Angular
Lab 9
Routing

1 Lab setup

The setup here is identical to that in Lab 1

2 Generating a new Angular project

The source code for this lab can be found in the `changes` subfolder of `Routing-Demo`

In either a separate command prompt (or shell terminal) or in an embedded terminal in Visual Studio Code, create a new Angular project with:

```
ng new routingdemo
```

Press enter to accept the default values for all the question prompts that follow regarding routing and stylesheet.

Navigate into the root project folder from the terminal with:

```
cd routingdemo
```

Build the app and serve it via Angular's live development server by typing:

```
ng serve
```

The final output line from this should indicate that the compilation process was successful and identify the port that the live development server is currently serving up the Angular app dynamically from (by default this will be port 4200)

Open a browser tab at:
http://localhost:4200/

You should be able to see the default landing page for all new autogenerated Angular projects.

To stop the development server, type Ctrl+C in the terminal window that it is running in. You can restart again anytime by typing `ng serve` in the project root folder.

3 Registering routes to use with RouterOutlet

The source code for this lab can be found in the `changes` subfolder of `Routing-Demo`

First, generate 3 components in a terminal in the root folder of this project with the following commands:

```
ng generate component home

ng generate component about

ng generate component contact
```

Modify the following files in `src/app` with the latest update from `changes`:

```
app.routes-v1.ts

app.component-v1.html

home.component-v1.html

contact.component-v1.html

about.component-v1.html
```

Type the following URLs manually into the browsers. These URL paths correspond to the routes specified in `app.config.ts` and will result in the corresponding component template view being rendered dynamically in place of the RouterOutlet directive

```
http://localhost:4200/home

http://localhost:4200/contact

http://localhost:4200/about
```

Typing in any other non-matching URL path (including just `http://localhost:4200`) will not result in any component view

In a typical single-page app (SPA) created using a framework such as Angular, it is more efficient to control what the user views by displaying portions of the overall view that correspond to particular components. This is opposed to rather than going out to the server to get a new page. Typically, as users perform application tasks, they will move between different views which in turn correspond to different routes that typically map to different URL paths.

To handle the navigation from one view to the next, you use the Angular Router. The Router enables navigation by interpreting a browser URL as an instruction to change the view, and maps that URL path to a corresponding component that needs to displayed at a specific point in the DOM tree.

Angular provides boilerplate configuration code for routing functionality to be implemented in all new generated projects from Angular 17 onwards (in `app.config.ts` and `app.routes.ts`)

In order to access the various components related to the various route paths configured in `app.config.ts`, we use the RouterOutlet directive which functions as a placeholder for the content which has to be dynamically displayed on the basis of the route path URL provided to the browser. This is handled by Angular dynamically in real time.

4 Using RouterLink and RouterLink Active to connect to and identify active routes

Modify the following files in `src/app` with the latest update from `changes`:

`app.component-v2.ts`

`app.component-v2.html`

`app.component-v2.css`

Clicking on the links in the main navigation bar available from the root template will now navigate to the previous route URLs, which in turn renders the component mapped to that route dynamically in the RouterOutlet as shown previously.

Notice however that if you manually type non-matching URLS paths into the browser bar, for e.g.

`http://localhost:4200/`

`http://localhost:4200/asdfasdf`

no component view is rendered below the main navigation bar.

Notice as well in the Network tab of Chrome Dev Tools, there is no further requests made to the development server to retrieve any additional code in response to the navigation activity. This is because the component view dynamically rendered in RouterOutlet was already preloaded into browser memory at the time of initial load of the Angular app.

The RouterLink attribute directive is used for binding a link to a specific route. It is commonly used in the template to create links that navigate between different views or pages in the application. When the RouterLink attribute is used with an anchor tag, the link behaves like a regular hyperlink and the browser's URL is updated with the route specified in the RouterLink attribute
Apart from the <a> tag, the RouterLink directive can be applied on the buttons or any element on which a click event can be used.
The RouterLinkActive directive can be used to add a specified class to the current active route (corresponding to the <a> tag that has just been clicked). This allows us to target the active route with appropriate styling rules.

5 Route redirecting and wildcard routes

Generate a component in a terminal in the root folder of this project with the following commands

```
ng generate component PageNotFound
```

Modify the following files in `src/app` with the latest update from `changes`:

`app.routes-v3.ts`

`page-not-found.component-v3.html`

Now notice that typing the following URL paths manually into the browser bar will have these effects:

`http://localhost:4200/`
now redirects to
http://localhost:4200/home

and any non-matching URL for e.g.

