**RK Angular Notes**

**What is Angular ?**

Framework to create Single Page Applications (DOM of page is dynamically changed without page refresh).

**Environment Setup**

1. Download and install latest version of NodeJS
2. Install Angular CLI with command sudo ng install –g @angular/cli

**Create a Project from command Line**

ng new my-project-name

**Start app from command line (navigate to project-folder)**

ng serve

* App is compiled, built and deployed on a local server. Available at <http://localhost:4200>
* A separate thread will monitor for all the changes. As soon as a file is edited and Saved, it is immediately deployed.

**Project Structure**

* **e2e** folder is used for End-to-end testing
* The project code is located in the **src** folder
* **index.html** is the single page that is served in the app.
* Folder “**assets**” is used to store static assets like images.
* “**environments**” folder is used for storing environment variables
* Folder “**app**” is the main folder where we will be working.
* Components are the things that are loaded and displayed.
* The specific html of that component is responsible for the html content, and the ts (typescript) file is responsible for variables and configurations of the component.
* HTML file is what we see, TypeScript is the source of business logic
* app.modules.ts is used to import and enable/unlock the features/components that will be used in this component

**What is TypeScript**

Superset to Javascript

Offers more features than vanilla Javascript

Classes, Interfaces and Types (data-types which will get checked at the time of creating variables and running the code )

Typescript doesn’t run in the browser, needs to compiled to Javascript

**Incorporating Bootstrap into angular Project**

npm install –save bootstrap

In file “**.angular-cli.json**” has a styles array under apps. This allows us to define multiple stylesheets that will apply to the whole application.

**(src/styles.css** is already referenced there.)

In “**.angular-cli.json**” add reference to the bootstrap css by adding one more entry : "../node\_modules/bootstrap/dist/css/bootstrap.min.css"

Now the application will start using the bootstrap styling.

**How Angular Starts working in the app**

“index.html” is the single page that is served by the server

In “index.html” we have <app-root></app-root>

Angular replaces this tag with the contents from “**app.component.html**” because inside file “**app.component.ts**” we have defined a component.

The component is identified by the **selector: ‘app-root’** (this gives the name of the tag)

And content for it comes from **templateUrl: './app.component.html'**

**But how does it know about app.component.ts ?**

In “**src/main.ts**” this code is executed platformBrowserDynamic().bootstrapModule(AppModule)

i.e. it initializes the application with **AppModule**.

The AppModule comes from **src/app/appmodule.ts**

From **appmodule.ts** it knows about “**app.component.ts**” which in-turn has references to **app.component.html** (the component’s html) and **app.component.css** (css specific for that component)

**Components**

Component is a set of code that has some functionality.

The whole application is composed of putting together various components.

App-component is the root component where all the other components will be nested into.

This strategy aids Reusability (of components).

**Creating Components**

Usually Created in in “src/app” each component has it’s own folder.

**Angular uses “components” to build web-pages and uses “modules” to build packages**

Module: is basically a bundle of functionalities and gives angular information on which features my app uses.

**Manual way of creating component**

In the folder for the component, create a

**<my>.component.ts**

**import** {Component} **from** "@angular/core";  
  
@Component({  
 selector: 'app-server',  
 templateUrl: './server.component.html'  
})  
**export class** ServerComponent {  
  
}

@Component is the decorator and it has attributes ‘selector’ and ‘templateUrl’

**<my>.component.html**

<p>This is the Server Component</p>

in the **app.modules.ts** register the component

**import** {ServerComponent} **from** "./serverComponent/server.component";

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

Then you can start using it in the **app.component.html**

<app-server></app-server>

**Using CLI command to create component**

**ng generate component <component-name>**

or **ng g c <component-name>**

**Component Inline html template**

Instead of using (referring to) an external template file

@Component({  
 selector: 'app-server',  
 templateUrl: './server.component.html'  
})  
**export class** ServerComponent {  
}

we can also specify an inline html template

@Component({  
 selector: 'app-servers',  
 template: '<app-server></app-server> <br/> inline template <br/> <app-server></app-server>',  
 styleUrls: ['./servers.component.css']  
})  
**export class** ServersComponent **implements** OnInit {  
  
 **...**  
}

For multi-line template content we can use the back-tick

template: `  
 <app-server></app-server> 1  
 <br/> inline template <br/>   
 <app-server></app-server> 2`,

It is mandatory to have either **templateUrl** or **template** property.

**Component Styles**

styleUrls: ['./servers.component.css']

Also we can use inline styling using ‘**styles**’ array-attribute

styles: [`  
 h1 {  
 color: dodgerblue;  
 }  
`]

can’t combine **styleUrls** and **styles**. You can have only one of those

**Component Selectors**

1. **Tag-name**

selector: 'app-servers', // tag-name selector (default)

usage :

<app-servers></app-servers>

1. **Attribute-name**

selector: '[app-servers]', // attribute name selector

usage

<div app-servers ></div>

1. **Css-class-name**

selector: '.app-servers', // css class-name selector

usage

<input type="text" [(ngModel)]="myName">

**Data-Binding**

Refers to binding the html to data (properties or function returns) from the TS.

#### On HTML, properties declared in TS can be printed using **{{ property-name }}**

The property definition in TS

**export class** ServersComponent **implements** OnInit {  
  
 allowNewServer = **false**;

can be shown in the HTML using **Data-biding**

<p>{{ allowNewServer }}</p>

or we could also call a function that returns a value that could be converted to string **{{ getAllowNewServer() }}**

**Property-Binding**

Binding an HTML element’s property to a field from the TS class.

Similar to earlier example, HTML element properties also can be bound as below

<p [innerText]="allowNewServer"></p>

Here the property ‘innerText’ is bound to the allowNewServer field.

We are not allowed to use it as **[innerText] = {{ allowNewServer }}**

**Event Binding**

Binding HTML events to functions from the TS class, like call a function on Click of button.

Say in the TS you have a property and function as below:

serverCreationStatus = 'No Server was created';  
  
onCreateServer(){  
 **this**.serverCreationStatus = 'Server was created !';  
}

Now this can be invoked as below.

<button class="btn btn-primary" (click)="onCreateServer()"  
>Add Server</button>  
  
<p [innerText]="serverCreationStatus"></p>