Pontifícia Universidade Católica de Minas Gerais

Curso: Arquitetura de Software Distribuído
Disciplina: Plataformas Node.js
Professor: Samuel Martins
Valor: 35pts

Exercício 2

Introdução

Nesse exercício iremos implementar a aplicação de gerenciamento de projetos utilizando a Arquitetura CLEAN, juntamente com a persistência utilizando o padrão repository. Para concluir o exercício, iremos utilizar as seguintes ferramentas:

- Editor de código de sua preferência (VSCode, WebStorm, Emacs...);
    - Para o VSCode, iremos utilizar a extensão SQLite3 Editor.
- Postman para testarmos a API.

Informações sobre a entrega

A entrega deste exercício deve ser composta por:

- Código completo, seguindo o passo a passo do enunciado;
- Prints conforme orientado no último passo;

Passo 1 – Estrutura da aplicação

Neste primeiro passo iremos instalar alguns pacotes adicionais e criar toda a estrutura da nossa aplicação. Vamos criar todos os módulos necessários, os serviços para implementação dos casos de uso e repositórios. Os arquivos, pastas e módulos serão criados via linha de comando com ajuda do Nest CLI e ficarão vazios nesse primeiro momento.

Para começar, abra o terminal e rode o seguinte comando para instalarmos algumas dependências que serão utilizadas posteriormente:

- **npm install --save @nestjs/typeorm typeorm sqlite3 class-validator class-transformer**

OBSERVAÇÃO: é de SUMA importância que a ordem dos comandos seja seguida pois os módulos e arquivos precisam ser referenciados corretamente entre si. Os serviços são dependências de módulos e alguns módulos são dependentes de outros módulos. Ao rodar os comandos na ordem desse exercício, garantimos que as referências estarão aplicadas corretamente.

Estrutura de módulos

Vamos criar os módulos da aplicação e iniciar a construção da nossa estrutura. Lembre-se de que para que a estrutura fique aplicada de forma correta, este projeto deverá ter todas as pastas criadas no exercício anterior exatamente como elas foram sugeridas.

Execute os seguintes comandos, um por um, na seguinte ordem:

1. nest g module domain

2. nest g module domain/use-cases
3. nest g module domain/use-cases/projects
4. nest g module domain/use-cases/tasks
5. nest g module domain/use-cases/users
6. nest g module infrastructure
7. nest g module infrastructure/database
8. nest g module infrastructure/auth
9. nest g module gateways (lembrar de mover o controllers module pra ca);

Após a execução do passo 9, substitua o código do arquivo **gateways.module.ts** pelo código abaixo. Isso é necessário pois o módulo de controllers já foi criado anteriormente e precisamos referenciá-lo corretamente no módulo dos gateways:

```
import { Module } from '@nestjs/common';
import { ControllersModule } from './controllers/controllers.module';

@Module({
  imports: [ControllersModule],
})
export class GatewaysModule {}
```

Casos de uso – Projects

Execute os seguintes comandos para criar os casos de uso do domínio de Projetos. Repare que estamos passando a opção *–flat* para garantir que os arquivos serão criados dentro da pasta projects sem uma pasta adicional para cada caso de uso.

1. nest g service domain/use-cases/projects/get-all-projects –flat;
2. nest g service domain/use-cases/projects/get-project-by-id –flat;
3. nest g service domain/use-cases/projects/create-project –flat;

Casos de uso - Tasks

1. nest g service domain/use-cases/tasks/get-all-tasks –flat;
2. nest g service domain/use-cases/tasks/get-task-by-id –flat;
3. nest g service domain/use-cases/tasks/create-task –flat;
4. nest g service domain/use-cases/tasks/update-task –flat;

Casos de uso - Users

1. nest g service domain/use-cases/users/create-user –flat;
2. nest g service domain/use-cases/users/get-user-by-id –flat;

Repositórios

Os arquivos de repositório serão uma abstração em cima do padrão repositório já utilizado pelo TypeORM. Essa abstração garante que se precisarmos trocar a biblioteca padrão de ORM posteriormente, temos uma abstração em cima do padrão repositório aplicado para que

somente os arquivos de repositório sejam alterados. Estamos passando a opção *–no-spec* para não criarmos os arquivos de testes unitários, que não serão feitos neste momento.

1. nest g service infrastructure/database/repositories/projects.repository --flat --no-spec;
2. nest g service infrastructure/database/repositories/tasks.repository --flat --no-spec;
3. nest g service infrastructure/database/repositories/users.repository --flat --no-spec

Passo 2 – Estrutura de banco de dados

Iremos agora configurar a estrutura de persistência de dados da aplicação utilizando o TypeORM. Para isso, abra o arquivo **database.module.ts** e insira o seguinte código.

OBSERVAÇÃO: este trecho de código contém alguns imports de arquivos que serão criados posteriormente. Portanto, exclusivamente nesse passo, é normal termos alguns erros de build de projeto:

```
import { Module } from '@nestjs/common';
import { ProjectsRepositoryService } from
'./repositories/projects.repository.service';
import { TasksRepositoryService } from
'./repositories/tasks.repository.service';
import { UsersRepositoryService } from
'./repositories/users.repository.service';
import { TypeOrmModule } from '@nestjs/typeorm';

@Module({
  imports: [
    TypeOrmModule.forRoot({
      type: 'sqlite',
      database: 'db/sql.sqlite',
      entities: ['dist/**/*.entity{.ts,.js}'],
      synchronize: true,
      autoLoadEntities: true,
    }),
  ],
  providers: [
    ProjectsRepositoryService,
    TasksRepositoryService,
    UsersRepositoryService,
  ],
  exports: [
    ProjectsRepositoryService,
    TasksRepositoryService,
    UsersRepositoryService,
  ],
})
export class DatabaseModule {}
```

Aqui temos as instruções do projeto para utilizar o SQLite como nosso banco de dados, bem como algumas outras opções que irão facilitar o desenvolvimento do projeto como um todo.

