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

JEFTER ROBERTO MOTA TARGINO

ANÁLISE, PROJETO E DESENVOLVIMENTO DO SISTEMA DE GESTÃO DO PROGRAMA PRA NADAR

MOSSORÓ - RN
2022

JEFTER ROBERTO MOTA TARGINO

ANÁLISE, PROJETO E DESENVOLVIMENTO DO SISTEMA DE GESTÃO 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

JEFTER ROBERTO MOTA TARGINO

ANÁLISE, PROJETO E DESENVOLVIMENTO DO SISTEMA DE GESTÃO 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: 19/04/2022

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. Maximiliano Araújo da Silva Lopes
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

1.1 DESCRIÇÃO GERAL DO PROJETO

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.

1.2 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.

1.3 OBJETIVOS

1.3.1 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.

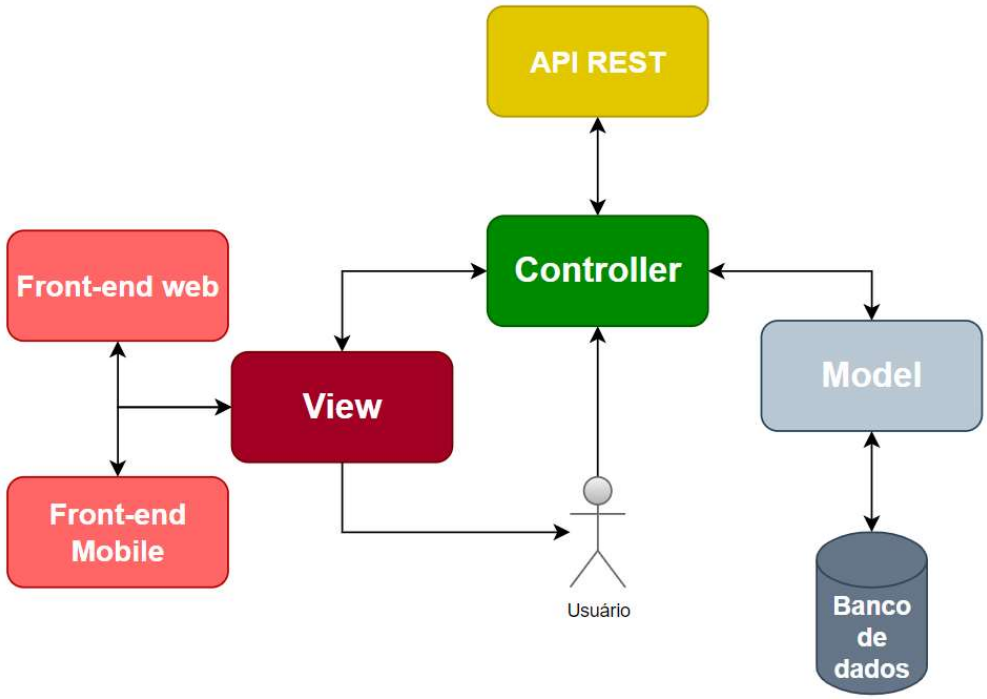
1.3.2 Objetivo específico

O trabalho tem como objetivo o levantamento e análise de requisitos, podendo assim construir o projeto de sistema, desenvolvendo as especificações do sistema, com diagramas de caso de uso, diagramas de classe e os requisitos funcionais para o sistema, a fim de indicar quais as funcionalidades seriam necessárias e qual o fluxo de funcionamento para cada usuário e suas ações no sistema. Como também o objetivo de elaborar a prototipagem de telas para os *front-end Web* e *Mobile*, para validação de projeto, além do propósito do desenvolvimento e gerenciamento do banco de dados.

2 ARQUITETURA DO SISTEMA

O sistema foi projetado utilizando da arquitetura MVC (do inglês, *Model-View-Controller*), que segundo Zucher (2020): "... é um padrão de arquitetura de software responsável por contribuir na otimização da velocidade entre as requisições feitas pelo comando dos usuários." A Figura 1 ilustra a estrutura do sistema Pra Nadar.

Figura 1: Estrutura MVC do sistema Pra Nadar



Fonte: Autoria Própria (2022)

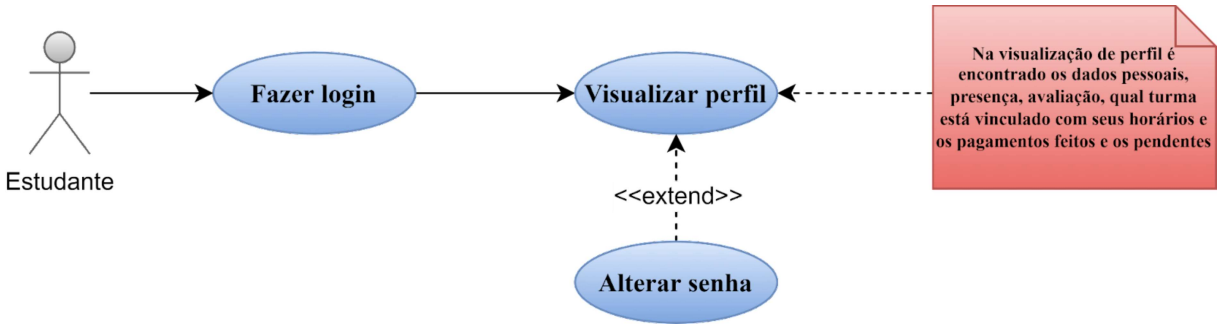
Detalhando a Figura 1, o *Model* seria o responsável pela manipulação dos dados e foi implementado utilizando de um banco de dados relacional, a *View* responsável por mostrar as informações do *Model* de forma apresentável ao usuário, sendo implementada através do *front-end Web* e *Mobile*, enquanto ao *Controller* com sua função de ligar o *Model* e a *View*, fazendo com que as informações do *Model* seja repassada às *Views* e vice-versa, foi implementado através de uma API REST (do inglês, *Application Programming Interface, Representational State Transfer*), por último se tem o usuário que interage de maneira indireta com o *controller* repassando os dados e interagindo com a *View* recebendo os dados do *Model*.

