UNIT 2

Angular

Exercise 1

Client-side Web Development
2nd course – DAW
IES San Vicente 2025/2026
Arturo Bernal / Rosa Medina

Index

Introduction

We are going to implement, more or less, the exercise we did in Unit 1 – part 3, with Angular. A single page with a list of (real state) properties and a form to add a new one (**reuse the HTML** from that exercise).

Create a project called **angular-inmosanvi**. Install also **angular-eslint** in this project. After creating it, set the prefix for the components to "" (empty).

Important: It must be a zoneless Angular project!

Add these classes to the body and app-root element in the index.html:

```
<body class="bg-gray-100">
  <app-root class="flex flex-col min-h-screen"></app-root>
</body>
```

Enable Tailwind

The project we've been developing in Unit 1 uses **TailwindCSS** as a style library/tool. This library requires an initial setup because it needs to analyze the CSS classes present in your project to generate the final CSS bundle.

Enabling TailwindCSS in Angular is very easy → https://angular.dev/guide/tailwind

npm install tailwindcss @tailwindcss/postcss postcss

Create the postcss (a tool that tailwind uses) configuration file → **.postcssrc.json** (project's root directory):

```
/* .postcssrc.json */
{
  "plugins": {
    "@tailwindcss/postcss": {}
  }
}
```

Finally. Import Tailwind in your main CSS file (src/styles.css):

```
/* styles.css */
@import "tailwindcss";
```

Creating a component

In this project create a component called **properties-page**. This will have almost all the HTML we need for this exercise. **Header**, **footer**, and the **main.container** element go in the **app component** HTML.

The app component's HTML has the main HTML that goes inside the body. Inside the **main.container** element, put the **properties-page** component selector. Also add the **grow** class to the main element.

```
<main class="container mx-auto px-4 py-8 mt-16 grow">
```

The properties-page component holds the rest of the HTML inside the main.container (form and list of properties). Don't copy the template (for now).

Copy the icons folder inside the public directory.

Also, add the **class="bg-gray-100"** to the body (index.html).

Create an interface called Property that will have the needed properties:

```
export interface Property {
  id?: number;
  province: string;
  town: string;
  address: string;
  title: string;
  price: number;
  sqmeters: number;
  numRooms: number;
  numBaths: number;
  mainPhoto: string;
}
```

We'll make some changes to this interface in future exercises as needed.

In the properties-page component, create an empty array of Property called **properties**, and a single Property object called **newProperty** (with empty fields and no id).

In the HTML form, reference each property of the **newProperty** object with the **[(ngModel)]** directive. Example:

```
<select id="province" name="province" required [(ngModel)]="newProperty.province" class="...">
```

Add the ngModel directive to the input[type=file]. Just reference a string that you will empty when resetting the form.

```
  <input type="file"  name="mainPhoto" id="mainPhoto" [(ngModel)]="filename"  ...>
```

Listing the properties

Generate the cards using **@for**. Copy the HTML inside the original template and bind the property attributes there. Add the **click** event to the delete button and remove the property from the array when that happens.

You don't need to format the price (yet). We'll do it in the next exercise.

If there are no properties in the array, show this message:

```
<p class="text-2xl text-gray-500">No properties found.</p>
```

Transform the image to Base64

The concept is the same as in plain JavaScript, but the events now are handled differently. The HTML for the file input will look like this:

```
<input type="file"  ...  #fileImage (change)="changeImage(fileImage)">
```

This would be the component's method:

```
#changeDetector = inject(ChangeDetectorRef); // Add this at the beginning of the class

changeImage(fileInput: HTMLInputElement) {
  if (!fileInput.files || fileInput.files.length === 0) {
    this.newProperty.mainPhoto = "";
    return;
  }

  const reader: FileReader = new FileReader();
  reader.readAsDataURL(fileInput.files[0]);
  reader.addEventListener('loadend', () => {
    this.newProperty.mainPhoto = reader.result as string;
    this.#changeDetector.markForCheck(); // Necessary in new Angular zoneless apps
  });
}
```

Submitting the form

To capture the submit event use the **ngSubmit** event.

When submitting the form all you have to do is:

- Assign an id to the (start with 1 and increment it)
- Add the cloned object to the array of project.
- Finally, reset the form: Reassign **newProperty** to a new object with empty values. Also reset the file name.

Recommendations and tips

- You can remove some id attributes from the original HTML. Just don't remove the ids on the input elements (the label elements are binded to those elements by using the id)
- You don't need to validate the form (that's something we'll learn in the future to do with Angular). You can omit any error messages.
- You'll need to include Angular's **FormsModule** in your events-page Component or directives like **[(ngModel)]**, or **ngSubmit** event won't work!
- When using **[(ngModel)]** inside a <form> element, the element **must have a name property** (we'll learn more about forms in the future).
- Instead of the class **hidden** (display: none) to hide the form's image, you can simply use **@if**.

- **Important:** Before uploading the project, delete the directories **node_modules** and **.angular** (Angular uses this for compilation cache). If there's a **dist** directory, also delete it.