UNIVERSIDADE FEDERAL DO CEARÁ
CURSO DE ENGENHARIA DE COMPUTAÇÃO

GABRIEL RIBEIRO CAMELO

TCC BD, UM SISTEMA DE PUBLICAÇÃO DE TRABALHOS ACADÊMICOS

SOBRAL
2022

GABRIEL RIBEIRO CAMELO

TCC BD, UM SISTEMA DE PUBLICAÇÃO DE TRABALHOS ACADÊMICOS

Trabalho de Conclusão de Curso apresentado ao Programa de graduação em Engenharia de Computação da Universidade Federal do Ceará, como requisito parcial à obtenção do título de Bacharel em Engenharia de Computação.

Orientador: Prof. Me. Fernando Rodrigues de Almeida Júnior

SOBRAL

2022

GABRIEL RIBEIRO CAMELO

TCC BD, UM SISTEMA DE PUBLICAÇÃO DE TRABALHOS ACADÊMICOS

Trabalho de Conclusão de Curso apresentado ao Programa de graduação em Engenharia de Computação da Universidade Federal do Ceará, como requisito parcial à obtenção do título de Bacharel em Engenharia de Computação.

Aprovada em:

BANCA EXAMINADORA

_____
Prof. Me. Fernando Rodrigues de Almeida Júnior (Orientador)
Universidade Federal do Ceará (UFC)

_____
Prof. Me. Erick Aguiar Donato
Universidade Federal do Ceará (UFC)

_____
Prof. Dr. Fischer Jonatas Ferreira
Universidade Federal do Ceará (UFC)

A Deus e a minha família por terem me guiado durante todo o curso, me ajudando em todos os momentos.

AGRADECIMENTOS

Primeiramente gostaria de agradecer principalmente a Deus por ter me ajudado a passar por todas as minhas experiências que tive em minha vida até o momento, o que me possibilitou ter chegado até aqui. Aos meus pais, Estela e Assis Júnior, por todo apoio e suporte durante o curso e a me ajudarem a não desistir do curso no meio dele.

Ao meu irmão mais velho Jayro Rafael por ter me inspirado a seguir esse caminho da programação.

Ao meu irmão mais novo Mikael pelos ensinamentos de vida.

Aos professores que tive durante o curso, em especial ao professor Me. Fernando Rodrigues de Almeida Júnior pela orientação durante o meu projeto de TCC e da minha monografia, e também pelas conversas e risadas.

Aos meus amigos de longa data, Alexandre, Maykson, Danillo, Derlan, dentre outros, aos quais eu tenho muito afeto e carinho, pela amizade, companheirismo e apoio ao longo dessa jornada.

À minha namorada Raquel por estar ao meu lado durante todos os momentos de aperreio, diversão e todos pelos quais passamos, por ser essa pessoa especial que me ajuda a deixar a minha cabeça no lugar em diversos momentos.

E finalmente aos meus amigos que cultivei ao longo do curso, em especial aos meus amigos Gabriel de Souza, Lucas Pedrosa, Felipe Barros, Lucas França, Gilson, Wilson, Stefane, Icaro e Robson por todos os momentos que passamos na faculdade, pelo companheirismo, por toda ajuda e suporte em diversos momentos do decorrer do curso.

“Truth can only be found in one place: the code.” (Robert Cecil Martin, Uncle Bob)

RESUMO

A academia todo ano produz textos científicos: seja os artigos de graduação, com alunos que estão se formando; seja as teses de doutorados com os novos doutores que conseguem esse título a cada ano ou ainda com as novas pesquisas que estão sendo desenvolvidas e ainda com as dissertações de concludentes do mestrado. Com tudo isso, há a necessidade de um local onde se tenha esses artigos acessíveis para as pessoas interessadas em ler ou estudar as novidades do mundo acadêmico. Para isso, as universidades disponibilizam suas publicações nos seus próprios repositórios online, tornando-os assim disponíveis para o público em geral. Entretanto, para se introduzir esses textos nessas plataformas há muita burocracia, visto que o trabalho passa por diversas etapas até ser disponibilizado efetivamente nas mesmas. Com esse intuito de facilitar a disponibilização dos textos acadêmicos, planejamos essa API (*Application Programming Interface* - Interface de programação de aplicações) para a construção de uma plataforma em que os próprios autores possam disponibilizar os seus textos de forma simples e segura.

Palavras-chave: Facilidade; Artigos; Academia; Publicação; API; repositório de publicações acadêmicas.

ABSTRACT

Every year, academy produces scientific texts, whether undergraduate articles, with students who are graduating, or doctoral theses with new doctors who obtain this title every year or with new research that is being developed and still with the dissertations of the master's graduates. In this way, there is a need for a place where these articles are accessible to people interested in reading or studying the news of the academic world. For this, universities make their publications available in their own online repositories, thus making them available to the general public. However, to introduce these texts on these platforms there is a lot of bureaucracy, since the work goes through several stages until it is effectively made available. In order to facilitate the availability of academic texts, we designed this API (Application Programming Interface) to build a platform where authors themselves can make their texts available in a simple and secure way.

Keywords: Ease; articles; Academy; Publication; API; repository of academic publications.

Lista de figuras

Lista de tabelas

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
BCSO	Biblioteca do Campus de Sobral
DIP	*Dependency Inversion Principle*
EESC	Escola de Engenharia de São Carlos
ISP	Interface *Segregation Principle*
JSON	*JavaScript Object Notation*
LSP	Liskov *Substitution Principle*
MER	Modelo Entidade Relacionamento
OCP	*Open-Closed Principle*
ORM	*Object-Relational Mapping*
SGBD	Sistema Gerenciador de Banco de Dados
SQL	*Structured Query Language*
SRP	*Single Responsability Principle*
TCC	Trabalho de Conclusão de Curso
UFRA	Universidade Federal Rural da Amazônia
UFC	Universidade Federal do Ceará
USP	Universidade de São Paulo

Sumário

01 Introdução

Com o avanço da tecnologia e o passar das gerações, cada vez mais as pessoas, de um modo geral vêm, buscando por informações fáceis de se obter, bem como a facilidade de poder deixar algo registrado ao alcance das pessoas. Isso não é diferente na academia, visto que, cada vez mais, se faz necessário acelerar a disseminação de informação pelo meio universitário.

Tendo em vista buscar facilitar o registro, bem como o acesso de novos ou antigos textos científicos, o presente trabalho visa implementar uma API para um sistema de repositório de textos científicos. Neste sistema, o próprio autor, seja ele aluno ou professor, orientando ou orientador, pode registrar um novo artigo científico tendo unicamente que fazer um prévio cadastro na plataforma. Em seguida, pode-se registrar o arquivo, executando poucos passos até finalizar o cadastro do documento.

Como o cenário atual de registro de artigos em plataformas digitais é muito burocrático e muitas vezes é até mesmo demorado, torna-se interessante o desenvolvimento de tal API, de modo que ela seja mais fluida e acessível para as pessoas de uma forma geral.

Em tal plataforma não seria necessário ser um membro da comunidade acadêmica para ter acesso a esses arquivos. Entretanto, para registrar um novo documento será necessário ter um e-mail com o domínio de alguma universidade do país.

Este trabalho foi estruturado em capítulo e em seções. O capítulo 2 apresenta a motivação do trabalho, seguido do capítulo 3 que a apresenta os objetivos do mesmo. Seguindo a sequência o capítulo 4 apresenta trabalhos que fizeram uso de técnicas e ideias que estão presentes em nosso trabalho. Em seguida o capítulo 5 foi elaborado de forma que o leitor possa entender a fundamentação teórica que engloba o projeto. Ele está dividido em seções que abordarão temas como princípios de programação, padrões de projeto, banco de dados, HTTP e *backend*. O capítulo 6 trata das técnicas utilizadas no presente trabalho. O capítulo 7 apresenta os resultados obtidos durante o desenvolvimento do sistema. O capítulo 8 apresenta as conclusões e por fim o 9 apresenta possíveis trabalhos futuros.

02 Motivação

Atualmente, a UFC (Universidade Federal do Ceará) conta com o sistema do Pergamum[1] para organizar e catalogar todo o seu acervo de suas mais diversas obras físicas e digitais. Dentre elas, se encontram os TCC's (Trabalhos de Conclusão de Curso) dos estudantes dos mais variados cursos da Universidade.

A diferença do sistema desenvolvido para o Pergamum e o Repositório institucional da UFC é que ele é alimentado pela comunidade sem a necessidade de passar por uma instituição de ensino. De modo que ele cresce a medida que a comunidade acadêmica insere novos trabalhos no mesmo, sem a presença desse mediador.

Quando queremos encontrar alguma monografia, dissertação ou tese, devemos selecionar o tipo de obra desejada e inserir um termo de pesquisa. Só então o sistema retorna as obras disponíveis que estão relacionadas com o termo inserido. Esse termo pode ser parte de um título, do nome do autor, do assunto a ser tratado ou pode ser uma palavra livre. Esse sistema é atualizado por um funcionário da biblioteca da Universidade. Tendo em vista que nem sempre esse funcionário tem acesso a todos os dados de todos os trabalhos que são realizados, o sistema acaba ficando desatualizado.

Com isso, a motivação deste trabalho é construir uma API (*Application Programming Interface* - Interface de programação de aplicações) que poderá ser utilizada para o desenvolvimento de uma plataforma colaborativa em que os próprios alunos autores possam ir atualizando o sistema, inserindo seus próprios trabalhos, incluindo dados como título, autores, data da publicação, universidade entre outros. Desta forma, a base de dados irá se tornar cada vez mais completa, bem como o trabalho de atualização do sistema ficaria descentralizado, promovendo uma melhor experiência para quem for procurar por trabalhos desse tipo.

Ao observar o modo como as atuais plataformas de disponibilização de TCC’s das Universidades funcionam, como o Pergamum e o Repositório Institucional da UFC, podemos perceber que, para novos trabalhos serem inseridos, necessitam passar primeiramente pela secretaria. Em seguida, o trabalho é enviado para a biblioteca e somente então, por fim, é inserido no sistema.

Como o repositório institucional da faculdade só pode ser manipulado por servidores, bolsistas e voluntários (no caso dos dois últimos, eles ainda teriam que passar

por um treinamento) isso acaba tornando o processo de disponibilização dos trabalhos acadêmicos um pouco mais demorado devido a burocracia envolvida.

O Pergamum é um sistema à parte do repositório institucional da UFC. Tendo isso em mente, os responsáveis por disponibilizar os trabalhos acadêmicos têm o trabalho de inserir em duas plataformas diferentes, que exigem informações específicas e diferentes, pois as suas exigências não são as mesmas[2][3].

03 Objetivos

3.1 Objetivos Gerais

Utilizar tecnologias *web* modernas e de banco de dados na construção de uma API para armazenamento e consulta de textos científicos de forma fácil e rápida. Os próprios autores dos textos poderão cadastrar os seus trabalhos, de modo que, assim, estariam disponibilizando-os com a finalidade de disseminar o conhecimento científico de forma mais rápida e eficaz em toda a sociedade.

3.2 Objetivos Específicos

- Levantar os requisitos do sistema;
- Modelar a base de dados do sistema;
- Construir o banco de dados;
- Construir uma API para comunicação com o banco de dados.

