TD 4-part 1 : Angular + SPA Features : Route, routerLink, Router, ActivatedRoute, @Component vs Service life-cycle, LocalStorage

TD Objectives

Pre-requisites = angular project already setup (IntelliJ, ng serve, Chrome DevTools)

This hands-on session corresponding to course CM3, is focusing on Single-Page-Application features of Angular. It should take 1h30. For the remaining 1h30, there is TD4-part2, focusing on @Input and @Output field bindings.

First, you will practice on Routing to do the mapping between URL and the @Component currently displayed in the <router-outlet>.

Then you will see the life-cycles of the angular features

@Component

< (shorter than) life-cycle of Service = life-cycle of angular Application

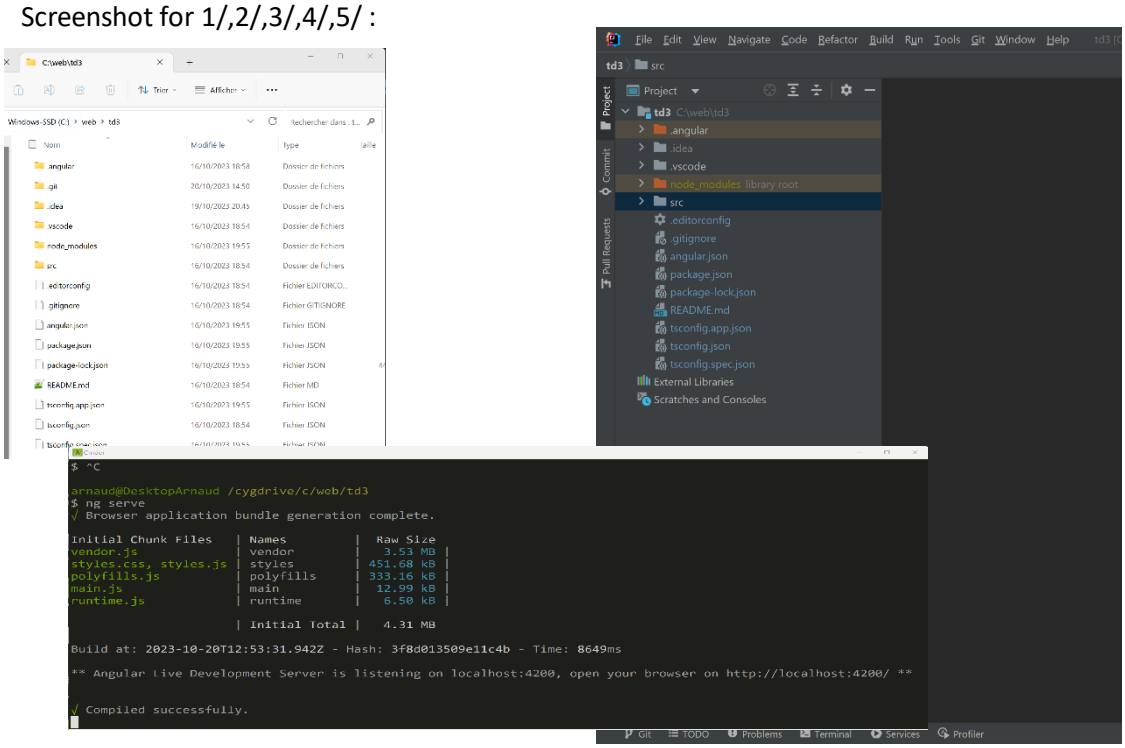
< (shorter than) localStorage.

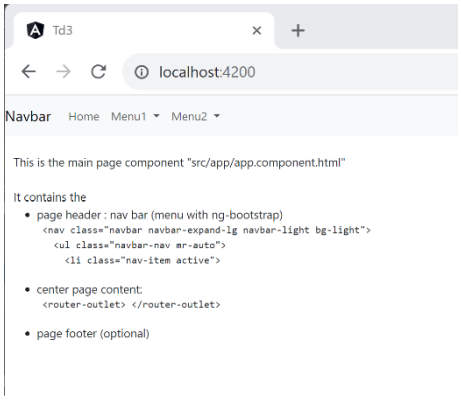
Step 1 : re-open your Angular project working environment from TD3

