SEG3502 – Lab 3

Angular – Composants

L'objectif de ce laboratoire est d'en savoir plus sur Angular en implémentant une application Carnet d'adresses (Address Book). Plus précisément, vous apprendrez:

- la création et la mise en œuvre de composants Angular,
- l'implémentation de formulaires basés templates,
- la communication entre composants avec des Inputs, des événements Outputs et des Observables

Le code source du laboratoire est disponible dans le référentiel Github https://github.com/stephanesome/angComponents.

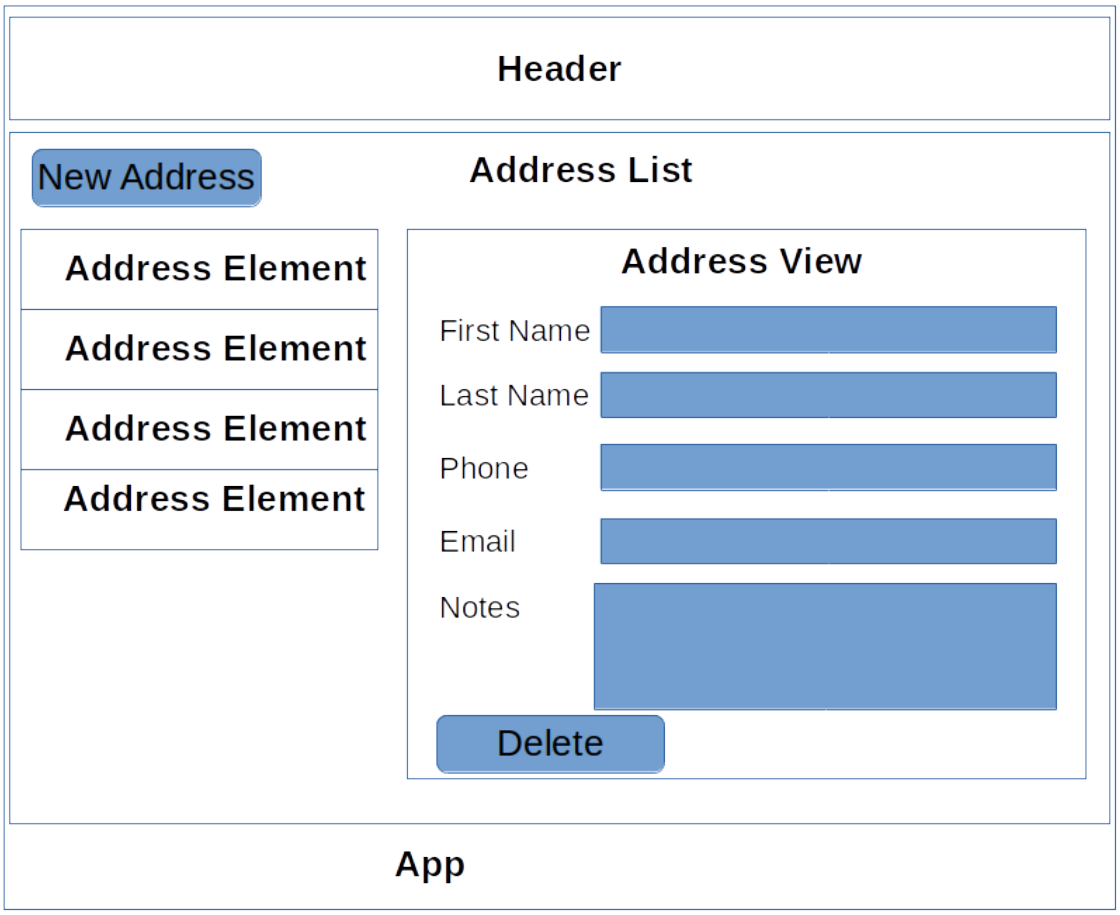
Application Address Book

Nous mettrons en œuvre une application pour enregistrer le carnet d'adresses d'un utilisateur. Les principales caractéristiques de l'application sont les suivantes.

