REPÚBLICA FEDERATIVA DO BRASIL

MINISTÉRIO DA ECONOMIA

INSTITUTO NACIONAL DA PROPRIEDADE INDUSTRIAL

DIRETORIA DE PATENTES, PROGRAMAS DE COMPUTADOR E TOPOGRAFIAS DE CIRCUITOS INTEGRADOS

Certificado de Registro de Programa de Computador

Processo Nº: **BR512022001664-8**

O Instituto Nacional da Propriedade Industrial expede o presente certificado de registro de programa de computador, válido por 50 anos a partir de 1° de janeiro subsequente à data de 31/03/2022, em conformidade com o §2°, art. 2° da Lei 9.609, de 19 de Fevereiro de 1998.

Título: Pra Nadar Web

Data de publicação: 31/03/2022

Data de criação: 30/03/2022

Titular(es): FUNDAÇÃO UNIVERSIDADE DO ESTADO DO RIO GRANDE DO NORTE - FUERN

Autor(es): ADALBERTO VERONESE DA COSTA; SEBASTIÃO EMIDIO ALVES FILHO; EXLLEY CLEMENTE DOS SANTOS; TIAGO DA SILVA MORAIS; JEFTER ROBERTO MOTA TARGINO; FRANCISCO CLEMENTINO MAIA JÚNIOR; MATHEUS DIOGENES DA SILVA; HÉLIO VICTOR APOLINÁRIO SOARES

Linguagem: JAVA SCRIPT; OUTROS

Campo de aplicação: AD-01; IF-01; SD-01

Tipo de programa: AP-01; GI-01; SO-05

Algoritmo hash: SHA-512

Resumo digital hash:
5acd83dcc7e7f499d85a34aecede682f6ba90dc3abf02fcb74b658ad37ad1797a34c15d9a2045824ffa3ae57b511b63b284c15bb11c673ff1fb64427f5f0683c

Expedido em: 12/07/2022

Aprovado por:

Joelson Gomes Pequeno

Chefe Substituto da DIPTO - PORTARIA/INPI/DIRPA Nº 02, DE 10 DE FEVEREIRO DE 2021

UNIVERSIDADE DO ESTADO DO RIO GRANDE DO NORTE – UERN
FACULDADE DE CIÊNCIAS EXATAS E NATURAIS – FANAT
DEPARTAMENTO DE INFORMÁTICA – DI

TIAGO DA SILVA MORAIS

API PRA NADAR: UMA API REST PARA GERENCIAMENTO DO PROGRAMA PRA NADAR

MOSSORÓ - RN
2022

TIAGO DA SILVA MORAIS

API PRA NADAR: UMA API REST PARA GERENCIAMENTO DO PROGRAMA PRA NADAR

Relatório apresentado ao curso de Ciência da Computação da Universidade do Estado do Rio Grande no Norte como requisito da disciplina de Trabalho de Diplomação, sob a orientação do(a) Prof. Dr. Sebastião Emidio Alves Filho e coorientação do Me. Exlley Clemente dos Santos.

MOSSORÓ - RN

2022

TIAGO DA SILVA MORAIS

API PRA NADAR: UMA API REST PARA GERENCIAMENTO DO PROGRAMA PRA NADAR

Registro de software apresentado como pré-requisito para a obtenção do título de Bacharel em Ciência da Computação da Universidade do Estado do Rio Grande do Norte – UERN, submetida à aprovação da banca examinadora composta pelos seguintes membros:

Aprovada em: 26/04/2022

Banca Examinadora

Prof. Dr. Sebastião Emídio Alves Filho
Universidade do Estado do Rio Grande do Norte - UERN

Me. Exlley Clemente dos Santos
Universidade do Estado do Rio Grande do Norte - UERN

Prof. Dr. Francisco Chagas de Lima Júnior
Universidade do Estado do Rio Grande do Norte - UERN

Prof. Dr. Isaac de Lima Oliveira Filho
Universidade do Estado do Rio Grande do Norte - UERN

SUMÁRIO

1 INTRODUÇÃO

O Programa Pra Nadar é um projeto de extensão fundado e dirigido pela Faculdade de Educação Física (FAEF) da Universidade do Estado do Rio Grande do Norte (UERN), oferecendo práticas de atividades aquáticas, como natação e hidroginástica. As práticas são oferecidas para a comunidade em geral, o que inclui estudantes, professores e servidores da UERN, além da comunidade externa. O processo de matrícula dos praticantes passa pela necessidade da entrega de atestado médico e um questionário pessoal, dando a devida condição para o exercício das atividades. O Pra Nadar funciona através das contribuições financeiras que podem ser mensais, semestrais ou anuais, provenientes da participação dos estudantes matriculados no programa.

O programa inicialmente era organizado pelos secretários e professores da faculdade e não existia algo para contribuir na gestão dos dados que estavam sendo inseridos e tratados. O processo de gerenciamento de turmas, alunos, instrutores e secretários no programa eram feitos de forma manual, ocasionando dificuldade no gerenciamento.

1.1 JUSTIFICATIVA

Diante da dificuldade no gerenciamento, contando com problemas de controle do número de estudantes em cada turma e vagas disponíveis para matrículas, registro das contribuições feitas durante o decorrer do programa, controle das frequências dos estudantes e instrutores, além de outros. Com estas adversidades, a FAEF necessitava de um sistema de gerenciamento para ter melhor controle do programa. Desse modo, foi desenvolvido um sistema que visa administrar os dados do programa de forma rápida, intuitiva e dinâmica.

Dentro desse sistema foi desenvolvido uma API responsável por realizar e tratar todos os dados referentes ao programa, prestando suporte a duas plataformas diferentes (web e mobile), unificando todos dados independente de qual forma de acesso foi utilizada, facilitando o trabalho de toda a equipe do programa.

2 OBJETIVOS

2.1 OBJETIVO GERAL

O presente trabalho tem a finalidade de fornecer um sistema de gerenciamento de informações para auxiliar a equipe de alunos e professores da FAEF envolvida no programa Pra Nadar.

