**Angular 2 Notes**

TO start the application -- npm start

ctrl c to stop

ng g c componentname -- command to create a new component using cli

ng g s servicename – command to create a service

ng g d directive name – create custom directive

interpolation -- string interpolation{{string}}

dependency injection

injecting or providing the dependencies of class into its constructor.

Property binding: [property name] = property value; ex <img [src] =<http://img.png> /> -- One way binding(component to view)

Attribute binding [attr.property name]

Event bubbling – event bubbles up in a dom tree.

Event binding -- (keyup.enter) = “methodname($event)” or you can write an expression

Template variables: #tempVariable will reference the element: used to simplify code

ngModel – Two way binding ex [(ngModel)] = propertyName -- ngModel imported from forms module

pipes : currency pipe, date pipe, decimal pipe used to format data.

Custom pipes:

Import Pipe,PipeTransform from core, then in transform method add your logic and return the output

@Input and @Output Properties

@Input() isSelected: boolean;

Input properties to pass input or state to a component, pass data from parent to child

Output properties to raise events from these components. Following are the steps to raise a custom event from component:

1. Add (change) event to html and in the component add the event that u want to be called once the change is done
2. Declare output property in the component where u have declared input output properties and initialize it with eventemitter.

@Output() change= new EventEmitter();

1. In Onclick event raise an event this.change.emit()

View Encapsulation: shadow dom

Ng-content – to replace with custom content in html. Provide custom content to reusable controller

Ng-container – render something without putting it under div

**Directives**

\* marks the structural directive – to modify structure of DOM

\* tells that angular converts the statement into ng-template and manually adds conditions.

\*ngIf – removes the element from dom if cond not satisfied

From ang4 <ng-template> introduced so that you can write ngIf condition based.

<ng-template #coursesList>

\*ngIf =” condition>0; then coursesList else course”

[hidden] – hides the element in dom should be used in case of small dom tree or if building the tree is costly affair

[ngSwitch] = bind value

\*ngSwitchCase – structural directive -- changes structure of dom

for of -- iterates over iterable

for in -- iterates over properties of object

\*ngFor=”let course of courses; trackBy:trackCourse”

trackBy – is used when you don’t want the dom to rerender every time on button click. It will track by whatever id returned from trackCourse method.

Safe Traversal object : task.assignee?.name – if assignee is not null then name otherwise null

**Custom Directives**

HostListner: listen to the events raised from the dom elements like onBlur or onFocus

Elementref add in constructor – to access the property of dom element

Pass data to a directive using input properties

Selector of the directive can also be used to pass data by making it as an attribute[] and pass data from it

ngOnchanges

passing data to a nested component thru @input parameter

u can declare a property of nested components (kind of directives) as [property name]

-------------------------------

**Template driven forms** – create directives, good for simple forms, easy to create

**Reactive forms** – create separate controls, good for complex forms and unit testable

Template driven: ngModel need to declare name attrib f or input elements

Validations: firstName.errors.required

FormControl : ngModel and formGroup : ngModelGroup

ngForm used for submit buton

[ngValue] can be used to bind complex object

**Reactive forms**

Abstract control is the base class for FormControl and FormControlGroup

formControlName and formGroup are directives used to bind the reactive form

use validators.required in component to do validation as a property of FormControl

----------------------------------------------------------------------------------------------------------------

**Services in Angular:**

Get,put,post,delete,**patch** 🡪 all return observables which we need to subscribe.

subscribing to observable

Patch use only if modifying few properties

Splice() method used to insert at starting of an array

---------------------------------------------------------------------------

**Lifecyclehooks**

Creates a component

Renders it

Creates and render its child component

Destroys component

Lifecyclehoook events: OnInit, OnChanges, DoCheck, AfterContentInit ……

Expected error and unexpected error

Observable.throw 🡪 factory methods in rxjs

Catch 🡪 operator methods in rxjs

To handle error at the global level create a class which implements ErrorHandler class from angular core and in providers add an object to use custom error handler in place of default error handler

{provide:ErrorHandler, useClass:AppErrorHandler}

Instead of calling response object and then calling the json method get array of objects…we use observable operator called map, map can transform items into observables.

observable map operator Applies a defined function to each item emitted by an Observable

Written in component

(response => {

this.posts1 = response.json();

});

Replaced by

.map(response=>response.json()); //written in service

(conversion to json done in service itself using map)

Optimistic update vs Pessimistic update

Generally, we do pessimistic updates where we wait for server to give response and then update the UI. In Optimistic approach we first update the UI then after getting response if there is an error we rollback our changes.

**Observables vs Promises**

Observables will not work until you put subscribe

Observables are lazy promises are eager

Observables provide more operators than we get in case of promises.

Promises: Represent a single value in future

Observables: Represent 0 or more values in future or now

Promises are asynch vs observables which can be either asynch or synch

Observables can be converted into promise

**Routing**

Configure the routes in app.module.ts

RouterModule.forRoot([

{

path:'posts',

component:HomeComponent

},

Add router-outlet element in html (routing will happen after this element)

Then add routerLink directive in the navigation page(routerLink in place of href as it supports spa)

<a routerLink=”/followers”> -- simple navigation

<a [routerLink]= “[‘followers’,follower.id]”> -- when you want to bind an argument(route params) along with route use property binding syntax – first element is path and second element is route argument

To catch the route param in components subscribe using this.route.paramMap

Observable is a collection of asynch data that arrives over time

Data comes in stream and anyone who is subscribed is notified

Anytime new data in param map the component which is subscribed will be notified

this.route.snapshot.paramMap.get("id"); -- if you don’t want to subscribe, user doesn’t have to navigate back to same component.

queryParamMap – to add optional params to route

**programmatic router navigation** is done using

thi.router.navigate([‘/users/login’]);

submit(){

this.router.navigate(['/followers'],{

queryParams:{page:1,order:'newest'}

});

}

---------------------------------------------------------------------------------------------------------------

Main.ts Loaded by webpack which loads app module, then index.html which loads app component

Angular does css encapsulation by applying different attributes to each elements during runtime(different for parent and different for child comp)

Route Activation and DeActivation(Route guard)

canActivate used during route mapping(as a controller) if you want to open error page in case of wrong url.

canDeactivate used during routing (as a function), in case of navigating away from page if you want deactivate movements. (Do you want to save changes before moving away?)