O próximo passo agora é criar as entidades de Project, Tasks e Users. Siga as instruções abaixo, criando os arquivos necessários e inserindo o código em cada um dos arquivos criados. Note que é normal que alguns erros apareçam ao importar os arquivos porque as entidades possuem dependências entre si e os arquivos serão criados um após o outro.

src/infrastructure/database/entities/project.entity.ts

```
import { IProject } from 'src/domain/interfaces/project.interface';
import { ITask } from 'src/domain/interfaces/task.interface';
import { IUser } from 'src/domain/interfaces/user.interface';
import {
  Column,
  Entity,
  JoinColumn,
  ManyToOne,
  OneToMany,
  PrimaryGeneratedColumn,
} from 'typeorm';
import { TaskEntity } from './task.entity';
import { UserEntity } from './user.entity';

@Entity('project')
export class ProjectEntity implements IProject {
  @PrimaryGeneratedColumn()
  id: number;

  @Column({ name: 'name', nullable: false })
  name: string;

  @Column({ name: 'description', nullable: false })
  description: string;

  @OneToMany(() => TaskEntity, (task) => task.project)
  tasks: ITask[];

  @ManyToOne(() => UserEntity, (user) => user.projects)
  @JoinColumn()
  user: IUser;
}
```

src/infrastructure/database/entities/task.entity.ts

```
import { IProject } from 'src/domain/interfaces/project.interface';
import { ITask } from 'src/domain/interfaces/task.interface';
import { IUser } from 'src/domain/interfaces/user.interface';
import {
  Column,
  Entity,
  JoinColumn,
  ManyToOne,
  PrimaryGeneratedColumn,
} from 'typeorm';
import { ProjectEntity } from './project.entity';
import { UserEntity } from './user.entity';

@Entity('tasks')
export class TasksEntity implements ITask {
  @PrimaryGeneratedColumn()
  id: number;

  @Column({ name: 'name', nullable: false })
  name: string;

  @Column({ name: 'status', nullable: false })
  status: 'pending' | 'completed';

  @ManyToOne(() => ProjectEntity, (project) => project.tasks, {
    cascade: true,
    nullable: false,
  })
  project: IProject;

  @ManyToOne(() => UserEntity, (user) => user.tasks)
  @JoinColumn()
  user: IUser;
}
```

src/infrastructure/database/entities/user.entity.ts

```
import { IUser } from 'src/domain/interfaces/user.interface';
import { Entity, PrimaryGeneratedColumn, Column, OneToMany } from 'typeorm';
import { ProjectEntity } from './project.entity';
import { TaskEntity } from './task.entity';
import { IProject } from 'src/domain/interfaces/project.interface';
import { ITask } from 'src/domain/interfaces/task.interface';

@Entity('user')
export class UserEntity implements IUser {
  @PrimaryGeneratedColumn()
  id: number;

  @Column({ name: 'firstName', nullable: false })
  firstName: string;

  @Column({ name: 'lastName' })
  lastName: string;

  @Column({ name: 'email', nullable: false })
  email: string;

  @Column({ name: 'password', nullable: false })
  password: string;

  @OneToMany(() => ProjectEntity, (project) => project.user)
  projects: IProject[];

  @OneToMany(() => TaskEntity, (task) => task.user)
  tasks: ITask[];
}
```

Passo 3 – Implementação dos repositórios

Agora que já temos as entidades implementadas, podemos criar a camada de repositórios. Cada repositório terá um conjunto de métodos que irá acessar o banco de dados e fazer as operações necessárias, de acordo com os casos de uso que temos implementados.

Os repositórios irão extender de um tipo genérico Repository do próprio TypeORM e irão implementar uma interface customizada. Vamos criar primeiro as interfaces:

src/domain/repositories/projects-repository.interface.ts

```ts
import { DeepPartial } from 'typeorm';
import { IProject } from '../interfaces/project.interface';

export interface IProjectsRepository {
  findAll(userId: number): Promise<IProject[]>;
  findById(id: number): Promise<IProject>;
  add(payload: DeepPartial<IProject>): Promise<IProject>;
}
```

src/domain/repositories/tasks-repository.interface.ts

```ts
import { DeepPartial } from 'typeorm';
import { ITask } from '../interfaces/task.interface';

export interface ITasksRepository {
  findAll(userId: number): Promise<ITask[]>;
  findById(id: number): Promise<ITask>;
  add(payload: DeepPartial<ITask>): Promise<ITask>;
  updateById(payload: DeepPartial<ITask>);
}
```

src/domain/repositories/users-repository.interface.ts

```ts
import { DeepPartial } from 'typeorm';
import { IUser } from '../interfaces/user.interface';

export interface IUsersRepository {
  findById(id: number): Promise<IUser>;
  add(payload: DeepPartial<IUser>): Promise<IUser>;
}
```

