@angular/router

routing with angular5

Web app routing

- user types a url in the address bar we need to direct him to the proper resource he is asking
- user press a link we need to pass him to the proper page and change the url
- user press back and forth we need to move him to the proper page
- user reloads the app we need to transfer the proper resource
- user search our app we need to add url query param (upon reload display the same search results)
- user add filters we need to adjust the url with proper query params (upon reload display the same filtering)
- user navigates to a resource we need to add the proper matrix param
- we need to add the proper slug for SEO
    - **/tasks/1/buy-groucery/**

@angular/router

- **@angular/router** is the angular package that helps us deal with routing
- to install :
  - **npm install @angular/router --save**
- **@angular/router** is installed when we start a **@angular/cli** project
- we need to add the proper module to our **imports** array

<base>

- we need to instruct the router how to build the urls for the app
- the urls will be appended to the **href** attribute of the base tag
- in most cases our app will be in the root url so we should add
  - **<base href=”/”>**
- **@angular/cli** already placed default base tag with href set to the root

Router Service

- there is a singleton router service for angular application that uses routing
- the job of the router service is to match routes and load the proper component according to the match
- by default there is no routes defined

RouterModule

- to add the **RouterModule** you have to static **creation** functions
  - **forRoot**
  - **forChild**
- the creation function requires us to pass first argument that describes the routes
- for each module we create (root or feature module) we will add the routing in seperate module
- for our **AppModule** we will create **AppRoutingModule** and will wrap the routing logic in it’s own module
- for our **AppRoutingModule** we will define the following pages:
  - **HomePageComponent**
  - **AboutPageComponent**
  - **Error404PageComponent**

RouterOutlet directive

- this directive control where the router needs to place the component after it has a match
- we will place the directive in the **app.component.html**
- we can place common things to all routes in the **app.component.html**
- let’s place the directive and check if our routing works

Navigation

- let’s add navigation bar to our app
- we can navigate by using the **routerLink** directive
- we can navigate by injecting the **Router** service and navigating by code
- let’s add a button to the home page and navigate by code to the about page
- **routerLink** can also be dynamic

adding class to the active route

- it’s common to mark in the navigation bar the currently active url
- **routerLinkActive** directive will place a class on the anchor when the route is active
- **routerLinkActiveOptions** gets a dictionary where you can set the matching to be exact

routing for feature module

- if a feature module should add routing then you should place an additional routing module for it
- in our todo application we created a **TodoModule**
- the todo application should have a home page
- the todo application should have the url **/tasks** which displays a list of all the tasks
- the todo application should have a route of **/tasks/:id** which displays detail about a single todo task.
- let’s create the todo module with the task list component
- we will create a route that will lead to a page displaying the list
- we will also create a page that displays a single component
- we will need to create a service which query the service for the list and for a single task

Passing matrix param

- the tasks list component will have to pass matrix param to the detail route
- to define a matrix param in the **Route** object you place in the path the following syntax
  - **/tasks/:id**
- you can inject the **ActivatedRoute** and grab the matrix params from there
- the **ActivatedRoute** holds the route associated with the current component

load feature routes

- we use **loadChildren** to point to our feature module
- angular will load those components lazily only when the user navigates to that page which will optimize performance

Search tasks

- in our above the task list we want to implement a search from the task list
- the rest server supports filtering by search string
  - https://nztodo.herokuapp.com/api/task/?format=json&search=**<search-query>**
- in our task service we will add a method to search the list of tasks
- we will add a search component
- the search component will contain an input to search that will alter the url and add a search query param
- the list component will list for the search param and apply the youtube filter to optimize the request

Common things to add to route

- **children** - with this property we can create hierarchy of common parent
- **redirectTo**
- **canActivate**

Summary

- we separate our app to root module and feature module
- every module we add has its own routing module
- in our angular application we have a singleton service called router
- with **ActivatedRoute** service we can subscribe to params and matrix params
- we also saw how we pass data with the url