngIf="showDiv" [ngStyle] ="{'color':colorName, 'background-color':bgcolorname}">Show Div message using NgIf directive value</div>
```

****

Angular 9 Tutorial For Beginners #15 - ngClass

- ngClass is is directive which is used to set the class name for DOM elements
- [ngClass]=""
- Examples:
    - simple using static class name

```
<div [ngClass]='one'>Example Of ngClass</div>
```

- dynamic- from the component
<div [ngClass]='clsOne'>Example Of ngClass
- array classes - to pass more than 1 class - we use array
<div [ngClass]="[clsOne,clsTwo]">Example Of ngClass
- ngClass with expressions to true or false
<div [ngClass]="{'one':false,'two':true}">Example Of ngClass

****

Angular 9 Tutorial For Beginners #17 - Data Binding

- Means to bind the data from view (template) to controller (component class) and vice versa
- Defines how the data flows and how the data gets updated based on business logic

Types of Binding:
- One Way databinding
Component to View

- Interpolation
- Property Binding
- Style Binding
- Attribute Binding

View to Component
- Event Binding
- Two Way Data Binding
- Data flows from view to component and back to component from the view

****

Angular 9 Tutorial For Beginners #18 - Interpolation

- Is a technique that allows the user to bind data from component to view
- The data flow only one-way ie. from component to view
- Can be used from strings, integers, objects, arrays, and much more
- Syntax - {{variable_name}}

Example 1:

app.component.ts
- subtitle ='this is subtitle of the page.. Interpolation binding';
app.component.html

- {{subtitle}}

Example 2:

app.component.ts
- user = {userid:10,name:'James'};
app.component.html
- {{user.userid}} {{user.name}}

****

Angular 9 Tutorial For Beginners #19 - Property Binding

- Is a technique that allows to the user to bind properties of elements from component to View (template)
- the data flow is one way -> component to view
- can be used for all properties like innerHTML, src etc
- [property]='"expression"'

app.component.html

```
<div [ngStyle]="{color:colorValue}">Example for Property Binding</div>
<p [innerHTML]='propertybindtext'></p>
<div [ngClass]='clsName'>This is property binding with using for class</div>
<div [ngClass]="'c1'">This is property binding with using for class</div>
<input [placeholder]='subtitle'/>
```

app.component.ts

```
subtitle ='this is subtitle of the page.. Interpolation binding';
  colorValue='red';
  propertybindtext = 'property bind text for html inner properties';
  clsName='c1';
```

app.component.scss

```
.c1
{
   background-color: royalblue;
}
```

****

Angular 9 Tutorial For Beginners #20 - Attribute Binding

- Is a technique that allows the user to bind attributes of elements from component to view
- Can be used for any existing properties or customer attributes
- [att.attribute_name]="'expression'"

app.component.html

```
<a [href]='hrefValue' [attr.updatedlink]='updatedlink' [attr.data-link]='updatedlink'>Google</a>
```

app.component.ts

```
hrefValue='www.google.com';
updatedlink='www.yahoo.com';
```

****

Angular 9 Tutorial For Beginners #21 - Event Binding

- Is a technique that allows the user to bind events of elements from view/template to component
- Can be used of all available events
- <button (event_name) = "function()">Click Me</button>

app.component.html

```
<button (click)="btnClick(101)">Click Me</button>
```

app.component.ts

```
btnClick(id){
    alert('Alert Message from ARC tutorials' + id);
    console.log('alert dialog opend');
  }
```

****

Angular 9 Tutorial For Beginners #22 - Two Way Data Binding

- Is a technique that allows the user to bind events of elements from view/templates to

component and vice versa

- The data flows from the both ways ie. view to component and component to view
- Two binding is combination of property binding and event binding
- We bind data using ngModel - square brackets of property bindings with parenthesis of event binding in a single notation
- <input [(ngModel)]= 'data'/>

app.component.html

```
<input [(ngModel)]="firstname"/>
<button (click)="btnReadFirstName(firstname)">ReadFirstName</button>
```

app.component.ts

```
firstname="";
   btnReadFirstName(firstname){
     alert('Entered First Name is' + this.firstname);
   }
```

app.module.ts

```
import { FormsModule } from '@angular/forms';

imports: [
   FormsModule
 ],
```

****

Angular 9 Tutorial For Beginners #23 - Pipes

- pipes are used to transform the data
- Pipes will take data input and convert/transform into desired format
- Pipes are written in using pipe operator (|)
- We can apply pipes to any view/template and to any data inputs

Built-in-pipes

- uppercase, lowercase, currency, date, percent, json

Parameterized pipes

- we can pass one or more parameters to pipe

Chaininng Pipes

- we can connect multiple pipes to a data input

Custom Pipes

- We can create our own pipes for various data formating

****

Angular 9 Tutorial For Beginners #24 - Built In Pipes

app.component.html

```
<div>{{userinfo.id}}, {{userinfo.fname | lowercase}}, {{userinfo.lname | uppercase}}</div>
<div>{{userinfo.DOB | date :'MMM dd'}}, {{userinfo.salary | currency:'USD':'code'}}</div>
<code>{{userinfo|json}}</code>
```

app.component.ts

```
userinfo = {
    id:10, fname:'James', lname:'Clark', DOB:'11-23-2020',salary:'10000'
  }
```

****

Angular 9 Tutorial For Beginners #25 - Parameterized Pipes

in this - 'MMM dd' is parameterized

app.component.ts

```
userinfo = {
    id:10, fname:'James', lname:'Clark', DOB:'11-23-2020',salary:'10000'
  }
```

app.component.html

```
<div>{{userinfo.DOB | date :'MMM dd' </div>
```

****

Angular 9 Tutorial For Beginners #26 - Chaining Pipes

- Using mutliple pipes on a data input is called as Chaining pipes
- We can pass one or more pipes to a data input
    - {{dob|date|uppercase}}

```
<div>Date - {{userinfo.DOB | date:'MMM' | uppercase}}</div>
```

****

Angular 9 Tutorial For Beginners #27 - Custom Pipes

- We can generate custom pipe using CLI -> ng generate pipe <pipe name>
- Pipes need to be added , but when we generate it will automatically added to module
- We need to import the Pipe and PipeTransform from the @angular/core
- Pipes are declared with the decorator @Pipe and provide the selector name

Example #1:

app.component.html

```
<div>{{userinfo.city|highlight:userinfo.city}}</div>
```

highligh.pipe.ts

```
transform(value: string, cityname :string): string {

   return "City Name Is - "+ cityname;
 }
```

Example #2:

highligt.pipe.ts

```
import { DomSanitizer} from '@angular/platform-browser';

export class HighlightPipe implements PipeTransform {
  constructor (private sanitizor:DomSanitizer){};

  transform(value: string, cityname :string): any {
```

```
   return this.sanitizor.bypassSecurityTrustHtml('<div
style="background-color:red">'+cityname+'</div>');
  }

}
```

****

Angular 9 Tutorial For Beginners #28 - Routing

1. All the paths/routes or navigation requirements in Angular are handled by Angular router package
2. We navigate from one component view to other using the routes
3. we can configure various types of routes - default, child, wild card routes,Query params,URL segments, Lazy loading
4. we create a routes array and whenever user requests a route
    - it will search in the routes array
5. import { Routes, RouterModule } from '@angular/router';
6. Router is singleton - which means there is only one instance of the application

****

Angular 9 Tutorial For Beginners #29 - Routing Strategies

Angular having 2 types of routing strategies - PathLocationStrategy (default) , HashLocationStrategy (we will see # in the url)

1. This is extremely important
2. \home \dashboard, \search?keyword=k1, /product, /product/10, /product/10/details, /product?search=param
3. We can create our own strategy there is no strategy. By default, PathLocationStrategy
4. We need to add this in Providers of our module app.module.ts

