git clone link

https://github.com/DariusPa/itacademybudget

Branch: “master”

IT Academy: Day 3

…

“The Real Stuff”

Contents

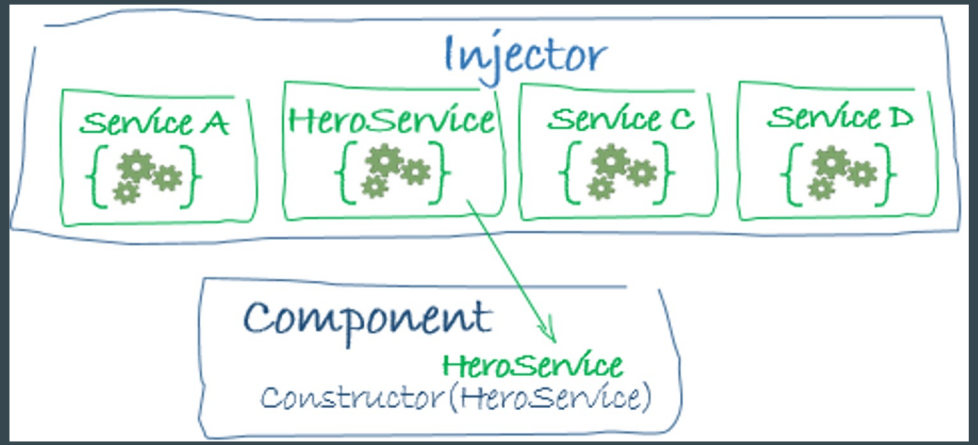
Dependency Injection

Dependency Injection:
Injecting dependency into a component

```
export class SomeComponent {
  constructor(private randomDependency: RandomDepService) {}
}
```

Dependency Injection: Basics

- One application-wide injector.
- Checks for existing instances of dependencies or creates if none exist.
- After all are available - constructor is executed



Routing & Navigation

What is routing?

- Entering a URL on the screen

https://www.swedbank.lt/private/cards/debit/debit_cards_main?language=LIT

- Clicking on a shady link

You've just won 1 BILLION dolleros!

- Any kind of script driven url change on screen.

Router

- Usually it's already imported and ready to use.
- RouterModule provided by '@angular/router'

```
import { RouterModule } from "@angular/router";
```

- Routes are defined by a Route[] interface, where a route has:
  - path - the string path for the route that is used
  - component - for the component that is bound to a route
    OR
  - redirectTo - a path that a user should be redirected to.
  - data - optional - route specific data object.
  - pathMatch - optional - "prefix" (default) or "full". "prefix" checks if at least a part of the URL matches a path - "full" requires a direct math.

Routes

- Navigation happens in a

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

- We can have more than one!
- So we can navigate to several routes at the same time

Router Navigation

There are two preferred ways of navigating between routes in Angular - from the template and from Typescript:

- routerLink=”route”
- this.router.navigate({path: “route”});

Example

```
const routes: Routes = [
  { path: '', component: FirstComponent },
  { path: 'second-component', component: SecondComponent },
  { path: '**', component: PageNotFoundComponent },  // Wildcard route for a 404 page
];
```

```
<a [routerLink]="['/heroes']">Heroes</a>
<a [routerLink]="['/hero', hero.id]">
  <span class="badge">{{ hero.id }}</span>{{ hero.name }}
</a>
```

```
<nav>
  <ul>
    <li><a routerLink="/first-component" routerLinkActive="active">First
Component</a></li>
    <li><a routerLink="/second-component" routerLinkActive="active">Second
Component</a></li>
  </ul>
</nav>
```

Task: Routing

Mike is really angry that application he is using does not load pages.. It shows blank page and sidebar menu is not working. We should fix that! He would really like to see “Home” as default page and be able to click on sidebar menu links! ☺

These are the steps you should do:

- Figure out what are the class names of components
- Add configuration to routing file with paths and components
- Add links to sidebar navigation

Route parameters

It is a common practice to have ids or numbers for entities and reflect them in the url:

https://news.ycombinator.com/item/22353596

In Angular Router, these path parameters are called “Route parameters” and they can be fetched from the ActivatedRoute of a component. ActivatedRoute refers to the data of the currently active route.

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

```
export class TestComponent implements OnInit {
  constructor(private activatedRoute: ActivatedRoute) {}

  ngOnInit(): void {
    // 1 way
    const id = this.activatedRoute.snapshot.paramMap.get('id');

    // 2 way
    this.activatedRoute.paramMap.subscribe( next: (data) => {
      const id = data.get('id');
    });
  }
}
```