04 Trabalhos Relacionados

No trabalho intitulado “Arquitetura Para Integração De Serviços Via Api Rest Para Plataforma De Acervos Digitais.”[4], a API Rest foi utilizada com a finalidade de aproveitar as vantagens de uma arquitetura descentralizada, de modo a utilizar o DSpace, ferramenta utilizada naquele trabalho, de modo mais eficiente, otimizando o processo de inserção. No presente trabalho também foi optado por desenvolver uma API, mas com a finalidade de poder oferecer os dados em mais de uma plataforma, no caso *web* e *mobile*. Ambos os trabalhos se relacionam a partir do pondo que eles se propõem a fazer uso de uma arquitetura descentralizada para o desenvolvimento do sistema.

No trabalho de Cristina Guerra Matos[5], apresentado na 10ª Conferência Luso-Brasileira De Ciência Aberta em Manaus, o objetivo foi a construção de uma biblioteca virtual de trabalhos acadêmicos para a Universidade Federal Rural da Amazônia (UFRA), igualmente ao trabalho da Larissa Herculano[4] eles utilizaram o DSpace, um software livre para armazenamento. A ideia é desenvolver uma biblioteca virtual de trabalhos acadêmicos para a UFRA, diferente do presente trabalho, em que pretendemos que autores de diversas instituições acadêmicas possam disponibilizar os seus trabalhos e pesquisas de modo a deixá-los aberto ao público.

No trabalho de Teresinha das Graças[6], intitulado “Biblioteca Digital De Trabalhos Acadêmicos Da Universidade De São Paulo: Desenvolvimento E Implementação Na EESC/USP”, o objetivo é desenvolver um repositório institucional para atender às necessidades da Escola de Engenharia de São Carlos, da Universidade de São Paulo (EESC/USP). Como podemos ver, ele também se propõe em fazer um repositório de uso institucional, o que vai de encontro a ideia proposta aqui.

05 Fundamentação Teórica

Para o desenvolvimento do projeto, nós utilizamos algumas estratégias de qualidade de software, como o S.O.L.I.D.[7] e o *Singleton*, que iremos abordar nas seções a seguir.

Esta seção apresenta os seguintes tópicos:

- 5.1 S.O.L.I.D.
  - 5.1.1 S - Single Responsibility Principle (Princípio da Responsabilidade Única)
  - 5.1.2 O - Open-closed principle (Princípio Aberto Fechado)
  - 5.1.3 L - Liskov Substitution Principle (Princípio da Substituição de LIskov)
  - 5.1.4 I - Interface Segregation Principle (Princípio da Segregação de Interface)
  - 5.1.5 D - Dependency Inversion Principle (Princípio da Inversão de Dependências)
- 5.2 Singleton
- 5.3 Injeção de Dependências
- 5.4 Sistemas gerenciadores de banco de dados (SGBD)
  - 5.4.1 Modelagem de banco de dados
  - 5.4.2 Formas de comunicação com o banco de dados
- 5.5 HTTP (Hypertext Transfer Protocol - Protocolo de Transferência de Hipertexto)
- 5.6 Backend
  - 5.6.1 API (Aplication Programming Interface – Interface de Programação de Aplicação)
  - 5.6.2 NodeJS
    - 5.6.2.1 Gerenciadores de pacotes do Node
    - 5.6.2.2 TypeScript

5.1 S.O.L.I.D.

S.O.L.I.D. é uma sigla para cinco princípios de programação, que são mais utilizados em programação orientada a objetos (POO). Cada letra é um princípio, que procuramos deixar mais claro nas seguintes subseções.

5.1.1 S - *Single Responsibility Principle* (Princípio da Responsabilidade Única)

Nesse princípio, cada arquivo (seja ele uma classe, um elemento ou uma função) deve ter apenas uma única responsabilidade, por exemplo: uma entidade tem apenas uma responsabilidade, ou seja cada entidade realiza apenas uma única tarefa.

O que implica na facilidade na hora de refatorar o código, traz facilidade no reaproveitamento de código, também traz mais facilidade na implementação dos testes automatizados e também menos *bugs*.

5.1.2 O - *Open-closed principle* (Princípio Aberto Fechado)

Esse princípio diz que classes, entidades ou funções devem estar abertas para extensão mas fechadas para modificações, ou seja, ele deve ser fácil de estender, mas não deve ser modificado, como se fosse o navegador com os seus *plugins*, o código fonte do navegador não é modificado, ou seja, ele é fechado para modificações, mas pode-se adicionar *plugins* que vão dar novas funcionalidades ao navegador. O navegador continua sendo ele mesmo, porém agora ele tem mais funções.

Aberto para extensão significa que, ao receber um novo requerimento, é possível adicionar um novo comportamento. Fechado para modificação significa que, para introduzir um novo comportamento (extensão), não é necessário modificar o código existente.

5.1.3 L - *Liskov Substitution Principle* (Princípio da Substituição de LIskov)

Toda classe filha deve ser capaz de substituir a classe pai sem quebrar o programa. Visto que, se a classe filho não pode exercer as mesmas funções da classe pai, isso pode acarretar em *bugs* em nossos programas.

Por exemplo, se criarmos uma classe “ave” e ela tem um método “voar”, em seguida desenvolvemos uma classe “pinguim” que herda da classe “ave”, ela não vai

poder implementar o método “voar”, pois pinguins não voam, com isso, o certo seria criar essa classe “ave” sem esse método “voar” e em seguida criar uma classe “avesQueVoam” que herda da classe “ave” e nela implementar o método “voar”.

5.1.4 I - *Interface Segregation Principle* (Princípio da Segregação de Interface)

Clientes não devem ser forçados a depender de métodos que eles não usam. Ou seja, uma classe não deve ser forçada a implementar uma interface com métodos que ela não vai precisar. Portanto, é melhor criar interfaces específicas do que criar uma interface genérica. Aqui podemos ver que os princípios se conectam, e assim podemos perceber que o ISP (*Interface Segregation Principle*) é bem parecido com o anterior.

5.1.5 D - *Dependency Inversion Principle* (Princípio da Inversão de Dependências)

Um módulo de alto nível não deve depender de detalhes de implementação de um módulo de baixo nível, deve existir uma abstração entre eles. Neste princípio devemos depender de abstrações e não de implementações.

Uncle Bob (Robert C. Martin)[7] diz que: Módulos de alto nível não devem depender de módulos de baixo nível. Ambos devem depender da abstração, e que abstrações não devem depender de detalhes. Na verdade, são os detalhes que devem depender de abstrações.

Esse princípio visa reduzir a dependência de classes de alto nível de classes de baixo nível através da introdução de uma interface.

5.2 *Singleton*

O *Singleton*[8] define uma operação *implementation* que permite aos clientes acessarem sua única instância, de modo que ela pode ser responsável pela criação de sua própria instância única. Seu objetivo é garantir que uma classe tenha apenas uma instância de modo que ela fornece um ponto global de acesso para a mesma.

Uma melhor solução seria tornar a própria classe responsável por manter o controle da sua instância única. A classe pode garantir que nenhuma outra instância seja criada (pela interceptação das solicitações para criação de novos objetos), bem como pode fornecer um meio para acessar sua única instância. Este é o padrão *Singleton*.

Este padrão é utilizado quando é preciso haver somente uma instância da classe, de modo que ela tenha que fornecer acesso aos clientes a partir de um ponto bem conhecido ou quando essa única instância tiver de ser extensível através de subclasses, abrindo a possibilidade do cliente usar uma instância estendida sem alterar o seu código.

Com isso podemos perceber que ele pode ser utilizado juntamente com o OCP (*Open-closed principle* – Princípio aberto fechado) visto na seção 5.1.2. A seguir podemos ver um exemplo de *Singleton* (Figura 1):

Figura 1: Exemplo de Singleton em TypeScript

```
class MyClass {
  private static INSTANCE: MyClass;

  private constructor(){}

  public static getInstance():MyClass {
    if(!this.INSTANCE){
      this.INSTANCE = new MyClass();
      return this.INSTANCE
    }

    return this.INSTANCE;
  }
}
```

Fonte: Próprio Autor

5.3 Injeção de Dependências

Injeção de Dependências[9][10] (*Dependency Injection*) é um conjunto de padrões de design de software que permite desenvolver código fracamente acoplado. No site *Stack Overflow*, formularam uma resposta para a seguinte pergunta: "*How to explain Dependency Injection to a 5-year old?*"[11] (Como explicar injeção de dependências para uma criança de 5 anos de idade?, tradução livre), a resposta mais votada foi a de John Munsch, que, em tradução livre, afirma:

"Quando você vai e pega coisas da geladeira por você mesmo, você pode causar problemas. Você pode deixar a porta aberta, você pode pegar algo que seu pai ou sua mãe não querem que você pegue. Você pode até mesmo estar procurando por algo que nós nem temos ou talvez tenha expirado.

O que você deve fazer é dizer qual é a sua necessidade: ‘Eu preciso de algo para beber com o almoço’, e então nós iremos garantir que você tenha algo quando você se sentar para comer”.

Com isso podemos ver que a criança depende dos pais para fazer o alimento. Nesse exemplo a criança seria o cliente, enquanto que os pais seriam o serviço. Em código orientado a objetos a classe “criança” depende da classe “pais” para poder usar o método “pegarComidaDaGeladeira”.

O objetivo da injeção de dependência é justamente fazer com que as classes fiquem cada vez mais desacopladas umas das outras, fazendo com que elas sejam cada vez mais independentes.

Quando instanciamos objetos de uma classe em outra classe, criamos uma classe dependente, normalmente isso deve ser evitado, separando o uso de um objeto da sua criação, deixando as classes o mais independente possível. Isso tem ligação direta com três princípios do SOLID, o princípio de responsabilidade única (SRP), o principio da substituição de Liskov (LSP), e o princípio de inversão de dependência (DIP).

Injeção de dependência também é conhecida como inversão de controle, na analogia da criança de cinco anos podemos perceber que antes o controle de pegar o alimento estava nas mãos da criança, mas, depois, foi para a mão dos pais, com isso podemos perceber essa inversão de controles. O mesmo ocorre no código de forma que nós movemos as responsabilidades secundárias de uma classe, ou um objeto, para outras dedicadas à função que se deseja. No entanto devemos passar essa responsabilidade para um mecanismo “dominante”, como um contêiner de tarefas específicas, visto que essa é uma configuração global, com isso nós invertermos o controle. Segue exemplo na Figura 2:

Figura 2: Exemplo de Injeção de dependência (inversão de controle) em JavaScript

```
// sem injeção de dependência
class Dependencia {}

class Dependente {
  dependencia = new Dependencia();
}

var dependente = new Dependente();

// com injeção de dependencia
class Dependente2 {
  constructor(dependencia){
    this.dependencia = dependencia;
  }
}

var dependencia = new Dependencia();
var dependente2 = new Dependente2(dependencia);
```

Fonte: Próprio Autor

Na Figura 2 podemos ver que a classe “Dependente” cria uma instância da classe “Dependencia” dentro dele, isso não ocorre quando utilizamos injeção de dependências. Utilizando a inversão de controle a dependência é passada como parâmetro para o método construtor da classe, assim como ocorrem em “Dependente2”. Assim nós criamos a instância da classe “Dependencia” em qualquer parte do código e passamos essa instância como parâmetro da classe. No nosso sistema fizemos uso da biblioteca *Tsyringe*[12], da qual explicaremos mais a frente na seção 6.3 Desenvolvimento da API.