```
 providers: [
   { provide: LocationStrategy, useClass: HashLocationStrategy }
  ],
```

****

Angular 9 Tutorial For Beginners #30 - Base HREF

- all Angular applicatin should have base href directive in index.html
- The default path of the href is '/' which means its pointing to the root of the server
- Its extremely important - why
    - if we configure the base href in wrong then application will not work
- <base href= "/">
- ng build - it always that you are deploying to the root folder
- http://google.com/ -> root folder , http://google.com/Search/ --> base href = '/search'
- how to configure - ng build --base-href="/search"
- after build -> you can verify in index.html -> base href="/search"

****

Angular 9 Tutorial For Beginners #31 - Routing Module

1. We need to import the modules from the package
    import { Routes, RouterModule } from '@angular/router'; -> available in app-routing.module.ts
2. We need to configure the route path in the form of array
    const routes: Routes = []; -> in app-routing.module.ts
3. Then we need to define our module
    @NgModule({
    imports: [RouterModule.forRoot(routes)],
     exports: [RouterModule]
    })
4. We need to export the module
    export class AppRoutingModule { } -> in app-routing.module.ts
5. import the module in app.module.ts
    import { AppRoutingModule } from './app-routing.module';
     imports: [
    BrowserModule,
    AppRoutingModule
    ],

****

Angular 9 Tutorial For Beginners #32 - Router Outlet

1. Every angular application must have one router outlet
2. We can have more than one outlet in our application
3. Router outlet let us define where output should be displayed
4. Router outlet can be specified at app module or in individual modules

****

Angular 9 Tutorial For Beginners #32- Configuring Routes

- We can configure routes to redirect route for various paths
    - path
    - Component
    - redirectTo
    - children

****

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import{TasksComponent} from './tasks/tasks/tasks.component';
import{ProductsComponent} from './products/products.component';
import{ProductEditComponent} from './product-edit/product-edit.component';
import{ProductViewComponent} from './product-view/product-view.component';

const routes: Routes = [
  {path:'products', component:ProductsComponent},
  {path:'productedit', component:ProductEditComponent},
  {path:'productview', component:ProductViewComponent}
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

****

Angular 9 Tutorial For Beginners #33 - Parameterized Routes

- We can configure and send parameters to our routes
- We need to configure the route and mention the value in dynamic

{path:'product/:id', component:'productidcomponent'}
{path:'product/:id/:id2', component:'productidcomponent'}

ProductIdComponent.ts

```
export class ProductIdComponent implements OnInit {
param1 ='';
param2='';
  constructor(private activeroute :ActivatedRoute) {
this.activeroute.params.subscribe(data=>{this.param1=data.id;this.param2=data.id2;});
```

****

Angular 9 Tutorial For Beginners #34 - Query Params in Routes

Example: employee.com/Search?toys=kids&country=India

Get method calls and visible to user

-make sure no sensitive information is captured via query params

- no passwords, no credit cards etc. they should be post calls

http://localhost:4200/Search?country=india&item=toy&currency=inr

Seach.Component.ts

```
export class SearchComponent implements OnInit {
querycountry=''; queryitem='';querycurrency='';
  constructor(private activatedRoute:ActivatedRoute) {
  this.activatedRoute.queryParams.subscribe(data=>{
console.log(data);
this.querycountry= data.country;
this.querycurrency=data.currency;
this.queryitem=data.item;

   });
```

****

Angular 9 Tutorial For Beginners #35 - Redirecting Routes

- when we want to a route to be redirected to another route - we will implement the redirectTo in our array

- {path: '', redirectTo: 'home', pathMatch:'full'}

- the empty path indicates that its the default route of the application
- the empty path also requeires us to mention that pathMatch should be 'full'

Syntax:

{path:'', redirectTo:'productview',pathMatch:'full'}

****

Angular 9 Tutorial For Beginners #36 - Wildcard Routes

- wild card intercepts any invalid URLs in our application
- When NO matching routes are found in the routes array, that router does not know where to go and hence results in console are errors
- wild card routes are defined in routes array is = {path:'**'}
- Usaually a PageNotFound component is a good example
- we can have more than one wild card routes but not suggestable. only one will be best option
-

****

Angular 9 Tutorial For Beginners #37 - Child Routes

- we can configure any number of child routes to parent route
- child routes - the syntax will be same as defining the routes array
- using logical grouping (products -> product-edit, product-view, product-update, product-delete)

app-routing.module.ts

```
const routes: Routes = [
{path:'products',
children:[
{path:'productedit',component:ProductEditComponent},
{path:'productview',component:ProductViewComponent}

]
},
```

****

Angular 9 Tutorial For Beginners #38 - Lazy Loading Modules

- By Default, NgModules are eagerly loaded, which means that as soon as the app loads, so do all the NgModules, whether or not they are immediately necessary
- For large apps with lots of routes, consider lazy loading - a design pattern that loads NgModules are needed
- Lazy loading helps keep initail bundle sizes smaller, which in turn helps decrease load times
- New with Angular8, loadChildren expects a function that uses the dynamic import syntax to import your lazy loaded module only when it's needed
- with lazy loaded modules in angular, it's easy to have features loaded only when the user navigates to their routes for the first time
- How to implement lazy loading
    - 1. create a feature module , child routes
    - 2. loadChildren: config in the app routing

- Syntax - ng generate module <orders> --route <orders> --module app.module
- It will generate a lazy loaded feature module , you can verify in console (F12), network tab
- when we navigate to orders module then only orders module will be loaded (lazy loaded feature)
- you can add your own routes in OrdersRoutingModule (routes array) - orders-routing.module.ts

```
{ path: 'orders', loadChildren: () => import('./orders/orders.module').then(m => m.OrdersModule)
}
```

Products -> Users, Orders, Cart, Items, Inventory, Vendors

****

Angular 9 Tutorial For Beginners #39 - Route Guards

- Route guards allows us to restrict users to NOT access certain routes or paths
- ex: if the user is not logged in then the user should not see/orders route
    /home, /profile, /todos, /tasks, /card
- whenever implement a route gauard - it will give Boolean value
- Based on this boolean value -> angular router will decide if user should accessing the route
- ng generage gaurd <gaurdname>
- Inject the gaurd in our module under providers
- there are various types of route gaurds available
    - CanActivate -> check to see if a use can visit a route
    - CanActivateChild -> Checks to see if a user can visit child routes

- CanDeactivae -> Checks to see if a user can exit a route
- Reslove - Performs route data retrieval before route activated
- CanLoad - Checks to see if a user can route to a module that lazy loaded

- The route gaurd resolves to true or false based on custom logic and functionality

- while implementing it will ask 4 interfaces that we need to implement

admin-guard.guard.ts

