LAB2 - Use Data Binding

In this lab, we are going to build a task manager, the goal is to build a simple user interface containing a form to let the user add a task. This lab will cover data binding basics (interpolation, property binding and event handling).

Create the "Task" Model

Before working on the view, we are going to create the model classes first:

1. In the `src/app` folder, create a new file `task.ts` and create the `Task` model with the following properties: id, description, completed

```
export interface Task {
    id: number;
    description: string;
    completed: boolean;
}
```

2. To add a priority field to the task model, we will have to create a enum first, in the same file add an enumeration called `Priority` with the three values: Low, Medium, High

```
export enum Priority {
    Low = 0,
    Normal = 1,
    High = 2
}
```

3. Now add a new property `priority` of type `Priority` to the `Task` interface

```
export interface Task {
    ///...
    priority: Priority
}
```

Create the view (HTML)

After setting up the necessary models, we are going to build the user interface which is basically a

form that contains an **input** for the task description, **dropdown** to select a priority and a button to **submit** the data.

1. We are going to work directly on the `AppComponent`, it comes with the Angular starter template by default, go to `app.component.html` and empty the file
2. In the same file, start by adding an HTML `input` element

```
<input  placeholder="Description"  />
```

3. Add a `select` HTML element, with an `option` for each priority value like the following:

```
<select>
    <option  value="0">
        Low
    </option>
    <option  value="1">
        Normal
    </option>
    <option  value="2">
        High
    </option>
</select>
```

4. Finally, add a submit `button` to the form

```
<button>
    Add task
</button>
```

Create the view model (TS)

Before binding the view with the view-model we need the actual data and properties, so in this section we are going to be working on the TS component class (`app.component.ts`)

1. Head over `app.component.ts` and create a new field `newTask` of type `Task`, and initialize it with default values, that will store the form data:

```
newTask: Task = {
    id:  0,
    description:  "",
```

```
      completed:  false,
      priority:  Priority.Normal
  };
```

2. In the previous section, we filled the priority options for the `select` field with static data, to reflect the actual `Priority` enum create three fields (`low, normal, high`) in the app component class, corresponding each to an object containing the value and a label (to show for the user)

```
low = {
     value:  Priority.Low,
     label:  "Low"
};
normal = {
    value:  Priority.Normal,
    label:  "Normal"
}
high = {
    value:  Priority.High,
    label:  "High"
}
```

3. Create a `tasks` field that will keep track of all the saved tasks

```
 tasks: Task[] = [];
```

4. Create a method `addTask` that will be reponsibile of saving the new task into `tasks`

```
 addTask(task: Task): void {
     //TODO implement
 }
```

5. Implement addTask method so it assign a new `id` to the task and save it in `tasks` field.

Bind data

Now, we can bind the property in the view model with the elements in the view

- **2-way binding** For the `input` field, use 2-way binding using `ngModel` since we want to show the description of the new task but also update it when the user type data:

```
<input  [(ngModel)]="newTask.description" placeholder="Description"   />
```

- **Property binding** For the `option` elements of the select, use property binding to bind the value of the option with the corresponding `priority` value

```
    <option  [value]="low.value">
         Low
    </option>
```

> Do the same for `normal` and `high` options

- **Interpolation** For the labels that face the user when selecting a prioprity from the select options, use interpolation to bind it with the corresponding `priority` label

```
<option  [value]="low.value">
         {{low.label}}
</option>
```

> Do the same for `normal` and `high` options

- **Event binding (Handling)** Now to submit the form and makes our code do what it's supposed to do, we have to call `addTask` method previously implemented when clicking the button, to acheive that use event binding

```
 <button  (click)="addTask(newTask)">
     Add task
</button>
```