5.4 Sistemas gerenciadores de banco de dados (SGBD)

Os sistemas gerenciadores de banco de dados[13] são ferramentas utilizadas para armazenar e gerenciar dados inter-relacionados. Normalmente essa coleção de dados é chamada de banco de dados, nele contem as informações mais relevantes para o sistema.

Quando vamos escolher um SGBD para a nossa aplicação temos de escolher entre bancos relacionais e não relacionais. Os bancos relacionais foram desenvolvidos com base em álgebra relacional, de modo que eles apresentam uma melhor eficiência no que diz respeito às consultas realizadas no banco.

Em contrapartida os banco não relacionais apresentam um relaxamento de algumas restrições[14] de consistência de dados tornando-os mais eficientes para sistemas de maior porte.

5.4.1 Modelagem de banco de dados

Os bancos de dados relacionais são desenvolvidos com base em modelos entidade relacionamento. Essas entidades representam algum personagem do nosso mundo, como autores, trabalhos acadêmicos, instituições de ensino, dentre outros exemplos. Tais entidades apresentam um relacionamento entre si, por exemplo um autor cria um ou mais trabalhos acadêmicos. Elas são definidas como tabelas em nosso banco.

Muitas vezes o modelo entidade relacionamento está diretamente ligado com o *script* de execução para criação do banco de dados do sistema, tal *script* também é conhecido como modelo físico do sistema. Para facilitar o desenvolvimento do banco de dados, existe um modelo de mais fácil compreensão, que é o modelo lógico do sistema.

5.4.2 Formas de comunicação com o banco de dados

Existem três formas[15] para a nossa aplicação se comunicar com o banco de dados, a primeira delas é utilizando um *driver* nativo do banco, a segunda é utilizando um *query builder* e por fim a terceira é utilizando um ORM (*Object-Relational Mapping* – Mapeamento Objeto-Relacional).

Ao utilizarmos um *driver* nativo nós fazemos uso direto de *queries* SQL em nosso código, o que dificulta a manutenção do código caso tenhamos que trocar o banco de dados. Em contrapartida, isso pode ser bem útil caso queiramos realizar alguma consulta específica em nosso banco.

O *query builder* apresenta um nível de abstração acima dos *drivers*, visto que ele faz uso de funções que geram a *query* nativa, de modo que escrevemos *queries* programaticamente.

Por fim, os ORMs, como o nome indica, mapeiam objetos de nosso código em tabelas de nosso banco de dados, de modo que isso facilita a legibilidade do código, tornando-o mais coeso, facilitando o desenvolvimento de aplicações. Com isso, nós optamos por utilizar um ORM para o desenvolvimento de nossa aplicação.

5.5 HTTP (*Hypertext Transfer Protocol* - Protocolo de Transferência de Hipertexto)

O HTTP[16] é um protocolo utilizado para acessar dados na *Web*, permitindo, assim, a obtenção de recursos como documentos HTML e JSON. Ele é a base de qualquer troca de informações que ocorre na internet e um protocolo cliente-servidor, de tal forma que a

comunicação é iniciada pelo destinatário, geralmente um *browser*, mas também pode ser um cliente de *frontend* ou *backend*.

Os clientes e os servidores se comunicam trocando mensagens individuais. As mensagens enviadas pelo cliente são conhecidas como *requests* (requisições), e as mensagens enviadas pelo servidor como *responses* (respostas).

As mensagens de requisições são compostas por uma linha de solicitação, um cabeçalho e, algumas vezes, um corpo. As mensagens de resposta são formadas por uma linha de status, um cabeçalho e, algumas vezes, um corpo[17].

A linha de solicitação da mensagem de requisição é utilizada para definir o tipo de solicitação, que são classificados em métodos. Os principais métodos estão listados na tabela a seguir (Tabela 1):

Tabela 1 - Métodos HTTP

Método	Ação
GET	Solicita dados ao servidor
POST	Envia dados do cliente ao servidor
PUT	Cria um novo dado ou substitui uma representação do dado de destino com os novas informações
DELET	O cliente solicita a remoção de algum dado do servidor

Fonte: Próprio Autor

Já a linha de status, da resposta do servidor, serve para indicar se a solicitação foi bem sucedida, ou não, e o motivo. Os códigos são formados por três dígitos. Códigos no intervalo 100 são apenas informativos, no intervalo 200 indicam uma solicitação bem-sucedida, no intervalo 300 são respostas de redirecionamento. Códigos no intervalo 400 indicam erro no lado do cliente, enquanto que erros no intervalo 500 indicam erros no lado do servidor.

Na Tabela 2 listamos os códigos mais utilizados durante o desenvolvimento da nossa aplicação.

Tabela 2 - Códigos de status

Código	Frase	Descrição
200	OK	A solicitação foi bem-sucedida
201	*Created*	Um novo recurso foi criado com sucesso, geralmente usado após uma requisição do tipo POST
400	*Bad Request*	Indica que o servidor não entendeu a solicitação

404	*Not Found*	O recurso solicitado não foi encontrado
500	*Internal Server Error*	Ocorreu um erro no servidor no qual ele não sabe lidar

Fonte: Próprio autor

As rotas de nosso servidor fazem uso do HTTP. Quando uma rota é chamada ela executa uma função que recebe como parâmetro uma requisição HTTP e retorna uma resposta que igualmente respeita o HTTP.

As requisições e respostas HTTP trafegam através de portas disponibilizadas em cada máquina, tais portas recebem uma numeração, sendo que da 0 até 1023 são reservadas par uso específico das máquinas.

5.6 Backend

Como o próprio nome sugere, o backend[18] é a parte que fica por de trás da aplicação, ou seja, os processos internos do nosso programa. Por exemplo, quando acessamos um site que não seja apenas informativo, por trás de toda a parte visual agradável há uma troca de informações realizada entre o banco de dados da aplicação e o cliente, no caso o navegador.

Com isso, o backend é responsável pelas regras de negócios da aplicação, bem como por tudo aquilo que está relacionado com essa troca de dados entre o cliente e o servidor, onde o nosso backend está sendo executado.

5.6.1 API *(Aplication Programming Interface* – Interface de Programação de Aplicação)

A API[19] é definida como uma interface que possibilita a comunicação entre sistemas com tecnologias distintas, de tal forma que ela é um conjunto de rotinas e padrões estabelecidos por um software para a utilização das suas funcionalidades por aplicativos que não pretendem envolver-se em detalhes da implementação do software, mas apenas usar seus serviços.

Consequentemente, as APIs conectam bancos de dados, aplicações, softwares e serviços.

5.6.2 NodeJS

Node.js é um software de código aberto, multiplataforma, baseado no motor V8 do Google e que permite a execução de códigos JavaScript em um ambiente de *desktop*, fora da *web*[20].

O *runtime* de JavaScript é constituído pelos seguintes comandos: node *package manager* (npm), e npx (node *package extractor*); onde o primeiro tem o propósito de executar código armazenado em um pacote de nodejs, instalando o software globalmente ou localmente, já o segundo tem o propósito de instalar a nível local o código instalado globalmente. Um exemplo disso é o “npx create-react-app” que tem como propósito instalar a nível local um template vazio de um site de react, pronto a ser usado, através de uma fórmula instalada com npm.

5.6.2.1 Gerenciadores de pacotes do Node

Por padrão, o Node vem com o npm como gerenciador de pacotes. Entretanto, ele apresenta alguns defeitos como a demora para instalar ou atualizar versões diferentes de dependências nas máquinas, o que pode acarretar em alguns defeitos de segurança.

Com isso, o *yarn* surge como uma solução paleativa ao npm. Ele é uma ferramenta de código aberto desenvolvida pelo Facebook com a colaboração da Google, Exponent e Tilde[21]. Portanto, optamos por utilizar o yarn em nosso projeto.

5.6.2.2 TypeScript

No projeto, utilizamos o typescript[22], um superset do javascript utilizado apenas para desenvolvimento.

Por padrão o node não consegue interpretar código typescript, portanto é necessário fazer uma transpilação do código typescript para o código javascript toda vez que for rodar o sistema. Entretanto, para não tornar tal tarefa algo maçante, nós fizemos uso da ferramenta “ts-node-dev” que faz a transpilação de todo o código typescript automaticamente para javascript de modo que o node vai interpretar esse código javascript e executá-lo, sem termos a necessidade de desligar o servidor e ativa-lo novamente.

Ao utilizarmos o “ts-node-dev” fizemos da seguinte forma: “ts-node-dev src/server.ts”. No caso, ele chama o arquivo *server.ts* localizado no diretório “src”. Para facilitar a utilização do comando, criamos um script no arquivo *package.json* da seguinte

forma: “dev”: “ts-node-dev src/server.ts”. Com isso podemos ativar o servidor utilizando apenas o seguinte comando de linha de comando: yarn dev.

06 Metodologia

Mediante a construção do banco de dados e da API, que vai servir para a comunicação com esse banco de dados, a aplicação, vai funcionar como um repositório de TCC's em que os próprios alunos podem cadastrar os seus trabalhos acadêmicos, que irão ficar disponíveis para o público em geral poder consultar esses trabalhos.

O banco de dados tem a finalidade de armazenar os dados das universidades, dos estudantes, dos professores orientadores, bem como armazenar os trabalhos que serão registrados na plataforma.

A API tem a finalidade de fazer a comunicação com o banco de dados para poder tanto fornecer os dados armazenados como também mantê-lo atualizado registrando, modificando ou deletando dados no mesmo.

O banco de dados criado para a realização trabalho acadêmico foi desenvolvido tendo como base o Modelo Entidade Relacionamento (MER), visto que, como explicamos na seção 5.4, modelos não relacionais apresentam um relaxamento nas restrições, nós não queremos isso em nosso banco. Com isso, ele foi construído de forma simples para facilitar o entendimento e o funcionamento do banco. Ele foi construído através do Oracle SQL Developer Data Modeler[23], que é uma ferramenta da Oracle para modelagem de banco de dados. Como o SGBD (Sistema Gerenciador de Banco de Dados) utilizado no trabalho foi o PostgreSQL 13.3, tivemos que fazer algumas alterações no *script* final.

A presente seção apresenta os seguintes tópicos, que foram os passos utilizados para o desenvolvimento do trabalho:

- 6.1 Levantamento de Requisitos
- 6.2 Desenvolvimento do Banco de Dados
- 6.3 Desenvolvimento da API
    - 6.3.1 Entidades
    - 6.3.2 Repositórios
    - 6.3.3 Camada de “UseCases”
    - 6.3.4 Rotas e Servidor
    - 6.3.5 Documentação
    - 6.3.6 Funcionamento do Sistema em Ambiente de Desenvolvimento
    - 6.3.7 Uso do Sistema em Ambiente de Desenvolvimento

6.1 Levantamento de Requisitos

O primeiro passo para o desenvolvimento da aplicação foi o levantamento de requisitos. Para isso, entramos em contato com a Biblioteca do Campus de Sobral (BCSO) perguntando como funciona a alimentação do sistema do Pergamum. Como resposta, obtivemos a informação que todos os TCC's que chegam na biblioteca através das coordenações são inseridos no Repositório Institucional. Esse trabalho é feito pela equipe técnica da BCSO, sendo que para inserir também no Pergamum, é preciso ter um vínculo com a biblioteca, como ser bolsista e passar por um treinamento, de tal modo que se pode acessar os TCC's a partir de duas plataformas.