```
import { Injectable } from '@angular/core';
import { CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot, UrlTree } from
'@angular/router';
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class AdminGuardGuard implements CanActivate {
  canActivate(
    next: ActivatedRouteSnapshot,
    state: RouterStateSnapshot): boolean{
      //TODO: write logic for authentication and authorization code here
      //TODO: call user service to check user has entered valid credentials or not


      return false //You can change to 'true' if  you want access admin module

    }
  }
```

app-routing.module.ts

```
import{AdminGuardGuard} from './admin-guard.guard';
const routes: Routes = [
{path:'Admin',component:AdminComponent,canActivate:[AdminGuardGuard]}
```

****

Angular 9 Tutorial For Beginners #40 - Install Bootstrap In Angular

- there are two ways to install bootstrap (CDN and NPM), preferable is NPM
- there maybe changes to URL if we use CDN and lead to issues in production environment
- copy links from - getbootstrap.com , if it is CDN and include them in index.html

- npm i bootstrap jquery popper --save (NOTE: It will download latest packages, you can download specific version by adding other conditions in this command)
- after installing bootstrap these entries will be in package.json
- we will need to import styles and scripts in order to use bootstrap - without bootstrap will NOT work
- go to -> angular.json add styles and scripts under 'architecture' section as shown below

```
"styles": [
  "src/styles.scss",
  "node_modules/bootstrap/dist/css/bootstrap.main.css"
],
"scripts": [
  "node_modules/jquery/dist/jquery_min.js",
  "node_modules/bootstrap/dist/js/bootstrap_min.js"]
```

****

Angular 9 Tutorial For Beginners #41 - Forms in Angular

- Forms are an very integral and essential building blocks of "almost" apps
- We can use any CSS - Bootstrap or Material design
- Forms allows us to gather information and data from users
- Good way to interact with users and almost all websites
- Two types of Forms in Angular
    - Template Driven Forms
    - Reactive Forms (also known as dynamic)
- Angular framework support for forms
    - Two data Binding, Change tracking, Validations, Error handling, Unit testing
- Template Driven Forms
    - Easy to use
    - Template forms are simple and straing forward
    - All the validations, from elements are all defined in the template file
    - We need to import 'FormsModule' in app module to work with template driven forms
- Reactive Forms
    - all the form elements, user interactions and validations are handled in component class
    - we will make use of inbuilt angular built in 'formGroup' and 'formControl'
    - we need to import 'ReactiveFormsModule' in our app module
    - More logic in component class and less morkup in html file
    - Can control better data binding
    - Exclusive define custom regular expression patterns of error handling
    - very flexible and allows users to define, develop complex requirements of forms

- Which is better (Template OR Reactive)
    Template Driven Forms
        - If your application simple and straight forward
        - Fixed static form fields and elements
        - No complex validations or pattern matching
    Reactive Forms
        - if your application forms are complex
        - uses multiple dynamic components
        - Advanced validation requirements
        - Dependent form elements
        - Dynamic form generation based on user preferences

****

Angular 9 Tutorial For Beginners #42- Template Driven Forms

- Easy to use
- Template driven forms are simple and straight forward
- Forms are tracked automatically
- Uses two-way data binding techniques to bind data
- Most of the code resides in template file
- validations are mostly the default HTML5 validations
- unit testing will be challenge
- Tracked form data traverses via various states (pristine etc)
- Import FormsModule while implementing (add it into app.module.ts), add this into array of imports
- ngForm - form name as template variable using "#" for e.g #loginForm
- ngModel - Every field should have a "name" attribute and ngModel attach to it

https://getbootstrap.com/docs/4.5/components/forms/

signincomponent.html

```
<div class="container">
<form #signInForm="ngForm" (ngSubmit)="loginSubmit(signInForm);">
   <div class="form-group">
     <label for="exampleInputEmail1">Email address</label>
     <input type="email" name="emailField" ngModel class="form-control"
id="exampleInputEmail1" aria-describedby="emailHelp">
     <small id="emailHelp" class="form-text text-muted">We'll never share your email with
anyone else.</small>
   </div>
   <div class="form-group">
     <label for="exampleInputPassword1">Password</label>
     <input type="password" name="pwdField" ngModel class="form-control"
```

```
id="exampleInputPassword1">
    </div>
    <div class="form-group form-check">
      <input type="checkbox" name="chkField" ngModel class="form-check-input"
id="exampleCheck1">
      <label class="form-check-label" for="exampleCheck1">Check me out</label>
    </div>
    <div class="form-group form-check">
        <input class="form-check-input position-static" name="genderField" ngModel type="radio"
id="genderField" value="Male" aria-label="...">
        <label class="form-check-label" for="blankRadio1">Male</label>
    </div>
        <div class="form-group form-check">
        <input class="form-check-input position-static" name="genderField" ngModel type="radio"
id="genderField" value="Female" aria-label="...">
        <label class="form-check-label" for="blankRadio1">Female</label>
      </div>
    <button type="submit" class="btn btn-primary">Submit</button>
  </form>

</div>
```

signincomponent.ts

```
ngOnInit(): void {
    }

    loginSubmit(signInForm:NgForm){
      console.log(signInForm.value.emailField);
      console.log(signInForm.value.pwdField);
      console.log(signInForm.value.chkField);
      console.log(signInForm.value.genderField);
  }
```

****

Angular 9 Tutorial For Beginners #43 - Validations in Template Driven Forms

- Validations are extremely important and integral part of any forms in apps
- we use validations to prevent unwanted data or junk data
- angular provides common vaidators like milLength, maxLength, required etc...
- Angular maintains state information of the forms all the times

- ng-touched, ng-untouched, ng-dirty, ng-pristine, ng-valid, ng-invalid
- Ways to handle validations in template driven forms are #3 types
- Highlight the errors (whenever you touched the control and it is required and invalid)

```
input.ng-invalid.ng-touched{
border:1px solid red}
```

- Disabling the submit button
- by adding attribute [disabled]="!formName.valid"
- Custom field level validation - show hide error messages - This can be done thru two-way (template) binding , add # to the field. #fieldname="ngModel"

```
<span *ngIf="firstname.touched && !firstname.valid">Enter Email</span>
```

signincomponent.html

```
<div class="container">
<form #signInForm="ngForm" (ngSubmit)="loginSubmit(signInForm);">
  <div class="form-group">
    <label for="exampleInputEmail1">Email address</label>
    <input type="email" name="emailField" ngModel email required #emailField="ngModel" class="form-control" id="exampleInputEmail1" aria-describedby="emailHelp">
    <small *ngIf="emailField.touched && !emailField.valid">Enter Email</small>
  </div>
  <div class="form-group">
    <label for="exampleInputPassword1">Password</label>
    <input type="password" name="pwdField" ngModel required #pwdField="ngModel" class="form-control" id="exampleInputPassword1">
    <small *ngIf="pwdField.touched && !pwdField.valid">Enter Password</small>
  </div>
  <div class="form-group form-check">
    <input type="checkbox" name="chkField" ngModel class="form-check-input" id="exampleCheck1">
    <label class="form-check-label" for="exampleCheck1">Check me out</label>
  </div>
  <div class="form-group form-check">
      <input class="form-check-input position-static" name="genderField" ngModel type="radio" id="genderField" value="Male" aria-label="...">
      <label class="form-check-label" for="blankRadio1">Male</label>
  </div>
      <div class="form-group form-check">
      <input class="form-check-input position-static" name="genderField" ngModel type="radio" id="genderField" value="Female" aria-label="...">
      <label class="form-check-label" for="blankRadio1">Female</label>
```

```
    </div>
  <button type="submit" [disabled]="!signInForm.valid" class="btn
btn-primary">Submit</button>
 </form>