E agora, as implementações de cada um dos repositórios:

src/infrastructure/database/repositories/projects.repository.service.ts

```
import { Injectable } from '@nestjs/common';
import { ProjectEntity } from '../entities/project.entity';
import { DataSource, DeepPartial, Repository } from 'typeorm';
import { IProject } from 'src/domain/interfaces/project.interface';
import { IProjectsRepository } from 'src/domain/repositories/projects-
repository.interface';

@Injectable()
export class ProjectsRepositoryService
  extends Repository<ProjectEntity>
  implements IProjectsRepository
{
  constructor(dataSource: DataSource) {
    super(ProjectEntity, dataSource.createEntityManager());
  }

  findAll(userId: number): Promise<IProject[]> {
    return this.findBy({ user: { id: userId } });
  }

  findById(id: number): Promise<IProject> {
    return this.findOneBy({ id });
  }

  add(payload: DeepPartial<IProject>): Promise<IProject> {
    return this.save(payload);
  }
}
```

src/infrastructure/database/repositories/tasks.repository.service.ts

```
import { Injectable } from '@nestjs/common';
import { DataSource, DeepPartial, Repository } from 'typeorm';
import { TaskEntity } from '../entities/task.entity';
import { ITask } from 'src/domain/interfaces/task.interface';
import { ITasksRepository } from 'src/domain/repositories/tasks-
repository.interface';

@Injectable()
export class TasksRepositoryService
  extends Repository<TaskEntity>
  implements ITasksRepository
```

```
{
  constructor(dataSource: DataSource) {
    super(TaskEntity, dataSource.createEntityManager());
  }

  findAll(userId: number): Promise<ITask[]> {
    return this.findBy({ user: { id: userId } });
  }

  findById(id: number): Promise<ITask> {
    return this.findOneBy({ id });
  }

  add(payload: DeepPartial<ITask>): Promise<ITask> {
    return this.save(payload) as Promise<ITask>;
  }

  updateById(payload: DeepPartial<ITask>) {
    return this.update(payload.id, payload);
  }
}
```

src/infrastructure/database/repositories/users.repository.service.ts

```
import { Injectable } from '@nestjs/common';
import { UserEntity } from '../entities/user.entity';
import { DataSource, DeepPartial, Repository } from 'typeorm';
import { IUser } from 'src/domain/interfaces/user.interface';
import { IUsersRepository } from 'src/domain/repositories/users-
repository.interface';

@Injectable()
export class UsersRepositoryService
  extends Repository<UserEntity>
  implements IUsersRepository
{
  constructor(dataSource: DataSource) {
    super(UserEntity, dataSource.createEntityManager());
  }

  findById(id: number): Promise<IUser> {
    return this.findOneBy({ id });
  }

  add(payload: DeepPartial<IUser>): Promise<IUser> {
```

```
    return this.save(payload) as Promise<IUser>;
  }
}
```

Após a execução dos passos acima, o módulo **database.module.ts** não deverá conter mais erros de import, visto que todos os arquivos necessários já foram criados corretamente.

Passo 4 – Implementação dos casos de uso

Primeiramente vamos implementar os DTOs, que serão responsáveis por modelar os dados que virão do controller e serão usados nos casos de uso para a nossa regra de negócio.

src/gateways/controllers/projects/dtos/create-project.dto.ts

```
import { IsNotEmpty, IsString } from 'class-validator';

export class CreateProjectDto {
  @IsNotEmpty({ message: 'O nome do projeto precisa de ser definido' })
  @IsString()
  name: string;

  @IsNotEmpty({ message: 'A descrição do projeto precisa de ser definida' })
  @IsString()
  description: string;
}
```

src/gateways/controllers/projects/dtos/update-project.dto.ts

```
import { PartialType } from '@nestjs/mapped-types';
import { CreateProjectDto } from './create-project.dto';

export class UpdateProjectDto extends PartialType(CreateProjectDto) {}
```

src/gateways/controllers/tasks/dtos/create-task.dto.ts

```
import { IsNotEmpty, IsString } from 'class-validator';

export class CreateTaskDto {
  @IsNotEmpty({ message: 'O nome da tarefa precisa ser definido' })
  @IsString()
  name: string;

  @IsNotEmpty({ message: 'O status da tarefa precisa ser definido' })
```

```
  @IsString()
  status: 'pending' | 'completed';

  projectId: number;
}
```

src/gateways/controllers/tasks/dtos/update-task.dto.ts

```
import { PartialType } from '@nestjs/mapped-types';
import { CreateTaskDto } from './create-task.dto';

export class UpdateTaskDto extends PartialType(CreateTaskDto) {
  id: number;
}
```

src/gateways/controllers/users/dtos/create-user.dto.ts

```
import { IsNotEmpty, IsString } from 'class-validator';

export class CreateUserDto {
  @IsNotEmpty()
  @IsString()
  firstName: string;

  @IsString()
  lastName?: string;

  @IsString()
  @IsNotEmpty()
  email: string;

  @IsString()
  @IsNotEmpty()
  password: string;
}
```

src/gateways/controllers/users/dtos/update-user.dto.ts

```
import { PartialType } from '@nestjs/mapped-types';
import { CreateUserDto } from './create-user.dto';

export class UpdateUserDto extends PartialType(CreateUserDto) {}
```

Iremos criar também uma interface base-use-case.ts para servir de padronização de implementação para nossos casos de uso:

src/domain/use-cases/base-use-case.ts

```
export interface BaseUseCase {
  execute(...args: unknown[]): Promise<unknown>;
}
```

Com os DTOs e interfaces criadas, vamos agora criar os nossos casos de uso. Olhando do ponto de vista de requisitos de negócio, a nossa aplicação precisa ter a capacidade de:

- Criar usuário;
- Listar usuário por ID;
- Criar projetos;
- Listar todos os projetos de um usuário;
- Listar projeto por ID;
- Criar tarefas;
- Listar todas as tarefas de um usuário;
- Listar tarefa por ID

Os casos de uso são em sua essência a implementação dos requisitos de negócio da aplicação. Sendo assim, crie os arquivos com os respectivos conteúdos seguindo as instruções abaixo:

Users

src/domain/use-cases/users/create-user.service.ts

```
import { Injectable } from '@nestjs/common';
import { BaseUseCase } from '../base-use-case';
import { CreateUserDto } from 'src/gateways/controllers/users/dtos/create-
user.dto';
import { UsersRepositoryService } from
'src/infrastructure/database/repositories/users.repository.service';
import { IUser } from 'src/domain/interfaces/user.interface';

@Injectable()
export class CreateUserService implements BaseUseCase {
  constructor(private readonly usersRepository: UsersRepositoryService) {}

  async execute(user: CreateUserDto): Promise<IUser> {
    const createdUser = await this.usersRepository.add(user);

    if (!createdUser) {
      throw new Error('Usuário não pôde ser criado');
```

```
    }

    return createdUser;
  }
}
```

src/domain/use-cases/users/get-user-by-id.service.ts

```
import { Injectable } from '@nestjs/common';
import { IUser } from 'src/domain/interfaces/user.interface';
import { UsersRepositoryService } from
'src/infrastructure/database/repositories/users.repository.service';

@Injectable()
export class GetUserByIdService {
  constructor(private readonly usersRepository: UsersRepositoryService) {}

  async execute(userId: number): Promise<IUser> {
    const user = await this.usersRepository.findById(userId);

    if (!user) {
      throw new Error('Usuário não encontrado');
    }

    return user;
  }
}
```

Projects

src/domain/use-cases/projects/create-project.service.ts

```
import { Injectable } from '@nestjs/common';
import { BaseUseCase } from '../base-use-case';
import { ProjectsRepositoryService } from
'src/infrastructure/database/repositories/projects.repository.service';
import { CreateProjectDto } from
'src/gateways/controllers/projects/dtos/create-project.dto';
import { IProject } from 'src/domain/interfaces/project.interface';
import { UsersRepositoryService } from
'src/infrastructure/database/repositories/users.repository.service';

@Injectable()
export class CreateProjectService implements BaseUseCase {
```

```
  constructor(
    private readonly usersRepository: UsersRepositoryService,
    private readonly projectsRepository: ProjectsRepositoryService,
  ) {}

  async execute(payload: {
    project: CreateProjectDto;
    userId: number;
  }): Promise<IProject> {
    // fetch user data
    const userData = await this.usersRepository.findById(payload.userId);

    if (!userData) {
      throw new Error('Usuário não encontrado');
    }

    const createdProject = await this.projectsRepository.add({
      name: payload.project.name,
      description: payload.project.description,
      user: { id: userData.id },
    });

    if (!createdProject) {
      throw new Error('Erro ao criar projeto');
    }

    return createdProject;
  }
}
```

src/domain/use-cases/projects/get-all-projects.service.ts

```
import { Injectable } from '@nestjs/common';
import { BaseUseCase } from '../base-use-case';
import { ProjectsRepositoryService } from
'src/infrastructure/database/repositories/projects.repository.service';
import { IProject } from 'src/domain/interfaces/project.interface';
import { UsersRepositoryService } from
'src/infrastructure/database/repositories/users.repository.service';

@Injectable()
export class GetAllProjectsService implements BaseUseCase {
  constructor(
    private readonly usersRepository: UsersRepositoryService,
    private readonly projectsRepository: ProjectsRepositoryService,
```

```
  ) {}

  async execute(userId: number): Promise<IProject[]> {
    // fetch user data
    const userData = await this.usersRepository.findById(userId);

    if (!userData) {
      throw new Error('Usuário não encontrado');
    }

    const projects = await this.projectsRepository.findAll(userId);

    if (!projects) {
      throw new Error('Erro ao recuperar projetos');
    }

    return projects;
  }
}
```

src/domain/use-cases/projects/get-project-by-id.service.ts

```
import { Injectable } from '@nestjs/common';
import { IProject } from 'src/domain/interfaces/project.interface';
import { ProjectsRepositoryService } from
'src/infrastructure/database/repositories/projects.repository.service';
import { UsersRepositoryService } from
'src/infrastructure/database/repositories/users.repository.service';

@Injectable()
export class GetProjectByIdService {
  constructor(
    private readonly usersRepository: UsersRepositoryService,
    private readonly projectsRepository: ProjectsRepositoryService,
  ) {}

  async execute(payload: {
    userId: number;
    projectId: number;
  }): Promise<IProject> {
    // fetch user data
    const userData = await this.usersRepository.findById(payload.userId);

    if (!userData) {
      throw new Error('Usuário não encontrado');
```

```
    }

    const project = await
this.projectsRepository.findById(payload.projectId);

    if (!project) {
      throw new Error('Erro ao recuperar projetos');
    }

    return project;
  }
}
```

