Angular 5(Formerly Angular 2)

Ronak Nayak

Angular is an open source JavaScript framework which simplifies binding code between JavaScript objects and HTML UI elements.

What is AngularJS?

1. AngularJS is client side JavaScript MVC framework which is used to create reactive Single Page Applications.
   Means it runs on user's browser as it runs on server.

2. In that, the views are specified using HTML + AngularJS's own template language.
   The models and controllers are specified using JavaScript objects and JavaScript functions.

   Means Angular = HTML created for dynamic applications

Why AngularJS?

1. AngularJS takes front-end application and map it with back-end application in very easier and simpler way.

2. In case of JavaScript/jQuery, we need to write bunch of code to update and manipulate DOM in case of event handling, routing and validations and to achieve many other things.

3. It supports two-way data binding i.e. connects your HTML (views) to your JavaScript objects (models) seamlessly. In this way your model will be immediately reflected into your view without the need for any DOM manipulation or event handling (with jQuery)

4. Dependency Injection, means giving a function an object. Rather than creating an object inside a function, we pass it to the function.

Angular 4 Environment set up:

- To install angular cli tool
  npm install -g angular/cli

- To create new project
  ng new PROJECT_NAME

- To build the code & deploy the project
  ng serve –o

- Install bootstrap
  npm install --save bootstrap (Import that into configuration file i.e.angular-cli.json)

- To create new directive
  ng g d DIRECTIVE_NAME --spec false(--spec false prevent to creating test files)

- To create new component
  ng g c COMPONENT_NAME --spec false

- ng build --env=prod

What is Typescript?

1. TypeScript is a superset of JavaScript. That means any valid JavaScript code is valid TypeScript code. But, it provides more features than Javascript like Types, classes, interfaces etc.
2. TypeScript performs strong type checking
3. However, TypeScript does not run in browser. Thus, TypeScript code converted into JavaScript code that browsers can understand.

Angular 4 Create Project:

- Once we create new project using angular-cli,
  i) It would set all default configuration like angular-cli.json, package.json
  ii) e2e folder used for end-to-end testing purpose
  iii) src/app folder contains all the project related files to be touched,
  src/assets folder contains all the static resources used in project like images,
  src/environments folder contains environment variable related configuration
  iv) node_modules folder contains all the third-party libraries that our project is dependent upon

- Few basic things about how angular app works,
  1. Angular trigger in application with **main.ts** and this file bootstrap the main module of our application i.e. AppModule

  2. In AppModule, there is bootstrap: [AppComponent] where we pass modules need at the time application loads first.
  Here, we pass the root component of our application.

  3. Now, the index.html is served by the server and this would be the single page to develop single page application.

  4. This index.html contains root component that CLI provides us i.e. app-root component (This is the selector that helps angular to find out component to load which it has recognized earlier and load that component inside <app-root></app-root>)

- **Few things about configuration:**
  1. If the module is utilizing other modules we define the modules in the **app.module.ts (AppModule)** file -> imports [ ]

  2. By default angular does not scan all the application components. So, to let angular know that which components required we have to register those components in
  app.module.ts (AppModule) file -> declarations [ ]

  3. Bootstrap section in AppModule defines the first component which will run.

  4. **angular-cli.json**: a configuration file for Angular CLI. For example, instead of importing bootstrap.css into our html page we can configure here to use it throughout the application.

- **Components** are key feature of angular. Components are simply TypesScript class. We build our application by composing multiple Components.

  Components will have the binding logic to bind the UI and the model.

  1. AppComponent acts as root component in angular and from which we can make other components

  2. Each component in application will have its own template, style and more importantly also its own business logic (properties and methods).

  Any properties and methods defined here can be accessible from the template and same way events occurs in template can be accessible in component.

  3. It allows split up your complex application into reusable components and we can reuse those components more than once

  4. @Component decorator declared above TypeScript class indicates its component. Within component there is,
  selector: The name of the tag that the component is applied to.
  templateUrl and styleUrls: These defines the HTML templated and stylesheets associated with the component.