</div>
```

****

Angular 9 Tutorial For Beginners #44- Reactive Forms

- all the form elements, user interactions and validations are handled in the component class
- we will make use of angular built in formGroup, and formControl
- using reactive forms we can control better data binding
- exclusive define custom regular expression pattern of error handling
- we need to import 'ReactiveFormsModule' in our app module
- very flexible and allows users to define, develop complex requirements of forms
- more logic in the component class and less in the html page
- Angular maintains state information of the forms all the times
    - ng-touched, ng-untouched, ng-dirty, ng-pristine, ng-valid, ng-invalid

Steps to Implement Reactive Forms: (#5 steps)

- Import ReactiveFormsModule in the app.module.ts
- Create the form in app.compoment.html. formGroup -> we need to use the directive FormGroup for the entire form and give it a name
- import the required modules in component class. import{Component,OnInit} from "@angular/core" , import {FormBuilder, FormGroup,FormControl,NgForm,Validators} from "@angular/forms";
- Inject the FormBuilder in the constructor (constructor (private formBuilder:FormBuilder))
- Create the form instance.
    this.checkoutform = formBuilder.group(
    {email: new formControl(), quantity:new formControl()});

https://getbootstrap.com/docs/4.5/components/forms/

checkout.html

```
<div class="container">
  <h4 class="pd-2">Checkout Form - Using Reactive Forms</h4>
  <form [formGroup]="checkoutform" (ngSubmit)="postdata()">
    <div class="form-group row">
      <label for="inputEmail3" class="col-sm-2 col-form-label">Email</label>
      <div class="col-sm-10">
```

```
          <input type="email" class="form-control" id="inputEmail3" formControlName="emailFld">
        </div>
      </div>
      <div class="form-group row">
        <label for="inputQuantity" class="col-sm-2 col-form-label">Quantity</label>
        <div class="col-sm-10">
          <input type="text" class="form-control" id="quantity" formControlName="qtyFld">
        </div>
      </div>
      <fieldset class="form-group" (ngSubmit)="postdata()">
        <div class="row">
          <legend class="col-form-label col-sm-2 pt-0">Radios</legend>
          <div class="col-sm-10">
            <div class="form-check">
              <input class="form-check-input" type="radio" name="gridRadios" id="gridRadios1"
value="option1" checked>
              <label class="form-check-label" for="gridRadios1">
                First radio
              </label>
            </div>
            <div class="form-check">
              <input class="form-check-input" type="radio" name="gridRadios" id="gridRadios2"
value="option2">
              <label class="form-check-label" for="gridRadios2">
                Second radio
              </label>
            </div>
            <div class="form-check disabled">
              <input class="form-check-input" type="radio" name="gridRadios" id="gridRadios3"
value="option3" disabled>
              <label class="form-check-label" for="gridRadios3">
                Third disabled radio
              </label>
            </div>
          </div>
        </div>
      </fieldset>
      <div class="form-group row">
        <div class="col-sm-2">Checkbox</div>
        <div class="col-sm-10">
          <div class="form-check">
            <input class="form-check-input" type="checkbox" id="trmChk"
formControlName="trmChk">
```

```
            <label class="form-check-label" for="gridCheck1">
              Agree to Terms & Conditions
            </label>
          </div>
        </div>
      </div>
      <div class="form-group row">
        <div class="col-sm-10">
          <button type="submit" class="btn btn-primary">Sign in</button>
        </div>
      </div>
    </form>
</div>
```

checkcomponent.ts

```
import { Component, OnInit } from '@angular/core';
import { FormGroup, FormControl, FormBuilder } from '@angular/forms';

@Component({
  selector: 'app-checkout',
  templateUrl: './checkout.component.html',
  styleUrls: ['./checkout.component.scss']
})
export class CheckoutComponent implements OnInit {

  checkoutform: FormGroup;
  constructor(private formBuilder: FormBuilder) {
    this.checkoutform = formBuilder.group(
      {
        emailFld: new FormControl(),
        qtyFld: new FormControl(),
        trmChk: new FormControl()
      }
    )


  }

  ngOnInit(): void {
  }

  postdata() {
```

```
    console.log(`Email Address: `+ this.checkoutform.value.emailFld);
    console.log(`Quantity: `+ this.checkoutform.value.qtyFld);
    console.log(`IsTermsChecked: `+ this.checkoutform.value.trmChk);

  }

}
```

****

Angular 9 Tutorial For Beginners #45- Reactive Forms - Validations

- Validations are extremely important and integral part of any forms in apps
- we use validations to prevent unwanted data or junk data
- angular provides common vaidators like milLength, maxLength, required etc...
- Angular maintains state information of the forms all the times
    - ng-touched, ng-untouched, ng-dirty, ng-pristine, ng-valid, ng-invalid
- 3 Ways to perform validations in Angular reactive forms
    - Highlighting the errors
    - Disabling Submit button
    - Custom Field Level Validations
- Import Validators into component file

checkout.component.html

```
<div class="container">
  <h4 class="pd-2">Checkout Form - Using Reactive Forms</h4>
  <form [formGroup]="checkoutform" (ngSubmit)="postdata()">
    <div class="form-group row">
      <label for="inputEmail3" class="col-sm-2 col-form-label">Email</label>
      <div class="col-sm-10">
        <input type="email" class="form-control" id="inputEmail3" formControlName="emailFld">
        <span *ngIf="checkoutform.get('emailFld').touched &&
checkoutform.get('emailFld').hasError('required')">Enter email Address</span>
        <span *ngIf="checkoutform.get('emailFld').touched &&
checkoutform.get('emailFld').hasError('email')">Enter Valid email Address</span>
      </div>
    </div>
    <div class="form-group row">
      <label for="inputQuantity" class="col-sm-2 col-form-label">Quantity</label>
      <div class="col-sm-10">
        <input type="text" class="form-control" id="quantity" formControlName="qtyFld">
        <span *ngIf="checkoutform.get('qtyFld').touched &&
```

```
checkoutform.get('qtyFld').hasError('required')">Enter Quantity</span>
        <span *ngIf="checkoutform.get('qtyFld').touched &&
checkoutform.get('qtyFld').hasError('minlength')">Minmum Length Not Matched</span>
      </div>
    </div>
    <div class="form-group row">
      <div class="col-sm-2">Checkbox</div>
      <div class="col-sm-10">
        <div class="form-check">
          <input class="form-check-input" type="checkbox" id="trmChk"
formControlName="trmChk">
          <label class="form-check-label" for="gridCheck1">
            Agree to Terms & Conditions
          </label>
        </div>
      </div>
    </div>
    <div class="form-group row">
      <div class="col-sm-10">
        <button type="submit" class="btn btn-primary"
[disabled]="!checkoutform.valid">Checkout</button>
      </div>
    </div>
  </form>
</div>
```

checkout.component.ts

```
import { Component, OnInit } from '@angular/core';
import { FormGroup, FormControl, FormBuilder, Validators } from '@angular/forms';

@Component({
  selector: 'app-checkout',
  templateUrl: './checkout.component.html',
  styleUrls: ['./checkout.component.scss']
})
export class CheckoutComponent implements OnInit {

  // checkoutform: FormGroup;
  // constructor(private formBuilder: FormBuilder) {
  //   this.checkoutform = formBuilder.group(
  //     {
  //       emailFld: new FormControl(),
```

```
  //       qtyFld: new FormControl(),
  //       trmChk: new FormControl()
  //     }
  //   )
  // }

  checkoutform: FormGroup;
  constructor(private formBuilder: FormBuilder) {
    this.checkoutform = formBuilder.group(
      {
        emailFld: ['',[Validators.required,Validators.email]],
        qtyFld: ['',[Validators.required,Validators.minLength(5)]],
        trmChk: ['',Validators.requiredTrue]
      }
    )
  }

  ngOnInit(): void {
  }

  postdata() {
    console.log(`Email Address: ` + this.checkoutform.value.emailFld);
    console.log(`Quantity: ` + this.checkoutform.value.qtyFld);
    console.log(`IsTermsChecked: ` + this.checkoutform.value.trmChk);

