**Angular Notes:**

1. Bootstrap install:
   * via CDN
     + copy <link> in head
     + copy script in body
   * via npm:
     + npm install bootstrap
     + in angular.json add:
     + in styles array: "./node\_modules/bootstrap/dist/css/bootstrap.min.css"
     + in scripts: "./node\_modules/bootstrap/dist/js/bootstrap.js"
2. angular-routing.module.ts

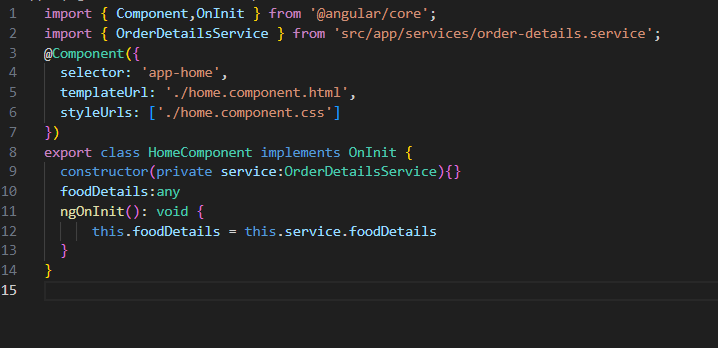
* This file is used to define and configure routes in the application by defining path to component mapping
* Several other responsibilities include:
* Lazy loading, which means loading modules and components on-demand instead of upfront.
* Route Guards: Route guards are used to control access to routes based on certain conditions. The routing module allows you to define and attach route guards to routes. Guards can be used to implement authentication checks, authorization checks, and other custom logic to determine if a user is allowed to access a particular route.
* Router Outlet: The routing module also defines the RouterOutlet directive, which is a placeholder where the router renders the corresponding component based on the current route. You typically place the RouterOutlet directive in your application's main template file, such as app.component.html.

1. Angular Terminologies:

* In the context of Angular, there are several key terminologies to understand. Let's go through some of them:
* Module: In Angular, a module is a container that organizes related components, directives, services, and other application elements. It is defined using the @NgModule decorator. A module encapsulates a specific feature or functionality of an application. It helps to keep the codebase modular and allows for better organization, reusability, and maintainability.
* Component: A component in Angular is a building block of the user interface. It represents a part of the application's UI and encapsulates its own logic, templates, and styles. Components are defined using the @Component decorator. They receive input data through properties and emit output events to communicate with other components.
* Template: The template in Angular represents the HTML view of a component. It defines how the component's UI should be rendered and displayed to the user. The template can contain HTML markup, Angular directives, binding expressions, and other Angular-specific syntax. It is associated with a component and controls the rendering of the component's data and structure.
* Directive: Directives are instructions that Angular applies to elements in the DOM (Document Object Model). They can modify the behavior or appearance of elements. Angular has two types of directives: structural and attribute directives. Structural directives, such as ngIf and ngFor, manipulate the DOM by adding or removing elements. Attribute directives, such as ngStyle and ngClass, modify the behavior or appearance of existing elements.
* Service: Services in Angular are singleton objects that provide specific functionality to components throughout an application. They are used for implementing business logic, data retrieval, communication with servers, and more. Services are typically injectable and can be injected into components, directives, or other services using dependency injection.
* Dependency Injection (DI): Dependency Injection is a design pattern used in Angular to manage the dependencies of components, services, and other objects. It allows for loose coupling and facilitates modular development. DI is performed by Angular's built-in injector, which creates and resolves dependencies automatically based on the configuration defined in providers.
* Routing: Routing in Angular enables navigation between different views or components based on the user's interaction or application state. The Angular Router is responsible for mapping URLs to specific components and updating the view accordingly. Routing allows for building single-page applications (SPAs) where different components are loaded dynamically without full page reloads.

1. **Directive-routerLink, routerLinkActive, routerLinkActiveOptions**: The **routerLink** and **routerLinkActive** directives are used in conjunction with the Angular Router module to create links and apply active CSS classes based on the current route.
   * + routerLink: The routerLink directive is used to create links that navigate to different routes within your application.
     + routerLinkActive: The routerLinkActive directive is used to add CSS classes to an element based on the current active route. It allows you to define one or more CSS classes that will be applied to the element when the route specified in the associated routerLink directive is active.
       - Eg: <a routerLink="/home" routerLinkActive="active">Home</a>
       - In the example above, when the route specified in the routerLink directive (/home) is active, the CSS class active will be applied to the <a> element. This allows you to style the active link differently, providing visual feedback to the user.
     + routerLinkActiveOptions: In Angular, the **routerLinkActiveOptions** is an input property of the **routerLinkActive** directive that allows you to customize its behavior and apply additional options for tracking the active route.
     + There are several properties that applies to this directive:
       - 1. **exact: boolean**: By default, **routerLinkActive** applies the active CSS class when the current route is a parent or child route of the linked route. Setting **exact** to **true** ensures that the active class is applied only when the current route exactly matches the linked route.
       - 2. **queryParams**
       - 3. **Fragment**
2. **Injector and @Injectable:**
   * + In Angular, injector is responsible for creating and managing instances of classes, called as dependencies. So, whenever you provide a dependency inside the constructor of a class or a service, injector is going to create an instance of that dependency and inject it into the constructor of the service class which requested it after it has created instance of the service class.
     + Injector Decorator: This decorator allows the service class to receive its dependencies through constructor injection.
     + Eg:
     + A screen shot of a computer program

       Description automatically generated with low confidence
     + Here, in the Injectable decorator, we have put ‘root’ in the providedIn tag, which specifies that this service should be provided at the root level, so it is available globally.
     + Using the **@Injectable** decorator is crucial for Angular to properly handle the dependency injection process and ensure that the required dependencies are available when the service is instantiated.
3. **Constructors in Angular:**
   * + In Angular, constructors play an important role in initializing components and handling dependencies.
     + In Angular, a component's constructor is responsible for receiving injected dependencies and performing any necessary setup or initialization tasks. The constructor is typically where you declare and receive the dependencies that the component requires to function properly.
4. Eg:



* The **@Component** decorator is used to define the metadata for the Angular component. It specifies the component's selector.
* implements OnInit signifies that the HomeComponent class implements the OnInit lifecycle hook interface. This interface requires the implementation of the ngOnInit method, which is called when the component is initialized.
* The ngOnInit method is implemented to execute logic during component initialization. In this case, it assigns the foodDetails property to the foodDetails property of the injected OrderDetailsService. This assumes that the OrderDetailsService has a foodDetails property accessible from outside the service.

1. routerLink=[a,fd.id]
   * + This is going to add /a/3 in the link, then it will search for this path in the app-routing.module.ts file and then navigate. fd.id is 3 here.
     + In the routing file, the configuration could be like this:
       - const routes:Routes = [

{path:”a/:id”,component:MenuComponent }

]

1. class=”form-control”
   * + probably used to include bootstrap styling of form than normal html.
     + Eg:

<input type="text" class="form-control">