Em seguida, pesquisamos quais os dados necessários para o cadastro dos TCC's nos sistemas que encontramos nos manuais de importação de dados do Pergamum e no manual de catalogação do sistema de bibliotecas da UFC. Fizemos um filtro manual dos dados que são obrigatórios e selecionamos os dados que consideramos mais relevantes para o sistema a ser desenvolvido.

Ao fazermos essa busca encontramos que os seguintes dados são exigidos:

- DESCRIÇÃO FÍSICA – INFORMAÇÃO GERAL – para monografias de graduação online, teses e dissertações não é exigido;
- ISBN - *INTERNATIONAL STANDARD BOOK NUMBER* - para monografia de graduação online não é exigido;
- FONTES DA CATALOGAÇÃO;
- CÓDIGOS DE DO IDIOMA;
- NÚMEROS DE CLASSIFICAÇÃO DECIMAL DE DEWEY (CDD);
- NÚMEROS DE CHAMADA LOCAL - para monografia de graduação e pós-graduação online não é exigido;
- ENTRADA PRINCIPAL - NOME PESSOAL;
- TÍTULO PRINCIPAL;
- CARACTERÍSTICAS DO ARQUIVO DE COMPUTADOR - para dissertações e teses não é exigido;
- ÁREA DA PUBLICAÇÃO, DISTRIBUIÇÃO ETC - para teses online não é exigido;
- DESCRIÇÃO FÍSICA - para teses online não é exigido;
- NOTA GERAL - para monografia de graduação e pós-graduação online, teses online não é exigido;
- NOTA DE DISSERTAÇÃO OU TESE;
- NOTA DE BIBLIOGRAFIA ETC;

- NOTA DE DETALHES DO SISTEMA - para dissertações e teses, pode se repetir em monografias não é exigido;
- ASSUNTO TÓPICO - pode se repetir em dissertações, teses, monografias;
- ENTRADA SECUNDÁRIA - NOME PESSOAL – 700;
- ENTRADA SECUNDÁRIA - ENTIDADE – 710;
- LOCALIZAÇÃO E ACESSO ELETRÔNICO - para dissertações e teses não é exigido;

Após esse levantamento, fizemos uma separação dentre quais dados são pedidos para Dissertações, Teses, Monografias de graduação online, monografias de pós-graduação online, teses online, dissertações online e quais dados não são exigidos para todos os documentos.

Com isso conseguimos observar que os seguintes dados são exigidos em todos os trabalhos:

- FONTES DA CATALOGAÇÃO
- CÓDIGOS DE IDIOMA
- NÚMEROS DE CLASSIFICAÇÃO DECIMAL DE DEWEY (CDD)
- ENTRADA PRINCIPAL - NOME PESSOAL
- TÍTULO PRINCIPAL
- NOTA DE DISSERTAÇÃO OU TESE
- NOTA DE BIBLIOGRAFIA ETC.
- ENTRADA SECUNDÁRIA - NOME PESSOAL
- ENTRADA SECUNDÁRIA - ENTIDADE

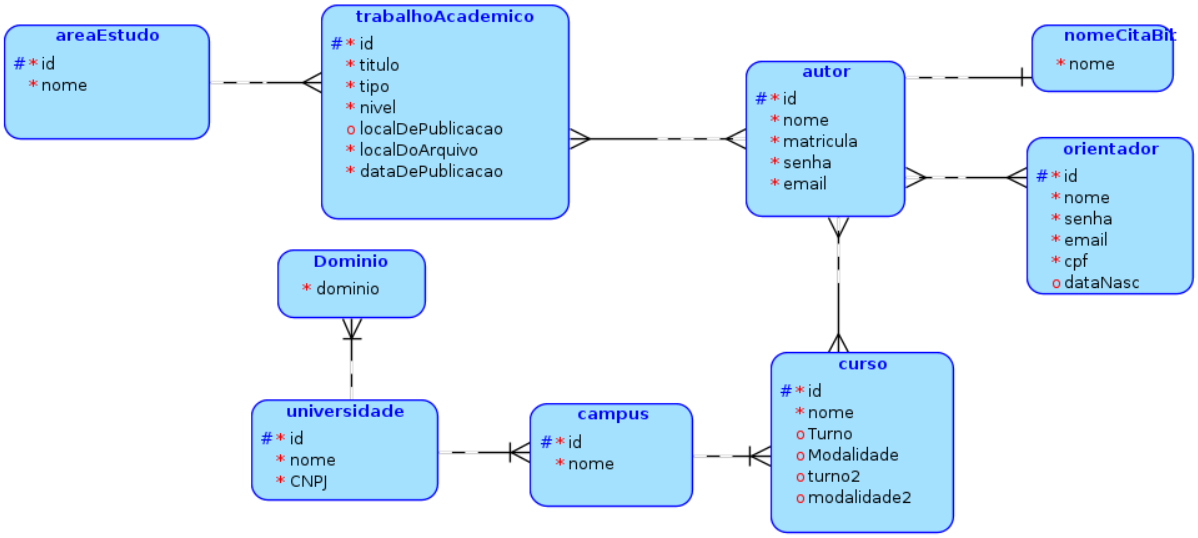
Por fim, separamos os que fariam mais sentido para o nosso sistema, sendo eles: entrada principal - nome pessoal e o título principal. Para complementar os dados requeridos optamos por pedir também o tipo de trabalho acadêmico (se ele é uma tese, dissertação ou monografia de graduação), o nível do trabalho (graduação, mestrado ou doutorado), o local de publicação (se o trabalho tiver sido publicado em alguma revista), e a data do trabalho.

6.2 Desenvolvimento do Banco de Dados

Após o levantamento de requisitos do sistema, utilizamos o SQL Developer Data Modeler[23] como ferramenta para modelagem do modelo lógico do sistema. A modelagem pode ser visualizada na Figura 3, onde elencamos todas as entidades que aparecem no

sistema, sendo a entidade principal, o autor do trabalho acadêmico, visto ser ele quem vai registrar o trabalho.

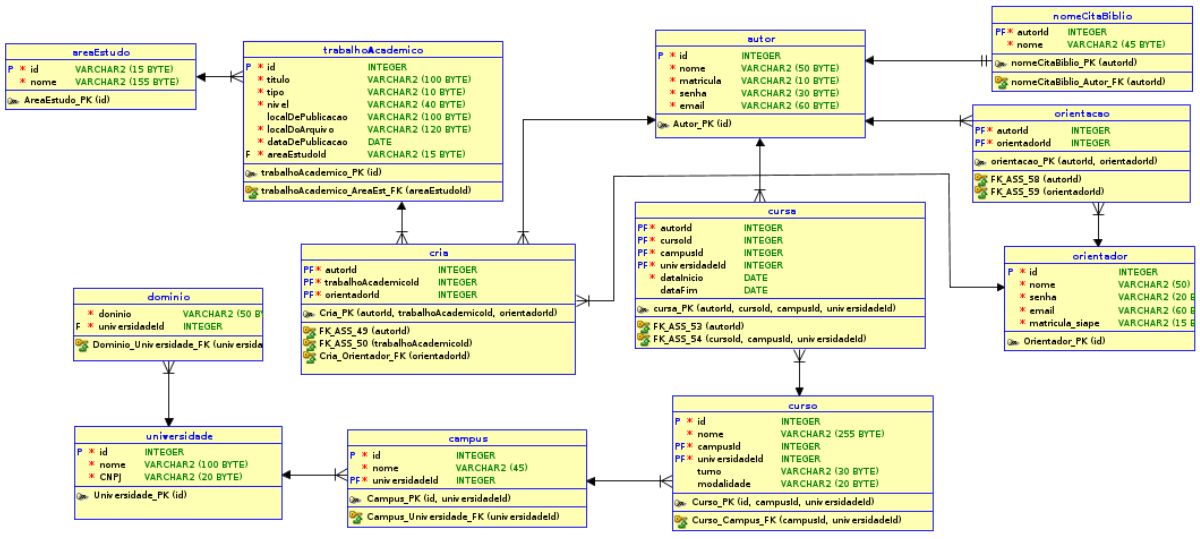
Figura 3: Modelo Lógico



Fonte: Próprio Autor

Em seguida, geramos o modelo relacional (Figura 4), onde os atributos aparecem com mais detalhes e as tabelas são mais detalhadas. Tivemos que realizar alguns ajustes, visto que o programa utilizado, Oracle SQL Developer Data Modeler, gerou algumas tabelas e relacionamentos a mais e criamos uma relação entre a tabela Orientador e a tabela trabalhoAcademico que o Data Modeler não havia desenvolvido previamente.

Figura 4: Modelo Relacional



Fonte: Próprio Autor

Logo após isso, geramos o modelo físico (o *script DDL - Data Descripition Language*), em que também tivemos que realizar algumas alterações no código, visto que a ferramenta gera código para o banco de dados da Oracle, então tivemos que adaptar para um script SQL (*Structured Query Language* - Linguagem de Consulta Estruturada) que o PostgreSQL possa interpretar, bem como também tivemos que adicionar algumas restrições no sistema. Um trecho de tal código pode ser visto a seguir na Figura 5.

Figura 5: Modelo Físico

```
-- Autor: Gabriel Ribeiro Camelo      You, 3 months ago • script postgress

-- CREATE schema tcc_bd;

-- set search_path to tcc_bd;

-- GRANT ALL ON SCHEMA tcc_bd TO postgres;
-- GRANT ALL ON SCHEMA tcc_bd TO tcc_bd;


CREATE TABLE area_estudo
  (
    id   VARCHAR NOT NULL ,
    nome VARCHAR NOT NULL
  ) ;
ALTER TABLE area_estudo ADD CONSTRAINT AreaEstudo_PK PRIMARY KEY ( id ) ;


CREATE TABLE autor
  (
    id        UUID NOT NULL ,
    nome      VARCHAR NOT NULL ,
    matricula VARCHAR NOT NULL ,
    senha     VARCHAR NOT NULL ,
    email     VARCHAR NOT NULL
  ) ;
ALTER TABLE autor ADD CONSTRAINT Autor_PK PRIMARY KEY ( id ) ;
```

Fonte: Prórpio Autor

Para o encerramento da parte do banco de dados, executamos o *script* para gerar as tabelas com os seus atributos e fizemos um povoamento inicial dos dados utilizando dados disponíveis publicamente, como as áreas do conhecimento[24], as universidades, seus campi[25], e os cursos da UFC[26].

6.3 Desenvolvimento da API

Por fim, desenvolvemos a API utilizando o NodeJS v16.13.0[27] com o Yarn 1.22.5[28] sendo o gerenciador de pacotes e dependências do sistema.