- Afficher une liste des entrées du carnet d'adresses enregistrées.
- Ajouter des entrées au carnet d'adresses.
- Afficher les détails des entrées du carnet d'adresses. Chaque entrée comprend un prénom, un nom, un numéro de téléphone, un e-mail et des notes (first name, last name, phone number, email, notes).
- Supprimer des entrées du carnet d'adresses.

Conception des composants

La conception de l'interface est indiquée pour toute application. La conception de l'interface permet de déterminer la disposition de l'interface et de s'assurer qu'elle supporte les utilisateurs de manière efficace. Cela nous permet également de comprendre les composants nécessaires pour une application Angular. Vous trouverez ci-dessous le wireframe de l'application.



Nous identifions les composants suivants et leurs relations.

- Composant *App*, le composant racine de l'application.
- Composant *Header* sous-composant de *App*.
- Composant *Address List* sous-composant de *App*. Ce composant contient une liste d'adresses et fournit des responsabilités associées telles que l'ajout d'une adresse à la liste ou la suppression d'une adresse de la liste.
- Composant *Address Element*, sous-composant de liste d'adresses. Ce composant est un élément sélectionnable pour une entrée de carnet d'adresses. Cliquer dessus affiche les détails de l'entrée correspondante.
- Composant *Address View* sous-composant de la liste d'adresses. Ce composant permet d'afficher et de modifier une entrée du carnet d'adresses.

Configuration de l'application

Créez une nouvelle application angular (`ng new address-book`). Le routage n'est pas nécessaire et nous utiliserons CSS comme format de feuille de style.

Nous utiliserons la bibliothèque CSS Angular Bootstrap. Exécutez la commande `ng add @ng-bootstrap/ng-bootstrap` dans le dossier racine du projet pour ajouter Bootstrap au projet..

Composant App

Le composant racine *App* contient le composant *Header* ainsi que le composant *Address List*. Spécifiez le template HTML (`src/app/app.component.html`) comme suit.

```
1. <app-header></app-header>
2. <div class="container">
3.     <app-address-list></app-address-list>
4. </div>
```

Composant Header

Le composant *Header* inclut uniquement la personnalisation de notre application. Exécutez la commande `ng generate component header` dans le dossier racine du projet. Angular CLI générera des fichiers pour un nouveau composant avec la sortie suivante.

```
CREATE src/app/header/header.component.css (0 bytes)
CREATE src/app/header/header.component.html (21 bytes)
CREATE src/app/header/header.component.spec.ts (628 bytes)
CREATE src/app/header/header.component.ts (275 bytes)
UPDATE src/app/app.module.ts (467 bytes)
```

Les fichiers générés incluent:

- la classe de composant dans `src/app/header/header.component.ts`,
- test du composant dans `src/app/header/header.component.spec.ts`,
- le template HTML de composant dans `src/app/header/header.component.css` and
- le fichier de feuille CSS du composant dans `src/app/header/header.component.css`.

Modifiez le template HTML comme suit

```
1. <nav class="navbar navbar-fixed-top">
2.   <div class="container-fluid">
3.     <div class="navbar-header">
4.       <a href="#" class="navbar-brand">Your Address Book</a>
5.     </div>
6.   </div>
7. </nav>
```

Composant Address List

Génèrez le composant de liste d'adresses (`ng generate component address-list`).

Modèle d'entrée d'adresse

Nous avons besoin d'une classe de modèle pour les entrées du carnet d'adresses. Ceci est juste une classe Typescript normale. Nous utilisons Angular CLI pour générer avec la commande `ng generate class address-list/address-entry`.

Modifiez le fichier `src/app/address-list/address-entry.ts` comme suit.

```
1.  export class AddressEntry {
2.    public firstName: string;
3.    public lastName: string;
4.    public phone?: string;
5.    public email?: string;
6.    public notes?: string;
7.
8.    constructor(firstName: string, lastName: string, phone?: string, email?: string, notes?: string) {
9.      this.firstName = firstName;
10.     this.lastName = lastName;
11.     this.phone = phone;
12.     this.email = email;
13.     this.notes = notes;
14.   }
15. }
```

La définition de classe spécifie des champs d'entrée d'adresse et un constructeur.

