ANGULAR UNIVERSITY

PREMIUM QUALITY
ANGULAR TUTORIALS

angular-university.io

Angular Router Crash Course - Build a Navigation Menu with Bootstrap 4

In this post we are going to learn how to use several features of the Angular Router in order to build a navigation system with multiple levels, similar to what you would find in an online learning platform or an online store like Amazon (but simpler).

We will do this step by step, the goal here is to learn how to configure the Angular Router by example and learn how to implement some of the very common routing scenarios that you will likely find during your everyday development.

Routing Scenarios Covered

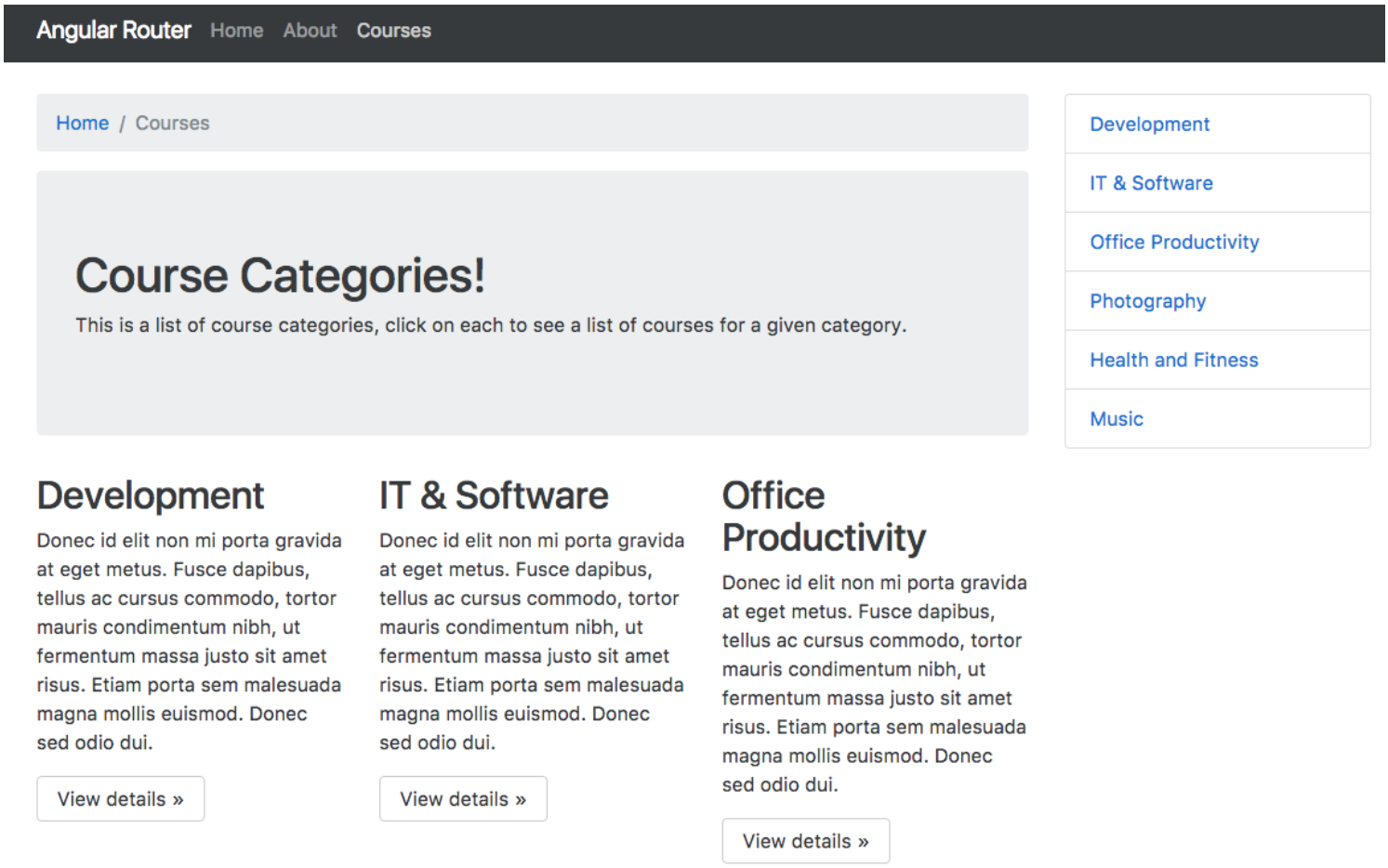
We are going to combine many features of the Router in order to build our menu, namely:

- Initial Router Setup
- Child Routes
- Nested Child Routes
- Auxiliary Routes
- Nested Auxiliary Routes

> *So let's get started building our navigation menu !*

The Menu System That We Are About To Build

This is how the menu system will look like:



We will have the following navigation elements:

- a top menu
- A navigation breadcrumb
- A side navigation menu, that only is displayed when we navigate into `Courses`
- navigation from the courses categories list to the course category, adapting the content of the side bar

Implementing the Top Menu

We want to start by implementing the top menu, which will always be visible no matter where we navigate inside the application. For that we are going to add the top menu at the level of the root component of the application:

```
<nav class="navbar navbar-fixed-top navbar-dark bg-inverse">
    <div class="container">
        <a class="navbar-brand">Angular Router</a>
        <ul class="nav navbar-nav" routerLinkActive="active">
            <li class="nav-item"><a class="nav-link" routerLink="home">Home
            <li class="nav-item"><a class="nav-link" routerLink="about">Abo
            <li class="nav-item"><a class="nav-link" routerLink="courses">C
        </ul>
    </div>
</nav>

<router-outlet></router-outlet>

```

If the menu is too big then a good alternative is to put it in a separate `top-menu.component.ts`. Notice the `routerLink` directives, linking to `home`, `about` and `courses`.

Also notice the `router-outlet` tag: this means the main content of the page below the top menu will be placed there. Also notice that there is no side navigation bar at this stage. The side navigation should only be visible if we click on `Courses`. Let's now write the router configuration for the top menu.

Configuring the Router Top Menu Navigation

The following initial configuration would cover the top menu scenario:

```
export const routerConfig: Routes = [
    {
        path: 'home',
        component: HomeComponent
    },
    {
        path: 'about',
        component: AboutComponent
    },
    {
        path: 'courses',
        component: CoursesComponent
    },
    {
        path: '',
        redirectTo: '/home',
        pathMatch: 'full'
    },
    {
        path: '**',
        redirectTo: '/home',
        pathMatch: 'full'
    }
];
```

As we can see, the `home`, `about` and `courses` Url paths currently map to only one component. Notice that we also configured a couple of redirects for the empty path case and a fallback route using the `**` wildcard. This is a good start, we have defined the home page, handled invalid urls and added a couple of common navigation items.

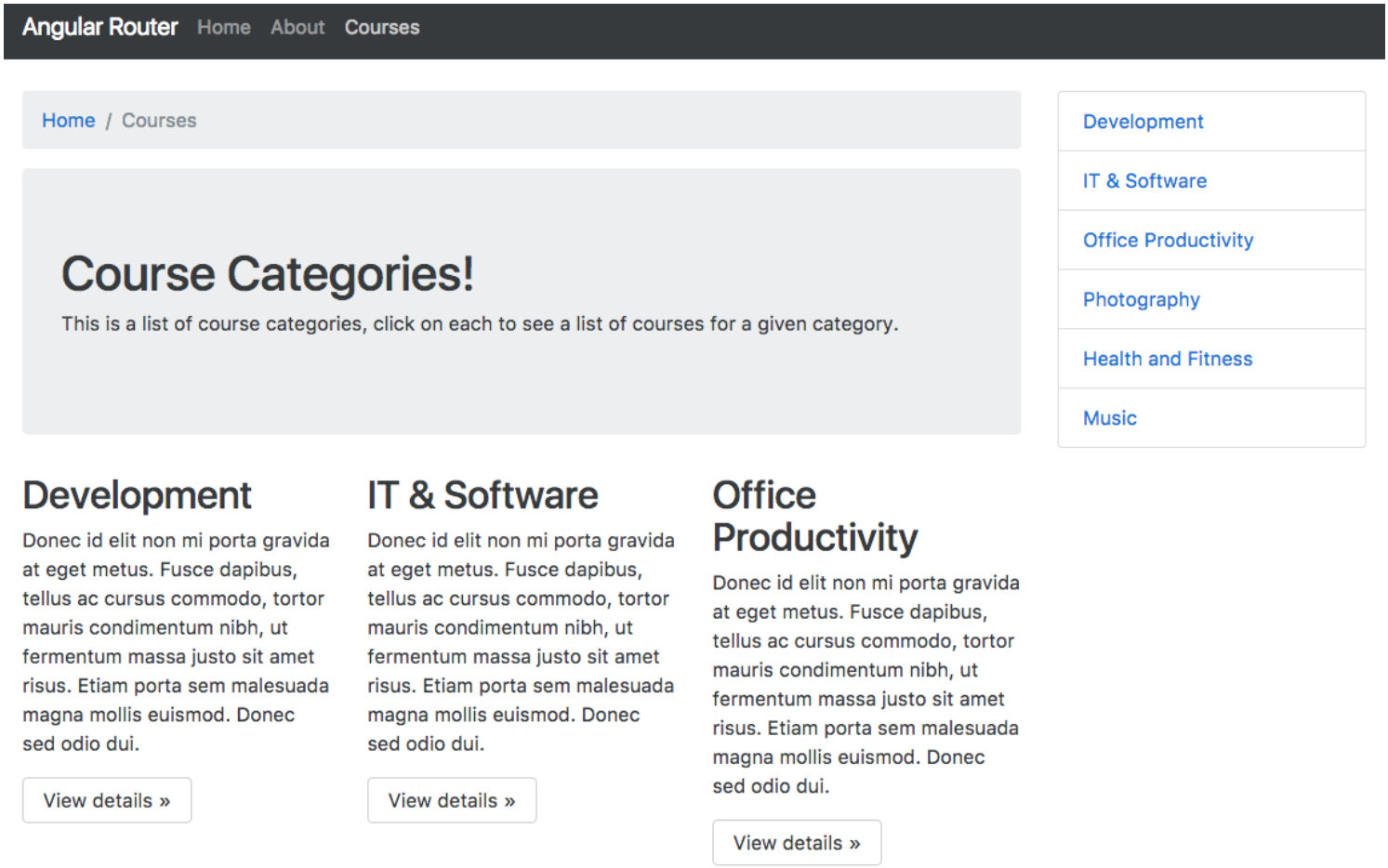
After setting up the top menu, we will now implement the following:

- we would like to show a side navigation to the user containing the list of course categories, but only if the user clicks on `Courses`

- also we would like to display a list of course category cards on the main body of the page

How will the Courses Categories page look like?

The goal is for the `CoursesComponent` to look like the following screenshot:



There are a couple of main elements in this screen:

- there is a main 'jumbotron' with a headline "Course Categories!"
- there is a list of course category cards below, containing 3 cards per line

- there is a side menu also containing a navigation link to each course category

As we can see, the main content of the page (everything below the top menu) was applied in place of the router outlet. But this Courses page will also have other internal routing scenarios as we will see further.

Implementing a side navigation menu using a Nested Auxiliary Route

In order to create the navigation side menu, we would like the `CoursesComponent` to contain a couple of router outlets itself:

- we would need a primary outlet inside the Courses component that contains a list of course categories.
- we would need an auxiliary router outlet inside the courses component to implement the side menu.

To implement this scenario we need to first start by going over Child Routes.

So how do we implement this very common routing scenario ?

Implementing the Course Category Cards with a Nested Child Route

Let's have a look at the `CoursesComponent` template, to see how we have implemented it:

```
<main>
    <div class="container">
        <div class="row row-offcanvas row-offcanvas-right">
            <div class="col-xs-12 col-sm-9">
                <ol class="breadcrumb">
                    <li class="breadcrumb-item"><a routerLink="/home">Home<
                    <li class="breadcrumb-item active">Courses</li>
                </ol>
                <div class="jumbotron">
                    <h1>Course Categories!</h1>
                    <p>This is a list of course categories, click on each t
                </div>
                <router-outlet></router-outlet>
            </div>
            <router-outlet name="sidemenu"></router-outlet>
        </div>
    </div>
</main>
```

Notice that there are a couple of `router-outlet` elements inside the courses component, which is itself being injected in place of a router outlet ! This is sometimes referred to as the "nested" route scenario.

We will go over the side menu outlet in a moment, but right now we want to configure the router to include the course category cards component in place of the unnamed router outlet.

Configuring the nested route scenario

In order to display the courses component and the course categories card inside it, we need the following router configuration:

```
{
        path: 'courses',
        component: CoursesComponent,
        children: [
            {
                path: '',
                component: CourseCardsComponent
            }
        ]
    }
```

This configuration means that if we hit the `/courses` url, the following occurs:

- we should display the `CoursesComponent` in the main `router-outlet` of our application (just below the top menu)
- in place of the `<router-outlet></router-outlet>` element of `CoursesComponent` we should display the `CourseCardsComponent`

But how do we display the side menu? For that we are going to need what is sometimes referred to an auxiliary route.