  }

}
```

****

Angular 9 Tutorial For Beginners #46- Reactive Forms - Get Values

- We can read the entire form fields/controls from the form
- Syntax to read the value of entire form -> this.formName.value
- Syntax to read the value of individual form control -> this.formName.get('fieldname').value

```
 postdata() {

    console.log(this.checkoutform.value);
    console.log(`Email Address using get: ` + this.checkoutform.value.emailFld);
    console.log(`Email Address: ` + this.checkoutform.get('emailFld').value);
    console.log(`Quantity: ` + this.checkoutform.value.qtyFld);
    console.log(`IsTermsChecked: ` + this.checkoutform.value.trmChk);
```

```
 }
```

****

Angular 9 Tutorial For Beginners #47- Reactive Forms - Set Values

- We can SET all the values to the entire form using setValue()
- Setting individual form field controls using - patchValue()

```
ngOnInit(): void {
   /*setValue - we need set all the values in the form. Otherwise, we will get error
   Must supply a value for form control with name:<fieldName>
   */
   // this.checkoutform.setValue(
   //   {
   //     emailFld: 'james@gmail.com',
   //         trmChk: true
   //     qtyFld: '12345', trmChk: true
   //   })

   /*patchValue - WE can assign some of values of the form using .patchValue method */

     this.checkoutform.patchValue(
       {
         emailFld: 'james@gmail.com',
         trmChk: true
       }

   )
 }
```

****

Angular 9 Tutorial For Beginners #48- Reactive Forms - Reset Forms

- It is important that we reset our form to avoid any duplicate values getting added

- ResetForm - reset()
- this.formname.reset();

.html

```
<button type="submit" class="btn btn-link" (click)="resetForm();" >Clear</button>
```

component.ts

```
resetForm(){
   this.checkoutform.reset();
 }
```

****

Angular 9 Tutorials For Beginners #49- Reactive Forms - Value Changes

- Form Group - valueChanges
  - valueChanges is yet another important property of FormControl, FormGroup and FormArray
  - valueChanges returns an obervable
  - We need to Subscribe to the observable to read the value
  - valueChanges is a property in AbstractControl
  - valueChanges will emit an even every time there is any change in the value of the control changes
  - Approach #1 - for Form controls
    - this.formname.get('fieldname').valueChanges.subscribe(data => {console.log(data);})
  - Approach #2 - for entire Form
    - this.formname.valueChanges.subscribe(data=>{console.log(data);})

```
ngOnInit(): void {

   this.checkoutform.get('emailFld').valueChanges.subscribe(data=>{console.log(data);})
}

ngOnInit(): void {
this.checkoutform.valueChanges.subscribe(data=>{console.log(data);})
}
```

****

Angular 9 Tutorial For Beginners #50- Reactive Forms : Status Changes

- statusChanges is yet another important propery for FormGroup, FormControl, FormArray
- statusChanges returns an Observable, we need to subscribe to the Observable to read the value
- statusChanges is a propery in AbstractControl
- statusChanges will emit an every time there is any change in the validaion status of the control changes
- Approach #1 - for Form controls

-

this.formname.get('fieldname').statusChanges.subscribe(data => {console.log(data);})

- Approach #2 - for entire Form

-

this.formname.statusChanges.subscribe(data=>{console.log(data);})

```
ngOnInit(): void {

   this.checkoutform.get('emailFld').statusChanges.subscribe(data=>{console.log(data);})
}

ngOnInit(): void {
this.checkoutform.statusChanges.subscribe(data=>{console.log(data);})
}
```

****

Angular 9 Tutorial For Beginners #51- Reactive Forms - Form Array

- Helps in building basic form with form elements
- for complex forms, we will need form arrays
- almost all modern applications will need us to work with form arrays
- DOM interactions in Angular Reactive Forms are implented using the form arrays
- Adding and removing elements can be handled in easy way by using Form Arrays
- Form Array can be group of FormControls
- Form Array can be group of FormGroups
- FormArray is 3 types - Simple,

FormArray - invalid //if any one of the formcontrol is invalid

[

FormControl - valid

FormControl - valid

FormContro- invalid

]

- FormArray
  - new FormControl(),
  - new FormControl()

Example:

```
[
    {itemid:1},
    {itemid:2},
    {itemid:3}
```

]

-FormArray // most applications will have this complex forms
- FormGroup()
-FormControl()
-FormControl()

Example:

[
{itemid:1, itemName:'test1', itemdesc:'test1desc'},
{itemid:2, itemName:'test2', itemdesc:'test2desc'},
{itemid:3, itemName:'test3', itemdesc:'test3desc'},
]

checkout.component.html

```
<div class="container">
  <h4 class="pd-2">Checkout Form - Using Reactive Forms</h4>
  <form [formGroup]="checkoutform" (ngSubmit)="postdata()">
    <div class="form-group row">
      <label for="inputEmail3" class="col-sm-2 col-form-label">Email</label>
      <div class="col-sm-10">
        <input type="email" class="form-control" id="inputEmail3" formControlName="emailFld">
        <span *ngIf="checkoutform.get('emailFld').touched &&
checkoutform.get('emailFld').hasError('required')">Enter
          email Address</span>
        <span *ngIf="checkoutform.get('emailFld').touched &&
checkoutform.get('emailFld').hasError('email')">Enter Valid
          email Address</span>
      </div>
    </div>
    <div class="form-group row">
      <label for="inputQuantity" class="col-sm-2 col-form-label">Quantity</label>
      <div class="col-sm-10">
        <input type="text" class="form-control" id="quantity" formControlName="qtyFld">
        <span *ngIf="checkoutform.get('qtyFld').touched &&
checkoutform.get('qtyFld').hasError('required')">Enter
          Quantity</span>
        <span *ngIf="checkoutform.get('qtyFld').touched &&
checkoutform.get('qtyFld').hasError('minlength')">Minmum
          Length Not Matched</span>
      </div>
```

```
    </div>
    <div class="form-group row">
      <label for="items" class="col-sm-2 col-form-label">Items</label>
      <div class="col-sm-10" formArrayName="items">
      <div *ngFor="let item of checkoutform.controls.items['controls']; let i=index"
[formGroupName]="i">
        <a [routerLink] (click) ="removeItem(i)">remove</a>
        <input class="form-control" type="text" formControlName="itemId" id=task{{i}}>
        <input class="form-control" type="text" formControlName="itemName" id=task{{i}}>
        <input class="form-control" type="text" formControlName="itemDescription" id=task{{i}}>
        <input type="checkbox" formControlName="itemDone"> Mark as done
      </div>
    </div>
</div>

<div class="form-group row">
  <!-- <div class="col-sm-2">Checkbox</div> -->
  <div class="col-sm-10">
    <div class="form-check">
      <input class="form-check-input" type="checkbox" id="trmChk" formControlName="trmChk">
      <label class="form-check-label" for="gridCheck1">
        Agree to Terms & Conditions
      </label>
    </div>
  </div>
</div>
<div class="form-group row">
  <div class="col-sm-10">
    <button type="submit" class="btn btn-primary"
[disabled]="!checkoutform.valid">Checkout</button>
    <button type="submit" class="btn btn-link" (click)="resetForm();">Clear</button>
  </div>
</div>
<div>valueChangeFld is : {{valueChangeFld}}</div>
</form>
</div>

checkout.component.ts

checkoutform: FormGroup;
  constructor(private formBuilder: FormBuilder) {
    this.checkoutform = formBuilder.group(
      {
```

