Angular

1. Angular Module
   1. Container for
      1. Components
      2. Services
      3. Pipes
      4. Directive
2. Decorator
   1. An attribute applied on Class/Method/Member to define behavior of it
   2. @NgModule
      1. Class is angular module
   3. @Component
      1. Class is component
   4. @Injectable
      1. Class will be used for DI
      2. Applied on class to be used as Angular Service
   5. @Input
      1. Applied on Member (public variable / property) to accept data from parent component
   6. @Output
      1. Applied on EventEmitter to emit event from Child component to parent component
   7. @ViewChild
      1. Applied on Component Instance to make component as DOM Child of Parent component
   8. @HostListener
      1. Applied on method with Event so that when an event occur the method will be auto-invoked
3. @angular/core
   1. Package that contains Object model for Angular Programming
      1. @NgModule, @Component, @Input, @Output, etc
4. @angular/compiler
   1. Ahead-of-Time (AoT) compilation service
      1. Optimized Build of the code
5. @angular/Common
   1. Common Module-loader for external modules
   2. @angular/common/http
      1. HttpClientModule Angular 4.0+
6. @angular/forms
   1. Form Object
      1. FormsModule
         1. Used for ngModel
      2. ReactiveFormsModule
         1. Data-Driven / Model-driven / Reactive Forms
            1. Model validation forms
7. @angular/router
   1. SPA
8. @angular/platform-browser
   1. Provides BrowserModule that loads Angular OM in browser
9. @angular/platform-browser-dynamic
   1. Dynamically executes Angular OM for
      1. Databinding
      2. Event-Binding
      3. Directives

**Angular Support Packages OR external dependent Libs**

1. Rxjs
   1. Reactive Extension for JavaScript, supported by DevLabs that is inspired from Rx.NET
   2. An Observable class, maintain state of An Async Operations
2. Core-js.js
   1. ES 6 compatibility to all browsers those who does not support ES 6 and only uses ES 3
      1. Class
         1. Access Specifier
            1. Public (Default)
            2. Private
            3. Protected
         2. Access Modifiers
            1. Static
            2. Async 🡪 await
      2. Datatypes (only in TypeScript)
      3. Arrow Operation/ Lambda =>
      4. Promises
      5. Template String
         1. Build an Immutable Html parsed string
            1. `${<Variable>}`
      6. Modules
         1. Export
         2. Import
      7. Transpiler
         1. Transform ES 6 to ES 6
         2. ES 6 Engines
            1. TypeScript

Microsoft

Npm install -g typescript

The **tsc** transpiler

* + - * 1. ES2015

High-Level JavaScript

Bable

Used by React

Used Angular

Tracure

* + - * 1. Dart

1. Zone.js
   1. Library used by platform-browser-dynamic for
      1. Browser level Error Logs and StackTrace

Package.json

* Application configuration file that contains
  + Runtime and Development time dependencies
  + ng build –aot
  + ng build –prod
  + ng serve –prod

Programming with Angular

1. Create a Component
   1. Class decorated with @Component() decorator
      1. Component class Name should be
         1. XXXXComponent
            1. E.g. SimpleComponent
      2. Component File Name
         1. app.XXXXX.component.ts
            1. E.g. app.simple.component.ts
   2. @Component properties
      1. The **selector** property
         1. This is a custom HTML tag to load component on HTML Page
         2. Naming Comventions
            1. app-<COMPONENT-Name>-component
      2. The **template** property
         1. Contains inline HTML Template means write HTML inside component’s file
      3. The **templateUrl** property
         1. Contains an Absolute path for external Html file
      4. The **style** and **styleUrls** properties
         1. Contains an inline and external CSS respectively
      5. The **provider** property
         1. Contains DI for Angular Service specific to component
2. Create a Angular Module
   1. Class decorated with @NgModule
   2. Properties for NgModule
      1. The **imports** an array type property, used to import all standard Angular Module those are needed for the current application
      2. The **exports** an array type property, used to export components, services, directives, ect from current application to other angular application
      3. The **declarations** an array type property, used to declare components and directives and piped of current application in module for **lazy loading.**
      4. The **providers** an array type property, used to declare Services in DI for current application.
      5. The **bootstrap** an array type property, used to define component(s) to be executed when the current application is loaded in browser. Note these components must be already declared in **declarations.**
3. General Folder Structure with Angular CLI
   1. Src🡪app🡪components🡪Folder-for-each-component
      * 1. 🡪services🡪app.XXXX.service.ts
        2. 🡪directives🡪app.XXXX.directive.ts
        3. 🡪pipes🡪 app.XXXX.pipe.ts
        4. 🡪models🡪 app.XXXX.model.ts

Databinding

* + - 1. Angular Interpolation
         1. Bind public member of Component class to UI using {{}}.
         2. Read-only binding
         3. Value can only be changed from component
      2. In Two-Way Binding, we use ngModel, and we need to import FormsModule from @angular/forms in NgModule.