Visando uma melhor performance do sistema, buscamos utilizar padrões de projetos de forma a deixá-lo mais manutenível com uma boa estruturação de forma a facilitar a leitura do mesmo. Para isso, utilizamos dois padrões de projetos, o *Singleton* e a inversão de controle, juntamente com os princípios do S.O.L.I.D.

6.3.1 Entidades

Primeiramente desenvolvemos as Entidades do nosso sistema, de modo que elas representam as tabelas de nosso banco de dados. Elas não são nada mais do que classes utilizadas para esse fim. Para isso, nós utilizamos o *typeorm*[29], uma biblioteca de ORM, para *typescript*.

Essas “Entidades” foram separadas em módulos para facilitar o seu desenvolvimento e manutenção. Cada módulo apresenta um conjunto de entidades que apresentam um contexto em comum. Por exemplo, as seguintes entidades: universidade, campus, curso e domínio, pertencem à um mesmo módulo, “universidades”, visto que todos pertencem ao contexto de universidade.

Ao desenvolvermos as entidades usando um ORM fizemos uso do princípio de inversão de dependência, de modo que o código de nossa API não depende de como o nosso banco de dados foi implementado, visto que fizemos uso dessa abstração de entidades.

6.3.2 Repositórios

Os repositórios são classes que utilizamos para implementar métodos de acessos às nossas entidades, de modo que através deles podemos consultar e registrar dados no nosso banco de dados.

Para desenvolvermos essa parte fizemos uso do princípio de segregação de interfaces, de modo que criamos as interfaces granulares de cada repositório antes de construirmos a sua implementação. Escolhemos fazer dessa forma para que pudéssemos fazer diferentes implementações desses repositórios, caso necessário.

Ao construirmos interfaces granulares, optamos também por desenvolver métodos que realizam tarefas específicas. Dessa forma, respeitamos o princípio da responsabilidade única, visto que cada método realiza uma única tarefa, seja buscar por id, criar um registro ou fazer uma listagem de dodos os dados daquela tabela.

Com isso, também respeitamos o princípio “aberto fechado”, visto que, se quisermos adicionar uma nova funcionalidade em nossos repositórios não precisamos modificar nem um outro código que fez uso das funcionalidades existentes.

Como esses repositórios permitem o acesso ao nosso banco de dados, eles devem ter apenas uma instância em nosso sistema, com isso fizemos uso do Singleton e da

Injeção de dependências para prevenirmos quaisquer inconsistências em nosso sistema, para tanto fizemos uso da biblioteca *Tsyringe*.

A biblioteca *Tsyringe* permite que criemos contêineres, onde fizemos todas as instâncias dos nossos repositórios, pois eles são utilizadas como dependências de outras classes, como se segue na Figura 6, onde nós criamos uma instância *Singleton* da classe “AreasEstudoRepository”.

Figura 6: Criação da instância da classe "AreasEstudoRepository" utilizando Tsyringe com a linguagem TypeScritp

```
container.registerSingleton<IAreaEstudoRepository>(
  "AreasEstudoRepository",
  AreasEstudoRepository
);
```

Fonte: Próprio Autor

Na Figura 7, podemos ver que a classe “AreasEstudoRepository” é passada como parâmetro para o método construtor da classe “CreateAreaEstudoUseCase” fazendo uso do *Tsyringe*.

Figura 7: Passagem como parâmetro da classe "AreasEstudoRepository" para a classe "CreateAreaEstudoUseCase"

```
@injectable()
class CreateAreaEstudoUseCase {
  constructor(
    @inject("AreasEstudoRepository")
    private areasEstudodoRepository: IAreaEstudoRepository
  ) {}

  async execute({ id, nome }: IRequest): Promise<void> {
    const areasEstudoAlreadyExists =
      await this.areasEstudodoRepository.findById(id);

    if (areasEstudoAlreadyExists) {
      throw new AppError("Area de Estudo ja está registrada");
    }

    this.areasEstudodoRepository.create({ id, nome });
  }
}
```

Fonte: Próprio Autor

Dessa forma, podemos injetar os repositórios onde forem necessários sem causar redundâncias em nosso código, de modo que podemos inseri-los onde forem necessários.

6.3.3 Camada de “*UseCases*”

Na camada de “*Use Cases*” desenvolvemos dois tipos de classes, as “*UseCases*” e as *Controllers*. As classes “UseCases” fazem uso dos métodos dos repositórios para realizar uma tarefa específica, como no exemplo da Figura 7, onde ela registra uma nova

área de estudo, antes verificando se ela já não está registrada no banco de dados, assim evitando que haja dados duplicados em nosso banco. Com isso, todo "*UseCase*" tem apenas um método, sendo ele o *execute*.

As classes "*Controllers"* também apresentam apenas um único método, sendo ele o método *handle.* Esse método é chamado por uma rota como uma função que irá executar conforme explicado na seção 5.5.

O *controller* faz uso do "*useCase"*, quando criamos uma instância através do método "*container.resolve"* do "*tsyringe*", para evitar que criemos uma segunda instância no nosso código, dessa forma os "*Use Cases*" permanecem sendo um *singleton.*

Por fim ele retorna uma resposta, sendo essa a resposta, no padrão HTTP, de nosso servidor mediante a requisição feita.

Com isso, tanto nos *controllers* como nos "*useCases*" respeitamos o princípio da responsabilidade única.

Essa camada serve como uma abstração entre o HTTP e o restante da nossa aplicação, de modo que, assim, respeitamos o princípio da inversão de dependência.

6.3.4 Rotas e Servidor

Para desenvolvermos as rotas e o servidor, fizemos uso do *express*[30]*,* um *framework* para *node.js* que fornece recursos mínimos para construção de uma aplicação *web.* Da mesma forma que separamos as entidades em módulos, fizemos a construção das rotas separadamente, deixando juntas somente as que fazem parte de um mesmo contexto. Com isso, temos uma maior facilidade na hora de realizar alguma refatoração no código ou encontrar algo que buscamos.

No desenvolvimento das rotas, primeiramente nós criamos uma instância do *controller* a ser utilizado naquela rota, para podermos fazer uso do método *handle.* Em seguida, definimos o método HTTP a ser utilizado seguido do caminho que aquela rota utiliza. Por fim definimos qual *controller* utilizar naquela rota.

Na sequência exportamos as rotas para um arquivo que irá agrupá-las, o qual chamamos de *index.ts,* que serve somente juntar as rotas em um único aquivo que irá ser exportado para o arquivo *server.ts.*

O *server.ts* é o primeiro arquivo a ser chamado no nosso sistema. É ele quem é responsável por escutar a porta da máquina que vai hospedar a nossa aplicação, bem como pelo redirecionamento de erros através de um *middleware* (uma rota intermediária).

Na seção 6.3.6 explicamos como o sistema funciona em ambiente de desenvolvimento.

6.3.5 Documentação

A documentação foi feita utilizando o *swagger*[31], uma ferramenta para documentação de APIs. Com ela, conseguimos desenvolver a documentação utilizando apenas um arquivo .json (*JavaScript Object Notation* - Notação de Objeto JavaScript).

Na documentação do sistema, nós definimos qual é a função de cada rota, de forma que descrevemos quais entradas e possíveis retornos do sistema, dada determinada entrada.

Com esse aquivo .json, o *swagger* consegue fazer a simulação das entradas no sistema, podendo ou não, dependendo da implementação, afetar diretamente o banco de dados.

O acesso à documentação do sistema também é através de uma rota que nós estabelecemos no nosso arquivo de servidor, a "/*api-docs*".

Para mais detalhes, segue o código do arquivo *swagger.json* em anexo (Anexo A).

6.3.6 Funcionamento do Sistema em Ambiente de Desenvolvimento

Para um sistema *web* funcionar em um ambiente de desenvolvimento, primeiramente é necessário instalar as dependências do sistema. No caso do presente projeto, as primeiras dependências são o nodeJS, para interpretar o código javascript na máquina, e o Yarn, que, como falamos anteriormente, é o gerenciador de pacotes utilizado no projeto.

Quando *server.ts* é chamado, ele inicia o *express* através de uma variável chamada *app* que recebe a função do *express.* Tal variável fica responsável por chamar as rotas no diretório *routes,* conforme explicado anteriormente (seção 6.3.4).

Por fim, ele faz uma requisição à máquina para ouvir uma porta do sistema, pela qual vão trafegar as requisições e respostas do sistema. Se ele conseguir acesso a essa porta, ele imprime no console a resposta: *"server is running!"*, e assim a aplicação está pronta para uso em ambiente de desenvolvimento.

Para mais detalhes, segue o código do arquivo *server.ts* em anexo (Anexo B).

6.3.7 Uso do Sistema em Ambiente de Desenvolvimento

Utilizamos a ferramenta *insomnia*[32] para verificarmos o funcionamento das rotas. Através dela, podemos fazer requisições do tipo post, get, updade e delete, de modo que podemos checar a resposta de cada rota de forma simples e fácil.

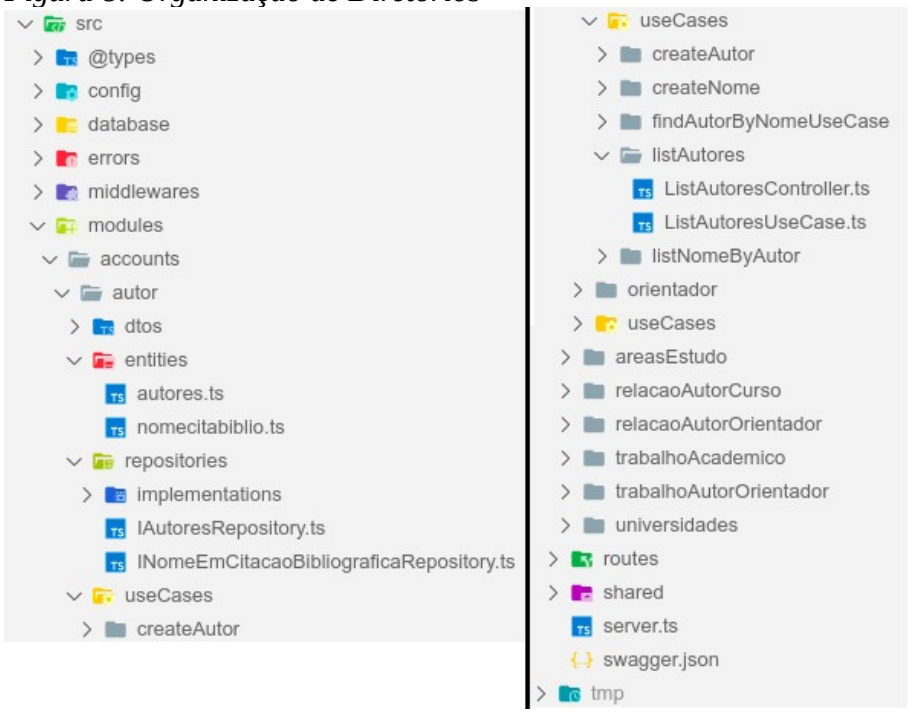
Para verificarmos alterações no banco de dados, fizemos uso do *Beekeeper Studio*[33], visto que ele consegue acessar o banco e mostrar todos os dados lá registrados, bem como conseguimos utilizar algumas *queries* SQL, para fazermos alterações pontuais no banco de dados.