Template du composant Address List

Le template HTML du composant *Address List* doit présenter un bouton pour ajouter une nouvelle entrée, une liste d'éléments de liste d'adresses (*Address List Element*) et un composant pour l'affichage d'adresses (*Address View*). Créez un template initial comme suit dans `src/app/address-list/address-list.component.html`.

```
1.  <div class="row">
2.    <div class="col-md-12">
3.      <button >New Address</button>
4.    </div>
5.  </div>
6.  <hr>
7.  <div class="row">
8.    <div class="address-list col-md-3" *ngIf="addresses.length > 0">
9.      <app-address-list-element
10.       *ngFor="let address of addresses"
11.       [address]="address"
12.     >
13.     </app-address-list-element>
```

```
14.   </div>
15.   <div class="col-md-9">
16.     <app-address-view></app-address-view>
17.   </div>
18. </div>
```

Lines 2-4 specifies the button to add a new address entry. In lines 8-12, we use a NgIf directive to loop over a collection of addresses and create a Address List Element component for each, passing the current address as input. Lines 15-17 embed an Address View component.
Les lignes 2 4 spécifient le bouton pour ajouter une nouvelle entrée d'adresse. Aux lignes 8 à 12, nous utilisons une directive ***NgFor*** pour boucler sur une collection d'adresses et créer un composant *Address List Element* pour chacune, en passant l'adresse actuelle en entrée. Les lignes 15 à 17 incorporent un composant *Address View*.

Classe de composant pour Address List

Ajoutez une propriété dans la classe AddressListComponent pour la collection d'adresses.

```
1.  addresses: AddressEntry[] = [];
```

Assurez-vous que la classe AddressEntry est ajoutée aux imports.

Address List Element

Générez un composant d'élément d'adresse
(ng generate component address-list/address-list-element). Un élément de liste d'adresses affiche un label identifiant l'entrée d'adresse. Nous utiliserons le nom complet construit à partir des champs *firstName* et *lastName*.

- Ajoutez la fonction suivante à la classe AddressListElementComponent de `src/app/address-list/address-list-element/address-list-element.component.ts`.

```
1.   getFullName(): string {
2.     return `${this.address.firstName}, ${this.address.lastName}`;
3.   }
```

- Nous devons également déclarer une propriété *address* dans la classe AddressListElementComponent. Cette instance sera transmise à l'instance de AddressListElementComponent par le parent AddressListComponent puisque les adresses doivent être détenues par cette classe.

  Nous utilisons la liaison d'entrée (input binding) pour cela. Ajoutez la déclaration

```
1.  @Input() address: AddressEntry | undefined;
```

  à la classe AddressListElementComponent. Vous devez également importer Input de @angular/core ainsi que AddressEntry de ./address-entry.

- Nous souhaitons également modifier l'apparence d'un Address List Element lorsqu'il est sélectionné. Ajoutez la déclaration suivante à la classe AddressListElementComponent.

```
selected = false;
```

Nous pourrons utiliser cette propriété pour définir les propriétés CSS du composant en fonction de leur statut sélectionné.

- Modifiez le modèle HTML d'élément de liste d'adresses (`src/app/address-list/address-list-element/address-list-element.component.html`) comme suit.

```
<button class="address-list-element ui positive"
        [ngClass]="{'address-list-element-selected': selected}">
  {{getFullName()}}
</button>
```

Le style est contrôlé par le directive ngClass qui ajoute la propriété address-list element-selected lorsque la propriété selected est true.

- Modifiez la feuille de style CSS de Address Element comme suit.

```
.address-list-element {
  border: 1px solid ;
  padding: 10px 2px;
  font-size: 16px;
  cursor: pointer;
  width: 200px;
  display: inline-block;
}
.address-list-element:active {
  color: green;
}
.address-list-element-selected {
  background-color: green;
  color: white;
  font-weight: bold;
}
```

Composant Address View

Génère un composant de vue d'adresse (Address View)
(ng generate component address-list/address-view).