Auxiliary Routes

Auxiliary route is a common term to describe this scenario where we have multiple portions of the screen whose content is dependent upon the value of the url. In terms of router configuration this corresponds to a route with a non-default outlet name.

The primary route is implicitly associated to an outlet without a name attribute (`<router-outlet></router-outlet>`), and after that we can have as many other outlets in a given routing level as long as we give them different outlet names.

Implementing the side menu component

The side menu will be displayed in the `sidemenu` router outlet, and we would want the content of the side menu links to be dynamic and depend on the url. For example when first viewing the Courses section we would like it to contain a list of Course Categories like in the screenshot above.

But when you navigate inside a course category, we might want to display for example a list of course sub-categories.

In order to do this, the `SideMenuComponent` needs to know what the current url is, in order to load its data depending on the url:

```
1  @Component({
```

```
  selector: 'app-categories-menu',
  templateUrl: './categories-menu.component.html',
  styleUrls: ['./categories-menu.component.css']
})
export class SideMenuComponent {

  constructor(route: ActivatedRoute) {

      route.params.subscribe(params => console.log("side menu id parameter"

  }
}
```

Notice that this is just a skeleton implementation of the side menu. We would need to inject a service and get the data, or even better we could load the data before creating the component using a Router Resolver.

We can see here that the side menu component gets notified whenever a change in the url occurs, which is the essential part of making the side menu adaptable to the url content.

Configuring the side menu via a nested auxiliary route

We can now put the side menu in place with the following router configuration:

```
{
        path: 'courses',
        component: CoursesComponent,
        children: [
            {
                path: '',
                component: CourseCardsComponent
            },
            {
                path: '',
```

```
                outlet: 'sidemenu',
                component: SideMenuComponent
            }
        ]
    }
```

What this will do is, whenever we navigate to `/courses` we will now get the following (refering back to the file `courses.component.html` above):

- we will have `CourseCardsComponent` inside the unnamed or primary `router-outlet` element
- the new `SideMenuComponent` will be displayed inside the router outlet named `sidemenu`

We can start seeing here a pattern: a large variety of many of the routing scenarios that we typically want to implement can be configured by using only a few configuration notions: routes, outlets, child routes.

Let's see a further example of this, where we will go one navigation level deeper in the application routing!

Implementing The Course Categories Component

Let's say that now if we are in courses and click for example in `Development`, we would like the following to occur:

- we would like the content of the `router-outlet` element (the primary outlet) inside Courses to be replaced with a new component the list the contents of the course category. This could be for example a list of sub-categories

- we would like at the same time that we click on `Development` for the side menu to change its content as well, by displaying a list of links specific to the `Development` course category.

So in this scenario we will have two different sections of the page that will react separately to the url change: the side menu and the main body of the `Courses` page.

Configuring a new level of nesting inside the Courses page

Now let's say that the user clicks on `Development`. This will cause the url to change to `/courses/development`. If the user clicks on `IT & Software`, the url changes to `/courses/it-software`, etc.

We want a new `CoursesCategoryComponent` component in this case to be displayed in the main body of the Courses page. For doing that, we need the following routing configuration:

```
    {
        path: 'courses',
        component: CoursesComponent,
        children: [
            {
                path: '',
                component: CourseCardsComponent
            },
            {
              path: ':id',
              component: CoursesCategoryComponent
            },
            {
                path: '',
                outlet: 'sidemenu',
                component: SideMenuComponent
```

```
            }
        ]
    }
```

Notice that we are using a `:id` path variable, to capture the url part of the courses section. The content of this variable will be for example `development` or `it-software`.

Navigating into a Course Sub Section

If we want to display the development section, we need in the Development card to do something like this:

```
    <div class="col-xs-6 col-lg-4">
        <h2>Development</h2>
        <p>... </p>
        <p><a class="btn btn-secondary" routerLink="development"  role="but
    </div>
    <div class="col-xs-6 col-lg-4">
        <h2>IT & Software</h2>
        <p>... </p>
        <p><a class="btn btn-secondary" routerLink="it-software" role="butt
    </div>
```

Note that this won't work in the sense that the side menu will not be notified of this navigation, more on this latter.

This would display the `CoursesCategoryComponent` in the main body of the `Courses` page as expected. But what if we wanted the side menu to adapt its contents according to the navigation as well?

With the current configuration this would not happen, the side menu would still display the same content.

