Catchup for Lost Notes

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.

Services

Services app: Service app connects to other components to outsource how something is done. They are just normal typescript files. To call a service initialized it into the constructor using private and add providers in the component with a link to the service.

Dependency injector: Injects an instance of a class into a component. Is a hierarchical injector.

If you inject into AppModule the same instance of the service is available application-wide.

If you inject into the AppComponent the same instance is available for all components but nor for other services.

If you inject into a single component with not child components will only be available for the single component.

If you inject a service into a service remember to add the @Injectable() tag at the top of the file.

Routing

Setup for router: add import to the top of the file, add a constant for you route ex: const appRoutes: Routes = []; inside of the Routes the notation is path: ‘’, component than in the imports section add RouterModuel.forRoot(nameOfRoutesConst) and in the html add router-outlet to where you want the pages to load.

When routing to a new page make sure to have a / in front of it

Absolute paths: /name

Relative path: name

Relative path will append to the end of the link.

../ go back one path and go to the name

routerLinkActive directive will let the active tab on a router link similar to class active in html. This can be added to the wrapping or the link itself. If you are using bootstrap you have to add it to the wrapping to get they styling added with it.

routnerLinkActiveOptions will tell angular on the home page to only add the active tag if a certain criteria is met. Ex: [routerLinkActiveOptions]=”{exact: true}” will only use router link active if the string for the link exactly matches instead of making home active the entire time.

Add a link programmatically: You can link a button or whatever you want to link in the front end to the ts file of that component in the component you have to import Router from @angular/router and initialize it in the constructor ex: private router: Router. Now in the method you created to link or do whatever calculations you need you would call this initialized variable in the constructor and tell it to navigate to wherever you want ex: this.router.naviate([‘/anotherTab’]);

ActivatedRoute is used to tell what the current route is and what it is called. Ex: private route: ActivatedRoute.

:id in path allows you to dynamically create pages based upon variables passed.

Params vs snapshot: params is a observable are objects that work with asynchronous task. If you know that none of your information is going to change use snapshot if your information can change use params with subscribe.

If you have your own observables you have to unsubscribe from them on your own by implementing ondestroy and adding a subscription variable to store the data.

Query params allows you to add variables to either allow you to enter certain areas. Ex: [queryParams]=”{allowEdit: ‘1’}”

To add a fragment: fragment=”whateverYouWantInURL” in the html file

You can setup the query params and the fragment in typescript by doing the router method programmatically and adding to it: , {queryParams: {parameter: ‘value’}, fragment: ‘value’}

Adding a plus in front of something will make sure that it is an number when it comes out instead of a string. Ex: +variableName

To add nested routes add to the original route a variable called children and put your children routes in an array inside of the route and in the class put a router outlet where you would want to access the data.

To configure navigation use queryParamsHandling. Preserve will keep no matter what even if you add new ones and merge will merge new and old parameters. By default it will drop all of the parameters. Ex: this.router.navigate([‘navigationLocation’], {relativeTo: this.route, queryParamsHandling: ‘merge’});

Relative to will allow you to create a relative path appending whatever name you have added in the navigate to the end of the path. Ex: this.router.naviate([‘navigationLocation’], {relativeTo: this.route});

Redirecting: If you don’t want to load a specific component you can add the redirectTo function in the module.ts file. Ex: { path: ‘something’, redirectTo: ‘/not-found’ } this will redirect the page something to the not-found f page instead. To catch all routes that do not exist use \*\* and make sure it is the final path.

Wildcard route: \*\* means catch all

To redirect to a blank path use pathMatch: ‘full’ which basically means that the path has to match in every character allowing you to match to an empty path.

If you have more than two or three routes add a new file called app-routing.module.ts. In the class you have to have @NgModule at the top. Add the routes array to the new file and import all of the components. In NgModule add RouterModule.forRoot(appRoutes) to imports and add exports array with RouterModule in it. Than in the old file delete the routes array, remove the routes, and routerModule from the imports and import AppRoutingModule.

Route guards: functionality which is executed once a route is loaded or when you leave a route.

In the route guard you have to implement the CanActivate method which in turn needs an ActivatedRouteSnapshot and RouterStateSnapshot this method should return an observable that is a Boolean or a Promise that is a Boolean or just a boolean. Then inject the Router method so that you can send them to another url if needed. To use the guard in the module.ts file add canActivate[AuthGaurd] to the paths you want to be guarded. Remember to add the service to the module.ts file so that it is accessible by the app.

To guard children add canActivateChild. You have to implement it just like canActivate and has the same signature as canActivate.

To control navigation with canDeactivate make sure you create an interface which uses the same return type as canActivate than in the class itself implement the interface you just created in the same file and have it take in the interface type you just created, activated route snapshot, router state snapshot, and another router state snapshot that will be the next state and have it return the same type as canActivate.

You can pass static data using data in the path. Ex: data: {‘Page not fount!’} and you have to inject your active route into the class.

In the resolve when you go to access the data it is the same thing you named it in the path on the module.ts.