  As we define selector in element style (<app-server></app-server>) we can also define our selector as an attribute in template.
  <div app-server>

  5. It is also where dependency injection occurs within constructor, which gives access to various services.

Data binding:

- Data binding means communication between TypeScript code business logic and HTML template.

- In current web application development, we have to write some code to update Model when View changes & need to update View when Model changes.

- To do same binding in JavaScript/jQuery, we need to write logic in both model and view layer. With this AngularJS start to come in picture to bind both the components without additional code.

1. We can achieve data binding from TypeScript -> template (display output data) by means of String Interpolation ({{data}}) or Property Binding ([property] = “data”).

Property binding is the same thing as interpolation, except you can set the properties and attributes of various HTML elements.

We can bind to an **element**, **component** or a **directive** property.

2. We can react to user’s events (template -> TypeScript) using event binding event binding ((event) = “expression”).

3. By passing $event while calling components method we can get details of entered input in component method. This way it bind UI data with the component. This way it provides one-way data binding only.

4. Combination of above both: Two-Way-Binding: [(ngModel)="data"]
[(ngModel)] is a directive which will help us send data from the object to UI and vice versa.
- Round brackets indicate data sent from UI to object.
- Square brackets indicate data is sent from object to UI.
- If both are present then it’s a two way binding.

Note:

To use ngModel, the FormsModule(from @angular/forms) has to be added in imports [] array in the AppModule.

Also, BrowserModule has components by which we can write IF conditions and FOR loop.

Directives:

Directives gives instructions AngularJS to manipulate a piece of the DOM. This could be add a class, hide this, create this to the HTML element etc.

In case of JavaScript/jQuery we have to manually change whatever going on in DOM in the memory representation of the HTML.

Components are such instructions in the DOM. Once we place the selector of the component in template at that point of time we are instructing Angular that to add content of component template and related business logic of typescript in our template.

Component is most commonly used directives and other two are,

Structural Directives: A structure directive basically deals with manipulating the DOM elements. Structural directives have a * sign before the directive. For example, *ngIf and *ngFor.

They affect a whole area in the DOM (elements get added and removed). For example, when *ngIf returns false whole view changed.

Attribute Directives: Attribute directives deal with changing the look and behaviour of the DOM element. We can create our own directives as well. For example, ngStyle, ngClass.

So Attribute directives are like normal HTML attribute with data binding or event binding. They only affect the element they are added to. For example, it changes some element background-colour on some condition.

Using **ngStyle** we can dynamically update the style on element whereas **ngClass** allow us to dynamically add or remove CSS class.

For ngClass directive it works as key-value pair where key is CSS class and value would be the condition determine whether this class to be attached or not.

When we our own directive we have to register it in main module as we have done in case to register new component.

Local reference variables (**#variable-name**):
A template reference variable is often a reference to a DOM element within a template. It can also be a reference to an Angular component or directive.

Use the hash symbol (#) to declare a reference variable on any HTML element. This reference not only contain value of HTML element but all the properties of that element.

We can use the reference variable only in the template where we declare it but not in the Typescript as the name itself says local reference variable.

Communication between Components:

1. Parent to Child: Sharing Data via **Input()**

Inputs allow you to pass data down to a child component where Input properties usually receive data values.

An Input property is a settable property annotated with a @Input decorator. Values flow into the property when it is data bound with a property binding.

We can also bind to a property of a different component. In such bindings, the other component's property is to the left of the (=).

The Angular compiler won't bind to properties of a different component unless they are Input or Output properties.

2. Child to Parent: Sharing Data via **Output()** and **EventEmitter**

Another way to share data is to emit data from the child, which can be listed to the parent. This approach is ideal when you want to share data changes that occur on things like button clicks, form entries, and other user events.

