Tuesday: Routing

Routing

In Angular, routing enables us to navigate from one view to another. This means users can click links or buttons that change whatever is displayed on the app. In our app, we will create a navigation bar, that will enable us to change to different views in order to understand how routing works.

Let's start by creating a component that will give more information about the app. We will call it the *about* component. On our terminals, let's execute this command:

```
$ ng generate component about
```

In the HTML template file of our about component, let's add a description of our app.

src/app/about.about.component.html

```
<div class="jumbotron">
  <h1 class="display-4 text-center">About Goals App</h1>
  <hr class="my-4">
  <p class="lead text-center">An Angular app that let's you create your goals and gives you quotes
 from the world of computing!</p>
</div>
```

We have used a bootstrap jumbotron and written a description of our app. Feel free to describe your app in your own words and style.

AppRoutingModule

To implement routing in Angular, we use a module known as the AppRoutingModule. We import it and feed it with simple instructions in the form of code and it does the navigation for us. Interestingly, when we generate a new app in angular with the command ng new <app-name>, this is one of the modules that the Angular CLI automatically adds to our app for us. If we recall using this command when creating our Goals app, the first prompt was whether or not we want to use Angular routing, to which we agreed. This is what made the CLI add this module for us. Under the directory src/app, there is a file named app-routing.module.ts which the CLI created for us and looks like this:

src/app/app-routing.module.ts

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';

const routes: Routes = [];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

This file has everything we will need to create routes in our app, which we will do soon. Checking our root level modules file, *app.module.ts*, we see the **AppRoutingModule** module is already imported and added to the @NgModule **imports** array meaning we can use it in our app.

We configure our routes using **Routes** which is a service that presents a given component view for a specific URL. We have imported it at the top of our routing module file.

Adding Routes

Routes tell the router which view to display when a user clicks a link or pastes a URL in the browser address bar. A typical Angular route has two properties, a path which is a string that matches the URL in the browser address bar and a component which is the component that the router should create when navigating to this route.

Let us add two routes, one that will navigate to our goals component and another to the about component we have just created.

src/app/app-routing.module.ts

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { GoalComponent } from './goal/goal.component';
import { AboutComponent } from './about/about.component';

const routes: Routes = [
  { path: 'goals', component: GoalComponent},
  { path: 'about', component: AboutComponent},
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

We have first imported our two components at the top of our file. Inside our routes array, we have defined the path and component for each route. The RouterModule.forRoot(routes) in the imports initializes the router and gets it listening to the browser for location changes. The forRoot() method makes the router configurable at root level and supplies the service providers and directives needed for routing. This is the same method we used when we configured the loader because it serves the same purpose, making a module available at root level and supplying any dependencies it needs.

RouterOutlet

It is now the router's job to take care of what is being displayed on the application. RouterOutlet is a directive of the RoutingModule that has a selector <router-outlet> which handles routing for us. Let's use this selector and put the router in charge of displaying different views.

src/app/app.component.html

```
<router-outlet></router-outlet>
```

We have replaced the <app-goal> selector with the <router-outlet> selector because the router will be responsible for views from now on. If we keep the <app-goal> selector, our app will display the goal component and its child components only, which beats our purpose for creating routes.

Let's run our server and try out these routes. On the browser address bar, put in your routes and see the difference, for example, **http://localhost:4200/goals (http://localhost:4200/goals)** to display the view for the goals, and **http://localhost:4200/about (http://localhost:4200/about)** to change the view to our about component. Works, right? Our router is functional which is great!