Labs

zenika
ARCHITECTURE INFORMATIQUE

Prérequis

Nécessaire pour les labs:

- Node.js / npm
- GIT

Lab 1: Ma première application Angular

Tout au long des labs, vous allez mettre en place une application Angular de ecommerce pour vendre de la bière.

Démarrer le projet

Clonez le project imt-ecommerce présent sur github. Pour cela, ouvrez un terminal et lancez la commande suivante:

```
git clone https://github.com/Orodan/imt-ecommerce.git
```

Déplacez vous dans le dossier télécharger et installez les dépendances:

```
npm install
```

Pour finir, lancer le projet:

```
npm start
```

Angular démarre pour vous un serveur web qui fournit votre application à l'adresse suivante:

```
http://localhost:4200
```

Mes premiers composants

Ouvez votre application avec votre éditeur de texte préféré (pour votre bien, celui-ci doit comprendre typescript, comme vscode par exemple. Libre à vous d'ouvrir un fichier typescript avec le block-note mais dans ce cas vous vous débrouillerez seul-e pour le reste de la session).

Prenez le temps de vous familiariser avec la structure du projet. Pour l'instant celui-ci contient très peu de code, présent dans le dossier `app`.

Ouvrez le fichier `app.component.html`

L'intégralité de votre application tient pour l'instant dans ce fichier, qui est pas conséquent assez volumineux et difficile à lire au premier abord. Vous allez découper votre application en plusieurs composants pour améliorer la lisibilité et mieux respecter le principe de `single responsibility`.

Créez un composant menu. Pour cela, la manière la plus simple et recommandée et d'utiliser le CLI d'Angular. A la racine de votre projet, lancez la commande suivante:

```
npx ng generate component menu
```

Déplacer dans votre composant `menu` tout le code du menu de votre application (tout le code de `<nav>` à `</nav>` inclue)

Créez un composant footer

```
npx ng generate component footer
```

Déplacez dans votre composant `footer` tout le footer de votre application (tout le code de `<footer>` à `</footer>` inclue)

Déplacer le style de votre footer dans votre composant

Utiliser votre `footer` component à l'endroit ou se trouver le code de votre navbar dans le fichier `app.component.html`.

Interpolation

Votre composant `app` afficher un header avec un titre. Vous allez plutôt utiliser le système d'interpolation d'Angular pour éviter d'avoir votre titre définit en dur dans votre fichier `app.component.html`.

Ouvrez votre fichier `app.component.ts` et définissez une propriété `title` possédant comme valeur `Bienvenue sur IMT Ecommerce` (sentez vous libre de mettre le titre qu'il vous plait, après tout c'est votre appli).

Remplacez le titre en dur dans le fichier `app.component.html` par votre propriété `title`. Pour rappel, l'interpolation s'utilise avec les doubles accolades dans un template Angular :

```
{{ expression }}
```

Vérifiez que votre application fonctionne toujours bien: vous devez avoir exactement le même affichage à l'écran.

Lab 2: Property & Event binding

Property binding

Pour ce deuxième lab, vous allez utiliser le property et l'event binding pour communiquer d'un composant à un autre.

Commencez par créer une propriété `beers` dans votre composant `app` contenant la liste des bières fournit en ressource.

Ensuite créez un composant `product`

```
npx ng generate component beer
```

Créez une propriété `data` dans ce composant de type `any`. Annotez cette propriété avec l'annotation `@Input()` provenant du package `@angular/core`. Cela permettra à un composant parent de modifier sa valeur.

Placer dans ce composant le template d'un produit (la balise `<div>` portant la classe css `product-item` - incluez la balise `<div>` en question) dans le fichier `app.component.html`.

Retirez les class css `col-md-3` `col-sm-6` de la balise `<div class="product-item">` dans votre composant `beer`.

Remplacer les données en dur que vous venez de copier par des données provenant de la propriété `data` de votre composant.

Exemple:

```
<h3>Avery Brown Dredge</h3>
```

devient

```
<h3>{{ data.title }}</h3>
```

Pour finir, utilisez le composant `beer` dans le template de votre composnat `app`. Ajoutez y également les classes css `col-md-3` `col-sm-6`:

```
<app-beer class="col-md-3 col-sm-6"></app-beer>
```

Répétez son utilisation autant de fois que nécessaire pour afficher toutes vos bières (fournissez une valeur différente à la propriété `data` à chaque fois).

Exemple:

```
<app-beer [data]="beers[0]"></app-beer>
```

Event binding

Maintenant que nous utilisons le property binding pour afficher nos bières, il serait intéressant d'être capable des les ajouter à un panier pour les acheter.

Créez un événement dans votre composant `beer` se nommant `addToBasket`. Cet évènement transportera les données de la bière sélectionnée.

Créez une méthode `onClick` dans votre composant qui émet l'événement que vous venez de créer, avec les données de votre bière.

Ecoutez l'événement `click` du bouton `Add to basket` et appelez votre méthode `onClick`.

Vous avez maintenant créé un événement custom émit lorsque votre utilisateur clique sur le bouton `Add to basket` d'une de vos bières. Il vous reste maintenant à gérer cet évènement.

Dans votre composant `app`, créez une propriété `basket` qui a comme valeur par défaut un tableau vide.

Créez une méthode `add` prenant en paramètre une bière et ajoutant cette bière à votre panier `basket`.

Ecoutez l'évènement `addToBasket` sur les composants `product` dans votre fichier `app.component.html` et liez lui la méthode `add`.

