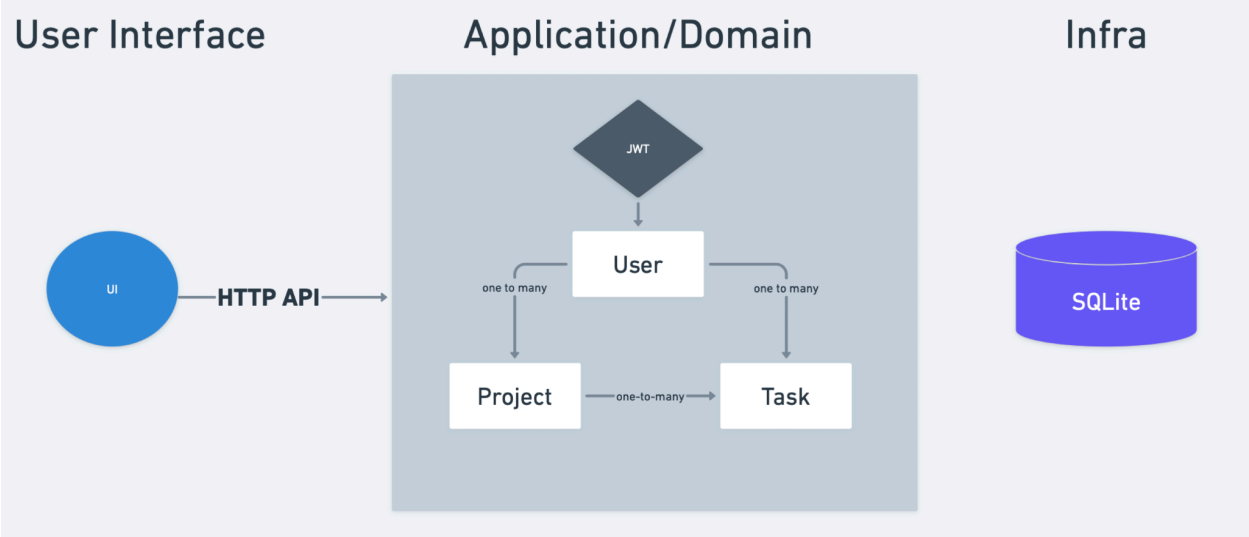
Pontifícia Universidade Católica de Minas Gerais

Curso: Arquitetura de Software Distribuído
Disciplina: Plataforma Node.js
Professor: Samuel Martins
Valor: 20pts

Exercício 2



O projeto final da disciplina consiste em desenvolver um produto de **gerenciamento de tarefas** como mostra o desenho arquitetural de alto nível acima. Este projeto será subdividido em etapas, sendo:

Etapa 1 - Setup do ambiente de desenvolvimento e modelagem inicial dos dados

Para esta etapa, iremos executar alguns passos iniciais que irão fazer com que o nosso projeto seja inicializado da forma correta. Antes de começar, ***verifique se você está utilizando a última versão do NodeJs*** em sua máquina local.

Passo 1

Instale o nest cli globalmente através do comando **npm i -g @nestjs/cli.** Em seguida, execute o comando **nest new project-manager-api** para inicializar o projeto com as configurações iniciais. Para garantir que a aplicação está funcionando, entre na pasta criada (cd project-manager-api) e execute o comando **npm run start:dev.** Este comando irá executar a aplicação em modo desenvolvimento, recarregando o servidor a cada nova alteração dos arquivos.

Ao acessar a página http://localhost:3000, você deverá ver uma saudação como esta:

Hello World

Passo 2

Vamos criar uma estrutura de pastas combinando a arquitetura CLEAN com os conceitos de DDD. Para isso, crie as pastas da aplicação de acordo com a seguinte estrutura:

```
.
└── src/
    ├── domain/
    │   ├── entities
    │   ├── interfaces
    │   └── use-cases/
    │       ├── projects
    │       ├── tasks
    │       └── users
    ├── gateways/
    │   ├── controllers/
    │   │   ├── projects
    │   │   ├── tasks
    │   │   └── users
    │   └── guards
    └── infrastructure/
        ├── auth
        └── database/
            ├── entities
            └── repositories
```

Iremos preencher essas pastas com os arquivos necessários na medida em que o projeto for avançando. Neste passo, vamos focar em criar as pastas e entender a responsabilidade de cada uma delas:

Domain

Essa pasta irá concentrar os nossos **casos de uso** que irão conter as regras de negócio de cada domínio, juntamente com as **entidades** que irão modelar a aplicação e as **interfaces** necessárias para os casos de uso e outras funções.