2.2 OBJETIVO ESPECÍFICO

Este trabalho tem como intuito a criação de uma API REST (do inglês, *Application Programming Interface*), utilizando dos princípios do REST (do inglês, *Representational State Transfer*), como o uso de requisições HTTP para facilitar a relação entre cliente e servidor (LIMA, 2020). A API tem como objetivo principal o gerenciamento de todas as informações do programa Pra Nadar, desde usuários críticos como administradores e secretários até dados mais simples como planos de aulas e frequências.

3 METODOLOGIA

Durante este trabalho a metodologia adotada foi a *Feature Driven Development (*FDD, em português, Desenvolvimento Dirigido por Funcionalidades) uma metodologia ágil focada no desenvolvimento baseado em funcionalidades, ou seja, uma metodologia que o seu desenvolvimento é dividido em ciclos onde nesses ciclos são decididos o que deve ser entregue (ROCHA, 2013).

3.1 TECNOLOGIAS UTILIZADAS

3.1.1 AdonisJS

A criação da API foi dada através do framework AdonisJS, um framework conhecido por sua facilidade por possuir diversas funcionalidades para otimizar o desenvolvimento (HOMBOLT TECHNOLOGY, 2021). Permitindo ao desenvolvedor ir modificando e adaptando o projeto de acordo com suas necessidades. Também contém uma estrutura de organização de pastas padrão que facilita durante o desenvolvimento.

3.1.2 Visual Studio Code

O Visual Studio Code é um editor de código disponível para todas as plataformas, com suporte para a maioria das linguagens utilizadas atualmente, como: C++, JavaScript e Python (MICROSOFT, [202?]). Para complementar, vem integrado com um conjunto de atalhos para agilizar e auxiliar no desenvolvimento, além de possuir diversas extensões que podem integrá-lo a outros softwares.

3.1.3 Insomnia

O Insomnia é um software desenvolvido pela empresa Kong, disponibilizado para sistemas Linux, Windows e MacOS (KONG, [2021?]). É utilizado para realizar as consultas como cliente de uma API REST, utilizando os protocolos HTTP. Nele é

possível fazer uma boa organização do ambiente, permitindo o uso de métodos HTTP diferentes em conjunto para uma só rota.

3.1.4 PostgreSQL

O banco de dados utilizado para o sistema foi o PostgreSQL. Um banco de dados relacional *open source* com mais de 30 anos de desenvolvimento, atualmente disponibilizado para todos os sistemas operacionais (POSTGRESQL, [202?]). O postgreSQL é bem otimizado e intuitivo devido sua fácil integração com o editor de código e execução de comandos SQL.

3.2 IMPLEMENTAÇÃO

3.2.1 Usuários

Durante as primeiras reuniões, o foco era organizar a hierarquia dos usuários. Então foi decidido que o sistema seria composto por 4 tipos de usuários: Administrador, secretário, instrutor e estudante.

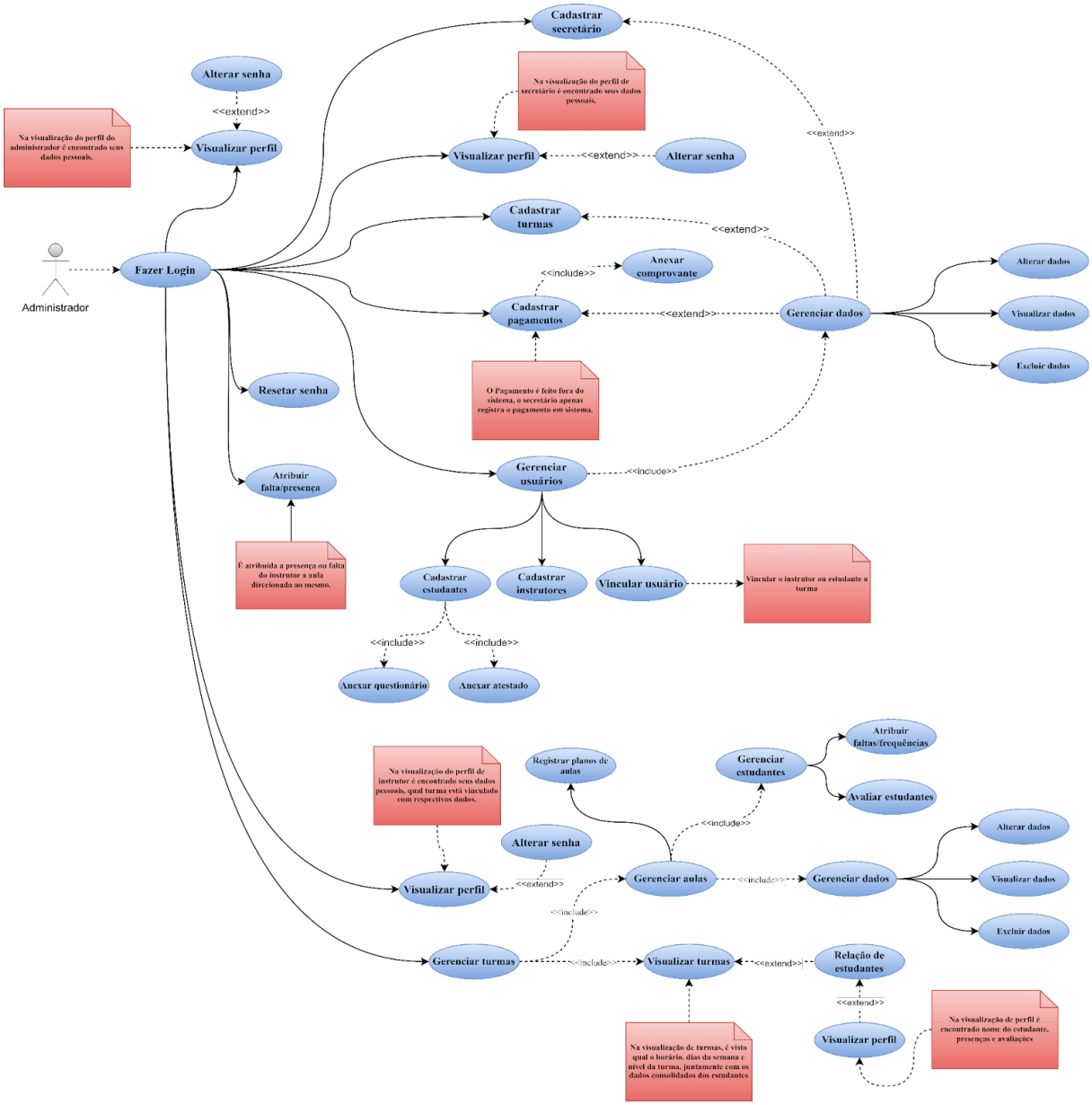
O administrador (Figura 1) é o usuário com o maior privilégio dentro do sistema, sendo capaz de realizar qualquer tarefa e podendo ter acesso a todas as rotas (permissões). A ideia principal do administrador é ser um usuário que deve ser usado para tarefas específicas, como a criação de novos secretários, já que estes não poderiam interferir em outros do mesmo cargo, como também é capaz de redefinir a senha de algum usuário, retornando a senha padrão criada no ato do cadastro.

