Modules and Routing
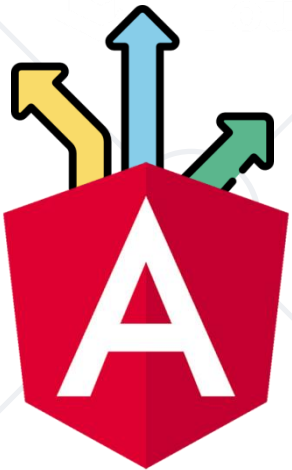
Creating Single-Page Applications

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

Table of Contents

Software
University

sli.do

#js-web

The NgModule

Building Blocks of the Application

Angular Modules Overview

- NgModules help **organize** an application into cohesive **blocks of** functionality

- An NgModule is a class **decorated** with **@NgModule**

```
import { NgModule } from '@angular/core';
```

- Many Angular **libraries** are NgModules
  - FormsModule, HttpClientModule, RouterModule

- Many **third-party** libraries are available as NgModules
  - **Material Design**, **Ionic**, **Angular Fire**

Creating Custom Modules

- Creating you own **modules** is **useful** when the application **grows**

- Only the **root** module should contain **BrowserModule**

- All custom-made modules should import **CommonModule**

```
import { CommonModule } from '@angular/common';
```

- Custom made modules have **exports** array

  - Components added in **declarations** are **private** by default

  - This is done because of **reusability**

Creating Custom Modules (2)

```typescript
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';

@NgModule({
    imports: [ CommonModule ],
    declarations: [
      CustomerListComponent,
      CustomerDetailsComponent ],
    exports: [ CustomerListComponent ],
    providers: [ CustomersService ]
})
export class CustomersModule { }
```

Export to render **outside** this module

Suggested Common Module

- **Shared Module** - to contain all **common** components, directives and pipes used by a **lot** of places

- **Core Module** - to contain **singleton** services and components needed only **once** in the application

- Authentication Module (**Register**, **Login**, **Logout**)

- **Feature Module** - to contain **feature** specific components

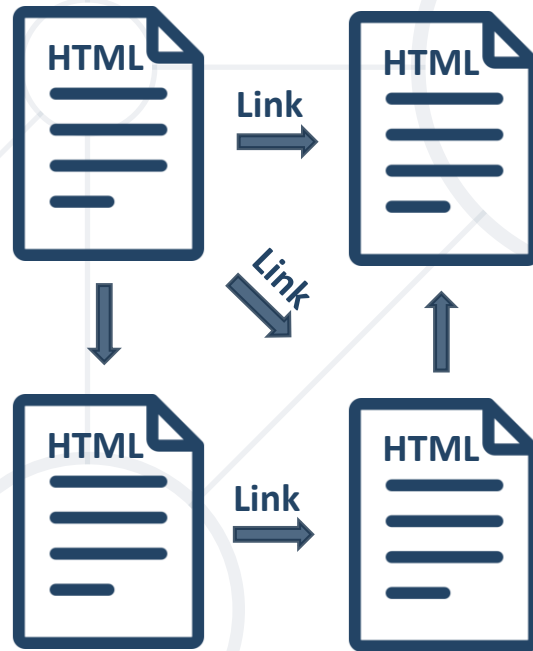- More info: **https://angular.io/guide/ngmodules**
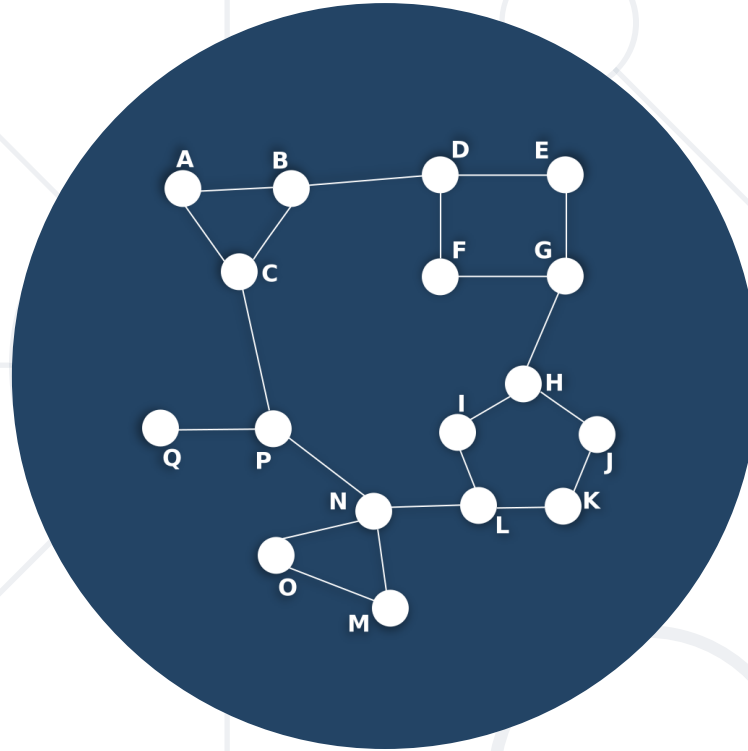
Routing Concepts

