LAB5 - Create Task Service

In the previous labs, we were creating and retreiving tasks from the component itself which is not a really good design if we want to respect the MVVM (Model View ViewModel), the component should be responsible only of the managing the view, other tasks should be delegated to services. In this lab, we will be creating a service that will manage tasks (loading, creation and removal).

Creating Task Service

In this section, we will create an abstract service and defining it's responsiblities

1. In the `src/app` folder, create a new file `task.service.ts`
2. In this file create an abstract class `TaskService`

```
export abstract class TaskService {
}
```

3. The task service should get tasks, create a task and remove task, define the correponding methods

```
export  abstract  class  TaskService {
    abstract  getAllTasks(): Task[];
    abstract  addTask(task: Task): Task[];
    abstract  removeTask(task: Task): Task[];
}
```

4. Don't forget to annotate the service with `@Injectable` decorator to be part of dependency injection system

> We created TaskService as an abstract class and not interface beacuse Angular's DI system works only with classes and abstract classes and not interfaces

Mocking the service

Since we do not have a real implementation yet with a backend server, we will create a mock service that implements the task service

1. In the same folder (`src/app`), create a new file `mock-task.service.ts`
2. In this file create a class `MockTaskService` and make it extends from the `TaskService` abstract class

```
export  class  MockTaskService  extends  TaskService {

    getAllTasks(): Task[] {
        throw  new  Error("Method not implemented.");
    }

    addTask(task: Task): Task[] {
        throw  new  Error("Method not implemented.");
    }

    removeTask(task: Task): Task[] {
        throw  new  Error("Method not implemented.");
    }
}
```

3. Don't forget to annotate the service with `@Injectable` decorator to be part of the dependency injection system
4. Add a private property `tasks` and initialize it with dummy data

```
private  tasks: Task[] = [
        { id:  1, description:  'fix bug', completed:  false, priority:  Priority.
        { id:  2, description:  'set up unit tests', completed:  false, priority:
        { id:  3, description:  'fix the UI',completed:  false, priority:  Priorit
];
```

5. Implement the three methods of the class

Registering the service

Before getting to use the service, we'll need to register it in the dependency injection

1. Go to `app.module.ts` file
2. To use `MockTaskService` you have to add a provider of type class provider, to tell the injector to return an instance of `MockTaskService` when asking for `TaskService` class type

```
providers: [
    {provide:  TaskService, useClass:  MockTaskService}
],
```

Using the service

Now, we can get rid of the old methods in the TaskCreatorComponent and AppComponent and directly use the `TaskService`

1. Head over `app.component.ts` and a `constructor`, inject `TaskService` in it

```
 constructor(private taskService: TaskService) {
 }
```

2. Implement `OnInit` interface,
3. In the `ngOnInit` method call the `getAllTasks` method of the service to get the list of tasks

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

4. Go to `task-creator.component.ts`
5. Inject a `TaskService` instance into the constructor of the `TaskCreatorComponent`

```
 constructor(private taskService: TaskService) {
  }
```

6. Change the `addTask` method implementation and use the task service to add a new task

```
 addTask(task:Task) {
     this.taskService.addTask(task)
     this.onTaskCreated.emit(task)
 }
```

7. Go to the `AppComponent` and replace the `onTaskCreated` implementation to update the list of tasks once a new task is added

```
onTaskCreated(task:Task) {
    this.tasks = this.taskService.getAllTasks();
}
```