```
      emailFld: ['', [Validators.required, Validators.email]],
      qtyFld: ['', [Validators.required, Validators.minLength(5)]],
      trmChk: ['', Validators.requiredTrue],
      items: this.formBuilder.array([
        this.formBuilder.group({
          itemId: ['1'],
          itemName: ['TeleVision'],
          itemDescription: ['SmartTv'],
          itemDone: ['', Validators.requiredTrue]
        })

      ])
    }
  )
}
```

****

Angular 9 Tutorial For Beginners #51- Reactive Forms - Nested Form Array

- We need Nested form array for complex form requirements involving dynamic items which are rendered into the form
- Ability to group multiple form controls and form groups into an Form Array
- ability to dynamically add or remove rows into the Form Array via form group
- FormArray aggregates the values of the "child" FormControl into an Array
- The status of the FormArray is calculated by reducing the statuses of the it's children
- If any of the child control is invalid then entire form array becomes invalid

****

Angular 9 Tutorial For Beginners #52- Reactive Forms - Add Rows Dynamically

- There will be use cases when we will not know how many rows or fields to expect
- That's when we will need to dynamically "Add Rows" to the form on the fly
- Adding new rows to form simply refers to appending/pushing the form groups or form controls to the form array at run time
- Form Array - Form Group, Form Control

component.ts

```
constructor(private formBuilder: FormBuilder) {
  this.checkoutform = formBuilder.group(
    {
      emailFld: ['', [Validators.required, Validators.email]],
```

```
      qtyFld: ['', [Validators.required, Validators.minLength(5)]],
      trmChk: ['', Validators.requiredTrue],
      items: this.formBuilder.array([
        this.formBuilder.group({
          itemId: ['1'],
          itemName: ['TeleVision'],
          itemDescription: ['SmartTv'],
          itemDone: ['', Validators.requiredTrue]
        })

      ])
    }
  )
}

//array name
get items() {
  return this.checkoutform.get('items') as FormArray;
}

//add rows to the form
AddRow() {
  const arraylength = this.items.length;
  const newitem = this.formBuilder.group({
    itemId: [arraylength+1],
    itemName: [''],
    itemDescription: [''],
    itemDone: ['', Validators.requiredTrue]
  });
    this.items.push(newitem);

}
```

component.html

```
<div class="container">
<h4 class="pd-2">Checkout Form - Using Reactive Forms</h4>
<form [formGroup]="checkoutform" (ngSubmit)="postdata()">
  <div class="form-group row">
    <label for="inputEmail3" class="col-sm-2 col-form-label">Email</label>
    <div class="col-sm-10">
      <input type="email" class="form-control" id="inputEmail3" formControlName="emailFld">
      <span *ngIf="checkoutform.get('emailFld').touched &&
```

```
checkoutform.get('emailFld').hasError('required')">Enter
          email Address</span>
        <span *ngIf="checkoutform.get('emailFld').touched &&
checkoutform.get('emailFld').hasError('email')">Enter Valid
          email Address</span>
      </div>
    </div>
    <div class="form-group row">
      <label for="inputQuantity" class="col-sm-2 col-form-label">Quantity</label>
      <div class="col-sm-10">
        <input type="text" class="form-control" id="quantity" formControlName="qtyFld">
        <span *ngIf="checkoutform.get('qtyFld').touched &&
checkoutform.get('qtyFld').hasError('required')">Enter
          Quantity</span>
        <span *ngIf="checkoutform.get('qtyFld').touched &&
checkoutform.get('qtyFld').hasError('minlength')">Minmum
          Length Not Matched</span>
      </div>
    </div>
    <div class="form-group row">
      <label for="items" class="col-sm-2 col-form-label">Items</label>
      <div class="col-sm-10" formArrayName="items">
      <div *ngFor="let item of checkoutform.controls.items['controls']; let i=index"
[formGroupName]="i">
        <a [routerLink] (click) ="removeItem(i)">remove</a>
        <input class="form-control" type="text" formControlName="itemId" id=task{{i}}>
        <input class="form-control" type="text" formControlName="itemName" id=task{{i}}>
        <input class="form-control" type="text" formControlName="itemDescription" id=task{{i}}>
        <input type="checkbox" formControlName="itemDone"> Mark as done
      </div>
    </div>
</div>

<div class="form-group row">
  <!-- <div class="col-sm-2">Checkbox</div> -->
  <div class="col-sm-10">
    <div class="form-check">
      <input class="form-check-input" type="checkbox" id="trmChk" formControlName="trmChk">
      <label class="form-check-label" for="gridCheck1">
        Agree to Terms & Conditions
      </label>
    </div>
  </div>
```

```
</div>
<div class="form-group row">
  <div class="col-sm-10">
    <button type="submit" class="btn btn-primary"
[disabled]="!checkoutform.valid">Checkout</button>
    <button type="submit" class="btn btn-link" (click)="resetForm();">Clear</button>
    <button type="submit" class="btn btn-link" (click)="AddRow();">Add Row</button>
  </div>
</div>
<div>valueChangeFld is : {{valueChangeFld}}</div>
</form>
</div>
```

****

Angular 9 Tutorial For Beginners #53- Reactive Forms - Remove Rows

- There will be use cases when we will have to remove dynamically added rows from the forms
- That's when we will need to dynamically "Remove Rows" on the fly
- We will need to capture the index of the row which we want to remove from the array and then use IndexAt method to remove the row

component.ts

```
 removeItem(itemId)
  {
    this.items.removeAt(itemId);
  }
```

****

Angular 9 Tutorial For Beginners #54 - Observable

1. Observable is part of RxJs library
2. import observable into our component where we want to make use of it
3. Observable is a sequence of data that is emitted over period of time
4. This data can be of any time - string , events, etc
5. Angular uses observable very frequently in most aysnc operations
    - HTTP, Routing, Event Handling
6. In order to listen and track the changes of observable - we need observer
7. Observer will continuously track the changes in observable
8. Observer has methods like - next (), error (), complete()
9. Observable as it - is useless unless we subsribe it

10. By subscribe we mean that we are processing the data/values by observable over period of time

11. We can have multiple subscribers to our observable

12. We can also unscribe from a subscriber

Example: Shopping Cart

- Initial Order - InProgress
- Processing
- Trasit
- Delivered
- Completed

component.ts

```
import { Component, OnInit } from '@angular/core';
import { Observable, Subscriber } from 'rxjs';

@Component({
  selector: 'app-observable',
  templateUrl: './observable.component.html',
  styleUrls: ['./observable.component.scss']
})
export class ObservableComponent implements OnInit {
  orderStatus: any;
  data: Observable<any>;
  constructor() { }

  ngOnInit(): void {
    this.data = new Observable(o => {
      //Business Logic comes here
      setTimeout(() => { o.next('In Progress'); }, 4000);
      //Business Logic comes here
      setTimeout(() => { o.next('Processing'); }, 8000);
      //Business Logic comes here
      setTimeout(() => { o.next('Trasit'); }, 12000);
      //Business Logic comes here
      setTimeout(() => { o.next('Delivered'); }, 16000);
      //Business Logic comes here
      setTimeout(() => { o.next('Completed'); }, 20000);