Navigation for Single Page Applications

What is Routing?

- Allows navigation, **without reloading** the page
- Pivotal element of writing **Single Page Applications**

Standard Navigation

Navigation using Routing

Single Page Applications

- A **Router** loads the appropriate content when the **location changes**
    - E.g. when the user manually enters an address
- Conversely, a change in content is reflected in the address bar
    - E.g. when the user clicks on a link
- Benefits
    - Load all scripts only once
    - Maintain state across multiple pages
    - Browser history can be used
    - Build User Interfaces that react quickly

Router Module

Setup, Links, Redirects, Parameters

Define the Template

- First add the **base** meta tag into the **index.html** file

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

Usually added by the CLI

- Add a **nav** tag so the **user** can navigate through the app

```
<nav>
  <a routerLink="/home">Home</a>
  <a routerLink="/about">About</a>
</nav>
```

Use routerLink instead of href

- Define the **router outlet** where the **content** will be **rendered**

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

Create Routes Module

- Import **NgModule**, **RouterModule** and **Routes**

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

- Define the needed **routes** as an **array** of **objects**

```
const routes: Routes = [
  { path: 'home', component: HomeComponent },
  { path: 'about', component: AboutComponent }
]
```

'/' is omitted

Create Routes Module

- Define the App Routes Module using the **decorator**

```
@NgModule({
  declarations: [
    HomeComponent,
    AboutComponent
  ],
  imports: [ RouterModule.forRoot(routes) ],
  exports: [ RouterModule ]
})
export class AppRoutesModule { }
```

Registers all app routes (done only once)

- Finally **import** the routes module in **app** module

```
import { AppRoutesModule } from './routes.module.ts'
// Other imports for core module

@NgModule({
  declarations: [ AppComponent ],
  imports: [
    BrowserModule,
    AppRoutesModule,
  ],
})
export class AppModule { }
```

The RouterLink Directive

- A basic usage of the **RouterLink** directive

```html
<a routerLink="/user/profile">Profile Page</a>
```

- Bind to the directive a pass an **array** of **parameters**

```html
<a
 [routerLink]="[ '/user', 1, 'profile' ]">
   Profile Page
</a>
```

Navigate Programmatically

- Inject the Angular Router in components

```
constructor(
  private router: Router
) { }
```

From "**@angular/router**"

- Use it to **navigate** from one component to another

```
loadData() {
  // Service call goes here
  this.router.navigate([ '/home' ])
}
```

Passing Parameters to Routes

- Define routes with parameters the following way

```
{ path: 'user/:id', component: UserDetailsComponent }
```

- Nested parameters

```
{
  path: 'user/:id/:username',
  component: UserProfileComponent
}
```

Fetching Parameters

- Inject **ActivatedRoute** in components

```
constructor(
 private route: ActivatedRoute
) { }
```

- Retrieve parameters directly from the snapshot

```
ngOnInit() {
    const id = this.route.snapshot.params['id']
}
```

Only runs one time when the component is initiated

Fetching Parameters Reactively

- To change the content of a component **inside the same one** use an **Observable** instead

```
ngOnInit() {
  this.route.params
    .subscribe((params: Params) => {
      const id = params['id']
    }
  )
}
```

Query Strings and Fragments

- To pass query parameters/fragments attach directives

```
<a
    [routerLink]="[ '/users', user.id, user.name ]"
    [queryParams]="{ search: 'Peter' }"
    fragment="loading"
</a>
```