**Resolve**: used when we want to fetch data first then load the ui(asynch)

The Angular 6 Router provides a resolve property that takes a route resolver and allows your application to fetch data before navigating to the route (i.e resolving route data).

<https://www.techiediaries.com/angular-router-resolve/>

{ path: 'events', component: EventsListComponent

, resolve: {events:EventListResolver} },

Above means, before resolving this route call the resolver service, once data is returned from resolver, add it as a property to route and same will be accessible from the component using activated routes property snapshot(this.events = this.route.snapshot.data['events'];)

**Resolve(){ }: automatically subscribes to observable, no need to call subscribe.**

routerLinkActive to activate the route or the menu buttons

**Feature Components**

It will have its own module which has following features:

1. Rather than BrowserModule it will call CommonModule

2. Also it will have RouterModule.forChild rather than forRoot

Add following entry in root route config file. load usermodule from following path

{path: 'user', loadChildren: './user/user.module#UserModule'}

This is also called **lazy loading** feature modules as none of these module’s and components are loaded with root modules. Loads the pages on call basis.

**Barrels** are used to simplify the imports in page, create an index file and add all imports in it ad then use this index file’s reference wherever required

Using ng-content directive we can reuse a component – content projection

If we add the tag <ng-content></ng-content> anywhere in our template HTML for our component. The inner content of the tags that define our component are then projected into this space.

**HTTPClient** for get put post and delete

getEvents():Observable<IEvent[]> {

return this.http.**get**<IEvent[]>('/api/events') }

saveEvent(event) {

let options = { headers: new HttpHeaders({'Content-Type': 'application/json'})};

return this.http.**post**<IEvent>('/api/events', event, options) }

put and post are same

Authentication can be persisted in app.component.ts’s OnInit()

currentUser = undefined --- log out

**Linting**: Pointing out and fixing potential problems (coding style changes) like missing semicolon, missing quotes. Use TsLint

**Producation Optimization:**

Minification

Bundling

Tree Shaking (identify Methods which are unused and not called anywhere)

DeadCode elimination

**AOT Compilation:** process that runs during angular build to compile templates and convert them to javascript, also helps in faster rendering, smaller angular framework to download, detect template errors and better security.

Preloading strategy in angular

**Unit testing in angular**

**Run npm test**

Jasmine is the most popular JavaScript testing framework (**Behavior Driven Development**)

|  |  |
| --- | --- |
| describe | This method is used to represent a group with related test blocks.  This method needs to execute with two arguments –   * Test name * A function |
| beforeEach | This method is fired before each test block. |
| it | This method executes a function to perform a test operation. |
| expect | This method evaluates the result from the test block and performs the asserts statements. |
| toEqual | This method is used to compare the expected result and the actual result. |
| beforeAll | This method is executed only once in the test block to provide the description of the test suites. |

Karma is basically a tool which lets us spawn browsers and run Jasmine tests inside of them, which are executed from the command line. This results of the tests are also displayed in the command line. Karma also watches our development file changes and re-runs the tests automatically.

1. Create spec.ts file for the service going to be tested.
2. Create a mock http service object using jasmine.createSpyOnj and also create service object in before each so that every time new instance is created
3. In describe method execute the delete and then expect and check the response

<http://thisdavej.com/how-to-watch-for-files-changes-in-node-js/>

<https://www.techiediaries.com/angular/>

to upgrade angular library versions:

<https://update.angular.io/>

<https://github.com/dotnetcurry>

------------------------------------------------------------------------------------------------

**Few Notes:**

2 to 4

if...else syntax in component HTML templates

Stand alone animation module

TypeScript's StrictNullChecks compliancy

Angular Universal adoption by team to live in core

Performance boost with FESM

**Wats new in Ang 5**

Build Optimizer -- makes application pure removes unnecessary add ons

-- remove angular decorators (only used by compilers)

Angular Universal State Transfer API -- easily share application state

between the server side and client side

Compiler Improvements

-- Faster rebuilds, fast AOT compilation

--Preserve Whitespace

--Improved Decorator Support

HttpClient in place of HttpLibrary

https://medium.com/@chriscordle/why-angular-2-4-is-too-little-too-late-ea86d7fa0bae

Wats new in ang 6

ng update

help you adopt the right version of dependencies,

and keep your dependencies in sync

ng add<package>

makes adding new capabilities to your project easy

Angular Elements

CLI Workspaces

RxJS v6

**ElementRef** : Provides access to the underlying native element (DOM element).

**Renderer2** : service that allows to manipulate elements of your app without

having to touch the DOM directly. Used mostly in custo directives.

**main.ts is the starting point of angular**

**pollyfill.ts imports javascripts**

**hot module reloading -- modify something in code and**

**it will be compiled automatically using webpack**

safenavigation operator ?

to make a copy of a js object completely use splice(0)

impure pipes

ngsubmit will hit submit function on any click

-----------------------------------------------ang

https://angular.io/guide/ajs-quick-reference

https://www.youtube.com/channel/UCdCOpvRk1lsBk26ePGDPLpQ/videos?view=0&flow=grid&sort=p

-----------------------------------------------------------