Gateways

Os gateways que também podem ser chamados de presenters ou application, irão compor a camada que terá interface direta com o mundo externo. Aqui, vamos colocar os **controllers** e os **guards,** que futuramente serão usados para a autenticação.

Infrastructure

A camada de infraestrutura existe para poder organizar a implementação de artefatos externos ao core business da nossa implementação. O banco de dados por exemplo nos conceitos do DDD/CLEAN é

considerado um driver externo. Isso porque em tese, a nossa aplicação deve ser independente de abstrações que fogem do escopo da implementação em si, assim como é o caso da autenticação (que será implementada utilizando o mecanismo JWT).

Passo 3

Vamos criar alguns arquivos dentro da nossa estrutura de pastas criada no passo anterior. Inicialmente, criaremos os controllers e algumas entidades. Crie um arquivo em **src/gateways/controllers** com o nome **controllers.module.ts.** Coloque o seguinte conteúdo no arquivo:

```
import { Module } from '@nestjs/common';
@Module({
controllers: [],
})
export class ControllersModule {}
```

Esse módulo irá conter as referências para todos os os controllers na nossa aplicação. Ele também precisa ser referenciado no nosso módulo principal. Portanto, substitua o arquivo **app.module.ts** pelo seguinte conteúdo:

```
import { Module } from '@nestjs/common';
import { AppController } from './app.controller';
import { AppService } from './app.service';
import { ControllersModule } from './gateways/controllers/controllers.module';

@Module({
imports: [ControllersModule],
controllers: [AppController],
providers: [AppService],
})
export class AppModule {}
```

Agora, execute os seguintes comandos para criar os controllers de projetos, tarefas e usuários:

1. **nest g controller gateways/controllers/projects**
2. **nest g controller gateways/controllers/tasks**
3. **nest g controller gateways/controllers/users**

Vamos criar agora algumas entidades que serão responsáveis por modelar a nossa aplicação. Primeiro, iremos criar uma interface para cada entidade. Portanto, crie as seguintes interfaces em seus respectivos arquivos:

src/domain/interfaces/user.interface.ts

```
import { IProject } from './project.interface';
import { ITask } from './task.interface';

export class IUser {
 id: number;
 firstName: string;
 lastName: string;
 email: string;
 password: string;
 projects: IProject[];
 tasks: ITask[];
}
```

src/domain/interfaces/task.interface.ts

```
import { IProject } from './project.interface';
import { IUser } from './user.interface';

export class ITask {
 id: number;
 name: string;
 status: 'pending' | 'completed';
 project: IProject;
 user: IUser;
}
```

src/domain/interfaces/project.interface.ts

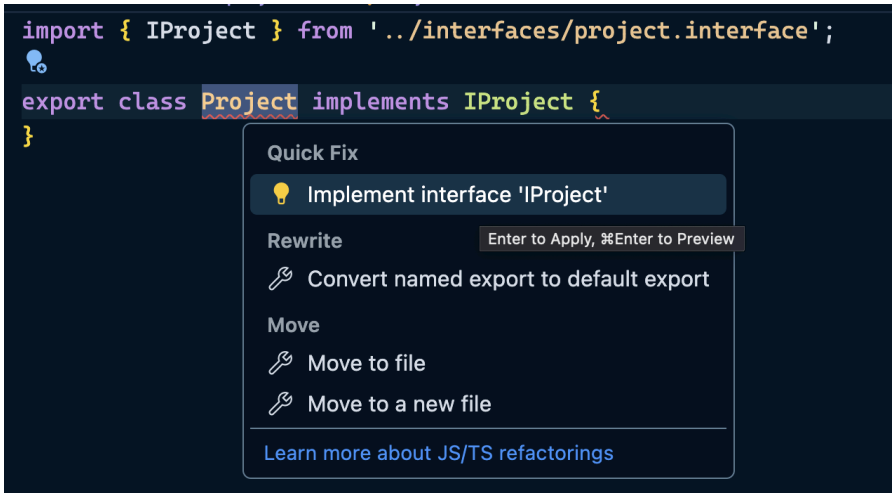
```
import { ITask } from './task.interface';
import { IUser } from './user.interface';

export interface IProject {
 id: number;
 name: string;
 description: string;
 tasks: ITask[];
 user: IUser;
}
```

Crie agora 3 arquivos:

- src/domain/entities/project.ts
- src/domain/entities/task.ts
- src/domain/entities/user.ts

Em cada arquivo, implemente a interface criada relacionada à sua entidade. Exemplo:



ℹ️**Dica**: crie a assinatura da classe como mostrado na imagem acima, clique no nome da classe e pressione **CTRL + .** Clique em “Implement interface IProject” e deixe o VSCode fazer o restante do trabalho.