O secretário (Figura 2) é o usuário que realmente vai realizar todo o gerenciamento do sistema. Desde criação de turmas, cadastrar pagamentos, estudantes e instrutores, também conseguindo realizar todos os vínculos dessas turmas com seus instrutores e estudantes, responsável por anexar os atestados e questionários que pertencem a cada estudante durante sua matrícula.

O instrutor (Figura 3) é o responsável por ministrar a aula, criando planos de aula e realizando a presença dos alunos. Assim como também é responsável por suas avaliações, possuindo funcionalidades básicas como visualizar seu próprio perfil e alterar sua senha, o instrutor é capaz de visualizar todas as informações a respeito das turmas criadas, desde alunos matriculados até o perfil destes em relação à turma (nome, presenças e avaliações).

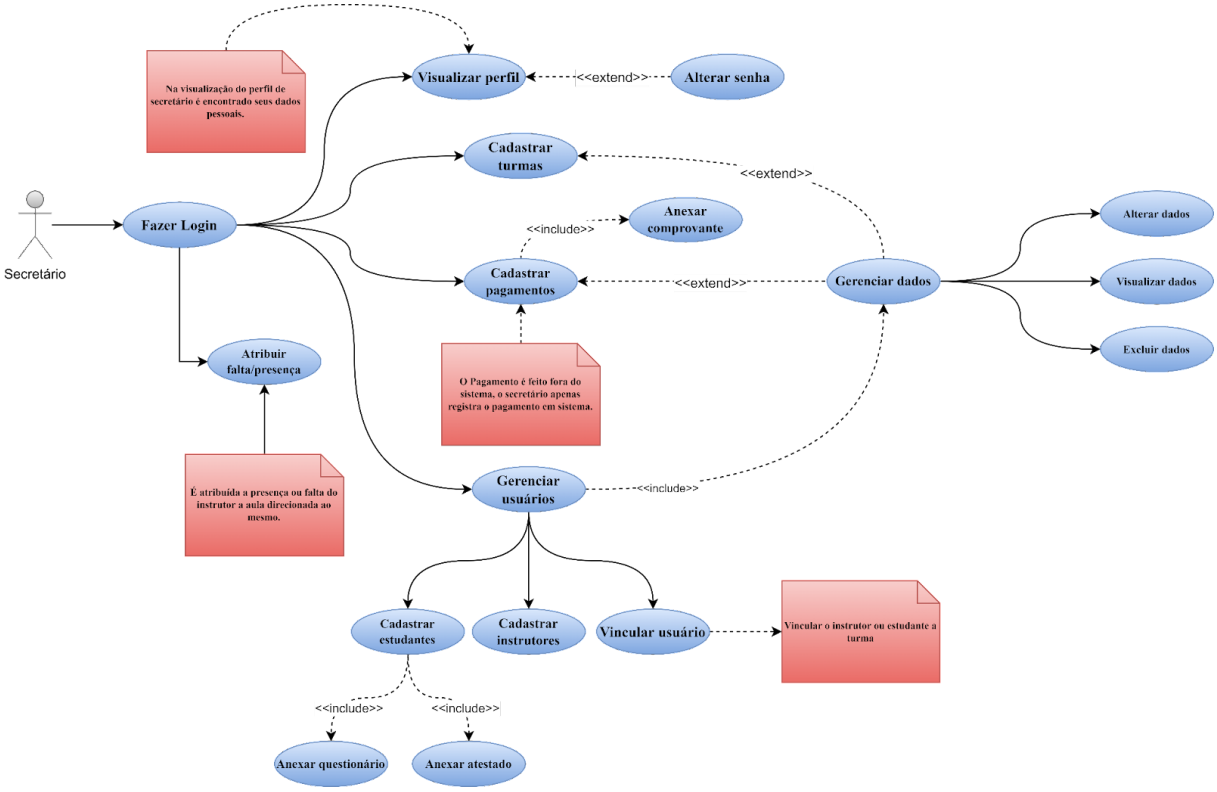
O estudante (Figura 4) é o cargo mais simples dentro do sistema. Possuindo apenas as funcionalidades de visualizar seu próprio perfil e alteração de sua senha.

