NestJS 102 -- Connecting to Database and Introduction to Repository Pattern

Lets start by setting up the repository Pattern

In repository pattern we have the flow of control as shown below

> controller --> Service --> Repository

We have the controller already created in the previous Section. So lets create the Service and Repository Classes

Create a entity **Task**

- create a file `task.entity.ts`
- add the following code the entity

```
@Entity()
export class Task{
    @PrimaryColumn()
    id:string;

    @Column()
    title:string;

    @Column()
    description:string;

    @Column()
    status:string;
}
```

Connecting to the database

- Open the app.module.ts file to add the connection details to connect to the database

```
@Module({

  imports: [  TypeOrmModule.forRoot({
    type: 'postgres',
    host: 'localhost',
    port: 5400,
    username: 'taskmgmt',
    password: 'taskmgmt',
    database: 'task',
    entities: ['dist/**/*.entity.{ts,js}'],
    synchronize: true,
```

```
}),TaskModule],
})
export class AppModule {}
```

- run the application. Connect to the pgAdmin and check if the table Task is created in the database

Add the Repository class

- create a new fiel `task.repository.ts`
- add the code snippet in the file

```
   @EntityRepository(Task)
export class TaskRepository extends Repository<Task>{
}
```

Add the service class

- create a class `task.service.ts`
- add the code snippet to the file

```
    @Injectable()
export class TaskService{

    constructor(@InjectRepository(TaskRepository) private
taskRepository:TaskRepository){}
}
```

Configure the Repository and Service in the `task.module`

- Open the `task.module.ts` file
- Add the **TaskRepository** in the imports array.
- Add the **TaskService** in the providers array.
- the code for task.module should look like shown below

```
@Module({
  imports:[TypeOrmModule.forFeature([TaskRepository])],
  controllers: [TaskController],
  providers:[TaskService]
})
export class TaskModule {}
```

Adding the flow between controller to the Service and Repository

- In the controller inject the Service in the constructor

```
export class TaskController {

    constructor(private readonly tasksService:TaskService){}
}
```

- The Repository is already injected into the service in the previous step.
- Invoke the service methods from the controller

```
export class TaskController {

    constructor(private readonly tasksService:TaskService){}

    @Get()
    getAll():Promise<Task[]>{
        return this.tasksService.getAll();
    }

    @Post()
    createTask(@Body() createTaskDto:CreateTaskDto):Promise<Task>{

        return this.tasksService.createTask(createTaskDto);
    }

    @Patch('/:id')
    updateTask(@Param('id') id:string,@Body() updateTaskDto:UpdateTaskDto):string{
       return this.tasksService.UpdateTask(id,updateTaskDto);
    }

    @Delete('/:id')
    deleteTask(@Param('id') id:string):string{
       return this.tasksService.deleteTask(id);
    }
}
```

- use the IDE option to generate the above methods in the service class
- modify the return types so that they match

```
@Injectable()
export class TaskService{

    constructor(@InjectRepository(TaskRepository) private
taskRepository:TaskRepository){}

    async getAll(): Promise<Task[]> {
        throw new Error('Method not implemented.');
    }
```

```
    async UpdateTask(id:string,updateTaskDto: UpdateTaskDto): Promise<Task> {
        throw new Error('Method not implemented.');
    }
    async  deleteTask(id: string): Promise<void> {
        throw new Error('Method not implemented.');
    }

    async createTask(createTaskDto:CreateTaskDto):Promise<Task>{
      throw new Error('Method not implemented.');
    }
}
```

- The createTask method we will implement in the repository, since it requires custom logic. for all remaining operations the code will be written in the service
- Below code snippet shows the final code for all service methods

```
@Injectable()
export class TaskService {

    constructor(@InjectRepository(TaskRepository) private taskRepository:
TaskRepository) { }

    async getAll(): Promise<Task[]> {
        return await this.taskRepository.find();
    }
    async UpdateTask(id:string,updateTaskDto: UpdateTaskDto): Promise<Task> {
        const {title, description,status} = updateTaskDto;
        const task = await this.taskRepository.findOne({where:{id}});
        task.title = title
        task.description = description;
        task.status = status;
        await this.taskRepository.save(task);
        return task;
    }
    async deleteTask(id: string): Promise<void> {
        const result = await this.taskRepository.delete(id);
        if (result.affected === 0) {
            throw new NotFoundException(`Task with ID :${id} not found in DB`)
        }

    }

    async createTask(createTaskDto: CreateTaskDto): Promise<Task> {
        return this.taskRepository.createTask(createTaskDto);
    }
}
```

- Below code snippet shows the final code for all repository methods

```
@EntityRepository(Task)
export class TaskRepository extends Repository<Task>{

    async createTask(createTaskDto: CreateTaskDto): Promise<Task> {
        const newTask = this.create({
      ...createTaskDto,
      status:'new'
        });
        await this.save(newTask);
        return newTask;
    }
}
```