Routing

BUILDING WEB APPLICATIONS USING ANGULAR

Contents

- Configuring the router
- Adding child routes
- Passing data using route parameters
- Using named outlets

Introduction

- Navigation buttons (back and forward), hyperlinks and deep linking into various pages of our website are behaviours our users will anticipate
- The Angular router is an external module that enables us to emulate the behaviour of a multi-page website, able to show different views of the application dependent on the URL

Configuring the Router

- As the Router is an external module, we must first import it

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

- We need to also configure our index.html to tell the Router how to compose URLs

```
<base href="/">
```

- The Router needs to know what routes it has available, and what component to display when that route is hit – we configure this by passing a Routes object (note the import we did above) to the RouterModule.forRoot method, passing the result to the imports array of the module

```
let appRoutes: Routes = [
   {path: 'users', component: UsersListComponent},
   {path: 'user/:user', component: UserComponent},
   {path: '', redirectTo: '/users', pathMatch:
'full'},
   {path: '**', component: PageNotFoundComponent}
]
```

```
@NgModule({
   imports: [RouterModule.forRoot(appRoutes)]
   …
})
```

Configuring the Router

- The Router Outlet tells Angular where to render the component for the matching route path. Angular places the component after the router outlet

```
<router-outlet></router-outlet>
<!- views go here -->
```

- Router links give the Router control over anchor tags and allow it to provide CSS class to the element with the current active route

```
<nav>
   <a routerLink="/users" routerLinkActive="active">List of Users</a>
   <a routerLink="/user/chris" routerLinkActive="active">Chris</a>
</nav>
```

Configuring the Router

- We can configure the Router at the root of our application, however it is best practice to separate it into its own module
- First we complete the configuration as previously stated, in a separate module (AppRoutingModule)
- Then in this new module we re-export RouterModule
- Import AppRoutingModule to the root of our application
- this gives our app access to the directives required, whilst keeping the routes configuration separate

ActivatedRoute

- The ActivatedRoute object is an injectable service and contains a lot of useful properties for use in our routed application:
  - url – an observable of the route paths, represented as an array of strings of each part
  - data – an observable containing data provided for the route
  - params – an observable containing the required and optional parameters for this route
  - queryParams – an observable containing the query parameters for all routes
  - fragment – an observable of the URL fragment available to all routes
  - outlet – the name of the RouterOutlet used for this route
  - routeConfig – the route configuration used for the route that contains the origin path
  - parent – an ActivatedRoute for the parent route
  - children –contains all child routes activated under the current route
  - firstChild – the first ActivatedRoute in the list of child routes

Route Parameters

- We saw earlier how to set route parameters in the configuration

```
{path: 'user/:user', component: UserComponent},
```

- In order to receive this in the component we need to examine the ActivatedRoute service

```
import { Router, ActivatedRoute, Params } from '@angular/router';
```

```
ngOnInit() {
  this.route.params
    .pipe(
        // (+) converts string 'id' to a number
        switchMap((params: Params) => this.someService.get(+params['id']))
    ).subscribe((user: User) => this.user = user);
}
```

Route Parameters

- In order to navigate to this component now we can either use the routerLink approach from earlier, manually type in a URL, or navigate imperatively
- To do it imperatively we add the Router to our component which we will navigate from, and then add a method to do the navigation

```
constructor(private router: Router, private userService: UserService) { }

onSelect(user: User) {
    this.router.navigate(['/user',user.id])
}
```

Optional Route Parameters

- Required route params are passed in the array as subsequent indices

```
this.router.navigate(['/user',user.id,foo,bar]) //constructs …/user/id/foo/bar
```

- Optional route params can be passed as an object

```
this.router.navigate(['/user', user.id,{foo: bar}]) //constructs …/user/id;foo=bar
```

- Note it's not a query string (separated by a ? And & symbols, but matrix URL notation separated by semicolons ;
- These parameters can be retrieved from the ActivatedRoute object in the same way as required parameters – but don't stop a router URL match from occuring

Child Routes

- Child routes can be configured in the same way, we simply create a router module for each feature module and use the RouterModule.forChild() method as opposed to .forRoot()

```
@NgModule({
   declarations: [],
   imports: [
      RouterModule.forChild(appRoutes),
   ],
   exports: [RouterModule]
})

export class UsersRoutingModule { }
```

Child Routes

- We then import the ‘feature module’ into the root module of our application

```
@NgModule({
   declarations: [
      AppComponent
   ],
   imports: [
      BrowserModule,
      UsersModule,
      AppRoutingModule
],
providers: [],
bootstrap: [AppComponent]
})
export class AppModule { }
```

- Note: it is important that our root AppRoutingModule is imported last because “last match wins” – this root Router will handle things such as our catch-all 404 pages

Router Outlet with Child Routes

- In order to display child components using child routes we need to update our configuration slightly
- Rather than have each route created in our Routes array as follows

```
{path: 'users/:id', component: UserComponent}
{path: 'users', component: UsersComponent}
```

- We specify to the router that our users route has child routes

```
{
   path: 'users', component: UsersComponent,
   children: [
      {path: ':id', component: UserComponent}
   ]
}
```

Router Outlet with Child Routes

- We then need to give our UsersComponent its own Router Outlet, where it will load each UserComponent as per the path specified
- And we update any imperative links to use relative paths over absolute paths

```
this.router.navigate([`./${user.id}`],{relativeTo: this.activatedRoute})
```

- By passing an options object to the navigate function we can specify that this navigation should be relative to the activatedRoute and thereby the router will construct the correct path using directory-like syntax
- When using the RouterLink directive you needn't specify to use the activatedRoute as we did above. This is implicit anyway so we simply use the same link parameters array.

Named Outlets

- Named outlets can be used to display multiple routes
- Each template can have one unnamed (primary) outlet, but any number of named outlets

```
<router-outlet name="aside-details"></router-outlet>
```

- We then need to update our routing module to link up this named outlet to a path and component

```
{
   path: 'details',
   component: 'DetailsComponent',
   outlet: 'aside-details'
}
```

Named Outlets

- To use this in a routerLink we need to bind the named outlet to the RouterLink

```
<a [routerLink]="[{outlets: {'aside-details': ['details']} }]">Show Details</a>
```

- We then need to update our routing module to link up this named outlet to a path and component

```
{
   path: 'details',
   component: 'DetailsComponent',
   outlet: 'aside-details'
}
```

Named Outlets

- Secondary routes persist as you navigate within a component

```
<a [routerLink]="[{outlets: {'aside-details': ['details']} }]">Show Details</a>
```

- If you wish to clear the component then you can do so imperatively

```
closeWindow() {
   this.router.navigate([{outlets: {'chat-window': null}}]);
}
```

Exercise

ADDING ROUTES TO YOUR APPLICATION