Figura 1: Caso de uso do usuário administrador



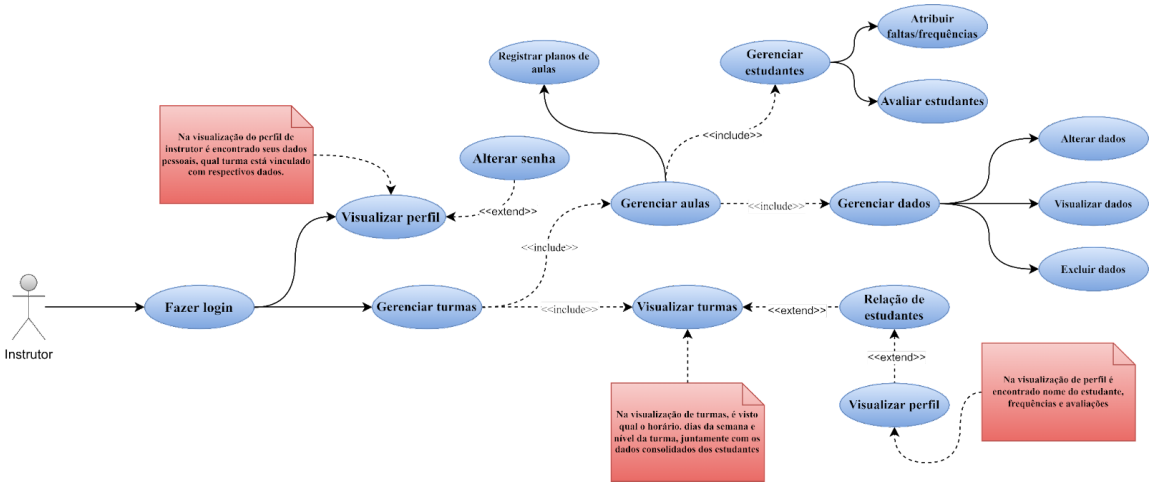
Fonte: Autoria própria (2022)

Figura 2: Caso de uso do usuário secretário



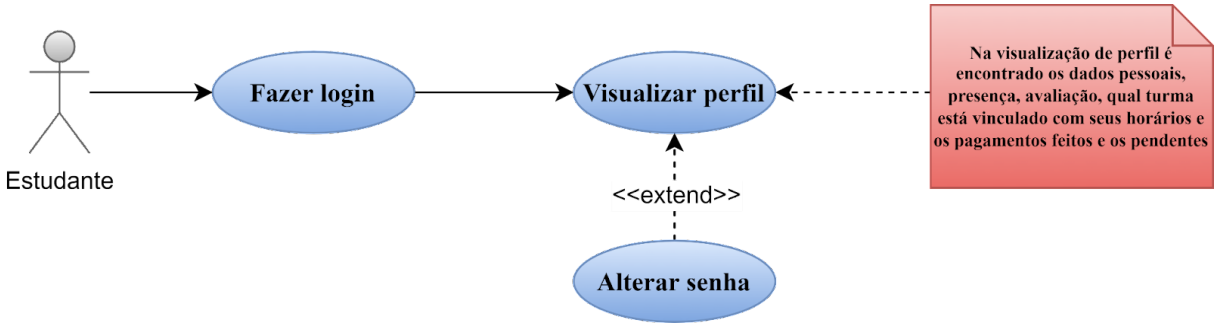
Fonte: Autoria própria (2022)

Figura 3: Caso de uso do usuário instrutor



Fonte: Autoria própria (2022)

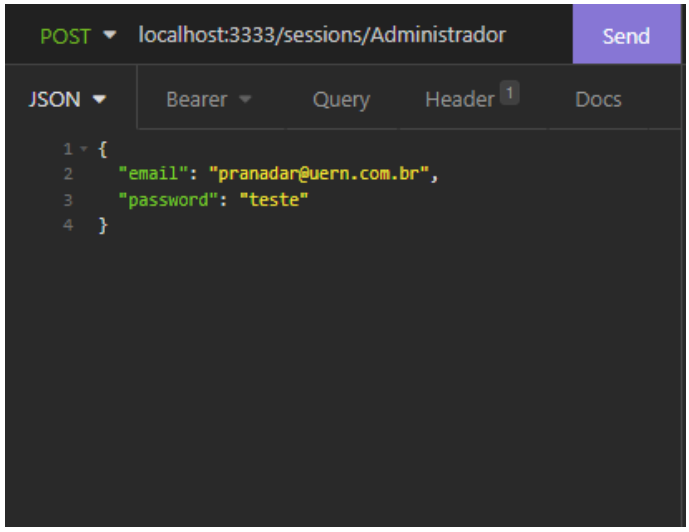
Figura 4: Caso de uso do usuário estudante



Fonte: Autoria própria (2022)

Todos os tipos de usuários apresentados possuem suas autenticações que lhe darão permissão para suas respectivas tarefas de acordo com o tipo de usuário passado na URL, como na Figura 5.

Figura 5: Autenticação no Insomnia



Fonte: Autoria própria (2022)

3.2.2 Autenticação

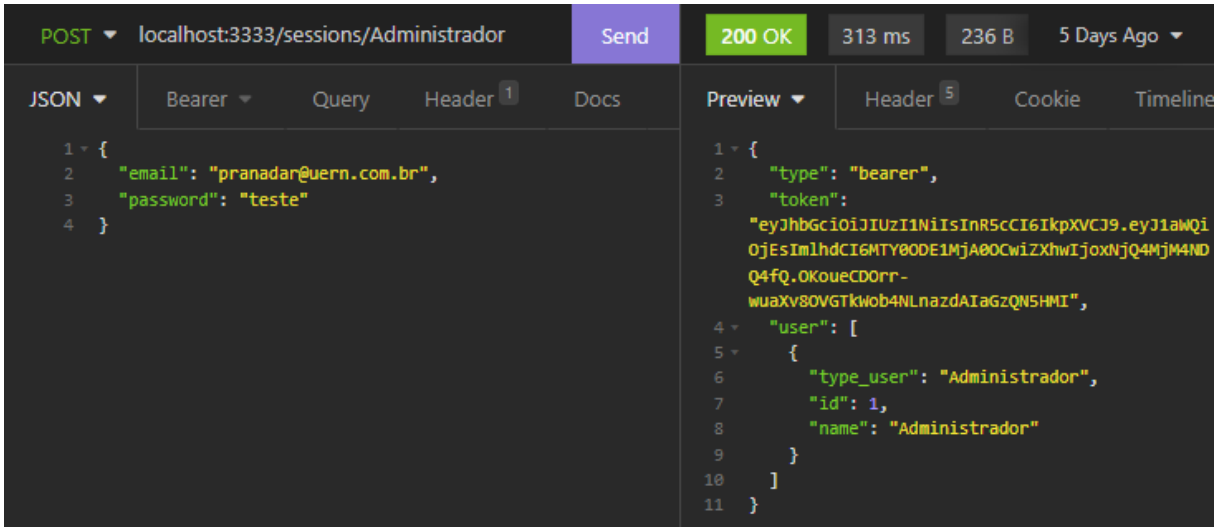
A autenticação dos usuários funciona de acordo com o cargo que foi escolhido no momento do login. Ao tentar efetuar o login a API irá verificar se o email e senha passados são compatíveis com os presentes no banco de dados. Em caso de sucesso o sistema irá gerar um token JWT (JSON Web Token), que de acordo com JWT ([202?], tradução nossa) "… é um padrão aberto (RFC 7519) que define uma forma compacta e autocontida para a transmissão segura de informações entre as partes como um objeto JSON". O token virá acompanhado com algumas informações do usuário para conceder suas permissões no sistema conforme a Figura 6.