In the child, we declare a variable with the Output decorator and set it equal to a new event emitter. Then we create a function that calls emit on this event with the message we want to send.

Parent have to subscribe to this event in order to receive data triggered from the user events.

Note:
As our application grows and number of components increased, using Input and Output it become difficult to make communication between two components if they are situated far away from each other.
For that it’s good to go to create separate services for such requirement.

ng-content:

Anything we place between opening and closing tag of our own component would simply remove from DOM. Thus, we can't get expected data.

e.g. <app-server *ngFor="let s of servers">
{{ s.serverName }}
</app-server>

We can get the rid of this problem using special directive <ng-content> </ng-content>.
Add this where you want to render the content and what does it do it find the content placed between opening and closing tag of our own component and project into our required component.

Angular Life-cycle:

ngOnChanges()
Called when at first components are loaded in angular and also called after bound input parameter changes.

This event used when we want to react to a change in input and want to do something with previous value.

ngOnInit()
Called once the component is initialized.

ngDoCheck()
Called during every change detection run.

In change detection run angular simply check whether there is any change in component property so that it can make respective changes in template.

It also get executed in user events like button click.

ngAfterContentInit()
Called after content(ng-content) projected into view.

ngAfterContentChecked()
Called every time the projected content has been checked.

ngAfterViewInit()
Called after the component view and child views has been initialized.

ngAfterViewChecked()
Called every time the component view and child views has been checked.

ngOnDestroy()
Called once the component is about to be destroyed and there we can perform some clean-up activity.

HostListener & HostBinding:

@HostListener() Decorator:

In Angular, the @HostListener() function decorator allows you to handle events of the host element in the directive class.

This is a function decorator that accepts an event name as an argument. When that event gets fired on the host element it calls the associated function.

@HostBinding() Decorator:

In Angular, the @HostBinding() function decorator allows you to set the properties of the host element from the directive class.

Let's say you want to change the style properties such as height, width, color, margin, border, etc., or any other internal properties of the host element in the directive class.

By using the @HostListener and @HostBinding decorators we can both listen to output events from our host element and also bind to input properties on our host element as well.

Services and Dependency Injection:

When developing an Angular app, we will most likely run into a scenario in which we need to use the same code across multiple components.

We may even need to make communication between components, or you may need to fetch data from a database of some sort. It's these times when creating an Angular service makes sense.

In fact, components shouldn't fetch or save data directly. They should focus on presenting data and delegate data access to a service.

Dependency injection allows us to inject dependencies in different components across our applications, without needing to know, how those dependencies are created.

When injecting a service (a provider) into your components/services, we specify what provider we need via a type definition in the constructor.

The DI in Angular basically consists of three things:

Injector - The injector object that exposes APIs to us to create instances of dependencies.

Provider - A provider is like a recipe that tells the injector how to create an instance of a dependency.

Dependency - A dependency is the type of which an object should be created.

Hierarchical Injector:

Depends on our requirement we can set where to add providers of the services.

1. AppModule: Same instance of service is available application wide.

2. AppComponent: Same instance of service is available for all components but not for other services.

3. Any other component: Same instance of service is available for the component and all its child components.

If we are using some service inside some other service then we should use @Injectable () annotation.

@Injectable () tells angular that service can be injectable means we should not @Injectable () to the service you want to inject but to the service where you want to inject service.

Routing:

When a user enters a web application or website, routing is their means of navigating throughout the application. To change from one view to another, the user clicks on the available links on a page.

Angular provides a Router to make it easier to define routes for the web applications and to navigate from one view to another view in the application.

Steps to create route:

1. First we need to setup some imports, like so:

import {Routes, RouterModule} from "@angular/router";

2. The mapping of URLs to Components we want displayed on the page is done via something called a Route Configuration, at its core it's just an array.

```
const routes: Routes = [
{ path: '', component: HomeComponent },
{ path: 'search', component: SearchComponent }
];
```

Routes is a typescript type of Route[], an array of individual Route instances.

