Advanced Angular

Labs

zenika
ARCHITECTURE INFORMATIQUE

Prerequisites

Installation

You should install the following on your system

- NodeJS (Version >= 10.9.x)
- Npm (It will be installed at the same time as NodeJS)
- Git

Lab 1: initiate your project

Installation

In this lab, you will initiate you **Angular** application with the Angular CLI. This project will be used along all labs.

First, you need to install the **Angular CLI** :

```
npm install -g @angular/cli
```

According to your system, you might need to run the command as an administrator. Also, you might already have the Angular CLI installed. If that's the case, make sure it is at least in version 9. If not, upgrade it.

Then create your application throught the installed CLI :

```
ng new zenika-ng-website
```

You will be displayed some options for your project. Add the routing module and choose the scss preprocessor.

Replace the generated `app` directory by the `app` directory given by your trainer. It contains the code of a basic application.

Include bootstrap by adding the code below in your `index.html` file:

```html
<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css"
rel="stylesheet">
```

To finish, launch your app with :

```
npm start
```

The server is composed of some randomness, 10% of request will send error 500

Lab 2: Test

Unit tests

In this lab we will add some tests to our application.

We will start with unit tests. They are the most important and the most common ones you can find in any app. Some unit tests are already defined in our app, go check them if you have any difficulties to start writing your own ones.

First make sure everything is working with your unit tests by launching them throught the Angular CLI :

```
npm run test
```

Open the file `product.component.spec.ts` .

Sometimes it can be hard to know what to test in a component. A good starting point is to ask ourselves :

`What is my component in charge of ?`

Our product component is in charge of two things :

- it should emit an event when its button is clicked
- it should add a specific class to its element when the product stock is equals to 1.

Write unit tests for these two functionalities.

Once you're done, open the file `home.component.spec.ts` .

We ask ourselves the same question. Our home component is a bit more complex than our product component, but we can easily isolate some functionalities to test :

- it should display the basket total
- it should display the product list
- it should not display a product with no stock

Write some unit tests for these functionalities too.

E2E tests

Unit tests are the most common tests in any application. But sometimes you want to test your application in its entirety, especially on some critical features. One way to do that is through e2e tests.

e2e tests are not defined at the same place as unit tests. Open you `e2e` folder and inspect what's already in there.

Launch your e2e tests with

```
npm run e2e
```

One of the already defined e2e test should fail, fix it.

We will add another e2e to test one of the most critical feature of our app : the checkout.
We consider that the checkout is working correctly if we are able to :

- go to our app
- add some products to our basket
- go to our basket page
- click on validate
- be redirected to our home page
- see a total of 0,00

To test it, create two new files:

- `basket.e2e-spec.ts`
- `basket.po.ts`

Use the page object file to write helper functions to keep our test code clean and readable. Look at `app.po.ts` if you have difficulties getting started.

One example of helper functions would be to :

- navigate to the basket page
- click on validate

Bonus

When testing a component, it's important to remember that a component is a typescript class and a template. Most of the time we only test our class methods. Update the unit tests you just wrote to make sure you're actually testing your template too.

Lab 3: Best practices

As you may seen, the code of the app smell in a bad way.

The goal of this labs is to refactor all the code.

Don't forget to refactor with unit-testing !

Step 1 - Make some services

The application use `window` object in various part of the project, this is very bad, especially when you have a Server Side Rendering (Server doesn't have `window` object available).

We have to replace this bad design by doing some services !

You may have notice two word that came recursively in PO's features: Catalog and Basket. They look like perfect to split our operational code into two distinct services.

Create the two services in the application:

- `ng generate service home/catalog`
- `ng generate service basket/basket`

Then progressively replace all operational code to put is into services. Ideally start by unit testing component and service before write code.

You can move the code of the `fakeServer` in the newly created `home/catalog` service.

Step 2 - organize you files

All component is in the root of the app and this is not sustainable. As long as the codebase grow, find the component to work on will be more and more difficult.

Each child component of a parent component should be a specific folder in the parent component folder.

If a component is shared by two parent component, depend of the case, but you can create a `commons` folder.

In this project Product component can be moved into home component folder.

Lab 4 - Architecture

To improve performance, set changeDetection to `onPush` on the HomeComponent. As you may see, Product should not display. Fix this !

Every data of components are initialized in the constructor. It's a bad pattern, you must use Angular hooks like ngOnInit.

Lab 5 - i18n

The application is available in only one language, english.

The goal of this lab is to translate each part of text in french.

Initialize ngx-translate

Add ngx-translate to the project with `npm i @ngx-translate/core`.

import the module into your application.

Set the default lang in english(en) in `app.component` via the *TranslateService*.

Export some labels and texts into en translation definition (*setTranslation*).

Fix your tests by importing the translate module into the test bed.

Switch language button

Set a translation definition (with *setTranslation*) in another language of your known (French for example - fr).

Add two button in the menu component.

Write tests about these two button to be sure these buttons will make the application use the given language.

Implement what is it describe in your tests.

Lab 6 - design

In this lab, we will initialize a design system based on material desgin.

Angular-Material

- Add material-ui dependencies to the projection
- Add a build-in Angular material theme in your CSS
- Create a module to initialize your design system : `ng generate module design-system`
- Import into the module all material component you need and don't forget to export it
- Import the designSystemModule into the main module

In product component change the button to use a mat-button (raised with primary color) component. Don't forget to remove the bootstrap classes.

Animation

When a product is the last available, start an animation of your choice. Show your creativity to the instructor !

Bonus

The application use bootstrap for styling. Feel free to remove all trace of bootstrap by switching to angular-material component.

Lab 7 - Forms

The goal of this lab is to build the better reactive form for the best User eXperience.

- import `ReactiveFormsModule`
- Build the Schema of your form in basket component :
    - Name is required

    - Address is require
    - CreditCard is required and should follow `/^\d{3}-\d{3}$/` pattern (ex: 123-123)

- Disable submit button when form is invalid
- Display an error message on each component when the field is invalid and is dirty

Lab 8 : Router

Usually, e-commerce website is compose of a catalog and purchase. Most of your visitor will consume the the catalog so don't make them load the purchase part of your app for no reason.

- When a user try to access to an URL that not exist in your app, display the catalog
- Split code into two modules, one for displaying the catalog and the other for the purchase part. Don't forget to have consistent naming.
- Lazy load the purchase part of your app
- Change the strategy to PreloadAllModules

Lab 9 : RxJS

As you may seen, the instructor made a fake-server based on promise to simulate server calls.

Replace this code by creating an observable with hand (use of `from` is prohibited).

Lab 10 : Universal

Follow the step in the slides to have a node.js server that render server side your angular application.

Lab 11 : NGRX

In this lab we will migrate the basket management of the app into a store.

- add ngrx to your project `ng add @ngrx/store`
- make an enum that contain 'addToBasket' action type
- create a reducer that add the product in the payload into the basket in the store
- import ngrx into the main module with your reducer
- access to the basket data into your components