3.2.3 Rotas e Controladores

Para todos usuários e funcionalidades do sistemas foram criados suas rotas e controladores para realização de seus métodos, as rotas são os caminhos que compõe a URL para o acesso de seus recursos e os controladores realizam a parte lógica durante a requisição, esses dois em conjunto formam o CRUD (Create, read,

update e delete) que é um acrônimo para as formas de manusear os dados recebidos, permitindo a criação, exibição, alteração e exclusão de seus dados. Utilizando respectivamente dos métodos HTTP: POST, GET, PUT e DELETE.

Figura 6: Autenticação no Insomnia



Fonte: Autoria própria (2022)

A tabela 1, apresenta o uso dos métodos HTTP do usuário administrador juntamente com suas rotas e funções.

Tabela 1 - Rotas do usuário administrador

Método	Rota	Descrição da rota
POST	/administrator	Cria um usuário do tipo administrador.
GET	/administrator	Exibe todos os usuários do tipo administrador.
GET	/administrator/:id	Exibe um administrador específico pelo ID.
PUT	/administrator/:id	Altera um administrador pelo ID.
DELETE	/administrator/:id	Deleta um administrador pelo ID.

Fonte: Autoria própria (2022)

De acordo com as Figuras 7 e 8 temos o desenvolvimento do CRUD do usuário administrador.

Figura 7: CRUD AdministratorController

```
183     async create({ request, response }) {
184         try{
185         const data = request.only(["name",
186                                    "email",
187                                    "password",
188                                    "cpf",
189                                    "birthday",
190                                    "rg",
191                                    "phone_number",
192                                    "category",
193                                    "status"]);
194         return await Administrator.create(data);
195         }catch(error){
196             return response.status(500).json({message: error })
197         }
198     };
199
200     async update({ params, request, response }) {
201         try{
202         const administrator = await Administrator.findOrFail(params.id);
203
204         const data = request.all([]);
205
206         administrator.merge(data);
207
208         return administrator.save();
209         }catch(error){
210             return response.status(500).json({message: error })
211         }
212     };
213
214     async index({ response }){
215         try{
216         const administrator = await Administrator.query().orderBy('name', 'asc').fetch();
217
218         return administrator;
219         }catch(error){
220             return response.status(404).json({message: error })
221         }
222     };
```

Fonte: Autoria Própria (2022)

Começando com o método *create,* partindo da linha 183 até a linha 198, no qual primeiramente a constante *data* recebe os dados pegos na requisição (dados esses destacados em laranja), após o armazenamento na constante o sistema retorna os dados, salvando diretamente no banco de dados assim criando um novo usuário administrador.

Partindo da linha 200 até a 212 se encontra o método *update,* método esse que permitirá o sistema atualizar algum usuário do tipo administrador. O método utiliza do parâmetro id passado para encontrar esse usuário já criado no sistema, após localizado, o método usa dos dados passados na requisição para mesclar com os dados já salvos no banco de dados, permitindo assim uma atualização.

O método *index* está sendo implementado da linha 214 a 222. Esse método não passa nenhum corpo na requisição e nem parâmetro, ele apenas possui o *response.* Quando este método é chamado o sistema retorna todos os usuários administradores já criados em ordem alfabética, como está implementado na linha 216, e encerra o evento.

Figura 8: CRUD AdministratorController

```
223
224     async show ({ params, response}) {
225         try{
226         const administrator = await Administrator.findOrFail(params.id);
227
228         return administrator;
229         }catch(error){
230             return response.status(404).json({message: error })
231         }
232     }
233
234     async delete({ params, response }){
235         try{
236         const administrator = await Administrator.findOrFail(params.id);
237         await administrator.delete();
238
239         response.json({
240             message: 'Administrador deletado'
241         });
242     }catch(error){
243         return response.status(404).json({message: error })
244     }
245     };
246 };
247
248 module.exports = AdministratorController
249
```

Fonte: Autoria Própria (2022)

O método *show* implementado nas linhas 224 a 232, permite ao usuário visualizar os dados de um administrador específico. Este método utiliza o uso do parâmetro id para encontrar o usuário escolhido, após isso o retorna.

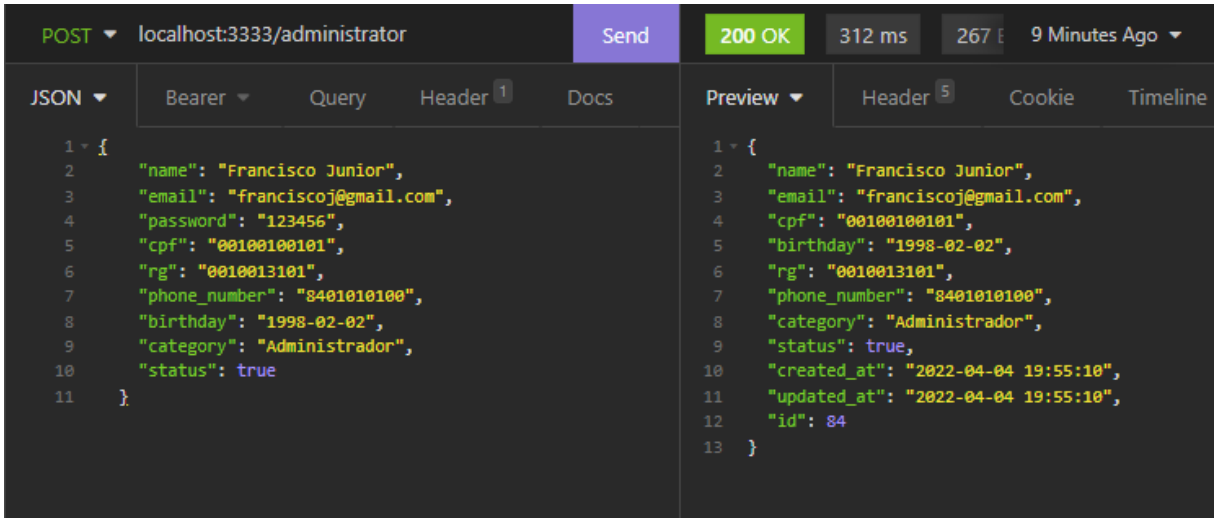
O método *delete* foi implementado nas linhas 234 a 246, permite a exclusão de dados de um usuário do tipo administrador. O método assim como o anterior faz uso apenas do id do usuário passado como parâmetro e após a busca no banco de

