

## SEG3102 – Lab 3

### Angular – Components

The objective of this lab is to learn more about Angular by implementing an Address Book application. More precisely, you will learn about:

- creating and implementing angular components,
- implementing template forms,
- communicating between components with inputs, output events and observables.

The source code for the lab is available in the Github repository  
<https://github.com/stephanesome/angComponents.git>.

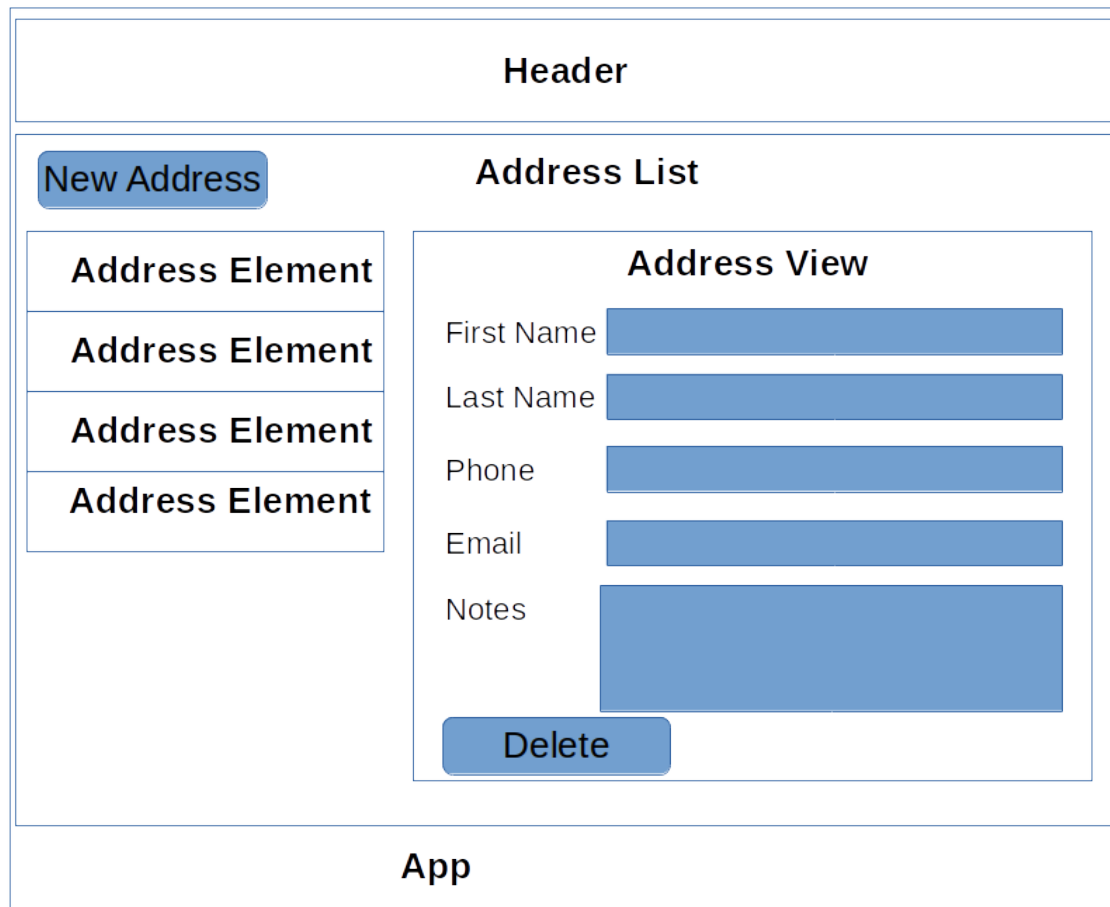
### Address Book Application

We will implement an application to record a User's address book. The main features of the application are as follows.

- Display a list of recorded address book entries.
- Add entries to the address book.
- View details of address book entries. Each entry includes a first name, last name, phone number, email and notes.
- Delete entries for the address book.

### Components Design

Interface design is indicated for any application. The interface design helps determine the interface layout and to ensure that it supports users in an effective way. It also allows us to figure out components needed for an Angular application. Below is the application wireframe.



We identify the following components and their relations.

- *App* component, the root component for the angular application.
- *Header* component, sub-component of *App*.
- *Address List* component, sub-component of *App*. This component holds a list of addresses and provides associated responsibilities such as adding an address to the list or removing an address from the list.
- *Address Element* component, sub-component of *Address List*. This component is a select-able element for an address book entry. Clicking on it prompts the display of the corresponding entry details.
- *Address View* component, sub-component of *Address List*. This component allows displaying and editing of an address book entry.

## Application Setup

Create a new angular application (**ng new address-book**). Routing is not needed and we will use CSS as stylesheet format.

We will use the [Angular Bootstrap CSS library](#). Issue command

`ng add @ng-bootstrap/ng-bootstrap` in the project root folder to add Bootstrap to the project.

## App Component

The root *App* Component contains a *Header* component and a *Address List* component. Specify the HTML template (`src/app/app.component.html`) as follows.