Total

Il serait intéressant d'afficher le total du montant de votre panier sur votre page.

Créez une méthode `getTotal` qui parcourt toutes les bières de votre panier et renvoie la somme de leur montant.

Remplacez les `...` de `Your basket amounts to ...` dans votre fichier `app.component.html` par un appel à `getTotal`.

Lab 3: Directives

Dans ce lab, vous allez utiliser les directives fournis par Angular pour modifier le DOM de votre application.

*ngfor

Dans votre composant `app`, vous répétez plusieurs fois votre composant `beer` pour afficher vos bières. Ce n'est pas très efficace et n'est pas viable si vous devez afficher 40 bières plutôt que 4.

Utilisez la directive `*ngFor` pour répérer votre composant `beer` autant de fois que vous avez de bières à afficher.

*ngIf

Jusqu'à maintenant, votre utilisateur pouvait ajouter autant de bières qu'il le souhaitait dans son panier. Cela rend sûrement très heureux votre utilisateur, mais niveau business vous risquez de vous retrouvez avec des commandes que vous ne pourrez pas honorer.

Pour éviter cela :

- Modifier la méthode `add` de votre composant `app` pour diminuer le stock de la bière ajoutée à votre panier.
- Faite disparaitre de votre écran la bière qui n'a plus de stock à l'aide de la directive `*ngIf`.

Attention, vous ne pouvez pas placer deux directives structurantes comme le `*ngif` et le `*ngFor` sur le même élément. Angular ne le permet pas.

Le premier réflèxe qu'on peut avoir dans ce cas est de placer une div supplémentaire portant la deuxième directive. Cela fonctionne. Cependant, cela peut casser le style de votre application et sémantiquement votre div n'a aucun intérêt à se trouver dans votre DOM.

Pour palier à ce soucis, il existe un composant Angular: le `ng-container`. Il a le même comportement qu'une div mais ne possède aucune représentation dans le DOM géréné par Angular.

Exemple:

```
<ng-container *ngIf="expression ">...</ng-container>
```

ngClass

Maintenant que vous faites disparaitre les bières que vous n'avez plus en stock, vous allez afficher un signal à votre utilisateur quand vous n'avez plus qu'un seul exemplaire d'une bière en stock !

Dans votre composant `beer`, rajoutez le style css suivant :

```
.last {
  background-color: #f80606a3;
}
```

Toujours dans le composant `beer`, ajouter la classe css `last` à la div portant déjà la classe css `product-item`, niquement quand la bière associée n'a plus qu'un exemplaire en stock.

Lab 4: Material

A travers les schématics, Angular nous donne accès à des commandes pour facilement rajouter des éléments dans notre projet. C'est le cas pour la librairie `@angular/material`.

Commencez par rajouter `@angular/material` dans votre projet avec la commande:

```
npx ng add @angular/material
```

Sélectionnez le thème que vous souhaitez, répondez `yes` aux autres questions.

Inspectez ce qu'a réalisé cette commande dans votre projet.

Maintenant qu'@angular/material est installé dans votre projet, vous allez utiliser le `mat-raised-button` dans votre composant `beer`.

Pour pouvoir utiliser ce bouton, importez la module `MatButton` dans votre module `app`.

Utilisez le `mat-raised-button` avec la couleur `primary` sur votre bouton `Add to basket`. N'oubliez pas de retirer les class boostrap que vous aviez sur ce bouton, vous n'en avez plus besoin.

Pour terminer, utilisez autant de composants @angular/material que vous le souhaitez pour améliorer le design de votre application. Pour rappel, la documentation de cette librairie est disponible à `material.angular.io`.

Lab 5: Pipe

Votre application affiche les prix de vos bières. Ces prix ont le symbole `$` inscrit en dur dans le template de votre composant `beer`. Vous allez changer cela grace aux pipes d'Angular.

Utilisez le pipe `currency` sur le prix de chacune de vos bières.

Affichez un `€` plutôt que `$` en fournissant un paramètre à votre pipe `currency`.

Lab 6: Services

Pour améliorer l'architecture de votre application, avoir des composants avec moins de responsabilités et être capable de plus facilement partager des données, vous allez créer des services dans votre application.

BeersService

Commencez par créer un nouveau service `BeersService` :

```
npx ng generate service beers
```

Le service créé va être en charge de fournir votre liste de bières, plutôt que cette liste soit uniquement définie dans votre composant `app`.

Déplacez la variable `beers` de votre composant `app` dans votre service nouvellement créé.

Injectez le service `BeersService` dans votre composant `app`.

Créez un getter `beers` dans votre composant qui renvoie la liste des bières contenue dans votre service. Pour créez un getter en typescript:

```
class myClass {
  get myList() {
    return this.myService.myList;
  }
}
```

Assurez vous que votre liste de bières est toujours affichée à l'écran.

BasketService

Créez un service `BasketService` en charge de la gestion de votre panier.

Déplacez la variable `basket` de votre composant `app` dans votre service nouvellement créé.

Créez dans votre service une méthode `getTotal` renvoyant le montant total de votre panier.

Créez dans votre service une méthode `add` prenant en paramètre une bière et l'ajoutant dans votre panier tout en diminuant son stock de 1.

Injectez ce service dans votre composant `app`.

Utilisez les méthodes de ce service pour interagir avec votre panier.

Assurez vous que vous pouvez toujours ajouter des bières à votre panier et que le montant total affiché à l'écran est toujours à jour.