      // //if any error
      setTimeout(() => { o.error(); }, 8000);
```

```
    // It will no more track or listen the changes.
    setTimeout(() => { o.complete(); }, 10000);
    setTimeout(() => { o.next('Msg after Completed executed'); }, 20000);


  });

  //first subscription
  this.data.subscribe(o => { this.orderStatus = o; });

  //second subscription -> val2
  // this.data.subscribe(val2 => { this.orderStatus = val2; });

 }

}
```

component.html

```
<p>Order Status - {{orderStatus}}</p>
```

****

Angular 9 Tutorial For Beginners #55 - Dependency Injection Explained

- Dependency injection is an important application design pattern
- Dependency injection is the ability to add the funcitonality of components at runtime
- The DI framework lets you supply data to a component from an injectable "serivce" class, defined on its own file
- Services are reusable classses which can be shared between components
- These services and dependencies are provided at runtime
- This helps in increasing effeciency of applications
- We will use @Injectable to let components know that this is a dependency

****

Angular 9 Tutorial For Beginners #58- Services

- Services allows us to create reusable common shared functionality between various models and components
- Services are singleton

- Services are injected into application using DI mechanism
- Services are commonly used for making HTTP requests to our endpoints to request and receive the resonse
- A service can have a value, methods, or a combination of both!
- Unlike components services need not be included in your modules
- We can create any number of services
- Importing and Injecting services into componets are called dependency injection
- Services are injected at runtime , this way code becomes highly efficient and easy to maintain
- ng generate servce <servicename>
- import {Injectable } from '@angular/core';
- We can generate the service in any of the folder we want
    - but best practice is always keep all services realted into modules
- @Injectable decorator inform to angular that we can inject it into components
- The service is ProvidedIn "root" will be available across the application
    - can be injected into any comonent
- Make the service as public in the constructor so that you can call service method from template(html) as well.

Example:

Contacts Module
- Create Contact
- View Contact
- Edit Contact
- Delete Contact

Contact Services
- HTTP
- Processing Data
- Cleaning Data

services.ts

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

@Injectable({
  providedIn: 'root'
})
export class ContactsService {
  constructor() { }
  getContacts()
  {
```

```
    const contactList=[
      {ContactId:1, ContactName:'James'},
      {ContactId:2, ContactName:'Clark'},
      {ContactId:3, ContactName:'Ruby'},
      {ContactId:4, ContactName:'Jack'}
    ]

    return contactList;
  }

  callingFromTemplate()
  {
          console.log('calling from template');
  }
}
```

component.ts

```
import { Component, OnInit } from '@angular/core';
import { ContactsService } from '../contacts.service';

@Component({
  selector: 'app-contacts',
  templateUrl: './contacts.component.html',
  styleUrls: ['./contacts.component.scss']
})
export class ContactsComponent implements OnInit {
  _contacts = [];
  constructor(private contactservice: ContactsService) {}
  ngOnInit(): void {
    this._contacts = this.contactservice.getContacts();
  }
}
```

component.html

```
<h4>List Of Contacts</h4>
<ul>
    <li *ngFor="let contact of _contacts">
        {{contact.ContactId}} -  {{contact.ContactName}}
    </li>
</ul>
<a (click)="contactservice.callingFromTemplate()">Calling From Template</a>
```

****

Angular 9 Tutorial For Beginners #59- HttpClient

- HTTPClient is used for performing HTTP requests and responses
- The HTTP service is available in the '@angular/common/http' package
- The HTTP client service is included in the HTTP Client Module which is used to initiate the http requests and responses in angular applications
- The HTTPClientModule needs to be imported into the module. Usually in the app module
- HTTPClient give other functionalities like - interceptors, headers etc...
- HTTPClient methods - get(), post(), put(), delete(), head(), jsonp(), optoins(), patch()
- Benefits of HTTPClient
    - includes observable API's
    - have HTTP headers in options
    - includes testability features
    - includes error handling
    - includes typed requests
    - includes response objects

    - Create - Post(), Read - GET(), Update - PUT(), Delete - Delete() (not permanent delete ONLY soft delete, set flag or so)
    - Configure json-server
        - https://www.youtube.com/watch?v=KZTu40cW4M0&feature=youtu.be
        - https://github.com/typicode/json-server

--SETUP Data

> npm i --save json-server (OR) npm install -g json-server
> json-server --watch .\db.json
-- default port run on json-server is - 3000 ie. localhost:3000
- http://localhost:3000 --> if it works fine then your json-server is up and running

****

Angular 9 Tutorial For Beginners #60- Http GET

- get('end-piont');
- get('url', options:{params{},headers{}})
- the response type will be observable
- whenever the response is observable then we need to subscribe in order to read the values
- Headers: is of type HttpHeader
- Params: is of type HttpParam

Steps to implement:

- 1. Import HttpClientModule in app.module
- 2. Import HttpClient in our sevice or component whereeve we are making the HTTP request
- 3. Inject the HTTPClient in the constructor method of the class
- 4. Implement the GET method call
- 5. Import the services into required calling component class
- 6. Call the method to make HTTP GET request

app.module.ts

```
import{HttpClientModule} from '@angular/common/http';
@NgModule({
  declarations: [ContactsComponent]

  imports: [HttpClientModule]
```

service.ts

```
  import { Injectable } from '@angular/core';
 import { HttpClient, HttpParams, HttpHeaders } from '@angular/common/http';

export class ContactsService {
  constructor(private httpClient: HttpClient) { }

  getProducts()
  {
    return this.httpClient.get('http://localhost:3000/products');
  }
```

component.ts

```
 import { Component, OnInit } from '@angular/core';
import { ContactsService } from '../contacts.service';

@Component({
  selector: 'app-contacts',
  templateUrl: './contacts.component.html',
  styleUrls: ['./contacts.component.scss']
})
export class ContactsComponent implements OnInit {
  _products: any;
  constructor(public contactservice: ContactsService) { }
```

```
  ngOnInit(): void {
    this._products = this.contactservice.getProducts().subscribe(data => {
      this._products = data;
    })
  }
}
```

component.html

```
<h4>List Of Products</h4>
<ul>
    <li *ngFor="let product of _products">
        {{product.id}} -  {{product.name}}
    </li>
</ul>
```

****

Angular 9 Tutorial For Beginners #61- HTTP POST

- Follow same as HTTP GET - like import respective modules and httpclient

component.ts

```
import { Component, OnInit } from '@angular/core';
import { ContactsService } from '../contacts.service';

@Component({
  selector: 'app-contacts',
  templateUrl: './contacts.component.html',
  styleUrls: ['./contacts.component.scss']
})
export class ContactsComponent implements OnInit {
  _products: any;
  msgAddProduct = false;
  constructor(public contactservice: ContactsService) { }
  ngOnInit(): void {
    this._products = this.contactservice.getProducts().subscribe(data => {
      this._products = data;})
  }

  addNewProduct() {

    const newFormData =
```

```
  {
    "id": 6,
    "name": "Bike",
    "category-id": 4,
    "description": "Motor Vehicle",
    "price": 1000,
    "is_available": true,
    "rating": 4,
    "reviews": 120,
    "vendor_name": "abcd",
    "warranty": 2,
    "delivery_date": "01-Dec-2020"
  };
```

service.ts

```
addNewProduct: any;
  createProduct(addNewProduct) {
  const httpHeaders = new HttpHeaders();
    httpHeaders.append('content-type','application/json');

    return this.httpClient.post('http://localhost:3000/products',
addNewProduct,{headers:httpHeaders});
     }
```

component.html

```
  <h3>Create New Product</h3>
<button (click)="addNewProduct()">Add Product</button>
<div *ngIf="msgAddProduct">Product added successfully...</div>
```