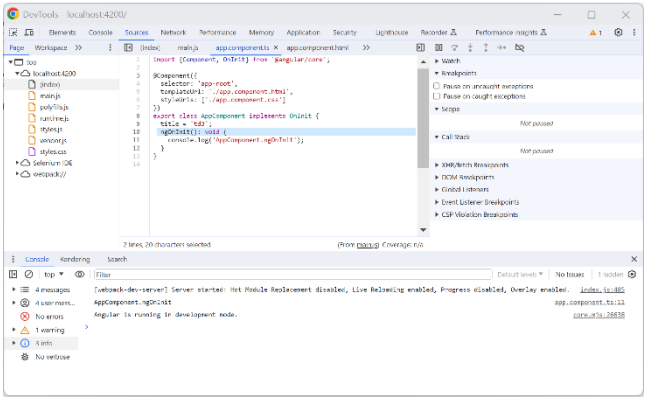
Your working environment consist of 5 things:

1/ A file system directory on your disk, for example “c:\web\td3”
This directory may be commited in a local git repository, for tracking history changes
2/ Your IDE opened on this directory (Intellij / VSCode / ..), and configured correctly
3/ The “ng serve” command running in a terminal, recompiling in real-time any change, and exposing on http://localhost:4200/
4/ Your Chrome web browser, opened on this URL
5/ the Chrome DevTools window opened (F12)

Screenshot for 1/,2/,3/,4/,5/ :





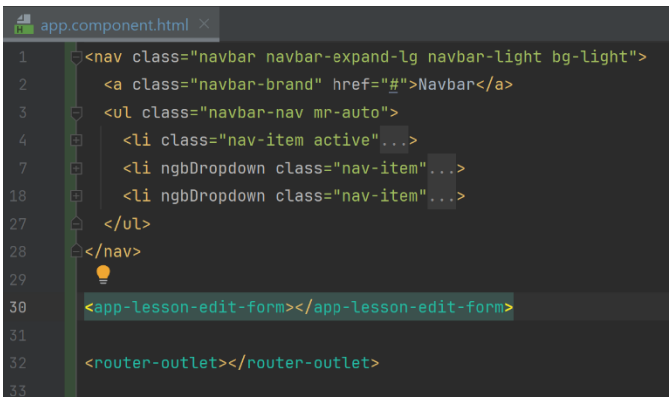


Step 2 : Display @Component in a page

From TD3, you already have created Angular component, using “ng g c lesson-edit-form”, which created 4 files (may be simplified to retain only 2 files) :


```
@Component({
  selector: 'app-lesson-edit-form',
  templateUrl: './lesson-edit-form.component.html'
})
export class LessonEditFormComponent {

}
```

To display this component statically from the main “app/app.component.html” page, you have added explicitly <app-lesson-edit-form></app-lesson-edit-form>



Step 3 : Define a route sub-URL for this page, retain only a link in the main menu page

In the main page, remove the line “<app-lesson-edit-form></app-lesson-edit-form>”, then add a link element “<a routerLink=”…”> in your menu nav bar.

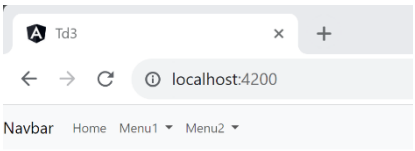
Be very careful to check you still have <router-outlet></router-outlet>

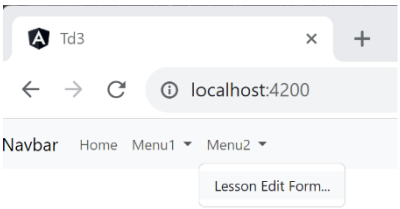
```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
  <a class="navbar-brand" href="#">Navbar</a>
  <ul class="navbar-nav mr-auto">
    <li ngbDropdown class="nav-item">
      <a ngbDropdownToggle id="navbarMenu2" class="nav-link dropdown-
toggle" role="button">
        Menu2
      </a>
      <div ngbDropdownMenu aria-labelledby="navbarMenu2" class="dropdown-
menu">
        <a class="dropdown-item" routerLink="/lesson-edit-form">Lesson Edit
Form...</a>
      </div>
    </li>
  </ul>
</nav>

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

Then declare in file “src/app/app-routing.module.ts”, that the chosen url path “lesson-edit-form” maps to component class “LessonEditFormComponent”

```
import {LessonEditFormComponent} from "./lesson-edit-form/lesson-edit-
form.component";

const routes: Routes = [
  { path:'lesson-edit-form', component: LessonEditFormComponent },
];
```

Your home page should be empty, with a menu to change the page:





When clicking on this menu item, your url changes to “http://localhost:4200/lesson-edit-form”, and it displays the form component. However, the angular page is not reloaded.

Step 4 : add a <a routerLink=”..”> to go back to the home page

In this lesson-edit-form page, add a link to navigate to the home page:

```
<a routerLink="/">Home</a>
```

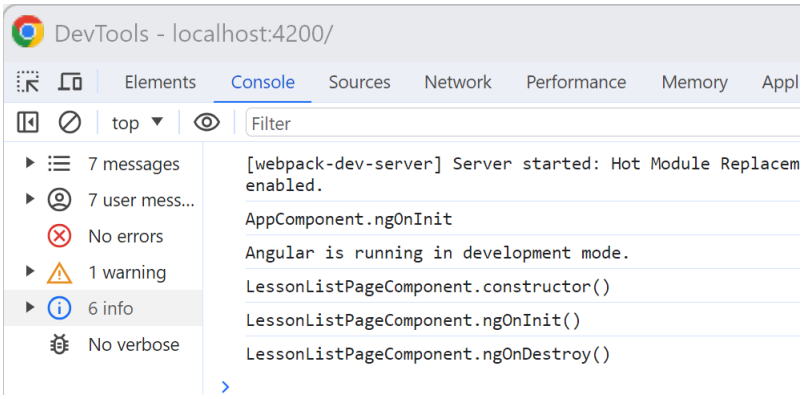
It lacks some icon, but it works. See optional Step 6 to add icon to it, otherwise skip it.

Step 5: check component life-cycle constructor-> ngOnInit-> ngOnDestroy

This is redundant with previous TD3, but redo the exercise of adding console.log() on 3 methods: constructor(), ngOnInit, and ngOnDestroy()

```
import {Component, OnDestroy, OnInit} from '@angular/core';

@Component({
  selector: 'app-lesson-list-page',
  templateUrl: './lesson-list-page.component.html'
})
export class LessonListPageComponent implements OnInit, OnDestroy {

  constructor() {
    console.log("LessonListPageComponent.constructor()");
  }

  ngOnInit(): void {
    console.log("LessonListPageComponent.ngOnInit()");
  }

  ngOnDestroy(): void {
    console.log("LessonListPageComponent.ngOnDestroy()");
  }
```

Navigate between page links, and check in DevTool console that each @Component is dynamically created, initialized, then destroyed.



[Optional] Step 6: add Fortawesome : angular wrapper for fontawesome icon library

To make you <a routerLink…> more graphical, add the “font-awesome” library, using the angular wrapper “fortawesome” (notice the spelling “font” <-> “fort”)

ng add @fortawesome/angular-fontawesome

Cf https://github.com/FortAwesome/angular-fontawesome

Add in the html page

```
<a routerLink="/"><fa-icon [icon]="faHome"/> Home</a>
```

And also add the corresponding field “faHome”, initialized from the import. Cf below lines 2 & 9.

```
import { Component } from '@angular/core';
import { faHome } from '@fortawesome/free-solid-svg-icons';

@Component({
  selector: 'app-lesson-edit-form',
  templateUrl: './lesson-edit-form.component.html'
})
export class LessonEditFormComponent {
  faHome = faHome;
}
```

The angular import + field declarations are very painful.

It is also possible to define the faLibrary once, then add like this:

```
<a routerLink="/"><fa-icon [icon]="['fas', 'home']"/> Home</a>
```

For this way of declaring, also edit in the main app/app.module.ts :

```
import {FaIconLibrary, FontAwesomeModule} from '@fortawesome/angular-
fontawesome';
import { fas } from '@fortawesome/free-solid-svg-icons';
import { far } from '@fortawesome/free-regular-svg-icons';
```

```
constructor(faIconLibrary: FaIconLibrary) {
  faIconLibrary.addIconPacks(fas, far);
}
```

Without the angular wrapper, icon should be added like this … This is what you find on internet, but it does not work (requires extra css / svg / fonts files ?).

```
<a routerLink="/"><i class="fas fa-home"></i> Home</a>
```

Step 7: create another component page “lesson-list-page.component.ts”, and add route “lesson-list”

This page will be filled later with a list of lessons. Currently, leave it empty.

Type in terminal

ng g c lesson-list-page

Then add in app.routing.module.ts

```
import {LessonEditFormComponent} from "./lesson-edit-form/lesson-edit-
form.component";
import {LessonListPageComponent} from "./lesson-list-page/lesson-list-
page.component";

const routes: Routes = [
  { path:'lesson-edit-form', component: LessonEditFormComponent },
  { path:'lesson-list', component: LessonListPageComponent }
];
```

Step 7: changing URL from typescript code, using Router

Edit you lesson-edit-form.component.html, add a button