dados em caso, em caso de sucesso, deleta todos os dados respectivos a esse usuário.

Em todos os métodos citados anteriormente foi utilizado o bloco *try catch,* esses blocos são utilizados para instruções. O sistema deverá tentar executar tudo que estiver dentro do bloco *try,* porém em caso de falha deverá executar o bloco *catch.*

Nas Figuras 9 e 10, logo abaixo, podem ser vistos exemplos práticos diretamente do Insomnia da utilização dos métodos *create* e *index* respectivamente:

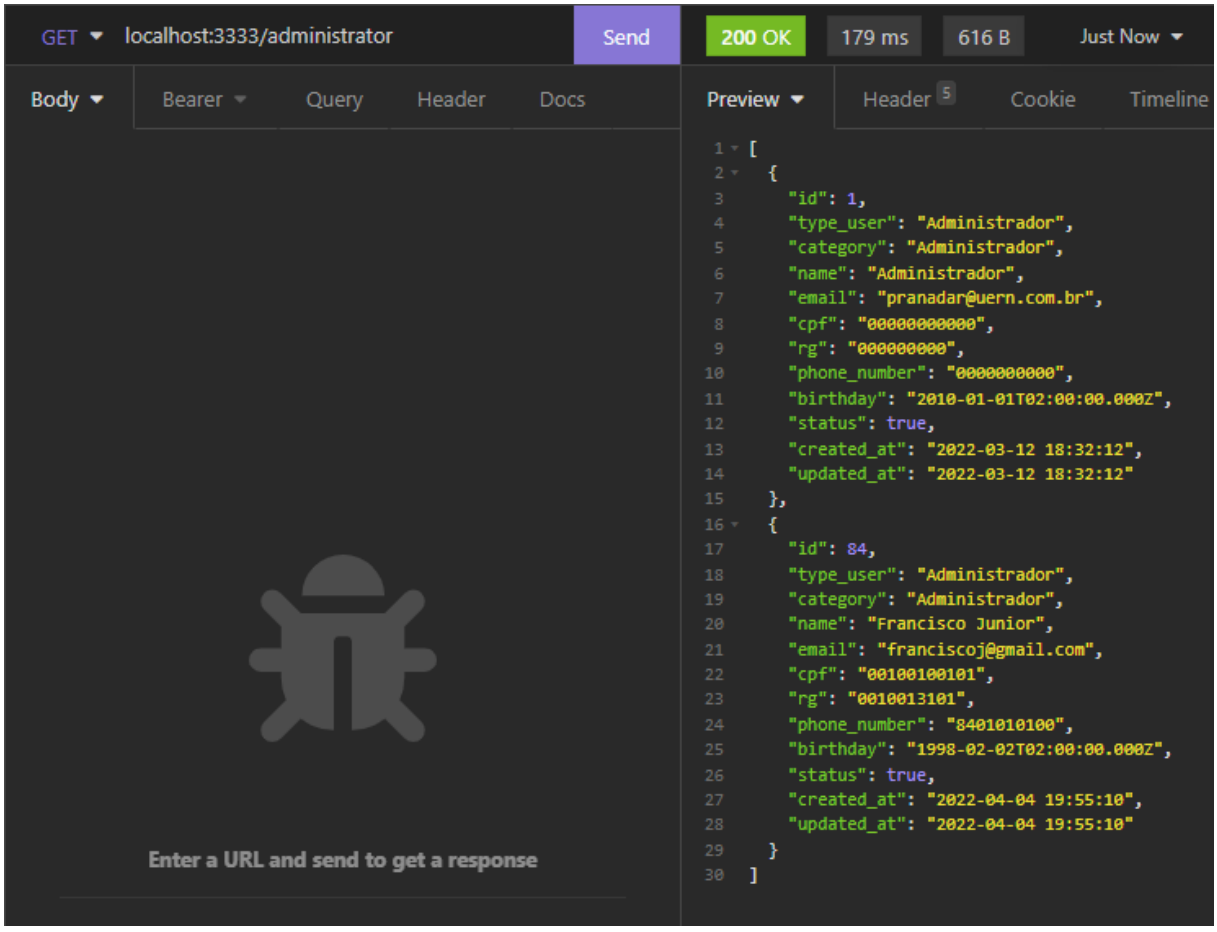
Figura 9: Método *create* diretamente do Insomnia



Fonte: Autoria Própria (2022)

O restante das rotas e controladores segue o mesmo padrão da tabela 1 e Figuras 7,8,9 e 10. No **Apêndice A** são apresentadas todas as rotas da aplicação, também podendo ser visualizadas diretamente em sua versão deploy no site https://pra-nadar-api.herokuapp.com/docs.

Figura 10: Método *index* diretamente do Insomnia



Fonte: Autoria Própria (2022)

3.2.4 Upload de arquivos

Para o upload dos questionários e atestados foram criados métodos específicos que necessitam previamente de um usuário do tipo estudante cadastrado no sistema para efetivação do seu cadastro. Neste método os tipos de arquivos aceitos são: PDF, JPEG,PNG e JPG. Cada arquivo tem seu tamanho máximo definido para 5MB (Megabytes). O caminho onde são armazenados os arquivos é registrado no banco de dados e vinculado ao ID do estudante. O método pode ser visto na Figura 11.