- Retrieve them from the **snapshot**

```
this.route.snapshot.queryParams
this.route.snapshot.fragment
```

Setting Up Child (Nested) Routes

- Create nested routing by defining **child routes** using the **children property** of a route

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

- New router outlet needed at **UsersComponent**

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

Using Wildcards and Redirects

- If the requested **URL** doesn't **match** any paths for routes, **show** a **404** Not Found Page

  - This is done by using a **wildcard** '**'

  ```
  { path: '**', component: PageNotFoundComponent }
  ```

  - To redirect from one path to another

  ```
  { path: '', redirectTo: 'home', pathMatch: 'full' }
  ```

  > **Telling the router how to match a URL to the path of the route**

Router Guards

Protecting Routes

Guards Overview

- Limiting access to a route is **needed** in every application

- In Angular there are route **guards**

    - Build a guard **service**

    - Register the **service** in an Angular **module**

    - **Add** the guard to a desired **route**

CanActivate Guard

- The CanActivate guard **checks** criteria before **activating** a route

- It **limits** route access to **specific** users (register users, admins..)

- Called when the url **changes**

```
import { Injectable } from "@angular/core";
import {
    Router, CanActivate,
    ActivatedRouteSnapshot,
    RouterStateSnapshot
} from "@angular/router";
```

Guard Example

- Create a **guard** that restricts **non-authenticated** users

```
@Injectable()
export class AuthGuard implements CanActivate {
    canActivate(
        route: ActivatedRouteSnapshot,
        state: RouterStateSnapshot) : boolean {
        return this.checkIfLogged(state.url);
    }

    checkIfLogged(url : string) : boolean {
        // Use the authentication service
    }
}}
```

Angular Router Resolver

- The Angular Router provides a **resolve** property

- It takes a route resolver and allows your application to fetch data **before** navigating to the route

```
path: 'users', component: ServersComponent, children: [
  {
    path: ':id',
    component: UserDetailsComponent,
    resolve: { user: UsersResolver }
  }
]
```

Implement the Resolver

- Create the Resolver Guard

```
@Injectable()
export class UserResolver implements Resolve<User> {
  resolve(route: ActivatedRouteSnapshot,
  state: RouterStateSnapshot) {
    return this.usersService.getUserById(route.params['id']);
  }
}
```

Inject the service inside the **guard**

Use It Inside a Component

- Inside a Component fetch the data from the **data property** of the **snapshot**

```
constructor (
  private route: ActivatedRoute
) {  }

ngOnInit() {
  this.user = this.route.snapshot.data['user'];
}
```