Template Address View

Ce composant affiche un formulaire pour les entrées d'adresses. Modifiez le template HTML (`src/app/address-list/address-view/address-view.component.html`) comme suit.

```
<form class="ui large form segment">
```

```
2.    <h3 class="ui header">Address Details</h3>
3.    <fieldset [disabled]="!edit ? 'disabled' : null">
4.      <div class="form-group">
5.        <label for="firstName">First Name</label>
6.        <input class="form-control" name="firstName" id="firstName" [(ngModel)]="address.firstName">
7.      </div>
8.      <div class="form-group">
9.        <label for="lastName">Last Name</label>
10.       <input class="form-control" name="lastName" id="lastName" [(ngModel)]="address.lastName">
11.     </div>
12.     <div class="form-group">
13.       <label for="phone">Phone Number</label>
14.       <input class="form-control" name="phone" id="phone" [(ngModel)]="address.phone">
15.     </div>
16.     <div class="form-group">
17.       <label for="email">Email</label>
18.       <input class="form-control" name="email" id="email" [(ngModel)]="address.email">
19.     </div>
20.     <div class="form-group">
21.       <label for="notes">Notes</label>
22.       <textarea class="form-control" rows="4" cols="60" name="notes" id="notes"
      [(ngModel)]="address.notes"></textarea>
23.     </div>
24.   </fieldset>
25.   <div class="btn-toolbar" role="toolbar">
26.     <button class="btn btn-danger" name="delete" (click)="delete()">Delete</button>
27.     <button  (click)="toggleEdit()"
28.              class="btn btn-secondary m-auto" *ngIf="edit">
29.       Protect
30.     </button>
31.     <button  (click)="toggleEdit()"
32.              class="btn btn-secondary m-auto" *ngIf="!edit">
33.       Edit
34.     </button>
35.   </div>
36. </form>
```

Nous spécifions des champs de formulaire pour l'entrée d'adresse (lignes 6, 10, 14, 18, 22) et nous lions avec des propriétés dans la classe de composant.

Assurez-vous que FormsModule est importé dans l'AppModule (`src/app/app.module.ts`) et ajouté aux ***imports*** du décorateur @NgModule.

Nous ajoutons également un bouton pour supprimer l'entrée (ligne 26) et un bouton pour basculer l'édition (lignes 27-34). L'événement cliquez sur le bouton *Delete* est géré par la fonction delete du composant. Afin d'afficher un bouton qui bascule de *Edit* à *Protect* lorsqu'on clique dessus, les lignes 27-34 spécifient deux éléments de bouton. Aux lignes 27-30, un bouton *Protect* est rendu lorsque la valeur edit est true aux lignes 31-34 un bouton *Edit* est rendu autrement.

Classe Address View

La classe du composant Address View, AddressViewComponent déclare les propriétés et les fonctions nécessaires au modèle HTML. Modifiez AddressViewComponent de `src/app/address-list/address-view/address-view.component.ts` comme suit.

```
1.  import {Component, EventEmitter, Input, OnInit, Output} from '@angular/core';
2.  import {AddressEntry} from '../address-entry';
3.
4.  @Component({
5.    selector: 'app-address-view',
6.    templateUrl: './address-view.component.html',
7.    styleUrls: ['./address-view.component.css']
8.  })
9.  export class AddressViewComponent implements OnInit {
10.   @Input() address!: AddressEntry;
11.   @Output() fireDelete: EventEmitter<AddressEntry> = new EventEmitter();
12.   edit = false | undefined;
13.
14.   constructor() { }
15.
16.   ngOnInit(): void {
17.   }
18.
19.   toggleEdit(): void {
20.     this.edit = !this.edit;
21.   }
22.
23.   delete(): void {
24.     this.fireDelete.emit(this.address);
25.   }
26. }
```

La liaison d'entrée (input binding) est utilisée à la ligne 10 pour transmettre l'adresse actuelle à modifier/afficher au composant. Un événement de sortie (output event - line 11) notifie le composant parent lorsque l'utilisateur clique pour supprimer l'adresse. L'événement est déclenché dans la fonction delete (lignes 23-25).

