Basics

Link to angular docs: <https://angular.io/docs>

Link to rxjs docs: <https://rxjs-dev.firebaseapp.com/>

All built in validators: <https://angular.io/api/forms/Validators>

Quick copy commands: cd C:\Users\GeddesT\Desktop\Source\, npm install, ng serve, ng g c , ng g d

Mass comment: ctrl k + c Mass uncomment: ctrl k + u Save all open files: ctrl k + s

@Input tag is used in conjuction with push to receive information from html template. Input tag means input into the component typescript file. Inside the input tag you can decide to have it called a different way if you don’t want confusing calls in the html file. Ex: @Input(‘notConfusingCall’) confusingCall will be called in the html by using notConfusingCall to push the call it would be notConfusingCall.push();

@Output tag is used with Even Emitter and emit to output information from typescript to html template. Output tag means output from the component typescript file. From the html if you have want to name it something different put it in quotes in the output what you want to be listened to. Ex: @Output(‘nonConfusingName’) confusingName = new EventEmitter<{variable: variableType}>(); this.confusingName.emit(variableWantingToBeSent);

@ViewChild tag uses local variables to get the value of the local without needing to send it through a click function. Ex: @ViewChild(‘localVariableName’, {static: false}) variableName: ElementRef; To access the variable in typescript you would type: this.variableName.nativeElement.value; The only time that static will be true is when you are using the function in the ngOnInit function.

@ContentChild lets you access information that is a child of the current component. This means if you have a ng-content called on a component item you can still receive the information that the ng-content is passing through. Ex: @ContentChild(‘localVariableName’, {static: false}) variableName: ElementRef;

@Directive you can create a directive similar to ngFor or ngIf. You have to import directive to be able to use the directive command. At the top of any directive file you need: @Directive({ selector: ‘[customDirectiveSelector]’}) and in the html where you want to use the directive use: <htmlElementTag customDirectiveSelector></htmlElementTag> The basic directive syntax is: In the constructor: constructor(private elementRef: ElementRef){} and in the init method: ngOnInit(){ this.elementRef.nativeElement.whateverYouWantToDo; The better directive syntax is: in the constructor: constructor(private elementReference: ElementRef, private renderer: Renderer2){} and in the onInit method: ngOnInit(){ this.renderer.setStyle(this.elementReference.nativeElement, ‘whateverTagYouWant’, ‘value’); } Link to renderer docs: <https://angular.io/api/core/Renderer2>

@HostListerner used to be able to bind events to a directive. Host listener need to be imported. Host listener syntax: @HostListener(‘eventName’) variableName(eventData: Event) { whateverYouWantToDoHere } ex: @HostListener(‘mouseenter’) mouseover(eventData: Event){ this.renderer.setStyle(this.elRef.nativeElement, ‘background-color’, ‘blue’ }

@HostBinding used to bind events to a directive without needing the renderer. This is used in conjuction with @HostListener to get rid of the need of a renderer. Needs to be imported in order to use it. Host binding syntax: @HostBinding(‘javascriptWayToAccessElements’) variableName: type; ex: @HostBinding(‘style.backgroundColor’) backgroundColor: string;

String interpolation: {{ variableName }} is used to post information to html from a corresponding typescript page. Meaning that the variable name has to be in the same component as the html template page.

Porperty Binding: [htmlVariableName] = ‘itemInComponent’ sets a variable from the html equal to something in the component typescript file. Typically use property binding over string interpolation.

Event Binding: (event)=”expression” used a lot for clicks ex: (click)=’doSomething()’. You can use custom events from other html files ex: (doSomething)=’someComoponentMethod()’.

Two way databinding: [(ngModel)]=”componentVariableName”

To generate a component in the console type ng generate component desiredComponentName or type ng g c desiredComponentName

To generate a directive in the console type ng generate directive desiredDirectiveName or you can type ng g d desiredDirectiveName

To generate a service in the console type ng generate service desiredServiceName or you can type ng g s desiredServiceName

To generate a model in the console type ng generate model desiredModelName or you can type ng g m desiredModelName

To generate a pipe in the console type ng generate pipe desiredPipeName or you can type ng g p desiredPipeName

ngFor syntax: \*ngFor=”let chosenVariableName of variableInComponent; let i = index”

ngIf syntax: \*ngIf=”booleanStatement; else otherStatement”

ngSwitch syntax: [ngSwitch]=”value” and \*ngSwitchCase=”valueInput” \*ngSwitchDefault=”defaultCase”

ngStyle syntax: [ngStyle]=”{stylingStatementUsingJavascript}”

ngClass syntax: [ngClass]=”{cssClassName: booleanThatYouWantForTheClassToFollow}”

To send an event to the component typescript file use $event inside of the click event and it will send the event data to the component ex: (click)=’doSomething($event)’ will send event data to the do something function in the component typescript file.

You can create a model a similar way to java the naming convention is: className.Model.ts

You can see the sourcemap of a program by going into the inspect element clicking on source, than going to webpack://, the file named . , than you go to src and you can see all of the file structure from the angular project.

Encapsulation: You can add encapsulation in the component which will either add the css tags as global the command for this is emulated, if you want it for just browsers that are supported that is native, and if you want the individual css files it is none.

Local reference: #variableName in html gives a reference to the entire html element not just a single value. Which you can pass along through click functions or other methods to get the full html value. You can single out these values by using variableName.value

To add value between an opening and closing custom element use the ng-content tag in the component that you are trying to call. Ex: if you are trying to call <app-variable-name>some information</app-variable-name> you would go to the variable name html component and put <ng-content></ng-content> wherever you want to add the information.

Lifecycle

ngOnChanges – called after a bound input property changes

ngOnInit – called once the component is initialized

ngDoCheck – called during every change detection run

ngAfterContentInit – called after content [ng-content] has been projected into a view

ngAfterContentChecked – called every time the projected content has been checked

ngAfterViewInit – called after the components view (and child views) has been initialized

ngAfterViewChecked – called every time the view (and child views) have been checked

Each of these lifestyles can be added to the component typescript files and each must be implemented and imported in the file to use them.

You should put as little logic as possible in html template.

In directives you can overwrite the variables in the directive by using the directive variables in the property binding tags ex: [defaultColor]=”’yellow’” to overwrite the default color to be yellow.

You can pass a directive with the same name as one of the directive variables by enclosing the directive in property binding.

\* are needed in angular because it tells angular that it is a structural directive.

You don’t have to use a \* in front of a structural directive if it is enclosed by ng-template tags and the statement is put in property binding.

When making a custom structural directive name the the input the same as the directive name.