CSS

TextBox 🡪 form-control

Top level Div 🡪 container

Inner DIV 🡪 form-group

Angular Reactive Forms

1. Form Object with HTML 5 validations
   1. The HTML 5 Compatible Browsers
2. Model Validation (IMP)
   1. ngForm 🡪 linked with <form> tag
   2. (ngSubmit) default event for form
   3. FormGroup
      1. Object that groups FormControlCollection
         1. Each object in FormControlCollection is FormControl
         2. FormControl 🡪 is an Editable element under <form> e.g. text, select, check, radio, etc.
         3. FormControl has constructor with 2 parameters
            1. First 🡪 The Model class property to be bind with UI element with attribute as **formControlName**
            2. Second 🡪 The Validation Rule

The **Validators** class with static methods as

Required()

MinLength()

MaxLength()

Pattern()

1. The Angular Component implements the **OnInit** interface’
   1. Lifecycle Event Interface.
   2. Provide **ngOnInit()** method, this is invoked immediately after constructor

**Angular Directives**

1. Objects those are used for
   1. Behavior of HTML Element
      1. Attribute Directives
         1. Create these for defining new attribute for DOM element to assign custom behavior for it.
         2. E.g.
            1. Valid
            2. Invalid
            3. Ng-Class
   2. New UI with Behavior and Functionality
      1. Component Directive
   3. Dynamic Generation of UI
      1. Structural Directives
         1. Dynamically generate DOM elements (Add/Remove)
         2. E.g.
            1. \*ngFor
            2. \*ngIf
            3. \*ngSwitch/ngSwitchCase
2. ES 6 Object Creation using
   1. Object.create()
      1. Create a new Blank Object from Source Object
      2. Var a = Object.create(b,{});
         1. Object a will be created as new schema of object b.
         2. Shallow Copy
   2. Object.assign()
      1. Create a new Object having same schema and value but on different ref. location
      2. Var a = Object.assign(b,{});
         1. Object a will be created at new ref. location having same schema and values from object b.
         2. Deep Copy
3. UI Element Validations based on HTML 5 Attributes and Angular Forms
   1. Step 1: Import ReactiveFormsModule in NgModule from @angular/forms
   2. Step 2: In Component import FormGroup, FormControl and Create an instance of FormGroup using FormControl and link FormControl with Model properties.
   3. Step 3: In the Html set the **formGroup** attribute for the <form> tag to the formGroup declared in component class.
   4. Step 3: In the Html set the **formControlName** attribute for each editable element to the Key of FormGoup object that is mapped/linked to the Model property using **FormControl** class
   5. Step 4: Remove the ngModel for elements those are applied with **formControlName**
   6. Step 5: Use the **value** property of FormGroup to read the submitted values
4. Implementating validations on FormControl Name
   1. <FormGroup>.controls.<formControlName>.dirty
      1. Listen to changes in FormControl
   2. <FormGroup>.controls.<formControlName>.valid/invalid
      1. Check the state of FormControl based on applied rules
   3. <FormGroup>.controls.<formControlName>.erros.<Rule>
      1. Rule 🡪 pattern/required/minLength/maxlength
   4. \*ngIf=”<FormGroup>.controls.<formControlName>.dirty

&&!<FormGroup>.controls.<formControlName>.valid”

Execute the Error Message

1. <FormGroup>.controls.<formControlName>.erros.<Rule>

Using Model Validators for the Angular Reactive Forms

1. Import Validators in Component class from @angular/forms’
2. Use Validators methods in FormGroup as a second parameter to FormControl() constructor

Angular Services

1. The class decorated with @Injectable() decorator, that has JSON object and its property ‘providedIn’ with value as ‘root’. This means that the service will be auto-injected in the DI container of the NgModule of current application. Angular 6.0+
2. Generally, Services are used for Http Calls
   1. Angular 4.0+, @angular/common/http package with following classes
      1. HttpClientModule
         1. The class used for providing Platform aka container for Http calls aka Ajax calls
      2. HttpClient
         1. Class that contains Http Methods
            1. get<T> 🡪 returns Observable<T>
            2. post<T> 🡪 returns Observable<T>
            3. put<T> 🡪 returns Observable<T>
            4. delete<T> 🡪 returns Observable<T>
         2. Where T can be Primitive Types e.g. number, string, etc. Can also be Complex Type
         3. Observable is imported from rxjs to maintain state of the external Http Calls.
   2. The get<T> (url, headerOptions?)
      1. headerOptions is of the type HttpHeaders class
   3. The post<T>(url, body, headerOptions)
   4. The put<T>(url, body, headerOptions)
   5. The delete<T>(url, headerOptions)

Communication Across Components

1. Case 1: Components has knowledge of each-other so that there is logical parent-child relationship established between them
   1. The child component is rendered in the context of parent component.
   2. If parent wish to send any data to child component using any type binding e.g. property binding, then the child must expose a property decorated with @Input decorator
      1. The Input decorator for Component Communication must be applied on get/set property on setter method
      2. When Input decorated property is updated in Component, the component updated itself and it updates all other properties those are dependent on property decorated using Input.
   3. In the parent component set the PropertyBinding for the child component using @Input Decorated Property
   4. Child can emit data to parent using @Output Decorator, and its uses an EventEmitter<T> object.
      1. The EventEmitter<T> is the object to emit data of the type T. Here T is called as PayLoad
      2. When Event is emitted by child, the parent must subscribe to it.
         1. The parent, read the payload data from child using the standard **$event** object
         2. To subscribe to emitted event, the parent must use the EventBinding