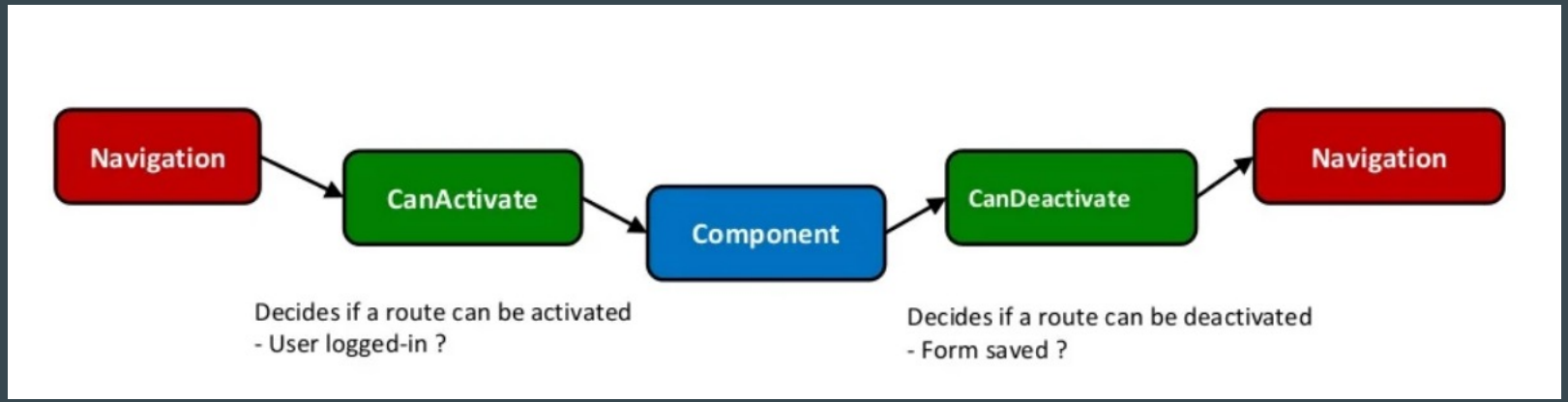
Guards

Route guards are services that define conditions under which a user can be allowed to enter a specific route. A guard implements an interface, with methods, that can define separate conditions for entry:

- canActivate - method to indicate if a path can be entered
- canDeactivate - method to indicate if a path can be exited
- canLoad - method to indicate if a lazy loaded path can be loaded
- canLoadChildren - method to indicate, whether lazy loaded children can be loaded.

```
{
  path: "home/:id",
  component: PostDetailsComponent,
  canActivate: [PostExistsGuard]
}
```

Route Guards



Task: Guard

Now we don’t want to let customer access certain page. For unknown reasons we should not let user to access expense inner page with id 2.

These are steps you should do:

- Add configuration to routing file for expense with id
- Apply links on expense name with id inside ExpensesComponent
- Create guard using command - `ng generate guard guards/filterExpense`
- Don’t let user access expense with id 2.

More examples

```
const routes: Routes = [
  {
    path: 'first-component',
    component: FirstComponent, // this is the component with the <router-outlet> in
the template
    children: [
      {
        path: 'child-a', // child route path
        component: ChildAComponent, // child route component that the router renders
      },
      {
        path: 'child-b',
        component: ChildBComponent, // another child route component that the router
renders
      },
    ],
  },
];
```

```
<!--AppComponent-->
<router-outlet>
  <!--Posts = /posts-->
  <!--Categories = /categories-->
  <!--Categories = /categories/:id = /categories/angular -->
  <!--Contacts = /contacts-->
  <!--Auth-->
  <router-outlet>
    <!--Login = /auth/login-->
    <!--Register = /auth/register-->
  </router-outlet>
</router-outlet>
```

Services

Angular Services

- Class with a narrow, well-defined purpose.
- Component >> View Presentation (How you see)
  Service >> Data Logic (What you see)
- Services should be used for encapsulating data processing tasks, such as fetch data, validating input or logging. Services are reusable (Singletons, ideally) in Angular.
- Generates via “ng generate service <name>”

Dependency Injection: @Injectable

@Injectable() - decorator that allows Angular to detect metadata and then pass the class a dependency into another class.

```
import { Injectable } from "@angular/core";

@Injectable({
  providedIn: "root"
})
export class PostService {
  constructor() {}
}
```

Providers

```
import { Injectable } from "@angular/core";

@Injectable({
  providedIn: "root"
})
export class PostService {
  constructor() {}
}
```

Can be added 3 different ways:

1. @Injectable({providedIn: ‘root’});
   Provides a single instance of Service to whole app - making it a Singleton.
2. @NgModule({ providers: [Service]});
   Provides a single Service instance to all components for this module.
3. @Component({ providers: [Service]});
   Provides a new instance of the Service for every instance of Component.

Example

```
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';
import { Category } from '../shared/category';
import { Post } from '../shared/post';

@Injectable({
  providedIn: 'root'
})
export class CategoriesService {

  constructor(private httpClient: HttpClient) { }

  getCategories(): Observable<Category[]> {
    return this.httpClient.get<Category[]>('/api/categories');
  }

  getCategory(id: string): Observable<Category> {
    return this.httpClient.get<Category>(`/api/categories/${id}`);
  }

  getCategoryPosts(id: string): Observable<Post[]> {
    return this.httpClient.get<Post[]>(`/api/categories/${id}/posts`);
  }
}
```

Task: Services

1. Generate a service -`ng generate services/expenses`.
2. Create method - `loadExpenses`- in the service.
3. Load expenses list from our fake API (DATA.ts inside shared folder).
4. Get posts from ExpensesService in component.
5. Bonus: create getExpense(id) in service to get single item for expense-details component

Useful links

- https://angular.io/guide/router
- https://www.youtube.com/watch?v=Np3ULAMqwNo

- https://angular.io/guide/architecture-services
- https://www.youtube.com/watch?v=pwuGBvOPFYI

The End