Como consta na Figura 11, o método *upload* foi implementado da linha 55 até a 98. Para o uso do método é necessário passar o parâmetro id, que neste caso é o id do aluno a quem pertence os arquivos do *upload*, como *request* deverá ser passado o diretório dos arquivos que vão ser anexados. O método primeiramente busca o

estudante por seu id, logo em seguida nas linhas 58 até 62 é implementado o *validationOptions* uma constante com as especificações aceitas para os arquivos que irão ser anexados. Nas linhas 64 e 65 são criados constantes para receber os arquivos atestado e questionário respectivamente, utilizando das especificações criadas acima. Logo em seguida, nas linhas 67 a 70 e 81 a 84, os arquivos que foram anexados tem seu nome modificado para a data em que foram enviados e são movidos a pasta upload dentro do sistema. Depois de anexadas e movidas, as linhas 72 a 79 e 86 a 93 trabalham de maneira semelhantes, mudando apenas o foco do arquivo (atestado e questionário), elas trabalham para verificar se esses arquivos realmente foram movidos e em caso de sucesso é preenchido a tabela com os respectivos dados.

Figura 11: Método de upload dos arquivos

```
55  async upload ({ params, request }) {
56      const student = await Student.findOrFail(params.id)
57
58      const validationOptions = {
59          types: ['image','pdf'],
60          size: '5mb',
61          extnames: ['jpg','jpeg','png','pdf']
62      };
63
64      const imageFile = request.file('attested', validationOptions);
65      const imageFile2 = request.file('quiz', validationOptions)
66
67      await imageFile.moveAll(Helpers.tmpPath('uploads'), file =>({
68          name: `${Date.now()}-${file.clientName}`,
69          overwrite: true,
70      }));
71
72      if (!imageFile.movedAll()) {
73          return imageFile.error();
74      }
75      await Promise.all(
76          imageFile
77            .movedList()
78            .map(image => student.file().create({ path: image.fileName, type: "Atestado" }))
79        )
80
81      await imageFile2.moveAll(Helpers.tmpPath('uploads'), file =>({
82          name: `${Date.now()}-${file.clientName}`,
83          overwrite: true,
84      }));
85
86      if (!imageFile2.movedAll()) {
87          return imageFile2.error();
88      }
89      await Promise.all(
90          imageFile2
91            .movedList()
92            .map(image => student.file().create({ path: image.fileName, type: "Quiz" }))
93        )
94
95      await Database.from('students').where('id', '=', params.id).update('status',true)
96
97      return 'Files uploaded';
98  }
99
```

Fonte: Autoria Própria (2022)

4 CONSIDERAÇÕES FINAIS

Com o fim desse projeto, foi entregue uma API que em conjunto com o *front-end* e *mobile* vão auxiliar todo o programa Pra Nadar na sua gestão de dados. A API faz o uso dos conceitos de arquiteturas orientadas a serviços e microsserviços que atualmente vem sendo uma tendência durante a implementação, permitindo um desenvolvimento de serviços de maneira bem mais independente, logo um serviço não necessariamente está vinculado a um usuário ou funcionalidade específica, na realidade, está fazendo uso de ambos, permitindo uma execução mais dinâmica e otimizada, facilitando o trabalho dos instrutores e aqueles que gerem o programa.

Como perspectiva para o futuro, pode ser implementado alterações na autenticação, atualmente o token não é salvo diretamente no banco de dados porém seria de mais fácil controle criar uma tabela para armazenar esses tokens e administrar essas sessões. Outra boa adição seria a funcionalidade de geração automáticas de planilhas com os dados que já estão presentes dentro do sistema. Ao todo a API foi feita de maneira bem genérica, portanto, é possível realizar expansões em relação a funcionalidades que serão necessárias de maneira simples como também a utilização em outros projetos que a necessitem.

REFERÊNCIAS

HOMBOLT TECHNOLOGY. **Introducing yourself to AdonisJs! A Node.js MVC framework in the style of Laravel.** [*S. l.*], 25 ago. 2021. Disponível em: https://hombolttech.com/blog/introduction-to-adonisjs/. Acesso em: 30 mar. 2022.

JWT. **JSON Web Token Introduction.** [S. l.], [202?]. Disponível em: https://jwt.io/introduction. Acesso em: 31 mar. 2022.

KONG. **Introduction to Insomnia.** [*S. l.*], [2021?]. Disponível em: https://docs.insomnia.rest/insomnia/get-started. Acesso em: 30 mar. 2022.

LIMA, Guilherme. **REST: Conceito e fundamentos.** [S. l.], 22 set. 2020. Disponível em: https://www.alura.com.br/artigos/rest-conceito-e-fundamentos?gclid=CjwKCAjwuYWSBhByEiwAKd_n_uq_o0oE6DdqN6coQuwpDBEmROVGLiiU0jXgFfteCfQaVpuOPblQXhoCICgQAvD_BwE. Acesso em: 29 mar. 2022.

MICROSOFT. **Getting Started.** [S. l.], [202?]. Disponível em: https://code.visualstudio.com/docs. Acesso em: 30 mar. 2022.

POSTGRESQL. **PostgreSQL: About.** [S. l.], [202?]. Disponível em: https://www.postgresql.org/about/. Acesso em: 30 mar. 2022.

ROCHA, Fabio Gomes. **Introdução ao FDD - Feature Driven Development.** [S. l.], 2013. Disponível em: https://www.devmedia.com.br/introducao-ao-fdd-feature-driven-development/27971. Acesso em: 29 mar. 2022.

