Angular Notes

# Basic Instructions & Information

## Console Commands

npm install -g @angular/cli@latest *// install angular globally; @latest is optional as it*

ng version *// check version of angular*

ng new *app-name // Create new Angular app, do not use “test” as a name*

*// This will create a folder “app-name” in the current dir*

ng new app-name –no-strict *// --no-strict to not use strict mode*

ng serve *// cd into app-name folder and run command*

*// starts dev server on port localhost:4200*

*// stays running until stopped (ctrl+C)*

## Basic Angular notes

{{*variable-name*}} *// using a variable in html file*

## Installing Bootstrap Via NPM

**Installing Bootstrap**

npm install --save bootstrap@3

*This installs latest ver3 of bootstrap; not having the 3 installs latest version*

Open **angular.json** file and add into “projectname/architect/build/options/styles”

Add above styles.css; “node\_modules/bootstrap/dist/css/bootstrap.css”,

Rerun **ng serve** to reload

Check dev console under Sources/style.css

If working, it should mention bootstrap in the output

## App Componenets

Initial/main componenet app of all angular files

**App.componenet.css** – css used by app.component.ts

**App.componenet.html** – template file used by app.component.ts

**App.componenet.ts** – main componenet.ts file, contains app-root selector which gets what’s in index.html

**App.module.ts** – module file that contains all componenets and calls them to be used.

All componenets and modules need to be imported and declared in this file

# Modules

Modules are packages Typescript needs to have declared for each component where certain functions are used

## Forms

Forms module is needed when working with forms and outputting it back into the DOM

EX: Create an input form, then output what user types back out into the DOM

In \*.modules.ts:

import {FormsModule} from '@angular/forms';

Under imports add: FormsModule

# Components

Components are custom “elements” that that have specific functions you create

These are usually created inside the src/app folder

## Creating new componenets manually

Create a folder using the name of the componenet (no spaces)

1. Create 4 fles inside this new folder *[name.components.ts/.css/.html/.spec.ts]*
2. Edit the app.modules.ts file (see below)
3. Edit the \*.components.ts file (see below)

### Adding to app.modules.ts

New components needs to be added to **app.modules.ts** in order for it to be used

Inside **app-module.ts** add the following:

Add new componenent into top of so it can be used

import { NameComponent } from ‘./name-component/name-componenent.component’; *// Do not add .ts at the end*

Add to **@NgModule** declarations after **AppComponenet**

declarations: [ AppComponenet, … , NameComponent ],

### Editing the \*.componenets.ts file

The \***.componenets.ts** file contains the selector and template

import {Component, OnInit} from ‘@angular/core’; *// This needs to be at the top of the file to make it work*

*// OnInit used to come after Componenet and made if made thru* ***ng g c*** *command*

@Component({

Selector: ‘app-name’, *// When creating selector names, try to use app-name as to not override any default html tags (ex: app-my-comp)*

TemplateUrl: ‘./app-name.html’, *// Templates can be directed to another file all together,*

*// or written inline enclosed in ` (Inline ex: template: `<p>Testing</p>`,)*

styleUrls: [‘./app-name.css’, ‘additional files if needed’] *// Inline example: styles:[` h3: {…}`]*

})

Export class NameComponent implements OnInit { *// This is where the main functions are created*

Constructor() { } *// OnInit is created if using automated* ***ng g c*** *command*

ngOnInit(): void { }

}

Inline html/styling in @Component.ts

**Use inline html** instead of linking to a .html file

Instead of *templateUrl:* use *template: “insert html”*

For multi inline html use `*insert multiline html*`

**Inline styling** instead of linking to css file using *styleUrls:* use *styles:[` h3: {…}`]*

### Selector Changing

You can change a custom component html tag to being an attribute or class

In \*.components.ts file, change the selector to the following

selector: ‘app-name’ *// default, html tag <app-name>*

selector: ‘[app-name]’ *// enclosing it in brackets [] makes it* ***an******attribute*** *<div app-name>*

selector: ‘.app-name’ *// adding a dot before the name changes it to* ***a class*** *<div class=”app-name”>*

Constructors are functions that are automatically ran upon instanciation of component.

## Create new components via CLI

Run the following console command: ng generate component *name* or ng g c *name*

This can be ran anywhere in the project folders (not need to be in root or app folder

Creates a *name* folder in src/app folder and creates necessary files for new component

*[name.components.ts/.css/.html/.spec.ts]*

*This also adds necessary syntax into* ***app.modules.ts*** *file*

*@ngModule({ declarations: [ NewComponent,….],…}*

### Create component without spec.ts file

ng g c componentName **--skipTests true**

this skips creating the spec.ts file when using ng g c

### Create component in sub-folder

ng g c **sub-folder/*name***

# Data Binding

Communication between typescript and html

## **String Interpolation** {{*name*}}

Outputting variables using **\*.component.ts** files

Create a variables inside export class *ComponenetName*

serverID: number = 10;

serverStatus: string = ‘offline’;

getServerStatus() {

return this.serverStatus;

}

Inside the **\*component.html** file, use the variables

*{{ "Server" }} with ID {{ serverID }} is {{ getServerStatus() }}*

## **Property Binding** [property] = “data”

You can set the property to logic.