To configure urls to use hash. This allows the url to be parsed before and after the hash symbol. Do not use this unless needed.

Observables

An observable is a data source.

Observer: handle normal data, handle error, handle completion. This is the code that you write.

You use observable to handle asynchronous tasks.

Observables are constructs in which you subscribe where you are informed in changes in data.

Some observables don’t stop emitting values just because you aren’t using them.

Observables provided by angular automatically unsubscribe.

If you have a custom observable remember to always unsubscribe.

Once an observable sends an error it kills the observable.

Once an observable is complete it is done and will not emit any more data.

Error cancels observable it does not complete it.

Subjects you can subscribe to but you can call next outside of the observable.

Use subject not event emitter. Subject is more efficient behind the scenes.

You have to unsubscribe from a subject.

Don’t use subject over emitter in Output.

Forms

Template Driven: Angular selects the structure of the form.

Reactive Approach: You set up the structure of the form .

Make sure you import forms in you module.ts

ngSubmit will be fired whenever the form is submitted.

Give the element an Id and make it equal to ngForm so that you can get some javascript data of everything submitted in the form. This is the normal use of forms.

Dirty means you changed something about the form.

Touched means you clicked into a field.

You can use viewchild to get the data when you submit or even earlier.

You can add required directive to make sure the user has to enter data for that input. You can also use the email directive to make sure that users are entering email addresses and not just random characters.

You can still use two way binding to update every keystroke and save the data on for you to see.

Radio buttons act like any other input tag.

setValue command will let you set a form value however it will overwrite any data that is currently in the form.

patchValue is the recommended use because it will only overwrite certain data and not all data.

If there is not a value on the html than use the class of the wrapper div to get values. Ex: this.signupForm.value.userData.username vs this.signupForm.value.username

.reset() will reset the form data and the state of anything.

TD approach will work in most use cases.

Reactive you create the form programmatically in typescript.

For reactive forms you need the ReactiveFormsModule from forms

To setup a new form group use new FormGroup({});

To control the data entered in use new FormControl(null, validators, asyncronousValidators) where the first value is the initial state and the validators. If there is more than one validator put them into an array. You can have the method without validators but it is good practice to validate data.

To connect to the formGroup use property binding to connect to the form and the typescript. In each input or form item add formControlName where you will end up connect to the FormControl.

To pass validators the syntax is Validators.whateverValidatorYouWant.

Use the get method to get access to the information you want.

You can group inside of form groups to make data a little more organized. However if you group inside of a group when you call the get command you have to use your group name to get the data. Ex: normal: signupForm.get(‘username’).valid, inside a smaller form group: signupForm.get(‘userData.username’).valid.

Form array is used to hold an array of controls. It is created by using new FormArray([])

To create a custom validator you send the method a control and make it return a type that is a javascript object and whatever variable or type you want, the type is most likely a Boolean because you want to make check whether it passed or not. Ex: forbiddenNames(control: FormControl): {[s:string]: Boolean]} and this will return something that looks like {‘nameIsForbidden’: true}. Always return null not false.

To be able to access the variable you will have to bind this to forbidden names so that you can access the data. Ex: this.forbiddenNames.bind(this)

-1 means true on an array

You can use error to handle or show data to allow your users to see the error message. Ex: signupForm.get(‘userData.username’).errors[‘nameIsForbidden’] would only show if name is forbidden erroring and the same would happen if you put an error for ‘required’.

In asynchronous validation the method should be sent a control and should return an observable or a promise. Ex: forbiddenEmails(control: FormControl): Promise<any> | Observable<any>

You can look at value changes and subscribe to them to see every change that is made.

You can look at status changes to see the status of a form.

You can use setValue to set the values of the form.

You can also use patchValue to just patch values instead of changing all of the values.

You can reset in the same was as the td method.

A single line is knows as a pipe and it looks like: |

All default pipes are on angular.io

You can chain pipes and are applied in the order they are in.

To create pipes the notation for the file is pipeName.pipe.ts

When you have parameters it shows like {{ server.name | shorten:12 }} and if you had a second parameter it would be {{ server.name | shorten:12:anotherParameter }}

If pure is false it will run the pipe anytime the any data is changed on the page.

To output a string that you know will happen but doesn’t happen at the very beginning use async in a pipe.

Don’t receive data straight from a database send it to a server or api.

http verb: these include post, get, put, etc.

url (api endpoint): /posts/1

headers (metadata): {“content-type”:”application/json”}

body: {title: “new post”}

Add the html clientmodule to the project in order to be able to handle http requests the import is from @angular/common/http

On other apis you have to have post or add in the url.

Post returns an observable.

all requests are only sent when you subscribe

Good practice to use observable operators.

The map operators allows you to get data and return other data and turns it into an observable which allows you to subscribe.

To assign type to response type use the angled brackets on what type it is. Ex: .get<{ [key: string]: Post}> will assign to a Post method.

Try to keep all of your posts into a service so that there is as little calculating methods in the component class as possible.