**20th September 2022: Angular Course Notes**

* Angular Application
  + Components
    - Has a template (HTML) // UI
    - Has a class (properties, methods using Typescript) // code
    - Has metadata // extra data
  + Services that provide functionality across components
  + If we want an instance of a service, define it as a dependency in constructtor
* Typescript remarks:
  + Most often we don’t need to declare data type: variableName = data;
  + Create function: functionName() returnType{}
  + ` backticks can be used to specify a JavaScript template literal, which allows us to use a placeholder insert expression in string
* app.module.ts remarks:
  + In @NgModule decorator:
    - declarations — for user defined components
    - imports — 3rd party/external Angular modules

**CREATING AN ANGULAR APPLICATION**

In command terminal:

ng new app-name --prefix pm 🡨 creates files for angular app

code . 🡨 opens code in vscode

npm start 🡨 there will be a link, press on it

**INTRODUCTION TO COMPONENTS**

* Component
  + Import 🡪 import { Component //, other members } from ‘@angular/core’;
  + Metadata & Template 🡪
    - Defined with Angular component function
    - Attach function to class as a decorator
      * Decorator: func that adds metadata to a class, its members, or its method arguments, always prefixed with an @

@Component({

selector: ‘pm-root’,

// selector references component in any HTML

// selector defines component’s directive name

template:`

// backticks allow composing a string over several lines

<div><h1>{{pageTitle}}</h1> // binding w/ class

<div>My First Component</div>

</div>

`

})

* + Class (contains properties) 🡪
  + Methods are declared in class in component.ts file

export class AppComponent { //AppComponent is component name

pageTitle: string = ‘Acme Product Management’;

// pageTitle is property name

}

* Component Checklist
  + Class
    - Clear name – Have the word ‘component’ in the name to indicate it is a component class
    - Keyword: export
  + Members of the class:
    - Appropriate data type, default value
  + Metadata
    - Component decorator: prefix w/ @, suffix w/ ()
    - Component name in HTML: selector
    - View’s in HTML: template
  + Import statement
    - Keyword: import
    - Member name
    - Path to file

**INTRODUCTION TO TEMPLATES**

* Inline templates (HTML defined in app.component.ts file)
* Linked template (app.component.ts file will have link to html file)
* Custom Directive
  + When a component has a selector defined, we can use component as a directive
  + Meaning: Can be used as a HTML tag
  + 1. Use the directive in a component

2. Add the directive’s component to the directives in the Angular module

* Interpolation: {{}}
  + One way binding: From component class property to element property
* Angular Built-In Directive
  + Structural Directives: Put in the HTML tags
    - \*ngIf = “statement” : If logic

<img \*ngIf="showImage"

                            [src]="product.imageUrl"

                            [title]="product.productName"

                            [style.width.px]="imageWidth"

                            [style.margin.px]="imageMargin"

* + - \*ngFor = “let … of … “: For loop
      * for(let…of…): iterates over iterable objects
      * for(let…in…): iterates over the properties of an object

<tr \*ngFor="let product of products">

**DATA BINDING & PIPES**

* Property Binding
  + Allows binding to types other than string
  + Done in component.html file
  + <img [src]=’product.imageUrl’>
    - [src]: element property
    - ‘product.imageUrl’: template expression
  + Interpolation: <img src={{product.imageUrl}}>
    - Always assigns a string
* Event Binding
  + Connects an event to a method
  + Done in component.html file
  + In button: <button class=’btn btn-primary’ (click)=’toggleImage()’>
    - When button is clicked, image will appear
* Two-Way Binding
  + Banana: [()] in component.html file in input HTML tag
* Transforming Data with Piping (Use | )
  + Transform bound properties before display
  + Built-in pipes: date, number, decimal, percent, currency, json, etc.
    - <https://codecraft.tv/courses/angular/pipes/built-in-pipes/>
  + Custom pipes – Create in pipe.ts files
    - Consists of:
      * @Pipe decorator
        + @Pipe({

name: ‘nameOfPipe’ })

* + - * export class pipeName implements PipeTransform{}
      * import {Pipe, PipeTransform} from “@angular/core”;
    - Add the import to the module that declares the component that needs the pipe (in example: product list component is in app module so add in app module)
    - Add to declarations in @NgModules