Style de Address View

Modifiez la feuille de style AddressViewComponent de `src/app/address-list/address-view/address-view.component.css` comme suit.

```
1.  .address-view-elt {
2.    vertical-align: top;
3.    padding: 15px;
4.  }
5.  .address-view-btns {
6.    padding-top: 20px;
7.  }
8.  .address-view-elt {
```

```
    padding: 10px;
}
```

Ajout d'une Address Entry

Nous associons un gestionnaire au bouton *New Address* dans le composant Address List pour ajouter une Address Entry. Modifiez AddressListComponent (`src/app/address-list/address-list.component.html`) pour qu'il soit comme suit.

```
1.  <div class="row">
2.    <div class="col-md-12">
3.      <button class="btn btn-success" (click)="addAddress()">New Address</button>
4.    </div>
5.  </div>
6.  <hr>
7.  <div class="row">
8.    <div class="address-list col-md-3" *ngIf="addresses.length > 0">
9.      <app-address-list-element
10.       *ngFor="let address of addresses"
11.       (click)='select(address)'
12.       [address]="address"
13.     >
14.     </app-address-list-element>
15.   </div>
16.   <div class="col-md-9">
17.     <app-address-view></app-address-view>
18.   </div>
19. </div>
```

Nous avons ajouté un gestionnaire d'événements pour ***click*** à la ligne 3.

Modifiez la classe de composant pour ajouter les fonctions addAddress et select, ainsi que la propriété currentAddress.

```
1.  export class AddressListComponent implements OnInit {
2.    addresses: AddressEntry[] = [];
3.    currentAddress: AddressEntry | null = null;
4.    constructor() { }
5.
6.    ngOnInit(): void {
7.    }
8.
9.    select(address: AddressEntry): void {
10.     this.currentAddress = address;
11.   }
12.
13.   addAddress(): void {
14.     const newAddress = new AddressEntry('New', 'Entry');
15.     this.addresses = [newAddress, ...this.addresses];
16.     this.select(newAddress);
17.   }
```

```
18. }
```

Modifiez AddressListComponent (`src/app/address-list/address-list.component.html`) pour transmettre une instance sélectionnée de AddressEntry au composant Address View.

```
1.  <div class="row">
2.    <div class="col-md-12">
3.      <button class="btn btn-success" (click)="addAddress()">New Address</button>
4.    </div>
5.  </div>
6.  <hr>
7.  <div class="row">
8.    <div class="address-list col-md-3" *ngIf="addresses.length > 0">
9.      <app-address-list-element
10.       *ngFor="let address of addresses"
11.       [address]="address"
12.       (click)='select(address)'
13.     >
14.     </app-address-list-element>
15.   </div>
16.   <div class="col-md-9">
17.     <app-address-view *ngIf="currentAddress !== null"
18.                       [address]="currentAddress"
19.                       (fireDelete)='deleteCurrent()'></app-address-view>
20.   </div>
21. </div>
```

Nous associons également une fonction gestionnaire à l'événement fireDelete émis par le composant Address View. Ajoutez la fonction gestionnaire d'événement à la classe AddressListComponent.

```
1.    deleteCurrent(): void {
2.      this.addresses = this.addresses.filter((address: AddressEntry) => address !== this.currentAddress);
3.      this.currentAddress = null;
4.  }
```

Cette fonction supprime l'entrée d'adresse de la collection d'entrées.

Notification de sélection

Lorsqu'un élément d'entrée d'adresse est sélectionné, l'élément précédemment sélectionné (le cas échéant) doit être désélectionné et son adresse associée remplace l'adresse précédente en tant qu'adresse dans le composant Address View. Nous ne pouvons pas utiliser les événements de sortie car les éléments d'adresse n'ont pas de relation parent-enfant entre eux. Nous utiliserons un observable pour communiquer.

Service de Notification

Générez un service avec la commande `ng generate service address-list/notification`. Modifiez `src/app/address-list/notification.service.ts` comme suit.

```
1.  import { Injectable } from '@angular/core';
2.  import {BehaviorSubject} from 'rxjs';
3.  import {AddressEntry} from './address-entry';
4.
5.  @Injectable()
6.  export class NotificationService {
7.    // Observable for selected elements
8.    selectedElement = new BehaviorSubject<AddressEntry | null>(null);
9.    constructor() { }
10.
11.   public selectionChanged(address: AddressEntry): void {
12.     this.selectedElement.next(address);
13.   }
14. }
```

NotificationService déclare un Observable (un BehaviorSubject) à utiliser pour transmettre des objets AddressEntry lorsque la fonction selectionChanged est appelée.