Making the Side Menu adjust to the current Url

In the current navigation scenario, once we get to the `Courses` page via the top menu, the side menu gets initialized and it stays in that state. In order to have the side menu also react to the url of the current course section (like load the `development` sub-sections when clicking on `Development`), we need to modify the routing config:

```
{
        path: 'courses',
        component: CoursesComponent,
        children: [
            {
                path: '',
                component: CourseCardsComponent
            },
            // ++ THIS PART WAS ADDED
            {
              path: ':id',
              component: CoursesCategoryComponent
            },
            // -- THIS PART WAS ADDED
            {
                path: '',
                outlet: 'sidemenu',
                component: SideMenuComponent
            },
            {
                path: ':id',
                outlet: 'sidemenu',
                component: SideMenuComponent
            }
        ]
    }
```

We have now configured a navigation to occur at the level of the side menu as well. If we navigate to `/courses/development` we might

expected the `SideMenuComponent` to receive the new `development` segment via the subscription to the `params` observable.

> *It turns out that won't be the case, so let's see why !*

What happened from the point of view of the side menu?

From the point of view of the side menu, we went from `/courses` to `/courses/development`, so we only affected the primary route so the content of the router outlet named `sidemenu` was kept the same.

If we want the side menu to be changed also we need to navigate in the `sidemenu` outlet as well. To understand this, try to imagine that its like the page has multiple browser windows each with its own url, one for the main content and the other for the side menu (the video above goes over this).

How to implement a Side Menu Navigation ?

In order to trigger a navigation in the side menu, we can do it directly in the template as well. In these more advanced navigation scenarios we can for convenience inject the router and use its navigation API to build the navigation.

You could also do this via the template, but it's great to be able to leverage Typescript auto-completion to fill in the needed navigation parameters. In these more advanced navigation scenarios auto-completion really helps as it acts like a form of documentation that is always up-to-date.

To do the navigation let's first change the HTML template of the course section card:

```
    <div class="col-xs-6 col-lg-4">
        <h2>Development</h2>
        <p>...</p>
        <p><a class="btn btn-secondary" (click)="navigate('development')"  r
    </div>
```

As we can see, we added a click handler in place of the `routerLink` directive, so we no longer configure the navigation directly in the template. Let's see how we can do the navigation programmatically.

Programmatic Navigation of Multiple Router Outlets

Let's have a look at the code, again this could have been done via the template as well:

```
@Component({
  selector: 'app-course-cards',
  templateUrl: './course-cards.component.html',
  styleUrls: ['./course-cards.component.css']
})
export class CourseCardsComponent {

  constructor(private router: Router, private route: ActivatedRoute) {

  }

    navigate(path) {
        this.router.navigate([{outlets: {primary: path, sidemenu:path}}],
                              {relativeTo: this.route});
    }

}

```

We can see that we are using the router navigation API to navigate in two outlets at the same time: the primary outlet and the `sidemenu` outlet.

This is what is happening in each outlet:

- the navigation is relative to the route we are currently in, because we are using the `relativeTo` property and passing in this property the current active route

- in the primary outlet we are hitting the `/courses/development` url, this should trigger the display of the `CoursesCategoryComponent` in the main `Courses` page.

- in the auxiliary `sidemenu` we are also hitting the url `/courses/development`, which will trigger the `SideMenuComponent` to get updated according to the url.

Remember that in the `SideMenuComponent` we have subscribed to the `route.params` observable of the active route. By now clicking on the course category of development, we will now have `development` printed to the browser console via `console.log`.

With this latest navigation logic the side menu is now notified of the url change. This means that the side menu could now load a different set of urls to be displayed if needed.

What does a multiple outlet url look like ?

By triggering the programmatic navigation call above, the browser will display the following url:

```
/courses/(development//sidemenu:development)
```

This url means:

- the courses url segment is active
- inside it the primary route is set to `/courses/development`
- the auxiliary child route 'development' is active for the outlet `sidemenu`

Conclusions

The new configuration API of the router is very powerful, because it allows us to implement a multitude of scenarios by using only just a few concepts: routes, outlets and child routes.

Notice that many of the terms that we usually use to describe routing scenarios like nested routes don't actually have a direct correspondence in the routing configuration terminology.

This is is because we can obtain those scenarios by combining a couple of more generic concepts: a nested route can be obtained via a child route with a different component than the parent route, an auxiliary route is just a normal route with a non-default outlet name, other than that it has the properties of a primary route, etc.

With this small number of configuration elements we can already easily set up a lot of the navigation scenarios that we will need in just about any application: a top menu, multiple levels of navigation, side menus and much more.