Para verificar os *logs* de erro do sistema ao longo do desenvolvimento, utilizamos o próprio terminal do sistema.

07 Resultados

Com isso, obtivemos um código organizado, fácil de ler e de se aplicar manutenções, quando necessário. Segue a organização dos arquivos (Figura 8):

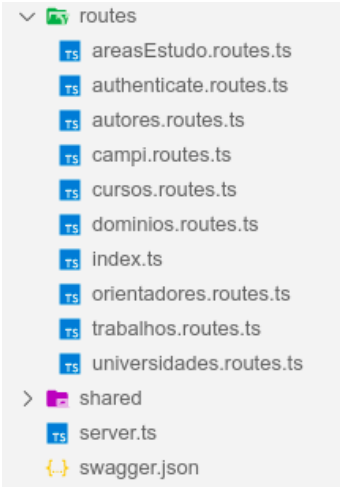
Figura 8: Organização de Diretórios



Fonte: Próprio Autor

As rotas ficaram separadas, conforme se segue na Figura 9, para facilitar a identificação de onde alguma função específica é chamada, de forma que elas são agrupadas no *index.ts* que é chamado pelo *server.ts* quando ele é iniciado no servidor, seja em ambiente de produção, ou em ambiente de desenvolvimento.

Figura 9: Organização das rotas



Fonte: Próprio Autor

No total foram construídas 23 (vinte e três) rotas contabilizando rotas dos tipos *GET* e *POST*, por onde são feitas as requisições para ler e inserir dados no banco, respectivamente. As rotas são as seguintes:

1. “/sessions” do tipo POST, rota de *loggin* no sistema;
2. “/universidades” do tipo POST, rota de cadastro de instituições de ensino superior;
3. “/universidades” do tipo GET, rota de listagem das instituições de ensino superior;
4. “/universidades/import” do tipo POST, rota de cadastro de instituições de ensino superior através de um arquivo .csv. Por essa rota pode-se registrar mais de uma instituição por vez;
5. “/dominios” do tipo POST, rota de cadastro dos domínios da universidade;
6. “/dominios” do tipo GET, rota de listagem dos domínios das universidades;
7. “/campus” do tipo POST, rota de cadastro dos campi das universidades;
8. “/campus:universidade_id” do tipo GET, rota de listagem dos campi de uma universidade específica;
9. “/cursos” do tipo POST, rota de cadastro dos cursos de uma universidade;
10. “/cursos:universidade_id” do tipo GET, rota de listagem dos cursos de uma universidade;
11. “/areasEstudo” do tipo POST, rota de cadastro de áreas de estudo;
12. “/areasEstudo” do tipo GET, rota de listagem das áreas de estudo;
13. “/areasEstudo/import” do tipo POST, rota de cadastro de áreas de estudo através de um arquivo .csv. Por essa rota pode-se registrar mais de uma área por vez;
14. “/autores” do tipo POST, rota de cadastro dos autores no sistema;
15. “/autores” do tipo GET, rota de listagem de autores registrados no sistema;
16. “/autores/createNome” do tipo POST, rota de cadastro de nomes que os autores usam em citações bibliográficas;
17. “/autores/listNome” do tipo GET, rota de listagem dos nomes que os autores usam em citações bibliográficas;
18. “/autores/orientacao” do tipo POST, rota de cadastro da relação entre autor e orientador;

19. “/orientadores” do tipo POST, rota de cadastro de orientadores;
20. “/orientadores” do tipo GET, rota de listagem de orientadores;
21. “/trabalhos” do tipo POST, rota de cadastro dos trabalhos acadêmicos;
22. “/trabalhos” do tipo GET, rota de listagem dos trabalhos acadêmicos;
23. “/trabalhos:titulo” do tipo GET, rota de listagem de trabalhos acadêmicos com nome específico.

Como exemplo de funcionamento, vamos pegar a rota “/orientadores” do tipo *GET*, para utilizarmos ela, primeiramente devemos executar o servidor, conforme explicado na seção 6.3.6. Em seguida, vamos em nosso navegador e digitamos o seguinte: “localhost:3333/orientadores”. Ele deve retornar uma resposta com status 200, e uma lista em formato .json. Caso ocorra algum erro, ele deve retornar um status 500 com uma mensagem descrevendo o erro ocorrido.

08 Conclusão

Os princípios de programação do S.O.L.I.D. se demonstraram bastante úteis, de modo que eles facilitam bastante o desenvolvimento, principalmente quando tinhamos que adicionar alguma funcionalidade no sistema. O *Singleton* se mostrou de grande valia para o sistema visto que ele acabou estando presente em grande parte do projeto, bem como a injeção de dependências.

O banco de dados ficou modelado conforme o planejado, de modo que ele está condizente com o que foi planejado no início. Com isso conseguimos desenvolver grande parte utilizável do sistema, todas as rotas *POST* e *GET*, o que viabiliza a utilização do sistema, conforme o planejado.

09 Trabalhos Futuros

Como o sistema foi planejado para ser utilizado por um usuário comum, somente pra ele registrar o seu trabalho e/ou ver outro trabalho que esteja cadastrado. Desta forma, ainda falta implementar as rotas de *update* e *delete*, bem como o *frontend* do sistema.

A implementação de testes unitários no sistema, para verificar a consistência do mesmo, é de suma importância. Por fim, resta colocar o sistema em produção em um servidor remoto, de modo que ele possa ser acessível a todos pela internet.

08 Referências

[1] Pergamum. O Pergamum. Pergamum, 2019. Disponível em: http://www.pergamum.pucpr.br/pergamum. Acesso em: 19. jan. 2022.

[2] UNIVERSIDADE FEDERAL DO CEARÁ. Importação de acervos da biblioteca nacional para o Pergamum. Ceará, 2014. 10 (dez) páginas. Disponível em: https://biblioteca.ufc.br/wp-content/uploads/2015/09/manual-importacao-dados-bn-pergamum.pdf. Acesso em: 08 ago. 2021.

[3] UNIVERSIDADE FEDERAL DO CEARÁ. Manual de catalogação do sistema de bibliotecas da Universidade Federal do Ceará. Fortaleza, 2017. 173 (cento e setenta e três) páginas. Disponível em: https://biblioteca.ufc.br/wp-content/uploads/2018/01/manual-de-catalogacao-atualizado-word-23-01-2018-1.pdf . Acesso em: 08 ago. 2021.

[4] HERCULANO, Larissa et al. Arquitetura para integração de serviços via api rest para plataforma de acervos digitais. Wayback Machine, 2014. Disponível em: https://web.archive.org/web/20200507005737id_/http://www.iadisportal.org/components/com_booklibrary/ebooks/201914L010.pdf. Acesso em: 17 fev. 2022.

[5] MATOS, Cristina Guerra et al. Biblioteca Digital de Trabalhos Acadêmicos (BDTA) da Universidade Federal Rural da Amazônia (UFRA): relato de experiência. Repositório Institucional da UFRA, 2019. Disponível em: http://repositorio.ufra.edu.br/jspui/bitstream/123456789/1441/1/BDTA-%20RELATO%20DE%20EXPERI%c3%8aNCIA.pdf. Acesso em: 17 fev. 2022.

[6] DAS GRAÇAS COLETTA, Teresinha et al. Biblioteca digital de trabalhos acadêmicos da universidade de são paulo: desenvolvimento e implementação na eesc/usp. Repositório FEBAB, 2020. Disponível em: http://repositorio.febab.libertar.org/files/original/48/5555/SNBU2010_207.pdf. Acesso em: 17 fev. 2022.

[7] MARTIN, R. C. Arquitetura Limpa: O Guia do Artesão para Estrutura e Design de Software. Rio de Janeiro: Alta Books, 2020.

[8] GAMMA, E. et. al. Padrões de Projeto: Soluções reutilizáveis de software orientado a objetos. Porto Alegre : Bookman, 2007.

[9] SEEMANN, M.; DEURSEN, S.V.; *Dependency Injection: Principles, Practices and Patterns*. Estados Unidos da América: Manning, 2019.

[10] MARTIN, R. C. Código Limpo: Habilidades Práticas do Agile Software. Edição de Brochura. Rio de Janeiro: Alta Books, 2009.

[11] MUNSCH John, et al. How to explain Dependency Injection to a 5-year old?. *Stack Overflow,* 2009. Disponível em: https://stackoverflow.com/questions/1638919/how-to-explain-dependency-injection-to-a-5-year-old. Acesso em: 31 jan. 2022.

[12] MICROSOFT. *GitHub: Where the world builds software*, 2020. *TSyringe, A lightweight dependency injection container for TypeScript/JavaScript for constructor injection*. Disponível em: https://github.com/microsoft/tsyringe. Acesso em: 01 fev. 2022.

[13] SILBERSCHATZ, Abraham, et al. Database Sistem Concepts. Seventh Edition, 2020. New York: McGraw-Hill Education, 2020.

[14] AMAZON. AWS Amazon, 2022. O que é NoSQL?. Disponível em: https://aws.amazon.com/pt/nosql/. Acesso: 18 fev. 2022.

[15] MATOSO, Douglas. Node.js e Banco de Dados: ORM, Query Builder, driver nativo. Web Dev Drops, 2020. Disponível em: https://www.webdevdrops.com/nodejs-banco-de-dados-orm-query-builder-driver-nativo/. Acesso em: 18 fev. 2022.

[16] UMA visão geral do HTTP. MDN Web Docs, 2022. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Overview. Acessado em: 21 fev. 2022

[17] FOROUZAN, Behrouz A. Comunicação de Dados e Redes de Computadores. Quarta edição, 2010. São Paulo: AMGH Editora Ltda. 2010.

[18] O QUE é front-end e back-end? Alura, 2021. Disponível em: https://www.alura.com.br/artigos/o-que-e-front-end-e-back-end. Acesso em: 21 fev. 2022.

[19] O QUE é back-end e qual seu papel na programação? TOTVS, 2020. Disponível em: https://www.totvs.com/blog/developers/back-end/. Acesso em: 21 fev. 2022.

[20] NODE.JS. Winkpédia, 2021. Disponível em: https://pt.wikipedia.org/wiki/Node.js. Acesso em: 21 fev. 2022.

[21] NPM vs Yarn. Alura, 2021. Disponível em: https://www.alura.com.br/artigos/npm-vs-yarn. Acesso em: 21 fev. 2022.

[22] MICROSOFT. TypeScript: JavaScript With Syntax For Types, c2012-2022. Página inicial. Disponível em: https://www.typescriptlang.org/. Acesso em: 01 fev. 2022.

[23] MODELAGEM de Dados com o Oracle SQL Developer. Ferramentas de Modelagem de Dados | Oracle SQL Developer Data Modeler | Oracle Brasil, c2021. Disponível em: https://www.oracle.com/br/database/technologies/appdev/datamodeler.html . Acesso em: 08 ago. 2021.

