Class 17 - Changing Pages with Routing && Course Project

Class 17 Course Content

Lesson Outline

Today we will learn:

1. How to add base routes for our application.
2. How to add child routes.
3. How to configure route parameters.
4. How to pass dynamic parameters through links.
5. How to programmatically edit our book links.

Lesson Notes

- **Router Outlet:** The `<router-outlet>` tag is how we dynamically render the component we want based on the URL path.

- **Router Link:** Every link needs to point to a route. The "*routerLink*" is used instead of an anchor tag to support the SPA methodology. The page will not refresh, even when you change routes!

Course Project Steps

STEP 1: Setting Up Our Basic Routes

app-routing.module.ts:

- Create the app-routing.module.ts file.

- Create the appRoutes array and add the main routes (root, bookshelf, library).

- Add the NgModule({}) configuration.

```
import { LibraryComponent } from "./library/library.component";
import { BookshelfComponent } from "./bookshelf/bookshelf.component";
import { NgModule } from "@angular/core";
import { RouterModule, Routes } from "@angular/router";

const appRoutes: Routes = [
  { path: "", redirectTo: "/bookshelf", pathMatch: "full" },
  { path: "bookshelf", component: BookshelfComponent },
  { path: "library", component: LibraryComponent },
];
```

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

app.module.ts:

- Add the `AppRoutingModule` to the imports array.

app.component.html:

- Replace the bookshelf and library tags with the `<router-outlet></router-outlet>` tag.
- Remove the `currentPage` property on the navigation tag.

app.component.ts:

- Remove the `onNavigatePage()` function and `pageDisplayed` variable.

STEP 2: Updating Our Navigation Component

shared/navigation/navigation.component.html:

- Remove the click listeners and "href" attributes.
- Add the `routerLink` attribute and point it to "/bookshelf" and "/library" respectively.
- Add the `routerLinkActive="active"` attribute to both links.
- Replace the dropdown menu (settings anchor tag) href with a "cursor: pointer" styles attribute.

```
<a class="navbar-brand brand" id="brand"
routerLink="/bookshelf">BookIt</a>

<!-- . . . -->

<li class="nav-item">
  <a class="nav-link" routerLink="/bookshelf" routerLinkActive="active"
    >Bookshelf</a
  >
</li>
<li class="nav-item">
  <a class="nav-link" routerLink="/library" routerLinkActive="active"
    >Library</a
  >
</li>

<!-- . . . -->

<a
  class="nav-link dropdown-toggle"
```

```
    style="cursor: pointer"
    id="navbarDropdownMenuLink"
    data-toggle="dropdown"
    aria-haspopup="true"
    [attr.aria-expanded]="show"
    (click)="show = !show"
    [class.show]="show"
  >
    Settings
  </a>
```

shared/navigation/navigation.component.ts:

- Remove the eventEmitter Output & `onSelectPage()` function.

shared/book/book.component.html:

- Remove the href attribute on the anchor tag.

- Add a style attribute and "cursor: pointer".

```
<a style="cursor: pointer" class="list-group-item clearfix">
  <!-- . . . -->
</a>
```

STEP 3: Adding Child Routes

Terminal:

- Create a bookshelf-home component inside the bookshelf folder.

```
ng g c skip-tests=true bookshelf/bookshelf-home
```

bookshelf/bookshelf-home.component.html:

- Add an `<h3>Please Select a Book!</h3>` tag inside the html.

app-routing.module.ts file:

- Add a children property to the bookshelf path.

- Add a route object for the bookshelf-home, book-details, and book-edit components. (Do them one at a time. . .)

```
const appRoutes: Routes = [
  { path: "", redirectTo: "/bookshelf", pathMatch: "full" },
  {
    path: "bookshelf",
    component: BookshelfComponent,
    children: [{ path: "", component: BookshelfHomeComponent }],
  },
  { path: "library", component: LibraryComponent },
];
```

bookshelf/bookshelf.component.html:

- Add a `<router-outlet>` in place of the book-details and ng-template.

shared/book/book.component.html/ts:

- Remove the `onBookSelected()` method & bookshelfService import.

STEP 4: Configuring Route Parameters

bookshelf/book-details/book-details.ts:

- Remove the @Input() before the "book" variable.
- Add `private router: ActivatedRoute` inside the constructor - (and import from "@angular/router").
- Add a subscription to the route.params observable inside `ngOnInit()`. Set an "idx" to the "+params['id']".
- Set "this.book" equal to the new "getBook(this.id)" method you will create from the bookshelfService.

```
book: Book;
idx: number;

constructor(
    private bookshelfService: BookshelfService,
    private route: ActivatedRoute
) {}

ngOnInit(): void {
    this.route.params.subscribe((params: Params) => {
    this.idx = +params['id'];
    this.book = this.bookshelfService.getBook(this.idx);
    });
}
```

bookshelf/bookshelf.service.ts:

- Add the "getBook(id)" method.

```
getBook(idx: number) {
  return this.myBooks.slice()[idx]
}
```

STEP 5: Passing Dynamic Parameters with Links

bookshelf/book-list/book-list.component.html:

- Bind the index from the `*ngFor` loop to a property on the `<app-book>` component tag.

```
<app-book [book]="bookElement" [idx]="i"></app-book>
```

shared/book/book.component.html/ts:

- Add an `Input()` for the idx we just passed to the book-list component.
- Add dynamic routing by placing `[routerLink]="[idx]"` on the anchor tag.
- Add a `routerLinkActive="active"` attribute to style the currently selected book.

```
<a
  style="cursor: pointer"
  class="list-group-item clearfix"
  [routerLink]="[idx]"
  routerLinkActive="active"
>
  <!-- . . . -->
</a>
```

STEP 6: Editing the Book

Terminal:

- Add a "bookshelf-editor" component inside the bookshelf folder.

```
ng g c bookshelf/bookshelf-editor
```

app-routing.module.ts:

- Register the new route

```
children: [
  { path: '', component: BookshelfHomeComponent },
  // { path: 'new', component: BookshelfEditorComponent },
  { path: ':id', component: BookDetailsComponent },
  // { path: ':id/edit', component: BookshelfEditorComponent },
],
```

bookshelf/book-editor/book-editor.component.ts:

- Register the `ActivatedRoute` inside the constructor.
- Add a subscription to the params observable inside `ngOnInit()`. Set a variable `this.idx = +params['id']`.
- Add another variable, "isEditMode" and use it to conditionally render based on what mode we are using / what route we are currently on.

```
export class BookshelfEditorComponent implements OnInit {
  idx: number;
  isEditMode = false;

  constructor(private route: ActivatedRoute) {}

  ngOnInit(): void {
    this.route.params.subscribe((params: Params) => {
      this.idx = +params["id"];
      this.isEditMode = params["id"] != null;
      console.log("%c  isEditMode: ", "color: red;", this.isEditMode);
    });
  }
}
```

STEP 7: Programmatically Edit Book Links

bookshelf/book-list/book-list.ts:

- Inject the Angular Router and Activated Route inside the constructor.
- Create a "onNewBook()" function that navigates to the new book route.

```
constructor(
  private bookshelfService: BookshelfService,
  private router: Router,
  private route: ActivatedRoute
) {}

// . . .

onNewBook() {
    this.router.navigate(['new'], { relativeTo: this.route });
}
```

bookshelf/book-list/book-list.html:

- Add a "(click)" listener to the "Add Book" button that calls "onNewBook()"

```
<button class="btn btn-primary" (click)="onNewBook()">Add New
Book</button>
```

bookshelf/book-details/book-details.ts:

- Inject the Angular Router and Activated Route inside the constructor.
- Create a "onEditBook()" function that navigates to the edit book route.

```
constructor(
  private bookshelfService: BookshelfService,
  private router: Router,
  private route: ActivatedRoute
) {}

// . . .

onEditBook() {
  this.router.navigate(['../', this.idx, 'edit'], { relativeTo: this.route
});
}
```

bookshelf/book-details/book-details.html:

- Add a "(click)" listener to the "Update Book" button that calls "onNewBook()"

```
<a class="dropdown-item" (click)="onEditBook()">Update Book</a>

<!-- If you haven't already! -->
<a class="dropdown-item" (click)="onRemoveBook()">Delete Book</a>
<!--   onRemoveBook() {
    this.bookshelfService.removeBook(this.idx);
  } -->
```

Additional Notes

Class Exercise

1. Generate a new Angular Application (without routing).
2. Generate three components: "Home", "About" and "Contact".
3. Redirect to the "HomeComponent" on the root route.
4. The "HomeComponent" should dynamically render either the "About" or "Contact" page depending on the URL.
5. Highlight the active route.
6. Create two buttons that programmatically route to both components.
7. Create two child components of the "About" page... the routes being: "about/bob" and "about/susan".
8. Create a button on the "About" page that displays one of these components at a time.

Bonus: Add a 404 page not found that displays when URL is not recognized

Routing Notes

- On the routerLink, if you omit the preceding "/"... you are creating a relative route to the page you are currently on while adding the slash creates an absolute path to the base URL of your website.

- Dynamically set the active anchor tab in the navigation bar by setting "routerLinkActive" property to "active" on every item.

- You can navigate between pages programmatically by using the built-in "@angular/router" router.navigate() method.

- Add parameters to your route by adding the ":custom-slug" to the path variable. eg: { path: "/servers/:id" }, where id is whatever is passed in the url after /servers.

- We can get access to the data passed as the ":custom-slug" by using the snapshot.params object that is available by importing {ActivatedRoute} from "@angluar/router".

- Every link has a bindable property "[queryParams]" which allows you to send key-value pairs through the URL. This can also be done programmatically. We retrieve this information similar to the last step.

- You can nest routers by adding a children property on the path that will hold all the child routes.

- To catch all routes that aren't covered by your app, add a new route at the end of your routes array with a path="**" and redirect to whatever component you want (usually a not-found page).

- It is common practice to have an `app-routing.module.ts` file that loads all of your routes.

- To protect certain routes from being accessed by users without permission, create an AuthGaurd Class that implements Angluar router's "CanActivate" or "CanActivateChild".

- Keep user from accidentally navigating away by using "CanDeactivate" Gaurd

---

Resources

- Angular Docs - Router Reference

- Angular Docs - Common Routing Tasks

- Angular 12 Blog - Routing Tutorial App