Injection de Service

Nous utiliserons l'injecteur de AddressListComponent pour injecter le service puisqu'il est parent de tous les composants Address Entry. Modifiez la classe AddressListComponent comme suit.

```
1.  import {Component, OnInit} from '@angular/core';
2.  import {AddressEntry} from './address-entry';
3.  import {NotificationService} from './notification.service';
4.
5.  @Component({
6.    selector: 'app-address-list',
7.    templateUrl: './address-list.component.html',
8.    styleUrls: ['./address-list.component.css'],
9.    providers: [NotificationService]
10. })
11. export class AddressListComponent implements OnInit {
12.   addresses: AddressEntry[] = [];
13.   currentAddress: AddressEntry = null;
14.   constructor(private notificationService: NotificationService) { }
15.
16.   ngOnInit(): void {
17.   }
18.
19.   select(address: AddressEntry): void {
20.     this.currentAddress = address;
21.     this.notificationService.selectionChanged(address);
```

```
22.   }
23.
24.   addAddress(): void {
25.     const newAddress = new AddressEntry('New', 'Entry');
26.     this.addresses = [newAddress, ...this.addresses];
27.     this.select(newAddress);
28.   }
29.
30.   deleteCurrent(): void {
31.     this.addresses = this.addresses.filter((address: AddressEntry) => address !== this.currentAddress);
32.     this.currentAddress = null;
33.   }
34. }
```

La ligne 9 spécifie l' injecteur de AddressListComponent comme injecteur pour NotificationService en l'ajoutant au tableau providers. Le service est injecté dans la classe à la ligne 14 et lorsqu'une nouvelle entrée d'adresse est sélectionnée, nous appelons la fonction selectionChanged du service de notification avec l'entrée d'adresse nouvellement sélectionnée à la ligne 21. Cela pousse l'entrée d'adresse dans l'observable.

Notification de changement d'Entrée

Les composants Address Entry sont informés des changements d'adresse en s'abonnant à l'observable fourni par le service de notification. Modifiez AddressListElementComponent comme suit.

```
1.  import {Component, Input, OnDestroy, OnInit} from '@angular/core';
2.  import {AddressEntry} from '../address-entry';
3.  import {NotificationService} from '../notification.service';
4.  import {Subscription} from 'rxjs';
5.
6.  @Component({
7.    selector: 'app-address-list-element',
8.    templateUrl: './address-list-element.component.html',
9.    styleUrls: ['./address-list-element.component.css']
10. })
11. export class AddressListElementComponent implements OnInit, OnDestroy {
12.   @Input() address!: AddressEntry;
13.   selected = false;
14.   subscription: Subscription;
15.
16.   constructor(private notificationService: NotificationService) {
17.   }
18.
19.   ngOnInit(): void {
20.     this.subscription = notificationService.selectedElement.subscribe(newAddress => {
21.       this.selected = newAddress === this.address ? true : false;
22.     });
23.   }
```