****

Angular 9 Tutorial For Beginners #62- HTTP PUT

- Making API calls to submit to "update" the existing data is reffered to as a PUT call
- put ('url', body)
- put ('ulr', body, options:{ })
- put ('url', body, options:{ } , params :{ })
- The response type will be Observable
- Since it is Observable, we need to use subscribe in order to read the values

component.html

```
<div *ngIf="msgUpdateProduct">Product Updated successfully...</div>
```

component.ts

```
updateProduct(productId) {

   const newFormData =
   {
     //"id": 6,
     "name": "Bike",
     "category-id": 4,
     "description": "Motor Vehicle_PUT",
     "price": 1000,
     "is_available": true,
     "rating": 4,
     "reviews": 120,
     "vendor_name": "abcd",
     "warranty": 2,
     "delivery_date": "01-Dec-2020"
   };
   this.contactservice.UpdateProduct(productId, newFormData).subscribe(data => {
     this.msgUpdateProduct = true;
   });
```

service.ts

```
UpdateProduct(id, updateProduct)
 {
   const httpHeaders = new HttpHeaders();
   httpHeaders.append('content-type','application/json');
   return this.httpClient.put('http://localhost:3000/products/' +
id,updateProduct,{headers:httpHeaders});
 }
```

****

Angular 9 Tutorial For Beginners #63- HTTP DELETE

- Making API calls to submit to "update" the existing data is reffered to as a PUT call
- delete ('url', body)
- delete ('ulr', body, options:{ })
- delete ('url', body, options:{ } , params :{ })
- The response type will be Observable
- Since it is Observable, we need to use subscribe in order to read the values

component.ts

```
 deleteProduct(prodId) {
   this.contactservice.DeleteProduct(prodId).subscribe(data => {
     this.msgDelProduct = true;
   })
 }
```

Service.ts

```
 DeleteProduct(id)
 {
   const httpHeaders = new HttpHeaders();
   httpHeaders.append('content-type','application/json');
   return this.httpClient.delete('http://localhost:3000/products/' + id,{headers:httpHeaders});

 }
```

component.html

```
 <button (click)="deleteProduct(6)">Delete Product</button>
 <div *ngIf="msgDelProduct">Product deleted successfully...</div>
```

****

Angular 9 Tutorial For Beginners #64- HTTP Headers

- We can send Headers with our HTTP Calls
- HTTPHeaders are immutable - they can not be modified
- Some of the headers are - Authorization , Content-Type
- We can add headers to - Post, GET, Delete, PUT
- Extreamely important when we work with RESTFul APIs
- HTTPHeaders consists of - Append, Has, Get, Keys, getAll, set, delete
- We don't send duplicate headers in the same request

Service.ts

```
 let httpHeaders = new HttpHeaders({
     'content-type': 'application/json',
     'Authorization': 'xeayxeecxyretreqeq#$'
   })

   httpHeaders.set('arc id', '110');

      return this.httpClient.get('http://localhost:3000/products', { headers: httpHeaders });
```

THESE HEADERS WILL BE SHOWN IN DEVELOPER TOOLS UNDER HEADER SECTION

****

Angular 9 Tutorial For Beginners #65- HTTP Params

- We can send Headers with our HTTP Calls
- HTTPParams are immutable - they can not be modified
- Some of the Params are - sending ID for update or delete API's
- We can add headers to - Post, GET, Delete, PUT
- Extreamely important when we work with RESTFul APIs
- HTTPParams consiste of - Append, Has, Get, Keys, getAll, Set, Delete, toString

service.ts

```
getProductById(id) {
    const httpParams = new HttpParams({
      fromObject: {
        id: id
      }
    })

    //'http://localhost:3000/products?id=5
   return this.httpClient.get('http://localhost:3000/products',{params:httpParams});
  }
```

component.ts

```
   getProductById(prodId) {
    this.contactservice.getProductById(prodId).subscribe(data => {
      this._products=data;
      console.log(data);
    }
  }
```

component.html

```
  <button (click)="getProductById(5)">Get ProductBy Id - 5</button>
```

****

Angular 9 Tutorial For Beginners #66 - HTTP Interceptors

- Interceptors handles HttpRequest and HttpResponse

- Intercepts()
- Intercepts() => parameters => req, next
- ng g interceptor <name>

Use Case #1: Create Contact -> log that data -> server
Use Case#2: Create Contact -> show loading icon -> server

interceptor.ts

```
import { Injectable } from '@angular/core';
import {
  HttpRequest,
  HttpHandler,
  HttpEvent,
  HttpInterceptor
} from '@angular/common/http';
import { Observable } from 'rxjs';

@Injectable()
export class LoginInterceptor implements HttpInterceptor {

  constructor() {}

  intercept(request: HttpRequest<unknown>, next: HttpHandler):
Observable<HttpEvent<unknown>> {
    console.log(request);
    console.log('call made ' + request.url);
    console.log('call made ' + request.urlWithParams);
    return next.handle(request);
  }
}
```

app.module.ts

```
import{HttpClientModule, HTTP_INTERCEPTORS} from '@angular/common/http';
import{LoginInterceptor} from './login.interceptor';
providers: [
    { provide: HTTP_INTERCEPTORS, useClass: LoginInterceptor, multi:true }
  ],
```

****

CRUD -
https://www.c-sharpcorner.com/article/crud-operation-in-angular-7-using-web-api/
https://www.youtube.com/watch?v=kGvNEAaOUvE

https://www.youtube.com/watch?v=AHqIrJ_PlPY&t=2628s
https://www.youtube.com/watch?v=np6e7McMVCM

https://www.youtube.com/watch?v=kGvNEAaOUvE
https://www.youtube.com/watch?v=AHqIrJ_PlPY&list=PLjC4UKOOcfDQfrxjOgGKM_UmydQig8pq5
https://www.youtube.com/watch?v=kGvNEAaOUvE&t=2450s

Material Design - https://www.youtube.com/watch?v=8e_nIPFc_7I

https://www.youtube.com/watch?v=BG9UqKQkXAI&t=297s
https://www.youtube.com/watch?v=I317BhehZKM&t=7s

https://www.youtube.com/watch?v=ZTKZBMWL2yA
https://www.c-sharpcorner.com/article/build-crud-operation-using-angular-9-and-dot-net-core-3-1-webapi-and-deploy-to-a/
https://www.c-sharpcorner.com/article/crud-operation-in-asp-net-core-mvc-using-visual-studio-code/
https://www.c-sharpcorner.com/article/create-angular-8-application-with-asp-net-core-3-0/
https://www.youtube.com/watch?v=DuME_6LCKBI
authentication:
https://www.youtube.com/watch?v=rC9rqSQRe38&list=PLC3y8-rFHvwg2RBz6UplKTGIXREj9dV0G&index=7

https://www.youtube.com/watch?v=B_CPqEnS98w
https://www.c-sharpcorner.com/article/angular-8-0-whats-new-and-how-to-upgrade/

ASP.NET CORE
https://www.youtube.com/watch?v=CP-zbZA5LAc&list=PLaFzfwmPR7_LTXu0Vz9Zz_Y0OMMC7ArHZ
https://www.youtube.com/watch?v=nO030SoF5Lo&list=PLrxstxPJs4QnjSNzG4YNCEk-csVvWDs2i