**MORE ON COMPONENTS**

* Using an interface – created in a separate file from components, in a separate .ts file
  + Have to be imported into component file
  + As a data type: Identify properties for a specific type
    - export interface interfaceName{}
    - Can specify this interface as the data type
    - In the interface: propertyName: propertyDataType;
  + As a feature set: Declare properties AND methods to implement a specific feature
    - export class classComponent implements interfaceName{}
    - In class, write code for each defined property and method
* Encapsulating Component Styles
  + In Component decorator: styles or styleUrls:[path to css]
* Lifecycle Hooks – these are interfaces, in component.ts file
  + OnInit: Perform component initialization, retrieve data
    - 1. export class componentName implements LifecycleHook{}
    - 2. import { LifecycleHook } from ‘@angular/core’
    - 3. ngLifecycleHook: void {}
  + OnChanges: Perform action after change to input properties
  + OnDestroy: Perform cleanup
* Getter and Setter
  + Used in filter, so can filter out something the user searched for
* Arrow Functions
  + Arrow function
    - (variableBeingPassed: dataTypeOfVariable) => functionBody()

**21st September 2022: Angular Course Notes**

* Nested Components
  + As a directive: app component or nested component
  + As a routing target:
    - Create a new component to be used in the html file of another component. Add the new component to the module in declarations.
    - Have to use @Input and property binding.
    - In example: container component – product-list, nested component – star
      * In nested component: @Input – container can pass data to it, @Output – nested can emit data to container
* Services (class)
  + Component creates instance of the service
  + Steps to building a service:

1. Create service class
2. Define metadata w/ decorator
3. Import what we need
   * Service has to be registered
     + Register in root: in @Injectable do providedIn: ‘root’
     + Register in component: in @Component do providers:[classNameOfService]
   * Injecting the Service:
     + Need constructor(){} in the class of component.ts
     + Dependencies become the parameters of constructor
       - E.g. constructor(private productService: ProductService) { }
       - ProductService is class name of service

**22nd September 2022: Angular Course Notes**

* Using an observable
  + Example:

import { operators and methods } from ‘rxjs’;

const source$: Observable<number> = range(0, 10); // number: type of data Observable will emit… after = is what Observable will emit

source$.pipe( // pipe emitted items through several operators

map( x => x \* 3), // x is the emitted item

filter(x => x % 2 === 0 )

).subscribe(x => console.log(x)) // calls the observable

* + Steps:

1. Start the Observable (subscribe)
2. Pipe emitted items through a set of operators
3. Process notifications: next, error, complete
4. Stop the Observable (unsubscribe)

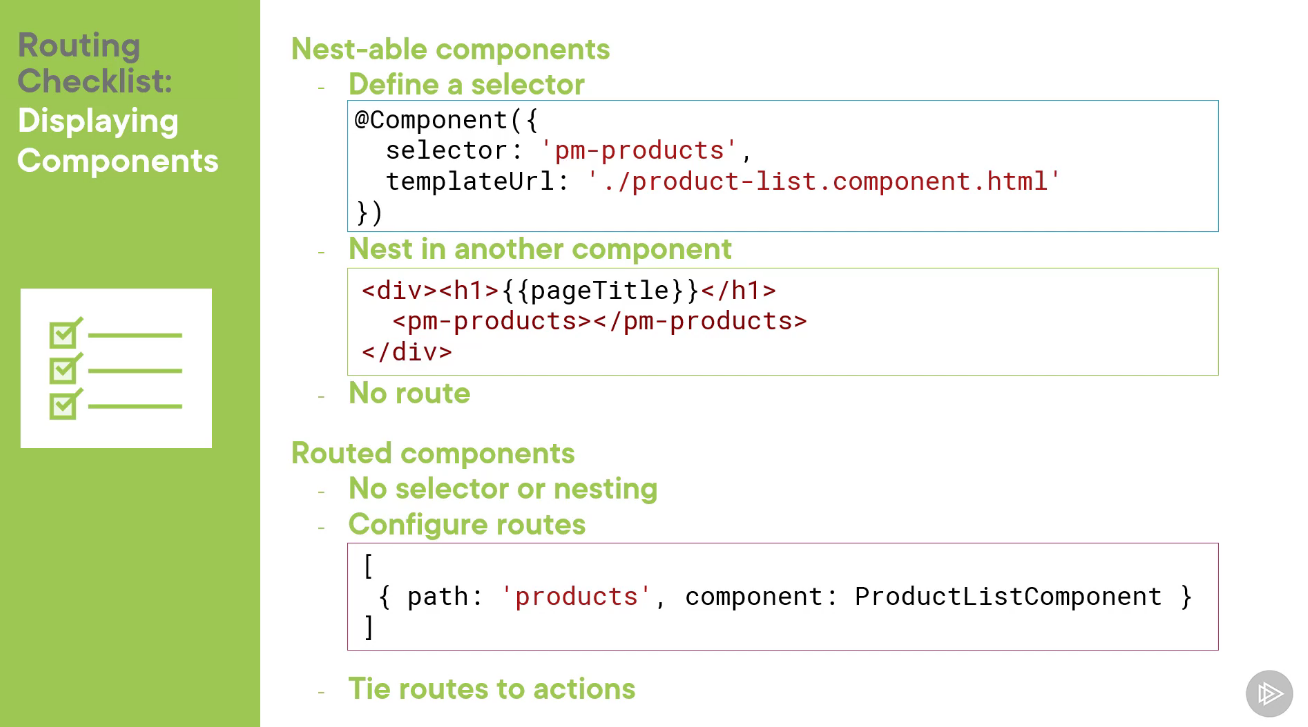
* Setting up an HTTP Request
  + Steps:

1. Import HTTPClient in service.ts
2. Inside class, specify the Url to the web server (private variable)
3. Add constructor inside the class (inject dependencies: private http:HttpClient)
4. Import HttpClientModule in module.ts
5. Add HttpClientModule in imports list of module.ts
6. In service.ts, to retrieve the data use a getter which returns an Observable<data type being processed>: return this.http.get<generic parameter data type the method will process>(this.Url) // this returns an observable

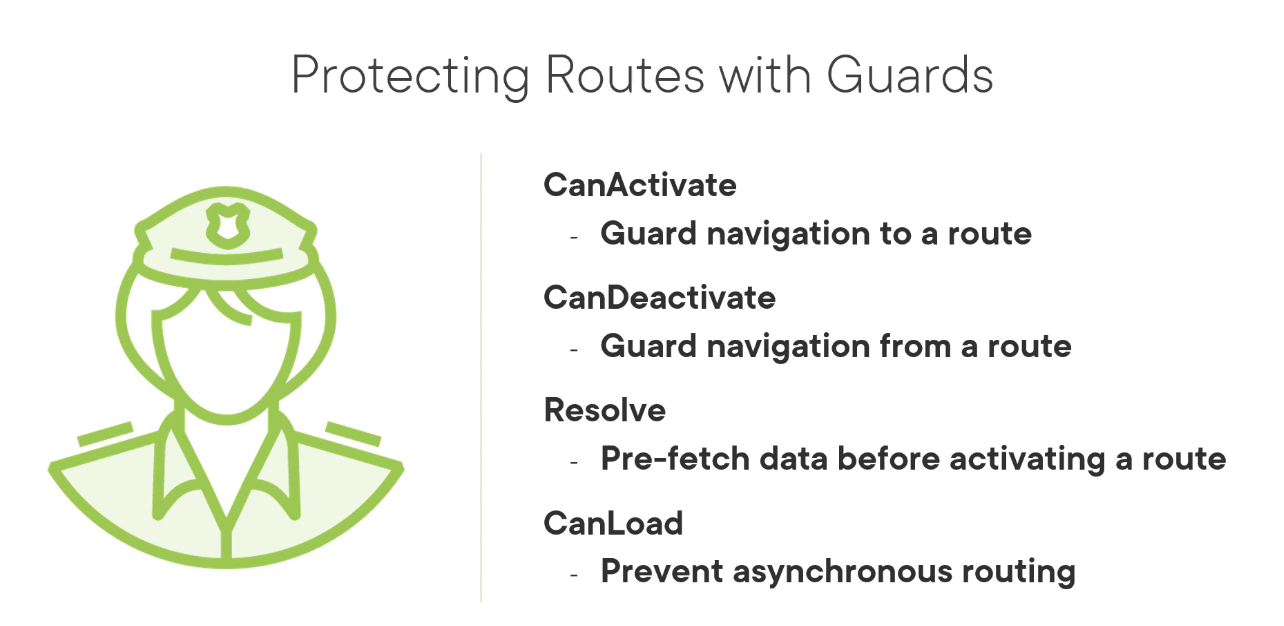
* Exception Handling
  + Should be added when doing HTTP Request
  + Steps:

1. After returning the observable, do tap( data => console.log(‘All : ‘, JSON.stringify(data))), and catchError(this.handleError)
2. Outside the Observable module but inside the class, create a new module private handleError(err: HttpErrorResponse){}

* Subscribing & Unsubscribing to an Observable
  + ngOnInit: subscribe, ngOnDestroy: unsubscribe
  + subscribe pass an observer object: next and error
* Routing
  + Configure a route for each component
  + Define options/actions
  + Tie a route to each option/action
  + Activate the route based on user action
  + Activating a route displays the component view
  + Uses RouterModule, add in module.ts and include in imports list
  + Router must be configured with a list of route definitions, each definition specifies a route object. Each route requires a path, and a component is specified (Add to RouterModule.forRoot([{path:…, component:…}])
* Tying the route to actions
  + Add in app.component.ts (the big one)
  + Use Angular directive: routerLink
* Routing vs Nesting:



* Reading parameters from a route
  + If path has : in front of the word, we have to get parameter
  + To get parameter from URL: use ActivatedRoute service
    - If we want an instance of a service, define it as a dependency in constructor
* Protecting Routes with Guards



* Build a guard
  + Create a class, use a decorator, import
  + Guard is a service so use Injectable decorator