3. Now we need to register these routes into our application by importing RouterModule.forRoot(routes) into our NgModule.

```
@NgModule({
        imports: [
                RouterModule.forRoot(routes, {useHash: true})
        ]
})
```

4. We need to add a directive called router-outlet somewhere in our template HTML. This directive tells Angular where it should insert each of those components in the route.

> When we click on any link new request is send to server and its refresh the page as some dynamic parameter comes in response. But, that's not desire behaviour as it restarts the app and does not provide good user experience.

For that angular provides routerLink attribute to display new view without getting page refresh.

<a routerLink="/servers">Servers

This is how routerLink works,

- routerLink catches click on the element
- prevent the default which would be send a request
- analyse what we have passed as routerLink value and finds the path from our routing configuration
- load the specified component on given path

> The **RouterLinkActive** directive on each anchor tag helps visually distinguish the anchor for the currently selected "active" route. The router adds the active CSS class to the element when the associated RouterLink becomes active.

The [**routerLinkActiveOptions**] input binding with the { exact: true } expression assures that a given RouterLink will only be active if its URL is an exact match to the current URL.

In Angular we can also programmatically navigate via a Router service we inject into our component, like so:

this.router.navigate(['/', 'red-pill']);

We can also navigate to relative path to current segment as below:

this.router.navigate(['/', 'red-pill'],{relativeTo : this.route});

ActivatedRoute service that is provided to each route component that contains route specific information such as route parameters, static data, resolve data, global query params, and the global fragment.

Also, we need to subscribe to the route parameters if we want to get the changes in current activated route parameters. Same way we can subscribe for query parameters of the route.

By default the query parameters are lost on any subsequent navigation action. To prevent this, you can set queryParamsHandling to either preserve or merge.

We can use merge instead of preserve if you're also passing new query params to the next route.

Route Resolvers:

One way to deal with getting and displaying data from an API is to route a user to a component, and then in that component's ngOnInit call a method in a service to get the necessary data. While getting the data, perhaps the component can show a loading indicator.

There's another way however using what's known as a route resolver, which allows you to get data before navigating to the new route.

For example, in a contacts application where we're able to click on a contact to view a contact's details, the contact data should've been loaded before the component we're routing to is instantiated, otherwise we might end up with a UI that already renders its view and a few moments later, the actual data arrives. Route resolvers allow us to do exactly that.

Routing Flow with Resolver,

1. User clicks the link.

2. Angular executes certain code and returns a value or observable.

3. You can collect the returned value or observable in constructor or in ngOnInit, in class of your component which is about to load.

4. Use the collected the data for your purpose.

5. Now you can load your component.

So basically resolver is that intermediate code, which can be executed when a link has been clicked and before a component is loaded.

Common Errors:

1. **Unhandled promise rejection: 'app-header' is not a known element.**

Solution:
If app-header is an Angular component then we have to register that component in the module. i.e., need to add required component in declarations part of main module (app.module) file.

2. **Cannot read property of undefined**

Solution:
Before using any object initialize it because if we keep it as is it can be anything for that object value.

3. **Can’t have multiple template bindings on one element. Use only one attribute named ‘template’ or prefixed with ***

Solution:
We can’t have more than one structural directive on same element. E.g. can’t add *ngIf and *ngFor together on same element.

4. **No provider for AppService!**

Solution:
Add providers inside Component or Directive decorator that tells the injector how to create an instance of a dependency.

5. **Can't match any routes. URL Segment: 'servers/servers'**

Solution:
If we assign relative path e.g. "servers" then it append our given path to the current path and it be "servers/servers".

So, we need to assign absolute path e.g. "/servers" as it removes path before localhost:8000 i.e. currently loaded segment and append "/servers" there.

But, that is also depends on from which path we access other path so no need to have absolute path for links defined in our navigation area because it's our root component and it would always be hosted on localhost:8000/