[24] CNPQ. Tabela das Áreas do Conhecimento do CNPq. Brasil, c2021. Disponível em: https://www2.ufmg.br/proex/content/download/2188/14753/version/2/file/Tabela+das+%C3%81reas+do+Conhecimento+do+CNPq.pdf. Acesso em: 08 ago. 2021.

[25] CAMPI da UFC. Universidade Federal do Ceará, c2021. Disponível em: https://www.ufc.br/contatos/677-campi-da-ufc. Acesso em: 08 ago. 2021.

[26] CURSOS de Graduação. Universidade Federal do Ceará PRÓ-REITORIA DE GRADUAÇÃO, c2021. Disponível em: https://prograd.ufc.br/pt/cursos-de-graduacao/?limit=todas. Acesso em: 08 ago. 2021.

[27] Docs | Node.js. *Documentation* | Node.js. Docs | Node.js, 2022. Disponível em: https://nodejs.org/en/docs/. Acesso em: 24 jan. 2022.

[28] *Introduction | Yarn – Package Manager. Introduction. Introduction | Yarn – Package Manager,* 2022. Disponível em: https://yarnpkg.com/getting-started. Acesso em: 24 jan. 2022.

[29] *TypeORM. TypeORM: Amazing ORM for TypeScript and JavaScript. TypeORM, 2022.* Disponível em: https://typeorm.io/#/. Acesso em: 24 jan. 2022.

[30] FUNDAÇÃO Node.js. Express - framework de aplicativo da web Node.js, c2017. Página inicial. Disponível em: https://expressjs.com/. Aecsso em: 01 fev. 2022.

[31] *Swagger*. API *Documentation* & Design *Tools* for *Teams: Swagger. Swagger,* 2021. Disponível em: https://swagger.io/. Acesso em: 24 jan. 2022.

[32] KONG Inc. Insomnia: The API Design Plataform and API Client, c2021. Página inicial. Disponível em: https://insomnia.rest/. Acesso em: 01 fev. 2022.

[33] MATTHEW RATHBONE. Beekeeper Studio: Open Source SQL Editor and Database Manager, c2022. Página inicial. Disponível em: https://www.beekeeperstudio.io/. Acesso em: 01 fev. 2022.

