LAB7 - Building a SPA

In this lab, we will going to build a fully functional SPA (Single Page Application), we will first integrate the Angular Material components into our app, setup the app skeleton and add client side routing useing Angular RouterModule to navigate between pages.

Using Angular Material components

In this section, we will integrate the Angular Material package into our app, use some of its component like input, select, button and toolbar.

Run the following command to add Angular Material into your project

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

The `ng add` command will additionally perform the following actions:

- Add project dependencies to `package.json`
- Add the Roboto font to your `index.html`
- Add the Material Design icon font to your `index.html`
- Add a few global CSS styles to:
  - Remove margins from `body`
  - Set `height: 100%` on `html` and `body`
  - Set Roboto as the default application font

After the installation is done, we will enhance our UI using Angular Material components.

TaskCreatorComponent

1. Go to `task-creator.component.html` and replace the form input, select and button with the corresponding angular material components

   - Replace the `input` element by the material component `MatInput` like the following:

```
<mat-form-field appearance="fill">
    <mat-label>Description</mat-label>
    <input [(ngModel)]="newTask.description" matInput>
</mat-form-field>
```

- Replace the `select` element by the material component `MatSelect` like the following:

```
<mat-form-field appearance="fill">
    <mat-label>Priority</mat-label>
    <mat-select [(ngModel)]="newTask.priority" >
        <mat-option *ngFor="let value of priorities" [value]="value">{{
    </mat-select>
</mat-form-field>
```

- Replace the native `button` element by the material component `MatButton` like the following:

```
<button (click)="addTask(newTask)" mat-flat-button color="primary">
    Add task
</button>
```

- Don't forget to import the corresponding modules from angular material in the `AppModule`

```
import {MatFormFieldModule} from '@angular/material/form-field';
import {MatInputModule} from '@angular/material/input';
import {MatSelectModule} from '@angular/material/select';
import {MatButtonModule} from '@angular/material/button';

@NgModule({
    ...
    imports: [
        ...
        MatFormFieldModule,
        MatInputModule,
        MatSelectModule,
        MatButtonModule
    ],
    ...
})
```

2. Add some styling to the task creation form, in order to do that add a title (`h3`) for the form, a

wrapper (`div`) and and a container (`div`), and apply those CSS classes

```
<div class="form-container">
    <div class="form">
        <h2>New task</h2>
        ...
        </div>
</div>
```

3. Implement the previous CSS classes (`.form-container` and `.form`) to apply the style of your choice. Example:

```
.form-container {
    margin: 32px;
    display: flex;
    flex-direction: row;
    justify-content: center;
}
.form {
    display: flex;
    flex-direction: column;
    justify-content: stretch;
    align-items: stretch;
    min-width: 300px;
}
```

TasksListComponent

For the list of tasks, we will arrange them in a grid component contained in cards insted of list items.

1. Go to `tasks-list.component.html`, and use angular material components instead of native HTML elements
    - Remove the list element (`ul`) and replace it by a `MatGridComponent`

    ```
    <mat-grid-list cols="4" rowHeight="2:1">
        <mat-grid-tile *ngFor="let task of tasks">
            ....
        </mat-grid-tile>
    </mat-grid-list>
    ```

- Now, replace the items (`li` elements) by a material card component : `MatCardComponent`

  ```
  <mat-card>
      <mat-card-title  [important]="task.priority==Priority.High"  [ngClass]="
          {{task.description}}
      </mat-card-title>
      <mat-card-subtitle>
          {{task.completed ? '(Completed)' : '(Not completed)'}}
      </mat-card-subtitle>
      <mat-card-actions>
          <button  mat-icon-button  aria-label="Delete">
              <mat-icon>delete</mat-icon>
          </button>
      </mat-card-actions>
  </mat-card>
  ```

  > Notice in the above snippet, we added an action for task removal, that we will implement later on.

- Don't forget to import the corresponding modules from angular material in the `AppModule`

  ```
  import {MatGridListModule} from  '@angular/material/grid-list';
  import {MatCardModule} from  '@angular/material/card';
  import {MatIconModule} from  '@angular/material/icon';

  @NgModule({
      ...
      imports: [
          ...
          MatGridListModule,
          MatCardModule,
          MatIconModule
      ],
      ...
  })
  ```

2. To add some styes to the tasks cards, add a wrapper (`div`) around the `mat-card` element and give it the CSS class `.task-container`

```
<div  class="task-container"  >
    <mat-card  >
        ...
```

```
    </mat-card>
<div>
```

3. Implement the CSS class the previous CSS class (`.task-container`) to apply the style of your choice. Example:

```
.task-container {
    padding: 16px;
    height: 100%;
    width: 100%;
}
```

Setting the app layout

Before activating routing, we will setup the layout of the main component (`AppComponent`)