Tasks

src/domain/use-cases/tasks/create-task.service.ts

```
import { Injectable } from '@nestjs/common';
import { BaseUseCase } from '../base-use-case';
import { TasksRepositoryService } from
'src/infrastructure/database/repositories/tasks.repository.service';
import { CreateTaskDto } from 'src/gateways/controllers/tasks/dtos/create-
task.dto';
import { ITask } from 'src/domain/interfaces/task.interface';
import { UsersRepositoryService } from
'src/infrastructure/database/repositories/users.repository.service';
import { ProjectsRepositoryService } from
'src/infrastructure/database/repositories/projects.repository.service';

@Injectable()
export class CreateTaskService implements BaseUseCase {
  constructor(
    private readonly usersRepository: UsersRepositoryService,
    private readonly tasksRepository: TasksRepositoryService,
    private readonly projcetsRepository: ProjectsRepositoryService,
  ) {}

  async execute(payload: {
    task: CreateTaskDto;
    userId: number;
  }): Promise<ITask> {
    // fetch user data
    const userData = await this.usersRepository.findById(payload.userId);

    if (!userData) {
```

```
      throw new Error('Usuário não encontrado');
    }

    const projectData = await this.projcetsRepository.findById(
      payload.task.projectId,
    );

    if (!projectData) {
      throw new Error('Projeto não encontrado');
    }

    const createdTask = await this.tasksRepository.add({
      name: payload.task.name,
      status: payload.task.status,
      project: projectData,
      user: { id: userData.id },
    });

    if (!createdTask) {
      throw new Error('Erro ao criar tarefa');
    }

    return createdTask;
  }
}
```

src/domain/use-cases/tasks/get-all-tasks.service.ts

```
import { Injectable } from '@nestjs/common';
import { BaseUseCase } from '../base-use-case';
import { ITask } from 'src/domain/interfaces/task.interface';
import { TasksRepositoryService } from
'src/infrastructure/database/repositories/tasks.repository.service';
import { UsersRepositoryService } from
'src/infrastructure/database/repositories/users.repository.service';

@Injectable()
export class GetAllTasksService implements BaseUseCase {
  constructor(
    private readonly usersRepository: UsersRepositoryService,
    private readonly tasksRepository: TasksRepositoryService,
  ) {}
```

```
  async execute(payload: { userId: number }): Promise<ITask[]> {
    // fetch user data
    const userData = await this.usersRepository.findById(payload.userId);

    if (!userData) {
      throw new Error('Usuário não encontrado');
    }

    const tasks = await this.tasksRepository.findAll(payload.userId);

    if (!tasks) {
      throw new Error('Erro ao listar tarefas');
    }

    return tasks;
  }
}
```

src/domain/use-cases/tasks/get-task-by-id.service.ts

```
import { Injectable } from '@nestjs/common';
import { BaseUseCase } from '../base-use-case';
import { ITask } from 'src/domain/interfaces/task.interface';
import { TasksRepositoryService } from
'src/infrastructure/database/repositories/tasks.repository.service';
import { UsersRepositoryService } from
'src/infrastructure/database/repositories/users.repository.service';

@Injectable()
export class GetTaskByIdService implements BaseUseCase {
  constructor(
    private readonly usersRepository: UsersRepositoryService,
    private readonly tasksRepository: TasksRepositoryService,
  ) {}

  async execute(payload: { taskId: number; userId: number }): Promise<ITask>
{
    // fetch user data
    const userData = await this.usersRepository.findById(payload.userId);

    if (!userData) {
      throw new Error('Usuário não encontrado');
    }

    const task = await this.tasksRepository.findById(payload.taskId);
```

```
    if (!task) {
      throw new Error('Erro ao listar tarefas');
    }

    return task;
  }
}
```

src/domain/use-cases/tasks/update-task.service.ts

```
import { Injectable } from '@nestjs/common';
import { BaseUseCase } from '../base-use-case';
import { TasksRepositoryService } from
'src/infrastructure/database/repositories/tasks.repository.service';
import { UsersRepositoryService } from
'src/infrastructure/database/repositories/users.repository.service';
import { UpdateTaskDto } from 'src/gateways/controllers/tasks/dtos/update-
task.dto';
import { ITask } from 'src/domain/interfaces/task.interface';

@Injectable()
export class UpdateTaskService implements BaseUseCase {
  constructor(
    private readonly usersRepository: UsersRepositoryService,
    private readonly tasksRepository: TasksRepositoryService,
  ) {}

  async execute(payload: {
    task: UpdateTaskDto;
    userId: number;
  }): Promise<ITask> {
    const userData = await this.usersRepository.findById(payload.userId);

    if (!userData) {
      throw new Error('Usuário não encontrado');
    }

    await this.tasksRepository.updateById(payload.task);
    const task = this.tasksRepository.findById(payload.task.id);

    if (!task) {
      throw new Error('Tarefa não encontrado');
    }
```

```
    return task;
  }
}
```

Passo 5 – Implementação dos controllers

Cada domínio deve ter um controller para receber as chamadas da API e redirecioná-las aos casos de uso. Vamos criar os controllers para Projetos, Tarefas e Usuários.

OBSERVAÇÃO: Como ainda não temos a autenticação implementada neste passo, iremos utilizar uma constante nos arquivos de controller com o userID = 1. Isso irá garantir que conseguimos fazer as chamadas para criar os projetos e tarefas associados a um usuário sem a necessidade de implementar a autenticação neste momento.