In this example you can set the button disabled if a variable is set to an empty string.

*<button class="btn btn-primary"* ***[disabled]="variableName === ‘f’ "****>Add Server</button>*

You can also setting an existing html property to bind to an instruction in the \*components.ts file

Example below shows binding the disabled property of a button to dynamically set to t/f scenario.

*<button class="btn btn-primary"* ***[disabled]="!allowNewServer"****>Add Server</button>*

Note disabled property is set to a variable in matching component.ts file

***allowNewServer*** *= false;*

*constructor() {*

*setTimeout(() => {*

*this.****allowNewServer*** *= true;*

*}, 2000);*

*}*

Button is initially set to disabled, but after 2 sec, it is set to enabled due to constructor

In a similar example you can set innertText to output to a p tag using property binding:

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

In this example **innerText** outputs the text of **allowNewServer** var equates to.

Similarly it does the same as string interpolation using the following:

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

## **Event Binding** (event) = “expression”

Similar to event binding you can set “EXPRESSION” to be an inline logic statement

In the example below, user is a variable that is being set to equal an empty string

*<button*

*class="btn btn-primary"*

*[disabled]="!allowNewServer"*

***(click)="username = ‘’"***

*>  Add Server*

*</button>*

Binding a component html event to a method in \*.component.ts file.

In this example, the button listens for an event (click), then the expression is ran

*<button*

*class="btn btn-primary"*

*[disabled]="!allowNewServer"*

***(click)="onCreateServer()"***

*>  Add Server*

*</button>*

Listening on click of button. When click, run function onCreateServer()

Example of Input element:

*<input*

*type="text"*

*class="form-control"*

***(input)="onUpdateServerName($event)"***

*placeholder="start typing"*

*/>*

The data in the input form is getting stored into **OnUpdateServerName($event)**

Note that $event is a reserved variable

In the component.ts file contains the following

***serverName*** *= '';*

*onUpdateServerName(event: Event) {*

*this.****serverName*** *=* ***(<HTMLInputElement>event.target).value****;*

*}*

The input gets stored into **serverName** variable which can be output back into the DOM via string interpolation

**Note: *(<HTMLInputElement>event.target)*** is used specifically to denote when an ***input*** element is used

## **Two-Way-Binding** [(ngModel)] = “data”

Variable data stored to its \*.componenet.ts file is output into the \*.componenet.html file. You can also output the variable using {{ var }}

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

In .components.ts file, set variable “name” to any string or empty string to be stored

Note: .modules.ts file needs the following

***import { FormsModule } from ‘@angular/forms’;***

*@ngModule ({*

*Imports: [ FormsModule ],*

*})*

To output the variable using {{ name }}

FormsModule is automatically added in if use ng g c

Similar to previous but now data is set to variable straight from html to ts file without function

<input

  type="text"

  class="form-control"

**[(ngModel)]="serverName2"**

  placeholder="start typing"

/>

<p>Server Name: {{ **serverName2** }}</p>

In the component.ts file will have **serverName2** equal a string

**serverName2** = 'Another Server';

Output: **Server Name: Another Server**

Once user types inside input box, it will replace **Another Server**

# Directives

Instructions in the html/DOM.

Ex: <p turnRed>This paragraph will turn red</p>

*In the \*component.ts file lies the instructions to turn this red where selector is [turnRed]*

Components are also certain Directives

## Structural Directives - \*ngIf; else

create a condition inside the html. Condition can only be true/false result only.

<p \*ngIf="serverCreated; else **noServer**">

  Server was created, server name is {{ serverName }}

</p>

serverCreated has to be created in components.ts file

<ng-template #**noServer**>

  <p>No Server was created</p>

</ng-template>

ng-template has to be used to create else condition

Note that **noServer** does not have to be created inside component.ts file.

## Attribute Directives – don’t add or remove elements, only change it

[ngStyle] – directive used to dynamically update styles

Following calls method to change the bg color

<p [ngStyle]="{ backgroundColor: getColor() }">Change bg color</p>

[ngClass] – dynamically change a class

[ngClass]="{ 'online': serverStatus === 'online' }"

'online' doesn’t need to be in quotes but if you have a class using -, then use quotes

**It only adds class if it’s true**

[ngFor] – using a for loop in the selector/tag property

<app-server \*ngFor="let server of servers"></app-server>

Creates a for-of loop - for (element of array)

server is the individual element and servers is an array set up in the .components.ts file

# Models

Filename.model.ts is the standard file naming protocol

A .ts file that is a blueprint/template for objects; it’s also treated like a custom type like string, or number

This can create a file type such as a specific array

In the .model, create a custom type of array

In another component, estanciate an array based off the the one in .model

**Remember:** import the .model into this component

Import { Recipe } from ‘../recipe.model’;

**Shortcut**: In creating a new model, you usually need to declare variable names and types then in the constructor call the parameters.

export class Ingredient {

  public name: string;

  public amount: number;

  constructor(name: string, amount: number) {

    this.name = name;

    this.amount = amount;

  }

}

Instead you can shorten it just calling the constructor and declaring the type and name inside as a parameter

export class Ingredient {

  constructor(public name: string, public amount: number) { }

}

Much shorter way of doing the same thing