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

## Header Component

The *Header* component only includes branding for our application. Issue command

`ng generate component header` in the project root folder. The angular CLI will generate files for a new component with the following output.

```
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)
```

The generated files include:

- the component class in `src/app/header/header.component.ts`,
- test for the component in `src/app/header/header.component.spec.ts`,
- the component HTML template in `src/app/header/header.component.css` and
- the component CSS sheet file in `src/app/header/header.component.css`.

Edit the HTML template as follows.

```
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>
```

## Address List Component

Generate the Address List component (`ng generate component address-list`).

## Address Entry Model

We need a model class for the address book entries. This is just a normal Typescript class. We use Angular CLI to generate with command `ng generate class address-list/address-entry`.

Edit the generated `src/app/address-list/address-entry.ts` as follows.

```
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. }
```

The class definition specifies an Address Entry fields and a constructor.

## Address List Component Template

The *Address List* Component HTML Template should present a button for adding a new entry, a list of *Address List Elements* and an *Address View* component. Create an initial template as follows in `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 **NgFor** 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.

### Address List Component Class

Add a property in class `AddressListComponent` for the collection of addresses.

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

Make sure class `AddressEntry` is added to the imports.

### Address List Element

Generate an Address Element component

(`ng generate component address-list/address-list-element`). An address list element displays a label identifying the address entry. We will use a full name constructed from `firstName` and `lastName`.

- Add the following function to class `AddressListElementComponent` in `src/app/address-list/address-list-element/address-list-element.component.ts`.

```
1. getFullName(): string {  
2.   return `${this.address.firstName}, ${this.address.lastName}`;  
3. }
```
- We need to also declare an *address* property in the `AddressListElementComponent` class. This instance will be passed to the instance of `AddressListElementComponent` by the parent `AddressListComponent` since addresses are to be held by that class.

We use input binding for that. Add the declaration

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

to class `AddressListElementComponent`. You also need to import `Input` from `@angular/core` and `AddressEntry` from `./address-entry`.

- We also want to change the appearance of an Address List Element when selected. Add the following declaration to class `AddressListElementComponent`.

```
1. selected = false;
```

We will be able to use this property to set CSS properties on the component according to their selected status.

- Edit the Address List Element HTML template (`src/app/address-list/address-list-element/address-list-element.component.html`) as follow.

```
1. <button class="address-list-element ui positive"
```

```

2.     [ngClass]="{'address-list-element-selected': selected}">
3.     {{getFullName()}}
4. </button>

```

The styling is controlled by the directive `ngClass` which adds the property `address-list-element-selected` when property `selected` is true.

- Edit the Address Element CSS style sheet as follows.

```

1. .address-list-element {
2.   border: 1px solid ;
3.   padding: 10px 2px;
4.   font-size: 16px;
5.   cursor: pointer;
6.   width: 200px;
7.   display: inline-block;
8. }
9. .address-list-element:active {
10.  color: green;
11. }
12. .address-list-element-selected {
13.  background-color: green;
14.  color: white;
15.  font-weight: bold;
16. }

```

## Address View Component

Generate an Address View component

(ng generate component address-list/address-view).

### Address View Template

This component displays a form for the address entries. Edit the HTML Template

(src/app/address-list/address-view/address-view.component.html) as follows.

```

1. <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>

```

We specify form fields for the Address Entry (lines 6, 10, 14, 18, 22) and bind with properties in the component class.

Ensure that **FormsModule** is imported to the **AppModule** (`src/app/app.module.ts`) and is added to the **imports** array of decorator **@NgModule**.

We also add a button to delete the entry (line 26) and a button to toggle editing (lines 27-34). Event click on button *Delete* is handled by the component class function **delete**. In order to show a button that toggles from *Edit* to *Protect* when clicked on, line 27-34 specify two button elements. In lines 27-30, a button *Protect* is rendered when value **edit** is true and in lines 31-34 a button *Edit* rendered otherwise.

### Address View Class

The Address View Class **AddressViewComponent** declares the properties and functions necessary to the HTML template. Edit **AddressViewComponent** in `src/app/address-list/address-view/address-view.component.ts` as follows.

```

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 | undefined;
11.   @Output() fireDelete: EventEmitter<AddressEntry> = new EventEmitter();
12.   edit: boolean | 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. }

```

Input binding is used in line 10 to pass the current address to be edited/displayed to the component. An output event (line 11) notifies the parent component when the user clicks to delete the address. The event is fired in the `delete` function (lines 23-25).

### Address View Style

Edit `AddressViewComponent` style sheet in

`src/app/address-list/address-view/address-view.component.css` as follows.

```

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 {
9.   padding: 10px;
10. }

```

### Adding an Address Entry

We associate a handler to button *New Address* in the *Address List* component to add an *Address Entry*. Edit `AddressListComponent` (`src/app/address-list/address-list.component.html`) so that it is as follows.

```

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>

```

We added an event handler for event **click** in line 3.