src/gateways/controllers/projects/projects.controller.ts

```
import {
  Body,
  Controller,
  Get,
  NotFoundException,
  Param,
  Post,
  UnprocessableEntityException,
} from '@nestjs/common';
import { CreateProjectService } from 'src/domain/use-cases/projects/create-project.service';
import { GetAllProjectsService } from 'src/domain/use-cases/projects/get-all-projects.service';
import { GetProjectByIdService } from 'src/domain/use-cases/projects/get-project-by-id.service';
import { CreateProjectDto } from './dtos/create-project.dto';

const userId = 1;
@Controller('projects')
export class ProjectsController {
  constructor(
    private readonly getAllProjectsUseCase: GetAllProjectsService,
    private readonly getProjectByIdUseCase: GetProjectByIdService,
    private readonly createProjectUseCase: CreateProjectService,
  ) {}

  @Get()
  async findAll() {
    try {
      return await this.getAllProjectsUseCase.execute(userId);
```

```
    } catch (error) {
      throw new NotFoundException(error.message);
    }
  }

  @Get(':id')
  async findOne(@Param('id') id: number) {
    try {
      return await this.getProjectByIdUseCase.execute({
        userId,
        projectId: id,
      });
    } catch (error) {
      throw new NotFoundException(error.message);
    }
  }

  @Post()
  async create(@Body() createProjectDto: CreateProjectDto) {
    try {
      return await this.createProjectUseCase.execute({
        userId,
        project: createProjectDto,
      });
    } catch (error) {
      throw new UnprocessableEntityException(error.message);
    }
  }
}
```

src/gateways/controllers/tasks/tasks.controller.ts

```
import {
  Body,
  Controller,
  Get,
  NotFoundException,
  Param,
  Post,
  UnprocessableEntityException,
} from '@nestjs/common';
import { CreateTaskService } from 'src/domain/use-cases/tasks/create-
task.service';
import { GetAllTasksService } from 'src/domain/use-cases/tasks/get-all-
tasks.service';
```

```
import { GetTaskByIdService } from 'src/domain/use-cases/tasks/get-task-by-
id.service';
import { CreateTaskDto } from './dtos/create-task.dto';

const userId = 1;
@Controller('tasks')
export class TasksController {
  constructor(
    private readonly getAllTasksUseCase: GetAllTasksService,
    private readonly getTaskByIdUseCase: GetTaskByIdService,
    private readonly createTaskUseCase: CreateTaskService,
  ) {}

  @Get()
  async findAll() {
    try {
      return await this.getAllTasksUseCase.execute({ userId });
    } catch (error) {
      throw new NotFoundException(error.message);
    }
  }

  @Get(':id')
  async findOne(@Param('id') id: number) {
    try {
      return await this.getTaskByIdUseCase.execute({
        userId,
        taskId: id,
      });
    } catch (error) {
      throw new NotFoundException(error.message);
    }
  }

  @Post()
  async create(@Body() createTaskDto: CreateTaskDto) {
    try {
      return await this.createTaskUseCase.execute({
        userId,
        task: createTaskDto,
      });
    } catch (error) {
      throw new UnprocessableEntityException(error.message);
    }
  }
}
```

src/gateways/controllers/users/users.controller.ts

```
import {
  Body,
  Controller,
  Get,
  NotFoundException,
  Param,
  Post,
  UnprocessableEntityException,
} from '@nestjs/common';
import { CreateUserService } from 'src/domain/use-cases/users/create-
user.service';
import { GetUserByIdService } from 'src/domain/use-cases/users/get-user-by-
id.service';
import { CreateUserDto } from './dtos/create-user.dto';

@Controller('users')
export class UsersController {
  constructor(
    private readonly getUserUseCase: GetUserByIdService,
    private readonly createUserUseCase: CreateUserService,
  ) {}

  @Get(':id')
  async findOne(@Param('id') id: number) {
    try {
      return await this.getUserUseCase.execute(+id);
    } catch (error) {
      throw new NotFoundException(error.message);
    }
  }

  @Post()
  async create(@Body() createUserDto: CreateUserDto) {
    try {
      return await this.createUserUseCase.execute({ ...createUserDto });
    } catch (error) {
      throw new UnprocessableEntityException(error.message);
    }
  }
}
```

Passo 6 – Conferência das dependências de módulos

Vamos garantir que as dependências dos módulos estão sendo corretamente utilizadas e exportadas entre os arquivos. Substitua o código dos arquivos abaixo pelos seus respectivos conteúdos:

src/domain/use-cases/use-cases.module.ts

```
import { Module } from '@nestjs/common';
import { ProjectsModule } from './projects/projects.module';
import { TasksModule } from './tasks/tasks.module';
import { UsersModule } from './users/users.module';

@Module({
  imports: [ProjectsModule, TasksModule, UsersModule],
  exports: [ProjectsModule, TasksModule, UsersModule],
})
export class UseCasesModule {}
```

src/gateways/controllers/controllers.module.ts

```
import { Module } from '@nestjs/common';
import { ProjectsController } from './projects/projects.controller';
import { TasksController } from './tasks/tasks.controller';
import { UsersController } from './users/users.controller';
import { UseCasesModule } from 'src/domain/use-cases/use-cases.module';

@Module({
  imports: [UseCasesModule],
  controllers: [ProjectsController, TasksController, UsersController],
})
export class ControllersModule {}
```

src/domain/use-cases/projects/projects.module.ts

```
import { Module } from '@nestjs/common';
import { GetAllProjectsService } from './get-all-projects.service';
import { GetProjectByIdService } from './get-project-by-id.service';
import { CreateProjectService } from './create-project.service';
import { DatabaseModule } from
'src/infrastructure/database/database.module';

@Module({
  imports: [DatabaseModule],
  providers: [
```

```
    GetAllProjectsService,
    GetProjectByIdService,
    CreateProjectService,
  ],
  exports: [GetAllProjectsService, GetProjectByIdService,
CreateProjectService],
})
export class ProjectsModule {}
```