`http://localhost:4200/asdfasdf`

will bring up the view from PageNotFoundComponent, typically a so-called 404 page that most well designed web apps will incorporate.
https://searchengineland.com/404-pages-best-practices-examples-436618

The classic use case of redirecting of routes is that navigating to the home page of the app (i.e. without any path suffix after the domain name) will redirect to the main page for that app.

6 Passing route parameters through dynamic routing

First, generate a component in a terminal in the root folder of this project with the following commands:

```
ng generate component hero
```

Modify the following files in `src/app` with the latest update from `changes`:

`app.routes-v4.ts`

`home.component-v4.ts`

`home.component-v4.html`

`hero.component-v4.ts`

`hero.component-v4.html`

Notice that clicking on the specific links in the Home page navigates to the hero route with the specific subpath portion after the domain name (for e.g. http://localhost:4200/hero/1/Peter)
You can also type in manually a URL that follows the format as above, that is 2 subpath portions separated by / (for e.g. http://localhost:4200/hero/15/Superman) and it will still navigate to hero route and extract those two parameters (15 and Superman)

Notice as well that the 2 subpath portions (`/x/y`) after the `/hero` is compulsory as it specified in the routing configuration, so if you type the following paths below manually into the browser address bar, it will not match the hero route and will instead end up matching the wildcard route and navigating to the PageNotFoundComponent

http://localhost:4200/hero
http://localhost:4200/hero/2

So far, we have been working with static routing, where the routes are hardcoded in their mapping with their respective components. Dynamic routing is enabled by passing parameters to the routes when we configure them in `app.routes.ts`. These parameters will contain the values specified in the URL path prefix portion - you can specify one or more parameters which are appended to the end of the static path portion separated by /.
Later in `home.component.html`, we can use the routerLink directive to bind component property values to the <a> tag which will then become the router parameters when they are clicked on.

In the HeroComponent component which receives the route parameters, there are several ways to extract the route parameters. The ActivatedRoute service class is injected through constructor injection and is then able to provide access to various properties and parameters associated with the current route, such as the route path, query parameters, route parameters, etcetera

Using the ActivatedRoute, there are 2 main ways to extract the route parameters:

a) Using the route snapshot which provides the initial value of the route parameter map. This is the easiest to implement
b) Using observables. This is useful if you want to ensure that the receiving component will always get the value of the parameter if is dynamically changed somewhere else. This will be clarified later in the lab session on nested / child routes.

7 Passing query parameters

Modify the following files in `src/app` with the latest update from `changes`:

```
contact.component-v5.ts

contact.component-v5.html

about.component-v5.ts

about.component-v5.html
```

Notice that clicking on the specific links in the Contact page navigates to the About route with the specific query parameters portion (for e.g. http://localhost:4200/about?person=engineer&status=1) You can also type in manually a URL that follows the format as above, that with either one or both of the query parameters (for e.g. http://localhost:4200/about?person=teacher&status=200) and it will still navigate to hero route and extract those two query parameters (teacher and 200)

However, query parameters are different from route parameters in that:

- They are not compulsory and thus are not specified in the routing configuration in `app.routes.ts`. If you attempt to navigate to the About page directly without any query parameters by typing in the URL directly in the address bar (http://localhost:4200/about) or clicking on the main link in the navigation bar, you will be able to do so successfully.
- Since query parameters are not expected or mandatory, the component class that attempts to extract them will typically provide a standby default value for the query parameter in the event it does not exist in the path URL (for e.g. `this.person = queryParamMap.get('person') || 'noone'`)

Just like in the case of route parameters, extracting query parameters requires the use of the ActivatedRoute service class which is again introduced via constructor injection, and for which there are again 2 main ways to extract the query parameters:

a) Using the route snapshot which provides the initial value of the query parameter map. This is the easiest to implement
b) Using observables. This is useful if you want to ensure that the receiving component will always get the value of the parameter if is dynamically changed somewhere else.

With these two options of passing data through a path URL when navigating to a route of a component, the best practice is to:

a) Use Route Parameters to identify a specific resource or resources. For e.g. a specific product or group of products (`/product/shoes/adidas` or `/product/catalogid/35`)
b) Use Query parameters to specify a sorting, filtering, pagination operation (`/products?sort=rating` or `products?page=2`)

8 Nested / child routes

First, generate 3 components in a terminal in the root folder of this project with the following commands:

```
ng generate component news
ng generate component boringNews

ng generate component excitingNews
```

Modify the following files in `src/app` with the latest update from `changes`:

```
app.routes-v6.ts

app.component-v6.html

news.component-v6.ts

news.component-v6.html

boring-news.component-v6.ts

boring-news.component-v6.html

exciting-news.component-v6.html

exciting-news.component-v6.ts
```

As the application grows more complex, you may wish to create routes that are relative to a component other than your root component. These types of nested routes are called child routes. The idea of nested / child routes also aligns with the hierarchy of components / views that are used to compose a typical Angular app.

