Angular Routing

Introduction to Angular Routing

Introduction

- Use certain **URLs** to access certain pages / components
- Use Browser navigation via **Browser-Stack**
- Navigate to different pages / components **via code**

Generate an application with routing enabled

- When creating a new Angular app, you will be asked if you would like to use Routing or not
- You can also pass the param **`--routing`** to enable routing (for a new app)

  ```
  ng new AngularRouting --routing
  ```

- This creates a **AppRoutingModule** (which is a `NgModel`) where you can configure your routes.

`app.module.ts`

- Import the **`AppRoutingModule`** to enable routing for the Angular app.

```
...
@NgModule({
    declarations: [AppComponent],
    imports: [BrowserModule, AppRoutingModule],
    ...
})
```

`app.component.html`

- Use the placeholder **`<router-outlet>`**.
- The Angular component, which is associated with a certain route (see later) is injected into this placeholder.

```
<h1>Welcome to my app.</h1>

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

Define a component for every page

- Create a normal Angular component for every page, e.g.:

```
ng generate component PageOne
ng gneerate component PageTwo
```

`app-routing.module.ts`

- Associate a route with a page.

```
const routes: Routes = [
    { path: 'page-one', component: PageOneComponent },
    { path: 'page-two', component: PageTwoComponent }
];
```

- The component `PageOne` is shown when navigating to `https://localhost:4200/page-one`

Default route

- If you leave the `path` empty, you can define a **defalt route** for your app.

```
const routes: Routes = [
    { path: '', redirectTo: '/page-one', pathMatch: 'full' },
    { path: 'page-one', component: PageOneComponent },
    ...
];
```

Undefined routes

- If an invalid route is used, you can define some kind of **"error page"**.
- **IMPORTANT:** Enter this route as the last entry in the `routes`-Array!

```
const routes: Routes = [
    { path: '', redirectTo: '/page-one', pathMath: 'full' },
    { path: 'page-one', component: PageOneComponent },
    ...,
    { path: '**', component: PageUnknownComponent },
];
```

Navigation - <a>

- Replace `href` with **`routerLink`** (this prevents a page refresh).

```
<h1>This is App.Component</h1>

<a routerLink="/page-one">Page One</a>
<a routerLink="/page-two">Page Two</a>

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

Navigation - <button>

- Use **`routerLink`** to define the target route.

```
<h1>This is App.Component

<button routerLink="page-one">Page One</button>
<button routerLink="page-two">Page Two</button>

<router-link></router-link>
```

Navigation - via Code

- Use the class `Router` to navigate to certain routes.

```
import { Router } from '@angular/router';
...
export class AppComponent {
    ...
    constructor(private router : Router) {}
}
...
navigateToPage(page: string) {
    this.router.navigateByUrl(page);
}
```

```
<input type="text" #page />
<button (click)="navigateToPage(page.value)">Navigate</button>
```

Navigation - Back

- You can simulate a click on the Browser's back button like this:

```
export class PageTwoComponent {
    constructor(private location: Location) {}

    goBack(): void {
        this.location.back();
    }
}
```

Parameters

- You can access URL parameters using **placeholders** in the routes.

```
const routes: Routes = [
    ...,
    { path: 'page-with-params/:id', component: PageWithParamsComponent
↪   },
    ...
];
```

Accessing Parameter Values

- Use the property **`paramMap`** from the class **`ActivatedRoute`** (via depency injection)

```
export class PageWithParamsComponent implements OnInit {
    id = -1;

    constructor(private activatedRoute: ActivatedRoute) {}

    ngOnInit(): void {
        this.activatedRoute.paramMap.subscribe(x => this.id =
↪   +x.get('id'));
    }
}
```

Accessing Query Parameters

- Use the property **`queryParamMap`** to access query parameters.

```
ngOnInit(): void {
    this.activatdRoute.queryParamMap.subscribe(x => x.keys.forEach(p =>
↪   console.log(`${p}: ${x.get(p)}`)));
}
```

Passing URL parameter via code

- Instead of **`navigateByUrl`** use **`navigate`** and pass the params as the second argument.

```
navigateToPageWithId(id: number): void {
    this.router.navigate(['page-with-params', id]);
}
```

Passing Query parameter via code

- Just add the params to the URL using **?**.

```
this.router.navigateByUrl(`edit?id=${id}`);
```

Passing complex objects

- Since params are just attached as **string** to the URL, you cannot extract complex objects just by using **activated route**.
- In general, there are two possibilites:

1. JSON.parse()
2. Use a service (**recommended**)

Using JSON.parse()

- You could serialize and deserialize the object using **`JSON.stringify()`** and **`JSON.parse()`**.

```
...,
{ path: 'page-with-complex-object/:person', component:
↪  PageWithComplexObjectComponent },
...
```

```
navigateToPerson() {
    const person: Person = { name: 'Hans', age: 66 };
    this.router.navigate(['/page-with-complex-object',
↪   JSON.stringify(person)]);
}

ngOnInit(): void {
    this.activatedRoute.paramMap.subscribe(x => this.person =
↪   JSON.parse(x.get('person')));
}
```

Using a service

- Simple version: Define a **public variable** in the service.
- Correct version: Use a **`Subject`** as learned already :)