```
<button type="button" class="btn btn-primary"
(click)="onClickSubmit()">Submit</button>
```

Then edit the corresponding lesson-edit-form.component.ts, to define the method callback, that changes the page dynamically from typescript.

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

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

onClickSubmit() {
  // could execute code (send save request to server)... then navigate
  this.router.navigate(['lesson-list']).then(res => {})
}
```

Check that when you click on the submit button, it changes the URL location to http://localhost:4200/lesson-list

Step 8: transmitting parameter to path

Create another component, “lesson-detail”

ng g c lesson-detail-page

Add a route declaration for it, taking an “id” in the path: “lesson/:id”

```
import {LessonDetailPageComponent} from "./lesson-detail-page/lesson-
detail-page.component";

const routes: Routes = [
  { path:'lesson-edit-form', component: LessonEditFormComponent },
  { path:'lesson-list', component: LessonListPageComponent },
  { path:'lesson/:id', component: LessonDetailPageComponent }
];
```

Add a button in you edit form, that navigate to the page “/lesson/1234”

```
<button class="btn btn-primary" [routerLink]="['/lesson', 1234 ]">Detail
for lesson 1234</button>
```

Notice routerLink is written with square bracket: [routerLink], so it interprets the attribute as an expression => need to escape literal with simple quotes ‘/lesson’

Here, as it is hardcoded value, you may also write it like constant attribute, without square-bracket, without array, and without simple-quote in value:

```
<button class="btn btn-primary" routerLink="/lesson/1234">Detail for
lesson/1234</button>
```

Step 9: get value of the path parameter “:id” from component (with ActivatedRoute)

Inject in constructor the angular ActivatedRoute:

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

Use it to get the id once from “snapshot.params”, and also subscribe to dynamically changing “params”

```
export class LessonDetailPageComponent {
  readonly initialId: number;
  id: number | undefined;

  constructor(activatedRoute: ActivatedRoute) {
    this.initialId = +activatedRoute.snapshot.params['id'];
    console.log('initialId:', this.initialId);
    activatedRoute.params.subscribe(currParams => {
      this.id = +currParams['id'];
      console.log('id:', this.id);
```

```
    });
  }
```

Add a button in your page that change the page to the next id page… This is a recap from question 8, using “Router”

In lesson-detail.component.html :

```
initialId: {{initialId}}
<br/>
id: {{id}}
<br/>
<button type="button" class="btn btn-outline-primary"
(click)="onClickGoNextPage()">Next</button>
```

in lesson-detail.component.ts:

```
onClickGoNextPage(){
  const nextId = (this.id)? this.id + 1 : 1;
  this.router.navigate(['/lesson', nextId]);
}
```

Notice that if you click on the button, the initialId is not changing (the component class is not reloaded), but if you change the value from the chrome URL toolbar, then the whole page is reloaded, so initialId is reset !

Step 10: configuring RouterModule useHash: true

To avoid reloading the whole application when the URL changes, you can use URL with a hash separator

Change your app.routing.module.ts

```
@NgModule({
  imports: [RouterModule.forRoot(routes, {useHash: true})],
```

Instead of default

```
@NgModule({
  imports: [RouterModule.forRoot(routes)],
```

Notice that now, url becomes http://localhost:4200/#/lesson/1234

instead of http://localhost:4200/lesson/1239

When changing only the part after the hash, your browser will consider that you are just scrolling inside the same page, and will not reload the page.

Change the URL, and see the difference with before (question 9). The initialId should not change any more (and the page is not reloaded). Check in Chrome DevTool > Network tab.

This behavior is also recommended for exposing your app from a static web page server (CDN, apache, nginx…), because the browser should always do “http GET /” query of root page http://localhost:4200/index.html (corresponding to http://localhost:4200/) and not the non-existing

page “http://localhost:4200/lesson/1239”. Internally, “ng serve” pretends this page exists, and return the index.html page instead (!!)

Step 11: Working with Service, to transmit global state variable (hidden from URL)

From your terminal, create a service “user-settings”:

ng g s user-settings

Add a field in the generated class: src/app/user-settings.service.ts

```
export class UserSettingsService {
  lastLessonId: number|null = null;
```

Inject it in the constructor in both components, and set/get the value:
in lesson-edit-form.component.ts:

```
import {UserSettingsService} from "../user-settings.service";

constructor(private router: Router, private userSettingsService:
UserSettingsService) {}

onClickSubmit() {
  this.userSettingsService.lastLessonId = 1234;

  // could execute code (send save request to server)... then navigate
  this.router.navigate(['lesson-list']).then(res => {})
}
```

in lesson-list.component.ts

```
constructor(private userSettingsService: UserSettingsService) {
  console.log("get lastLessonId:", userSettingsService.lastLessonId);
}
```

Verify that this value is transmitted (hidden from URL), and logged in the DevTool console.