Some key points regarding nested / child routes:

a) They are declared / configured in the main routing configuration file `app.route.ts` using the children property in the path object.
b) You can declare one ore more nested / child routes mapped appropriately to a given path. The path to the nested / child route will then take the form of *`parent/child`*. Using path portion suffixes here is similar conceptually to passing route parameters, except that the subpath portions (`/x/y`) after the parent path route maps to child components rather than represent route parameter values.
c) You can optionally pass route parameters / query parameters to the nested child components in the same way that we have demonstrated earlier (for e.g. `'exciting/:framework'`)
d) The template of the parent component (`NewComponent`) will now contain an additional `<router-outlet>` directive in which the appropriate child component will be rendered. This is in addition to the primary `<router-outlet>` directive in the root component (`AppComponent`)
e) Child components (`BoringNewsComponent`) can provide links that direct to their sibling components (`ExcitingNewsComponent`) by using relative routes (`'../exciting'`)

A subtle but important point to keep in mind here: When the child component `ExcitingNewsComponent` extracts its route parameters, it needs to use the Observables approach (Approach #2). If we use Approach #1, this will produce an unexpected result as follows:
When we initially navigate to BoringNewsComponent, and use the Exciting News link there to navigate back to the ExcitingNewsComponent, the route parameter is passed is ExpressJS. Now if we attempt

to navigate again to ExcitingNewsComponent using the link from the parent News Component, it will still show ExpressJS (and not the updated parameter value passed by NewsComponent (React). This is because the `ngOnInit` method (where the parameter value is used to initialize the component property in Approach #1) is only executed once when the ExcitingNewsComponent is loaded for the first time (following the initial navigation from the link in BoringNewsComponent). Subsequently, any further navigations to this component will no longer call `ngOnInit` method to update the component property with the latest parameter value. To avoid this issue, we use the Observables approach to obtain the parameters.

9 Programmatic navigation and setting page title

Modify the following files in `src/app` with the latest update from `changes`:

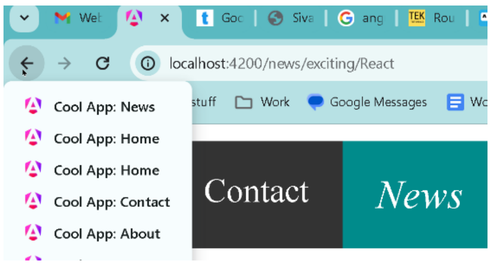
`app.routes-v7.ts`

`home.component-v7.ts`

`home.component-v7.html`

Up to this point of time, we have been navigating to different components by clicking on <a> links which link to specific routes via the routerLink directive. Alternatively, we could also programmatically navigate to a different URL path, which would allow navigation to occur on another event (for e.g. clicking on a button or typing in a textbox) and the usage of other relevant component properties in the URL path to navigate to. We use the Router service and the navigateByUrl or navigate method to do this.

In addition, we can add a title property to the route objects in the routing configuration in `app.routes.ts`. This allows us to associate a title with all the components that we have navigated to in the navigation history of the browser, which can be viewed and accessed by right clicking on the back arrow button in the browser address bar.



10 Integrating dynamic routing with HTTP REST API calls

We will use JSON server package that we installed in a previous lab. Navigate into the directory that holds the installation for this package along with the `heroes.json` file.
If you have deleted it already, you can reinstall it again globally as a NPM package in an empty directory with:

```
npm install -g json-server
```

Copy the following files into the empty directory from `changes`:

```
heroes.json
```

In the command prompt in this directory, start up the JSON server to serve the content from this file via a REST API in response to any HTTP requests that it receives:

```
json-server --watch heroes.json
```

In a new browser tab, navigate to:

http://localhost:3000/

You should be able to see the landing page for JSON server.
You can retrieve the entire content of the json file that you specified in the watch parameter with:

http://localhost:3000/heroes

Modify the following files in `src/app` with the latest update from `changes`:

```
app.config-v8.ts

hero.component-v8.ts

hero.component-v8.html

home.component-v8.ts

home.component-v8.html
```

Create the following files `hero.ts` and `localAPI.service.ts` in `src/app` with the latest update from `changes`:

```
localAPI.service-v8.ts

hero-v8.ts
```

A common theme in most apps is to use either the route parameter or query parameter values received by a component to be incorporated in a HTTP call to a backend REST API end point, which the returned response being formatted and displayed accordingly.