src/domain/use-cases/tasks/tasks.module.ts

```
import { Module } from '@nestjs/common';
import { GetAllTasksService } from './get-all-tasks.service';
import { GetTaskByIdService } from './get-task-by-id.service';
import { CreateTaskService } from './create-task.service';
import { UpdateTaskService } from './update-task.service';
import { DatabaseModule } from
'src/infrastructure/database/database.module';

@Module({
  imports: [DatabaseModule],
  providers: [
    GetAllTasksService,
    GetTaskByIdService,
    CreateTaskService,
    UpdateTaskService,
  ],
  exports: [
    GetAllTasksService,
    GetTaskByIdService,
    CreateTaskService,
    UpdateTaskService,
  ],
})
export class TasksModule {}
```

src/domain/use-cases/users/users.module.ts

```
import { Module } from '@nestjs/common';
import { CreateUserService } from './create-user.service';
import { GetUserByIdService } from './get-user-by-id.service';
import { DatabaseModule } from
'src/infrastructure/database/database.module';
```

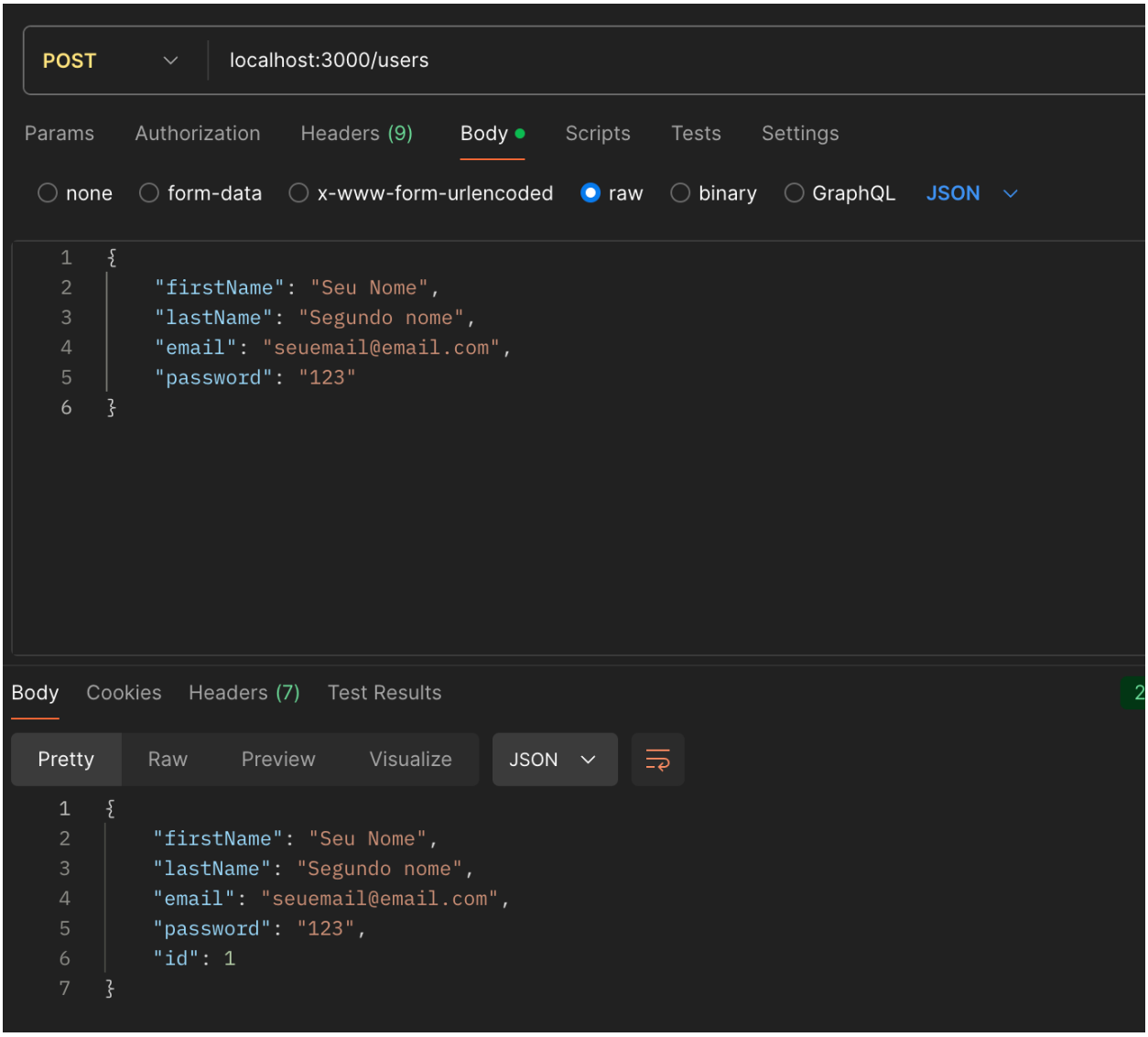
```ts
@Module({
  imports: [DatabaseModule],
  providers: [CreateUserService, GetUserByIdService],
  exports: [CreateUserService, GetUserByIdService],
})
export class UsersModule {}
```