You can also verify that when reloading the page (F5), the value is lost.

Step 12 : persisting JSON value in localStorage

If you want to persist some value on the local storage of your PC, with a lifetime longer than an angular Service (a page reload), you can change your service to transparently save/reload to localStorage (this is a w3c standard spec, not specific to Angular).

First, change the field to be private, with an underscore prefix: “private _lastSessionId”, and define get/set to it, so that is transparently call get/set methods, without changing code that used it.

```
private _lastLessonId: number|null = null;
get lastLessonId(): number|null { return this._lastLessonId; }
set lastLessonId(p: number|null) { this._lastLessonId = p;
this.saveInfoToLocalStorage(); }
```

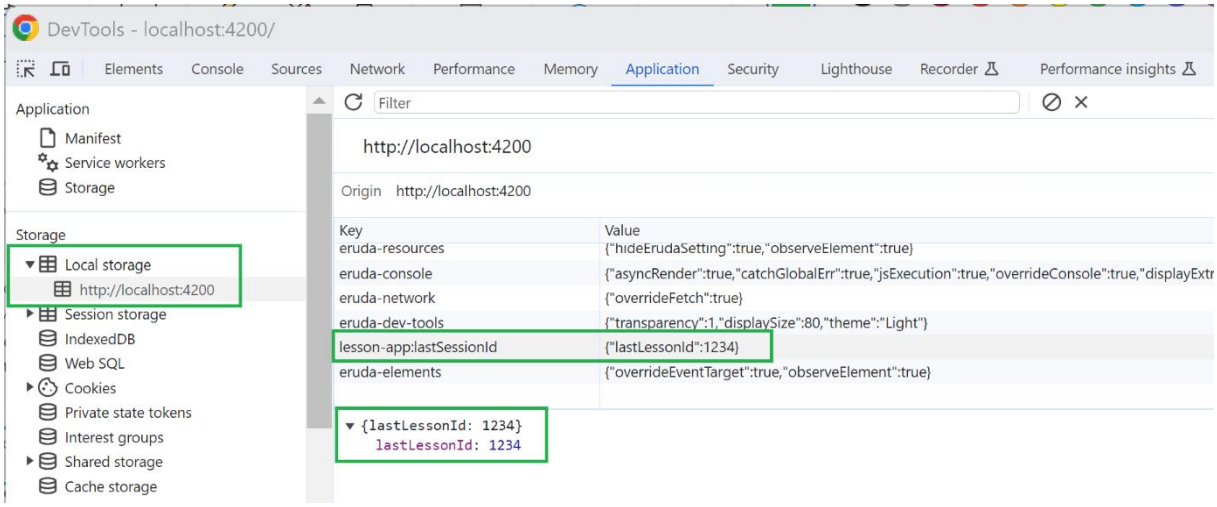
For clean code, declare what will be loaded/saved in json, using a typed interface:

```
const STORAGE_KEY = 'lesson-app:lastSessionId';
interface StoredInfo {
  lastLessonId?: number|null;
}
```

Then add a load/save mechanism to use the builtin “window.localStorage” object:

```
  constructor() {
    const storedInfo = this.loadInfoFromLocalStorage();
    this.lastLessonId = storedInfo.lastLessonId || null;
  }

  loadInfoFromLocalStorage(): StoredInfo {
    const storedJson = window.localStorage.getItem(STORAGE_KEY);
    let res = (storedJson)? JSON.parse(storedJson) : { lastLessonId: null
};
    console.log('reading localStorage item ' + STORAGE_KEY, res)
    return res;
  }

  saveInfoToLocalStorage() {
    const info: StoredInfo = { lastLessonId: this.lastLessonId };
    let json = JSON.stringify(info);
    console.log('writing localStorage item ' + STORAGE_KEY, info)
    window.localStorage.setItem(STORAGE_KEY, json);
  }
```

Open Chrome DevTool to check that the value is still available after refreshing page, or even after restarting Chrome.



Coffee Break & Next TD Steps

You can draw a mind map, to summarize what you have done so far on life-cycles of Component < Service = Angular App < LocalStorage,

and Route / routerLink / Router / ActivatedRoute / RouterModule hash