ANEXO A – CÓDIGO DO ARQUIVO *SWAGGER.JSON*

```
{
  "openapi": "3.0.0",
  "info": {
    "title": "API do TCC databank",
    "description": "Essa API tem como objetivo a manipulação de dados que acontece por
trás do projeto TCC DB",
    "termsOfService": "http://localhost:3000/terms",
    "contact": {
      "email": "ganriel201@gmail.com"
    },
    "version": "1.0.0"
  },
  "servers": [
    {
      "url": "http://localhost:3333/",
      "description": "API de testes"
    }
  ],

  "components": {
    "securitySchemes": {
      "bearerAuth": {
        "type": "http",
        "scheme": "bearer",
        "bearerFormat": "JWT"
      }
    }
  },

  "paths": {
    "/sessions": {
      "post": {
        "tags": ["Loggin"],
        "summary": "rota de loggin no sistema",
        "description": "É por essa rota que os autores teram acesso ao sistema e poderem
cadastrar os trabalhos academicos",
        "requestBody": {
          "content": {
            "application/json": {
              "schema": {
                "type": "object",
                "properties": {
                  "matricula": { "type": "string" },
                  "password": { "type": "string" },
                  "isOrientador": { "type": "boolean" }
                },
                "example": {
                  "matricula": "401092",
```

```
              "password": "123456",
              "isOrientador": false
            }
          }
        }
      }
    },
    "responses": {
      "200": { "description": "OK" },
      "content": {
        "application/json": {
          "schema": {
            "type": "object",
            "item": {
              "type": "object",
              "properties": {
                "token": { "type": "string" },
                "user": {
                  "type": "object",
                  "propeties": {
                    "name": { "type": "string" },
                    "email": { "type": "string" },
                    "matricula": { "type": "string" }
                  }
                }
              }
            }
          }
        }
      }
    }
  },
  "/universidades": {
    "post": {
      "tags": ["Universidade"],
      "summary": "Cadastro de Instituições de Ensino Superior",
      "description": "Essa rota é responsável por cadastrar as Instituições de Ensino
Superior",
      "requestBody": {
        "content": {
          "application/json": {
            "schema": {
              "type": "object",
              "properties": {
                "nome": { "type": "string" },
                "cnpj": { "type": "string" }
              },
              "example": {
                "nome": "Universidade Teste",
                "cnpj": "Teste CNPJ"
              }
```

```
            }
          }
        }
      },
      "responses": {
        "201": { "description": "Created" },
        "400": { "description": "Universidade ja está registrada" }
      }
    },
    "get": {
      "tags": ["Universidade"],
      "summary": "Listagem de Instituições de Ensino",
      "description": "Essa rota lista todos as Instituições de Ensino cadastradas no banco
de dados",
      "responses": {
        "200": {
          "description": "success",
          "content": {
            "application/json": {
              "schema": {
                "type": "array",
                "items": {
                  "type": "object",
                  "properties": {
                    "id": { "type": "string" },
                    "nome": { "type": "string" },
                    "cnpj": { "type": "string" }
                  }
                }
              }
            }
          }
        }
      }
    }
  },
  "/universidades/import": {
    "post": {
      "tags": ["Universidade"],
      "summary": "Cadastro de Instituições de Ensino Superior atravez de Upload de
arquivo",
      "description": "Através dessa rota você pode cadastrar diversas universidades de
uma vez com um arquivo CSV contendo o nome o cnpj da universidade
RESPECTIVAMENTE",
      "requestBody": {
        "content": {
          "multipart/form-data": {
            "schema": { "type": "object" },
            "properties": {
              "file": { "type": "string", "format": "binary" }
            }
          }
```

```
        }
      },
      "responses": {
        "201": { "description": "Created" }
      }
    }
  },
  "/dominios": {
    "post": {
      "tags": ["Universidade"],
      "summary": "Cadastro de dominios",
      "description": "Essa rota é responsável por cadastrar os dominios das
universidades",
      "requestBody": {
        "content": {
          "application/json": {
            "schema": {
              "type": "object",
              "properties": {
                "dominio": { "type": "string" },
                "universidade_id": { "type": "string" }
              },
              "example": {
                "dominio": "ufc.br",
                "universidade_id": "4ea50475-2cb6-4576-ba55-2a9bffacc8fc"
              }
            }
          }
        }
      },
      "responses": {
        "201": { "description": "Created" },
        "400": { "description": "domino ja está registrado!!!" }
      }
    },
    "get": {
      "tags": ["Universidade"],
      "summary": "Listagem de dominios",
      "description": "Essa rota lista todos os dominios cadastrados no banco de dados",
      "responses": {
        "200": {
          "description": "success",
          "content": {
            "application/json": {
              "schema": {
                "type": "array",
                "items": {
                  "type": "object",
                  "properties": {
                    "dominio": { "type": "string" },
                    "universidade_id": { "type": "string" }
                  }
```

```
            }
          }
        }
      }
    }
  }
},
"/campus": {
  "post": {
    "tags": ["Universidade"],
    "summary": "Cadastro de campi",
    "description": "Essa rota é responsável por cadastrar os campi das universidades, insira o nome do campus na propriedade campus, e o id da universidade a qual o campus pertence (consultar na rota de get /universidades)",
    "requestBody": {
      "content": {
        "application/json": {
          "schema": {
            "type": "object",
            "properties": {
              "nome": { "type": "string" },
              "universidade_id": { "type": "string" }
            },
            "example": {
              "nome": "UFC Campus de Sobral",
              "universidade_id": "8d3888a7-cc28-468e-bcea-23397bdd5f51"
            }
          }
        }
      }
    },
    "responses": {
      "201": { "description": "Created" },
      "404": {
        "description": "Universidade não existe!!!\nVocê não pode registrar um campus sem uma universidasde!!!"
      },
      "400": { "description": "esse campus ja foi registrado!!!" }
    }
  }
},
"/campus/{universidade_id}": {
  "get": {
    "tags": ["Universidade"],
    "summary": "Listagem de campi",
    "description": "Essa rota lista todos os campi de uma universiade cadastrados no banco de dados",
    "parameters": [
      {
        "name": "universidade_id",
        "in": "path",
```

```
          "description": "a rota utiliza o id da universidade pra fazer a busca dos campus pelo banco de dados",
          "required": true
        }
      ],
      "responses": {
        "200": {
          "description": "success",
          "content": {
            "application/json": {
              "schema": {
                "type": "array",
                "items": {
                  "type": "object",
                  "properties": {
                    "id": { "type": "string" },
                    "nome": { "type": "string" },
                    "universidade_id": { "type": "string" }
                  }
                }
              }
            }
          }
        },
        "404": { "description": "Universidade não registrada" }
      }
    }
  },
  "/cursos": {
    "post": {
      "tags": ["Universidade"],
      "summary": "Cadastro de cursos de uma universidade",
      "description": "Essa rota é responsável por cadastrar os cursos das universidades, as propriedades turno e modalidade só aceitam tipos específicos. \n Para o turno: \"diurno\", \"noturno\" e \"integral\". \n Para modalidade: \"licenciatura\", \"bacharelado\" e  \"bacharelado e licenciatura\" ",
      "requestBody": {
        "content": {
          "application/json": {
            "schema": {
              "type": "object",
              "properties": {
                "nome": { "type": "string" },
                "campus_id": { "type": "string" },
                "universidade_id": { "type": "string" },
                "turno": {
                  "type": "\"diurno\" | \"noturno\" | \"integral\""
                },
                "modalidade": {
                  "type": "\"licenciatura\" | \"bacharelado\" | \"bacharelado e licenciatura\""
                }
              },
```

```
            "example": {
              "nome": "Engenharia de Computação",
              "campus_id": "5002ee8f-d56e-4707-bd7f-1186ba490937",
              "universidade_id": "8d3888a7-cc28-468e-bcea-23397bdd5f51",
              "turno": "integral",
              "modalidade": "bacharelado"
            }
          }
        }
      }
    },
    "responses": {
      "201": { "description": "Created" },
      "400": { "description": "curso ja fou registrado!!!" },
      "404": {
        "description": "você pode se deparar com duas mensagens para esses erro, uma informando q a universidade não existe e outra se referindo ao campus"
      }
    }
  }
 },
 "/cursos/{campus_id}": {
  "get": {
    "tags": ["Universidade"],
    "summary": "Listagem de cursos de um campus de uma universidade",
    "description": "Essa rota é responsável por listar todos os cursos de um capmus de uma universidade.",
    "parameters": [
      {
        "name": "campus_id",
        "in": "path",
        "description": "a rota utiliza o id do campus pra fazer a busca dos cursos pelo banco de dados",
        "required": true
      }
    ],
    "responses": {
      "200": {
        "description": "success",
        "content": {
          "application/json": {
            "schema": {
              "type": "array",
              "items": {
                "type": "object",
                "properties": {
                  "id": { "type": "string" },
                  "nome": { "type": "string" },
                  "turno": { "type": "string" },
                  "modalidade": { "type": "string" },
                  "campus_id": { "type": "string" },
                  "universidade_id": { "type": "string" }
```

```
                }
              }
            }
          }
        }
      },
      "404": { "description": "campus não encontrado" },
      "400": { "description": "entrada invalida" }
    }
  }
 },
 "/areasEstudo": {
  "post": {
   "tags": ["Áreas de Estudo"],
   "summary": "Cadastro de Áreas de Estudo",
   "description": "Essa rota é responsável por cadastrar as Áreas de Estudo, o id é o
número identificador listado no CNPQ",
   "requestBody": {
    "content": {
     "application/json": {
      "schema": {
       "type": "object",
       "properties": {
        "id": { "type": "string" },
        "nome": { "type": "string" }
       },
       "example": {
        "id": "1.00.00.00-3",
        "nome": "Ciências Exatas e da Terra"
       }
      }
     }
    }
   },
   "responses": {
    "201": { "description": "Created" },
    "400": { "description": "Category already exists" }
   }
  },
  "get": {
   "tags": ["Áreas de Estudo"],
   "summary": "Listagem de Áreas de Estudo",
   "description": "Essa rota lista todos as Áreas de Estudo cadastradas no banco de
dados",
   "responses": {
    "200": {
     "description": "success",
     "content": {
      "application/json": {
       "schema": {
        "type": "array",
        "items": {
```

```
                "type": "object",
                "properties": {
                  "id": { "type": "string" },
                  "nome": { "type": "string" }
                }
              }
            }
          }
        }
      }
    }
  },
  "/areasEstudo/import": {
    "post": {
      "tags": ["Áreas de Estudo"],
      "summary": "Cadastro de Áreas de Estudo atravez de Upload de arquivo",
      "description": "Através dessa rota você pode cadastrar diversas Áreas de Estudo de uma vez com um arquivo CSV contendo o id e nome da Área de Estudo RESPECTIVAMENTE",
      "requestBody": {
        "content": {
          "multipart/form-data": {
            "schema": { "type": "object" },
            "properties": {
              "file": { "type": "string", "format": "binary" }
            }
          }
        }
      },
      "responses": {
        "201": { "description": "Created" }
      }
    }
  },
  "/autores": {
    "post": {
      "tags": ["Autores"],
      "summary": "Cadastro de Autores",
      "description": "Essa rota é responsável por cadastrar os Autores,\nATENÇÃO: o email tem que ser um email instucional registrado no banco de dados;\nATENÇÃO: o campo data_fim é opcional",
      "requestBody": {
        "content": {
          "application/json": {
            "schema": {
              "type": "object",
              "properties": {
                "email": { "type": "string" },
                "nome": { "type": "string" },
                "senha": { "type": "string" },
                "matricula": { "type": "string" },
```

```
            "curso_id": { "type": "string" },
            "campus_id": { "type": "string" },
            "universidade_id": { "type": "string" },
            "data_inicio": { "type": "string" },
            "data_fim": { "type": "string" }
          },
          "example": {
            "email": "exemple@alu.ufc.br",
            "nome": "Alguem",
            "senha": "senha de exemplo",
            "matricula": "123456",
            "curso_id": "b2617a42-31c0-4b64-8fe5-49d2e1030e1a",
            "campus_id": "92a8cbdd-b228-49a9-8e7c-1eb7fbc59694",
            "universidade_id": "58ec464f-2833-480f-b5cf-6068ef21fe86",
            "data_inicio": "2017-03-23"
          }
        }
      }
    }
  },
  "responses": {
    "201": { "description": "Created" },
    "400": { "description": "erro ao cadastrar autor" }
  }
},
"get": {
  "tags": ["Autores"],
  "summary": "Listagem de Autores",
  "description": "Essa rota lista todos os Autores cadastradas no banco de dados",
  "responses": {
    "200": {
      "description": "success",
      "content": {
        "application/json": {
          "schema": {
            "type": "array",
            "items": {
              "type": "object",
              "properties": {
                "id": { "type": "string" },
                "nome": { "type": "string" },
                "matricula": { "type": "string" },
                "email": { "type": "string" }
              }
            }
          }
        }
      }
    }
  }
},
```

```
    "/autores/createNome": {
      "post": {
        "tags": ["Autores"],
        "summary": "Cadastro de nome em citações bibliográfias",
        "description": "Essa rota é responsável por cadastrar os nomes em citações bibliográficas dos autores, visto que um autor pode ter vários nomes diferentes de citações bibliográficas",
        "security": [{ "bearerAuth": [] }],
        "requestBody": {
          "content": {
            "application/json": {
              "schema": {
                "type": "object",
                "properties": {
                  "nome": { "type": "string" }
                },
                "example": {
                  "nome": "Alguem A."
                }
              }
            }
          }
        },
        "responses": {
          "201": { "description": "Created" },
          "401": { "description": "User does not exists!" }
        }
      }
    },
    "/autores/listNomes": {
      "get": {
        "tags": ["Autores"],
        "summary": "Listagem de nomes em citações bibliográfias",
        "description": "Essa rota é responsável por listar os nomes em citações bibliográficas dos autores, visto que um autor pode ter vários nomes diferentes de citações bibliográficas",
        "security": [{ "bearerAuth": [] }],
        "responses": {
          "200": {
            "description": "success",
            "content": {
              "application/json": {
                "schema": {
                  "type": "array",
                  "items": {
                    "type": "object",
                    "properties": {
                      "nome": { "type": "string" },
                      "autor_id": { "type": "string" }
                    }
                  }
                }
```

```
            }
          }
        }
      }
    }
  },
  "/autores/orientacao": {
    "post": {
      "tags": ["Autores"],
      "summary": "Cadastrar relação entre autor e orientador",
      "description": "Essa rota é responsável por cadastrar as relações entre autores e orientadores",
      "security": [{ "bearerAuth": [] }],
      "requestBody": {
        "content": {
          "application/json": {
            "schema": {
              "type": "object",
              "properties": {
                "orientador_id": { "type": "string" }
              },
              "example": {
                "orientador_id": "9cc109c4-a109-42bc-83fe-a8037c02fd52"
              }
            }
          }
        }
      },
      "responses": {
        "201": { "description": "Created" },
        "404": { "description": "Autor ou orientador não foi encontrado" },
        "400": { "description": "erro do usuário" }
      }
    }
  },
  "/orientadores": {
    "post": {
      "tags": ["Orientadores"],
      "summary": "Cadastro de Orientadores",
      "description": "Essa rota é responsável por cadastrar os Orientadores, ATENÇÃO: o email tem que ser um email instucional registrado no banco de dados",
      "requestBody": {
        "content": {
          "application/json": {
            "schema": {
              "type": "object",
              "properties": {
                "email": { "type": "string" },
                "nome": { "type": "string" },
                "senha": { "type": "string" },
                "matricula_siape": { "type": "string" }
              },
```

```
            "example": {
              "email": "exemple@alu.ufc.br",
              "nome": "Alguem",
              "senha": "senha de exemplo",
              "matricula_siape": "123456"
            }
          }
        }
      }
    },
    "responses": {
      "201": { "description": "Created" },
      "400": { "description": "Orientador Already Exists!!!!" }
    }
  },
  "get": {
    "tags": ["Orientadores"],
    "summary": "Listagem de Orientadores",
    "description": "Essa rota lista todos os Orientadores cadastradas no banco de
dados",
    "responses": {
      "200": {
        "description": "success",
        "content": {
          "application/json": {
            "schema": {
              "type": "array",
              "items": {
                "type": "object",
                "properties": {
                  "id": { "type": "string" },
                  "nome": { "type": "string" },
                  "email": { "type": "string" }
                }
              }
            }
          }
        }
      },
      "500": { "description": "aconteceu alguma coisa muito errada aqui" }
    }
  }
},
"/trabalhos": {
  "post": {
    "tags": ["Trabalhos Acadêmicos"],
    "summary": "Cadastro de Trabalhos Acadêmicos",
    "description": "Essa rota é responsável por cadastrar os Trabalhos Acadêmicos",
    "security": [{ "bearerAuth": [] }],
    "requestBody": {
      "content": {
        "multipart/form-data": {
```

```
            "schema": { "type": "object" },
            "properties": {
              "titulo": { "type": "string" },
              "tipo": { "type": "\"TCC\" | \"artigo\" | \"resenha\"" },
              "nivel": {
                "type": "\"graduação\" | \"mestradro\" | \"doutorado\""
              },
              "arquivo": { "type": "string", "format": "binary" },
              "local_publicacao": { "type": "string" },
              "area_estudo_id": { "type": "string" },
              "orientador_id": { "type": "string" }
            }
          }
        }
      },
      "responses": {
        "201": { "description": "Created" },
        "404": {
          "description": "erro em algum dos campos, autor, orientador ou trabalho não
encontrado"
        }
      }
    },
    "get": {
      "tags": ["Trabalhos Acadêmicos"],
      "summary": "Listagem de Trabalhos Acadêmicos",
      "description": "Essa rota lista todos os Trabalhos Acadêmicos cadastradas no banco
de dados",
      "responses": {
        "200": {
          "description": "success",
          "content": {
            "application/json": {
              "schema": {
                "type": "array",
                "items": {
                  "type": "object",
                  "properties": {
                    "titulo": { "type": "string" },
                    "tipo": { "type": "string" },
                    "nivel": { "type": "string" },
                    "local_publicacao": { "type": "string" },
                    "data_publicacao": { "type": "string" },
                    "area_estudo_id": { "type": "string" }
                  }
                }
              }
            }
          }
        },
        "500": { "description": "um erro muito doido aqui mano" }
      }
```

```
      }
    },
    "/trabalhos/{titulo}": {
      "get": {
        "tags": ["Trabalhos Acadêmicos"],
        "summary": "Listagem de Trabalhos Acadêmicos com esse título",
        "description": "Essa rota lista todos os Trabalhos Acadêmicos que tenham esse
título",
        "parameters": [
          {
            "name": "titulo",
            "in": "path",
            "description": "a rota utiliza o titulo do trabalho academico pra fazer a busca dos
trabalhos pelo banco de dados",
            "required": true
          }
        ],
        "responses": {
          "200": {
            "description": "success",
            "content": {
              "application/json": {
                "schema": {
                  "items": {
                    "type": "object",
                    "properties": {
                      "arquivo": { "type": "string", "format": "binary" }
                    }
                  }
                }
              }
            }
          },
          "404": { "description": "trabalho não encontrado" }
        }
      }
    }
  }
}
```

ANEXO B – CÓDIGO DO ARQUIVO *SERVER.TS*

```
import "reflect-metadata";
import express, { NextFunction, Request, Response } from "express";
import swaggerUi from "swagger-ui-express";

import "./database";
import "./shared/container";
import { router } from "./routes";
import { AppError } from "./errors/AppErros";
import swaggerDocs from "./swagger.json";

const app = express();

app.use(express.json());

app.use("/api-docs", swaggerUi.serve, swaggerUi.setup(swaggerDocs));

app.use(router);

app.use((err: Error, req: Request, res: Response, next: NextFunction) => {
  if (err instanceof AppError)
    return res.status(err.statusCode).json({
      message: err.message,
    });

  return res.status(500).json({
    status: "error",
    message: `Internal Server Error - ${err.message}`,
  });
});

app.listen(3333, () => console.log("server is running!"));
```