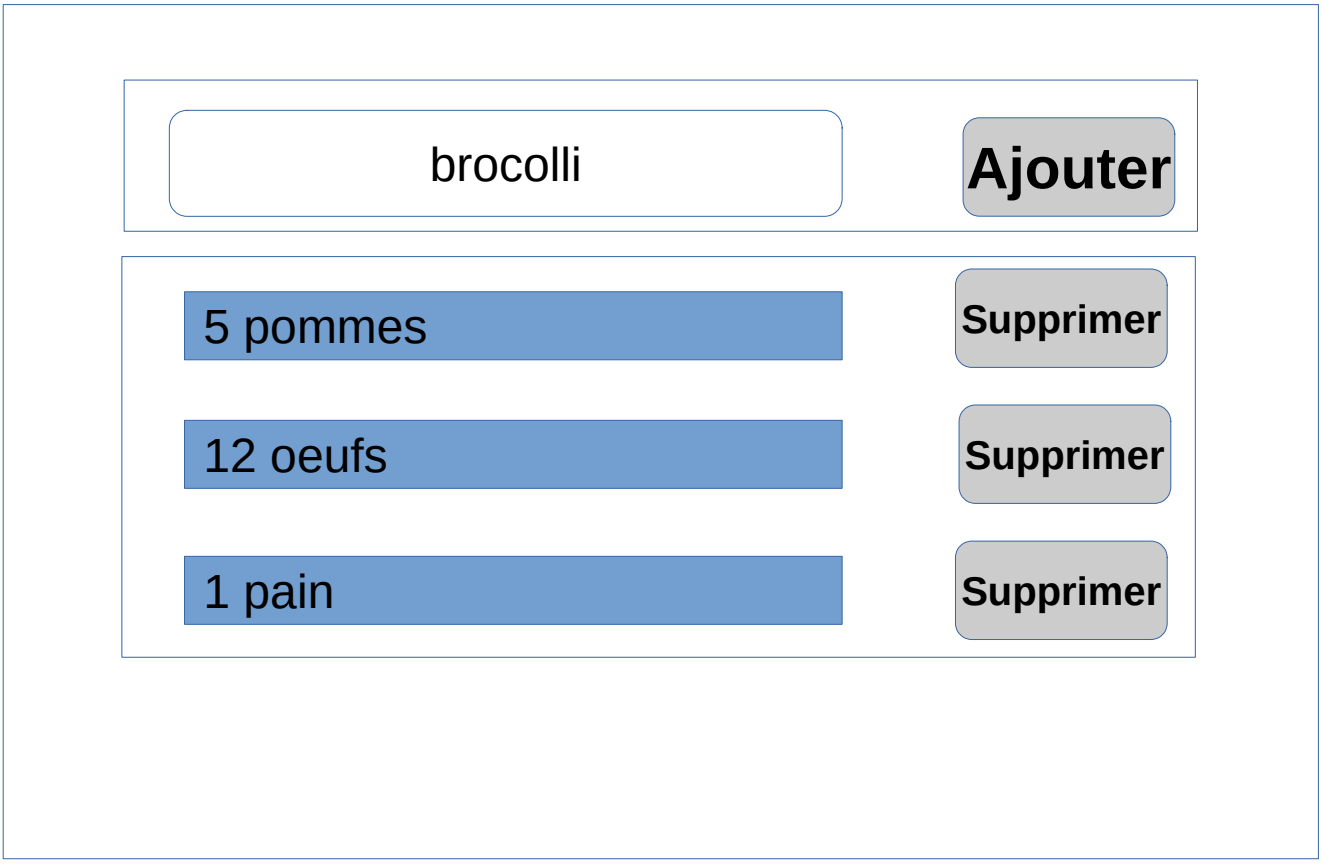
```
24.
25.   getFullName(): string {
26.     return `${this.address.firstName}, ${this.address.lastName}`;
27.   }
28.
29.   ngOnDestroy(): void {
30.     this.subscription.unsubscribe();
31.   }
32. }
```

Le service de notification est injecté à la ligne 16 et abonné aux lignes 20-22. Lorsqu'une nouvelle entrée est reçue, la propriété selected property prend comme valeur true si la nouvelle entrée est la même que celle détenue par le composant et false sinon. Le composant implémente également la fonction de suivie du cycle de vie (life-cycle hook) ngOnDestroy aux lignes 29-31, pour se désabonner de l'observable lorsque celui-ci est détruit.

Exercise

L'exercice suivant est le livrable pour le laboratoire. Complétez et enregistrez le code dans Github Classroom avant la date limite. Seul le travail de ce exercice sera évalué.

Implémentez une application Angular pour une liste de courses (shopping list). L'application doit afficher une interface similaire à la suivante.



L'application présente un champ de texte et un bouton. Lorsqu'un utilisateur saisit un item dans le champ de texte et appuie sur le bouton, ce nouvel item doit être ajouté à une liste affichée des items de la liste de courses, avec un bouton associé permettant de supprimer l’item de la liste.

Votre implémentation devra compter deux composants Angular au moins : un composant avec le champs d’entrée de texte et le bouton ‘Ajouter’ ainsi qu’un composant présentant la liste des éléments et les boutons ‘Supprimer’ y associés.