1. Go to `app.component.html` file, and add a header for the application, use MatToolbar

```
<mat-toolbar color="primary">
    <span>Task manager</span>
</mat-toolbar>
```

2. Add some actions to the toolbar, (you will use them for the navigation later)

```
<mat-toolbar color="primary">
    <span>Task manager</span>
    <div style="flex: 1"></div>
    <button mat-icon-button>
        <mat-icon>add</mat-icon>
    </button>
    <button mat-button>
        My tasks
    </button>
```

</mat-toolbar>

3. Add the toolbar module to the `AppModule`

```
import {MatToolbarModule} from '@angular/material/toolbar';

@NgModule({
    ...,
    imports: [
        ...
        MatToolbarModule,
    ],
    ...
})
```

Routing configuration

Currently, the tasks list and the creation form are in the same destination (`AppComponent`), in this section we will separate them into two routes, one for tasks list and one for the creation form for a better use experience, we will use Angular `RouterModule` .

1. Before registering our routes, you need to do the following refactoring since that our component tree will change using routing:

   - Remove the `app-task-creator` and `app-tasks-list` elements from the `app-component`
   - Remove all the `@Input` (s) and `@Output` (s) from all the components and use `TaskService` instead to
   - Refactor the `TaskService` so it fullfills the necessary requirements

   The following files should contains the following:

   `app.component.ts` : (Empty)

   ```
   constructor() {}

   ngOnInit(): void {}
   ```

   `tasks-list.component.ts` :

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

   constructor(private  taskService: TaskService) { }

   ngOnInit(): void {
       this.tasks = this.taskService.getAllTasks();
   }
   ```

   `task.service.ts` : (New abstract method)

```
 abstract  getDefaultPriority(): Priority
```

`mock-task.service.ts` : (New method implemented)

```
 getDefaultPriority(): Priority {
     return  Priority.Normal
}
```

`task-creator.component.ts` :

```
newTask: Task = {
    id:  0,
    description:  "",
    completed:  false,
    priority:  Priority.Normal
};

constructor(private  taskService: TaskService) {}

 ngOnInit(): void {
     this.newTask = {
         id:  0,
         description:  "",
         completed:  false,
         priority:  this.taskService.getDefaultPriority()
    };
}
```

2. After doing the previous refactoring, we can now register our routes, head over `app.module.ts` and create a new constant `routes` at the top of file, which will contain the mapping between a path and its correponsding component:

```
import { RouterModule, Routes } from  '@angular/router';
const  routes: Routes = [
    {path:  "tasks", component:  TasksListComponent},
    {path:  "new-task", component:  TaskCreatorComponent}
]
```

3. Register the `routes` using `RouterModule` like the following:

```
@NgModule({
    ...,
    imports: [
        ...,
        RouterModule.forRoot(routes),
    ]
})
```

4. If we run the application and go the url http://localhost:4200/tasks, nothing is shown, and that's because Angular doesn't know where to put the components yet, in order to make the routing works we need to place an outlet. Go to `app.component.html` and add a `router-outlet` element below the toolbar

    ```
    <div  style="padding: 20px;">
        <router-outlet></router-outlet>
    </div>
    ```

    > You can wrap the outlet inside a `div` and add some `padding` for a better display

5. Now the navigation shoulds works, navigating to /tasks should take us into the the tasks list, and /new-task should take us into the the task creating form. The missing behavior is when navigating to the root (http://localhost:4200), it show a blank and that's because `routes` configruation there is no default route. Add a `redirectTo` configuration at the top of `routes` array

    ```
    const  routes: Routes = [
        {path:  "", redirectTo:  "tasks", pathMatch:  "full"},
        ...
    ]
    ```

6. When navigating to an existing url, we can redirect to a `NotFoundComponent` or easier than that redirect to the default route (/tasks) Add a wildcard (catch-all) route in the end of `routes` array:

    ```
    const  routes: Routes = [
        ...
        {path:  "**", redirectTo:  ""},
    ]
    ```

Using `routerLink` directive

Go the `app.component.html` and add `routerLink` directives to the action buttons of the toolbar

```
<button  routerLink="/new-task"  mat-icon-button>
    <mat-icon>add</mat-icon>
</button>
<button  routerLink="/tasks"  mat-button>
    My tasks
</button>
```

Using `Router` service

If you test the application and create a new task, you will notice that you have to return to the tasks list manually after completing the creation, we want to improve that experience by automatically navigating to the tasks list.

1. Go to `task-creator.component.ts` and inject the `Router` service in the constructor

   ```
   constructor(private router:Router, ...) {
   }
   ```

2. In the `addTask` method, after adding the task use the Router to navigate to /tasks route

   ```
   addTask(task:Task) {
       this.taskService.addTask(task)
       this.router.navigate(["/tasks"])
   }
   ```