Edit the component class to add functions **addAddress** and **select**, as well as property **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. }

```

Edit **AddressListComponent** (`src/app/address-list/address-list.component.html`) to pass a selected instance of **AddressEntry** to the **Address View** component.

```

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>

```

We also associate a handler function to the fireDelete event emitted by the Address View component. Add the handler function to class AddressListComponent.

```

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

```

This function removes the Address Entry from the collection of entries.

## Notifying of selection

When an address entry element is selected, the previously selected element (if any) must be de-selected and its associated address entry replaces the previous entry as the address shown in the Address View component. We can not use output events since Address Elements do not have a parent-child relation among each other. We will use an observable to communicate.

### Notification Service

Generate a service with command `ng generate service address-list/notification`. Edit `src/app/address-list/notification.service.ts` as follows.

```

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 declares an Observable (a BehaviorSubject) to be used to pass AddressEntry objects when function selectionChanged is called.

### ***Service Injection***

We will use the AddressListComponent Injector to inject the service since it is parent to all the Address Entry components. Edit class AddressListComponent as follows.

```
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;  
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. }
```

Line 9 specifies the AddressListComponent injector as an injector for NotificationService by adding it to the providers array. The service is injected into the class on line 14 and when a new address entry

is selected we call the `selectionChanged` function of the notification service with the newly selected address entry on line 21. This pushes the address entry in the observable.

### **Entry Change Notification**

Address Entry components are notified to address entry changes by subscribing to the observable provided by the notification service. Edit `AddressListElementComponent` as follows.

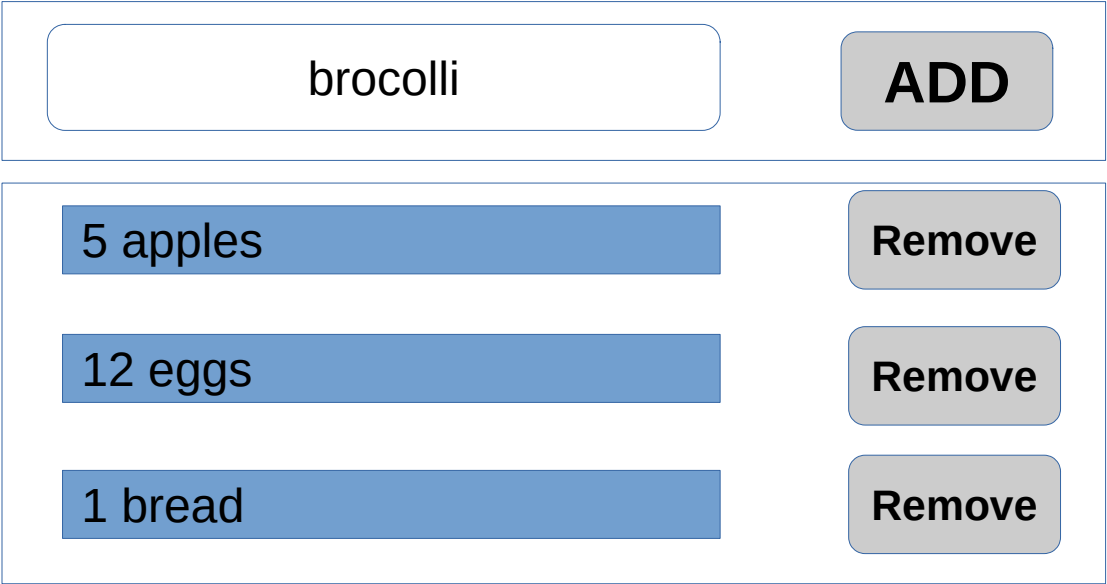
```
1. import {Component, Input, OnDestroy, OnInit} from '@angular/core';
2. import {AddressEntry} from '../address-entry';
3. import {NotificationService} from '../notification.service';
4. import {Subscription} from 'rxjs';
5.
6. @Component({
7.   selector: 'app-address-list-element',
8.   templateUrl: './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. }
```

The notification service is injected at line 16 and subscribed to at lines 20-22. When a new entry is received, the `selected` property is set to `true` if the new entry is the same as the one held by the component, and `false` otherwise. The component also implements the `ngOnDestroy` life-cycle hook (lines 29-31) to unsubscribe from the observable when it is destroyed.

## Exercise

The following exercise is the deliverable for the lab. Complete and check-in the code to Github Classroom before the deadline. Only this exercise will be evaluated.

Implement an Angular application for a shopping list. The application shall show an interface similar to the follow.



The mockup shows a shopping list application interface. At the top, there is a text input field containing the word "broccoli" and a grey button labeled "ADD". Below this, there is a list of three items, each in a blue box with a corresponding grey "Remove" button to its right:

- 5 apples
- 12 eggs
- 1 bread

The app presents a text field and a button. When a user enters an item in the text field and press the button, this new item should be added to a list of displayed items, with an associated button to remove the item from the list.

Your implementation should include at least two Angular components: a component for list element entry with the text field and 'Add' button, and a component with the listed elements and associated 'Delete' buttons.