Passo 7 – Testes via Postman

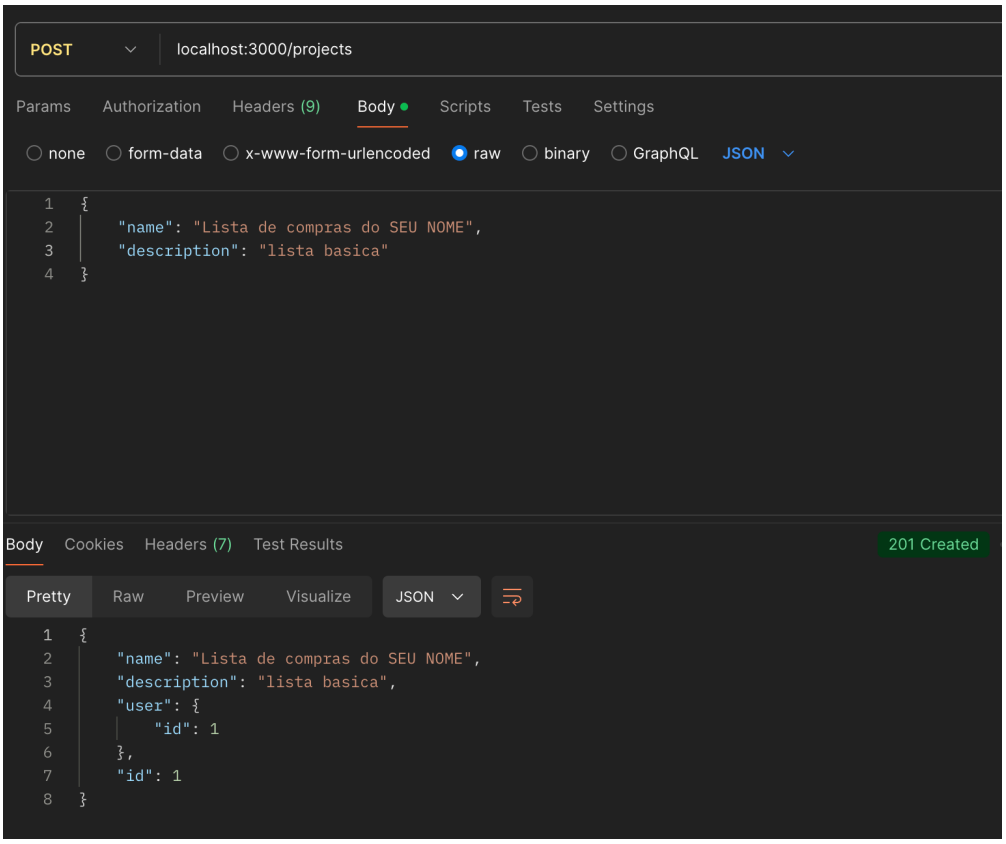
Para garantir que tudo funciona conforme o esperado, abra o Postman (ou qualquer outro cliente de requisições de sua preferência) e faça as seguintes requisições nas URLs.

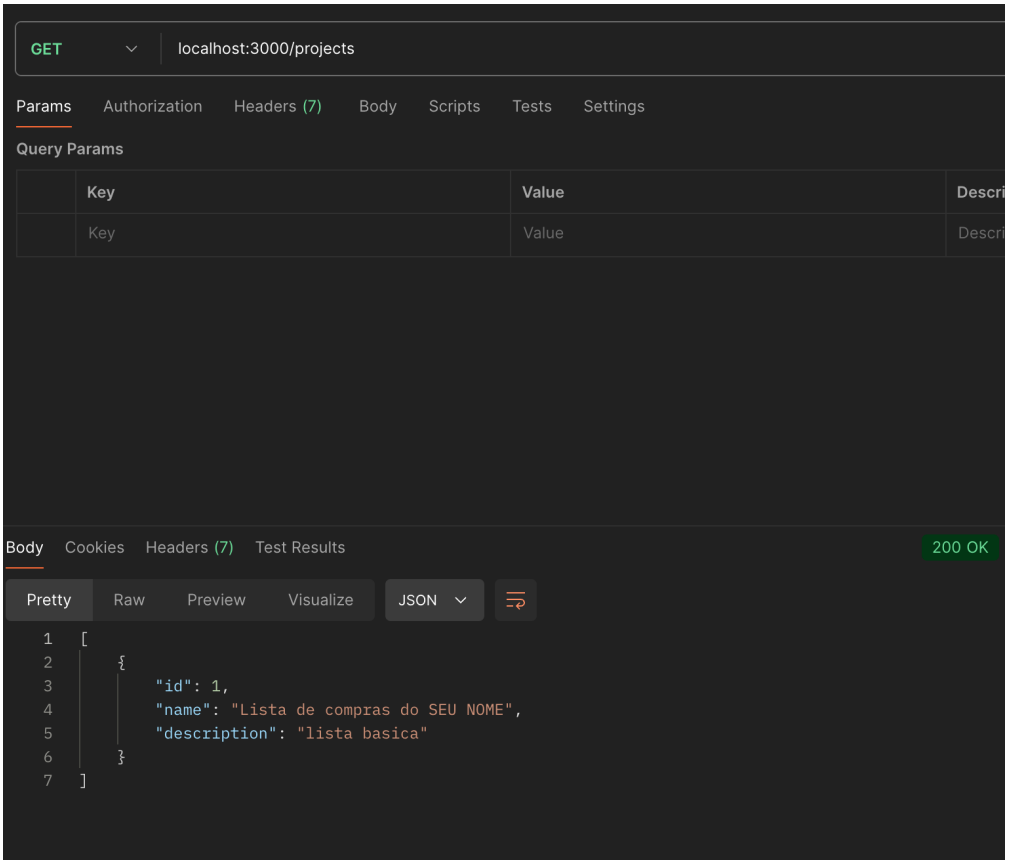
Tire um print de cada requisição e envie como forma de entrega do exercício. Substitua os dados do print pelos SEUS dados (exemplo: Seu Nome = Samuel...)

Requisição /users



Requisição /projects





Requisição /tasks