APÊNDICE A - ROTAS DISPONÍVEIS NA API

Rotas sobre administradores

Método	Rota	Descrição
POST	/administrator	Cria um administrador
GET	/administrator	Mostra todos os administradores
GET	/administrator/id	Mostra os valores de um administrador específico com o parâmetro id
PUT	/administrator/id	Atualiza dados de um administrador com o parâmetro id
DELETE	/administrator/id	Deleta um administrador com o parâmetro id

Rotas sobre secretários

Método	Rota	Descrição
POST	/secretary	Cria um secretário
GET	/secretary	Mostra todos os secretários
GET	/secretary/id	Mostra os valores de um secretário específico com o parâmetro id
PUT	/secretary/id	Atualiza dados de um secretário com o parâmetro id
DELETE	/secretary/id	Deleta um secretário com o parâmetro id

Rotas sobre instrutores

Método	Rota	Descrição
POST	/instructor	Cria um instrutor
GET	/instructor	Mostra todos os instrutores
GET	/instructor/id	Mostra os valores de um instrutor específico

		com o parâmetro id
PUT	/instructor/id	Atualiza dados de um instrutor com o parâmetro id
DELETE	/instructor/id	Deleta um instrutor com o parâmetro id

Rotas sobre estudantes

Método	Rota	Descrição
POST	/student	Cria um estudante
GET	/student	Mostra todos os estudantes
GET	/student/id	Mostra os valores de um estudante específico com o parâmetro id
PUT	/student/id	Atualiza dados de um estudante com o parâmetro id
DELETE	/student/id	Deleta um estudante com o parâmetro id
GET	/student/frequency/id	Lista frequências de um estudante específico pelo id

Rotas sobre usuários

Método	Rota	Descrição
GET	/users	Mostra todos os usuários
GET	/users/id	Mostra os valores de um usuário específico com o parâmetro id
PUT	/users/id	Atualiza dados de um usuário com o parâmetro id
DELETE	/users/id	Deleta um usuário com o parâmetro id

Rotas sobre turmas

Método	Rota	Descrição
POST	/classroom	Cria uma turma
GET	/classroom	Mostra todas as turmas
GET	/classroom/id	Mostra os valores de uma turma específica com o parâmetro id
PUT	/classroom/id	Atualiza dados de uma turma com o parâmetro id
DELETE	/classroom/id	Deleta uma turma com o parâmetro id
GET	/classroom/class	Mostra id, nome e status das turmas
GET	/classroom/classplans/id	Lista todos os planos de aula de uma turma pelo parâmetro id
GET	/classroom/frequency/id	Lista todas as frequências de uma turma pelo parâmetro id

Rotas sobre os relacionamento de estudantes

Método	Rota	Descrição
POST	/studentrelationship	Cria um relacionamento
GET	/studentrelationship	Mostra todos os relacionamentos
GET	/studentrelationship/id	Mostra os valores de um relacionamento específico com o parâmetro id
PUT	/studentrelationship/id	Atualiza dados de um relacionamento com o parâmetro id
DELETE	/studentrelationship/id	Deleta um relacionamento com o parâmetro id
GET	/studentrelationship/bound	Retorna os estudantes que estão vinculados
GET	/studentrelationship/unbound	Retorna os estudantes que não

		estão vinculados
GET	/studentrelationship/listClass/id	Retorna os estudantes vinculados a uma turma específica

Rotas sobre os relacionamentos de instrutores

Método	Rota	Descrição
POST	/instructorrelationship	Cria um relacionamento
GET	/instructorrelationship	Mostra todos os relacionamentos
GET	/instructorrelationship/id	Mostra os valores de um relacionamento específico com o parâmetro id
PUT	/instructorrelationship/id	Atualiza dados de um relacionamento com o parâmetro id
DELETE	/instructorrelationship/id	Deleta um relacionamento com o parâmetro id
GET	/instructorrelationship/bound	Retorna os instrutores que estão vinculados
GET	/instructorrelationship/unbound	Retorna os instrutores que não estão vinculados
GET	/instructorrelationship/listClass/id	Retorna os instrutores vinculados a uma turma específica

Rotas sobre planos de aula

Método	Rota	Descrição
POST	/class	Cria um plano de aula
GET	/class	Mostra todos os planos de aula
GET	/class/id	Mostra os valores de um plano de aula específico com o parâmetro id

PUT	/class/id	Atualiza dados de um plano de aula com o parâmetro id
DELETE	/class/id	Deleta um plano de aula com o parâmetro id

Rotas sobre frequências de estudantes

Método	Rota	Descrição
POST	/frequency	Cria uma frequência
GET	/frequency	Mostra todas as frequências
GET	/frequency/id	Mostra os valores de uma frequência específica com o parâmetro id
PUT	/frequency/id	Atualiza dados de uma frequência com o parâmetro id
DELETE	/frequency/id	Deleta uma frequência com o parâmetro id

Rotas sobre presenças de instrutores

Método	Rota	Descrição
POST	/presence	Cria uma presença
GET	/presence	Mostra todas as presenças
GET	/presence/id	Mostra os valores de uma presença específica com o parâmetro id
PUT	/presence/id	Atualiza dados de uma presença com o parâmetro id
DELETE	/presence/id	Deleta uma presença com o parâmetro id

Rotas sobre avaliações dos alunos

Método	Rota	Descrição
POST	/assessment	Cria uma avaliação
GET	/assessment	Mostra todas as avaliações

GET	/assessment/id	Mostra os valores de uma avaliação específica com o parâmetro id
PUT	/assessment/id	Atualiza dados de uma avaliação com o parâmetro id
DELETE	/assessment/id	Deleta uma avaliação com o parâmetro id

Rotas sobre pagamentos

Método	Rota	Descrição
POST	/payment	Cria um pagamento
GET	/payment	Mostra todos os pagamentos
GET	/payment/id	Mostra os valores de um pagamento específico com o parâmetro id
PUT	/payment/id	Atualiza dados de um pagamento com o parâmetro id
DELETE	/payment/id	Deleta um pagamento com o parâmetro id

Rotas sobre autenticação

Método	Rota	Descrição
POST	/sessions/type_user	Efetua o login
PUT	/sessions/change/type_user	Atualiza a senha de um usuário
PUT	/sessions/reset/id	Reseta a senha de um usuário

Rotas sobre arquivos dos alunos (Questionário e atestado)

Método	Rota	Descrição
POST	/files/student/id/upload	Faz o upload dos arquivos
GET	/files/download/fileName	Faz o download dos arquivos

Rotas sobre recibos de pagamento

Método	Rota	Descrição
POST	/files/receipt/id	Faz o upload dos recibos
GET	/files/download/receipt/fileName	Faz o download dos recibos