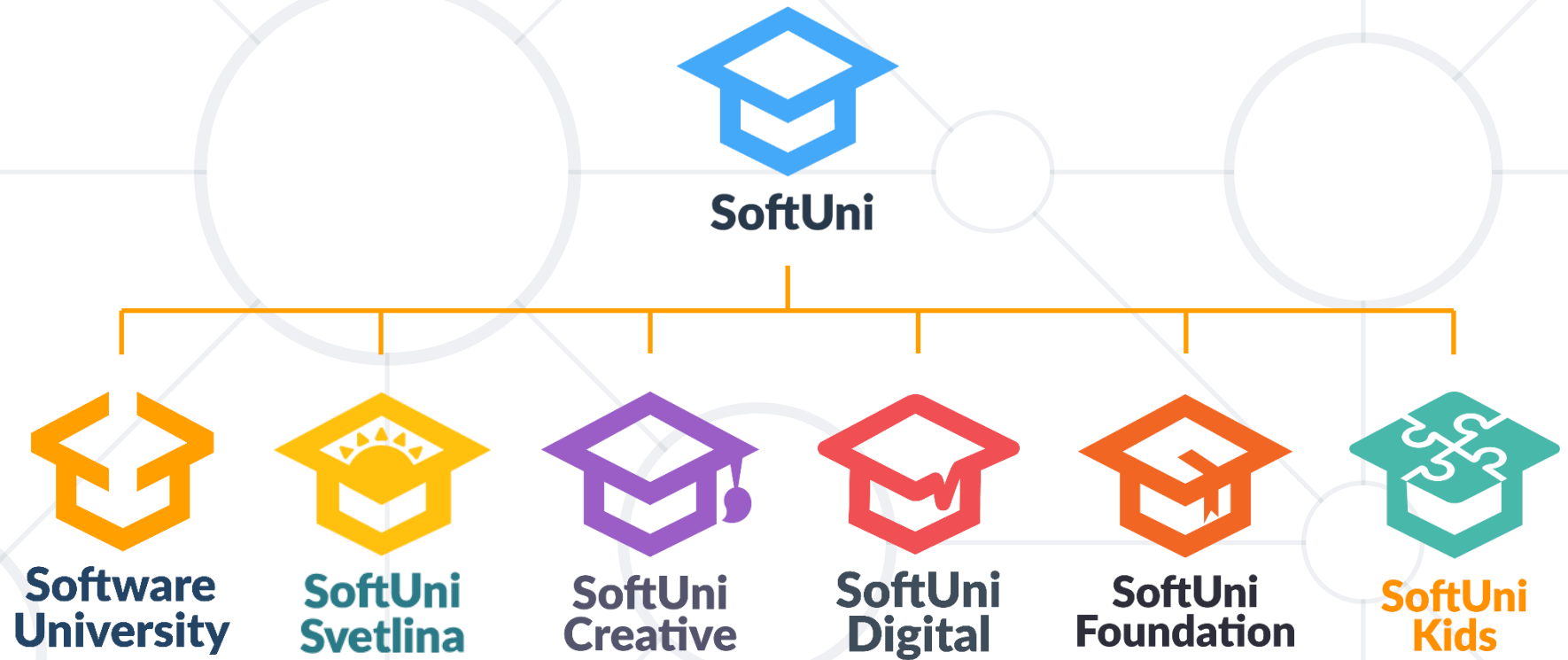The name bound **inside** the route resolver

Summary

- NgModules help **organize** an application

```
import { NgModule } from '@angular/core'
```
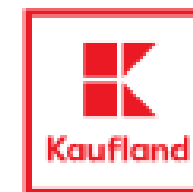
- Routing allows **navigation** without **reloading** the page

- The **Router Module** in Angular is a **powerful** tool

  - It supports routing with **params**, **child** routes, route **guards**, **resolvers** and more
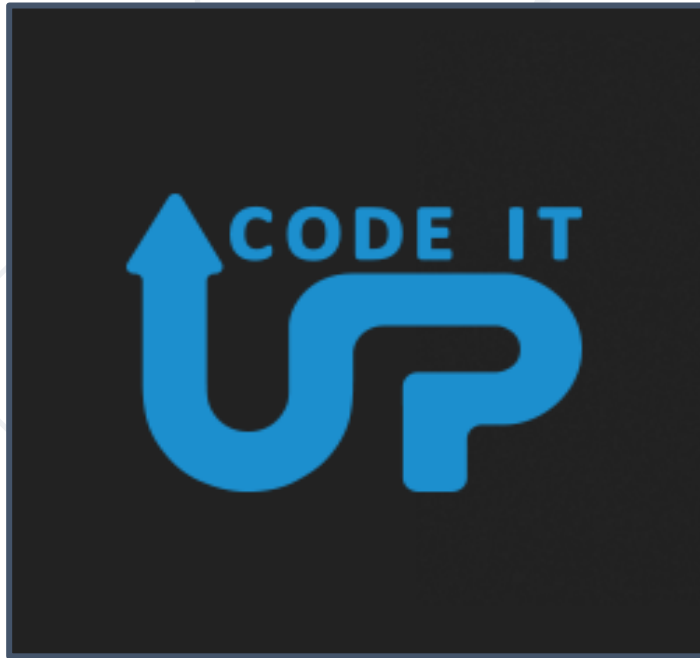
Questions?

SoftUni Diamond Partners

Educational Partners

CODE IT UP

VIRTUAL RACING SCHOOL

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers

  - softuni.bg, softuni.org

- Software University Foundation

  - softuni.foundation

- Software University @ Facebook

  - facebook.com/SoftwareUniversity

- Software University Forums

  - forum.softuni.bg

License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**

- Unauthorized copy, reproduction or use is illegal

- © SoftUni – https://softuni.org

- © Software University – https://softuni.bg