ANGULAR

Introduction to Angular :

Angular is a development platform, built on TypeScript. As a platform, Angular includes:

- A component-based framework for building scalable web applications
- A collection of well-integrated libraries that cover a wide variety of features, including routing, forms management, client-server communication, and more
- A suite of developer tools to help you develop, build, test, and update your code

Basically, the fundamental concept of Angular is to create an SPA(Single page application). Angular makes that by modularizing the code blocks such that modifying these modules effects the output and is also easier by reducing the work load.

The Basic concepts of ANGULAR is divided simply as,

- Components
- Modules(or NgModules)
- Directives
- Services and Dependency Injection(DI)

Decorators :

The **Decorators** are what that provides the information(i.e, meta data) to Angular the this is a component or an injectable or a module.

Examples: @Component(), @Injectables, @Input etc

Without the decorator Angular App cannot understand what to with the data and it will be just a regular Class.

NgModules:

The root module is the starting point of the application. It provides the whole content and directions to the application.Every file you use in the app are located here as long as you use them.

Example:

@NgModule(){

declaration: [components];

imports: [Modules];

providers:[Services etc];

bootstrap:[RootComponent]

}

Components

Components are the building blocks that compose an application.

A component includes a TypeScript class with

- a @Component() decorator,
- a HTML template, and
- Styles
- A html selector

The LifeCycle hooks like onInit() etc helps in loading content based in component actions.

The @Component() decorator specifies the following Angular-specific information:

- A CSS selector that defines how the component is used in a template. HTML elements in your template that match this selector become instances of the component.
- An HTML template that instructs Angular how to render the component.

Ex: -> The following is a minimal Angular component :

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

@Component({

selector: 'hello-world',

template: ` <h2>Hello World</h2> <p>This is my first component!</p> `,

})
```

```
export class HelloWorldComponent {
// The code in this class drives the component's behavior.
}
```

-> To use this component, you write the following in a template:

Syntax : <hello-world></hello-world>

-> When Angular renders this component, the resulting DOM looks like this:

```
<hello-world>
   <h2>Hello World</h2>
   <p>This is my first component!</p>
</hello-world>
```

Templates

Every component has an HTML template that declares how that component renders. You define this template either inline or by file path.

Most of the web-application working exists in these templates.

Interpolation, Property Binding, Event Binding, Two Way Binding, Pipes.

Pipes : Pipes in Angular help in providing better user experience using inbuilt function of variable types.

Ex: {{ “ravi” | upperCase }}   Output: RAVI

-> One application of this feature is inserting dynamic text, as shown in the following example :

```
<p>{{ message }}</p>
```

-> The value for message comes from the component class:

```
import { Component } from '@angular/core';
@Component ({
  selector: 'hello-world-interpolation',
  templateUrl: './hello-world-interpolation.component.html'
})
```

```
export class HelloWorldInterpolationComponent {
  message = 'Hello, World!';
}
```

-> When the application loads the component and its template, the user sees the following:

```
<p>Hello, World!</p>
```

-> Angular also supports property bindings, to help you set values for properties and attributes of HTML elements and pass values to your application's presentation logic.

```
<p [id]="sayHelloId" [style.color]="fontColor">You can set my color in the component!</p>
```

-> You can also declare event listeners to listen for and respond to user actions such as keystrokes, mouse movements, clicks, and touches. You declare an event listener by specifying the event name in parentheses:

```
<button (click)="sayMessage()" [disabled]="canClick">Trigger alert message</button>
```

Directives :

Directives are classes that add additional behavior to elements in your Angular applications. With Angular's built-in directives, you can manage forms, lists, styles, and what users see.

The different types of Angular directives are as follows:

1. Components—directives with a template. This type of directive is the most common directive type.
2. Attribute Directives—directives that change the appearance or behavior of an element, component, or another directive.
3. Structural Directives—directives that change the DOM layout by adding and removing DOM elements.

Built-In Attribute Directives :

Attribute directives listen to and modify the behavior of other HTML elements, attributes, properties, and components.

Many NgModules such as the RouterModule and the FormsModule define their own attribute directives. The most common attribute directives are as follows:

- NgClass—adds and removes a set of CSS classes.
- NgStyle—adds and removes a set of HTML styles.
- NgModel—adds two-way data binding to an HTML form element.

Built-In Structural Directives:

Structural directives are responsible for HTML layout. They shape or reshape the DOM's structure, typically by adding, removing, and manipulating the host elements to which they are attached.

This section introduces the most common built-in structural directives:

- NgIf—conditionally creates or disposes of subviews from the template.
- NgFor—repeat a node for each item in a list.
- NgSwitch—a set of directives that switch among alternative views.

Services :

The services are typescript classes defines by the Decorator Injectable() which provided the content in it to the required components based on the hierarchy of angular component.

Services can fetch and load data independent of components allowing us to modularize the code and change the fetch operations independent of the the component.

Dependency Injection :

Dependency injection, or DI, is a design pattern in which a class requests dependencies from external sources rather than creating them.

Dependencies help us providing the services to the components that subscribe to them and not to those that do not.

These service injections are mostly used to fetch data and also providing a common point to access common functions (such as alerts, logs, database services)

HTTP Services:

A lot of such libraries available in Angular that provides us the DataBase connections to fetch and post data.

Angular CLI :

The Angular CLI is the fastest, easiest, and recommended way to develop Angular applications. The Angular CLI makes a number of tasks easy. Here are some examples:

ng build - Compiles an Angular app into an output directory.

ng serve - Builds and serves your application, rebuilding on file changes.

ng generate - Generates or modifies files based on a schematic

ng test - Runs unit tests on a given projecting

ng e2e - Builds and serves an Angular application, then runs end-to-end tests.

Angular is a vast concept and contains a lot of developer options with a rare need of external inputs. This component based model needs a lot more to be explained but most of the concepts are sub-branches of the above mentioned. With constant upgrades to the software it is current Angular version is 9.0.0.

- L. N. S. S. Ravi Teja