2.1 DIAGRAMA DE CASO DE USO

Após levantamento de requisitos, ficou acordado que o sistema teria a 4 atores: o administrador, secretário, instrutor e estudante; todos com sua devida hierarquia no sistema e suas respectivas funções. Todos os atores contém uma funcionalidade em comum, sendo esta, a visualização de perfil, permitindo a possibilidade de alterar a sua própria senha.

Descrevendo o diagrama de caso de uso de estudante ilustrado na Figura 2, é referente ao usuário do mesmo nome, é o usuário com cargo mais baixo no sistema, foi destinado funcionalidades da visualização do seu perfil, com acesso a quais dados estão cadastrados, quais as turmas está vinculado, quais suas notas, frequências e a relação dos pagamentos feitos e pendentes.

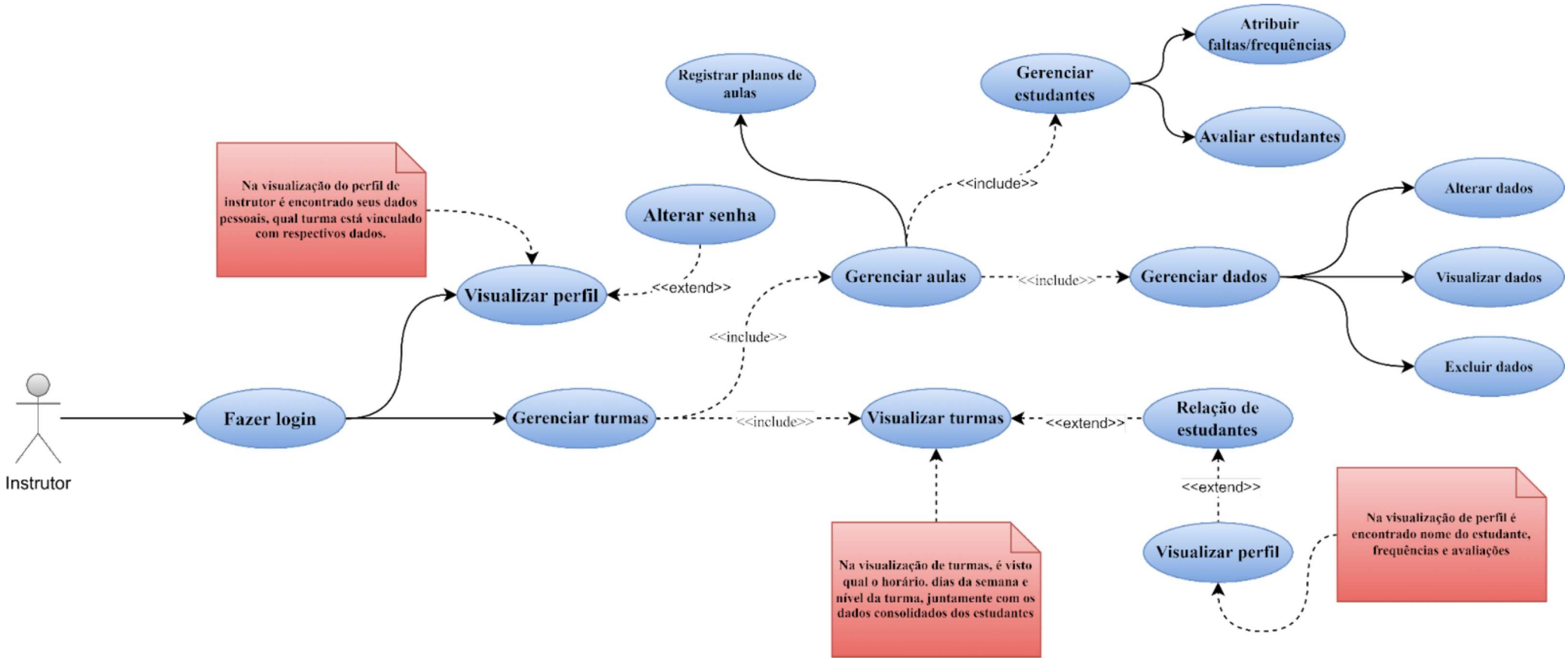
Figura 2: Diagrama de caso de uso do estudante



Fonte: Autoria Própria (2022)

No diagrama de caso de uso de instrutor apresentado na Figura 3, ilustra os casos de uso direcionados ao usuário tipo instrutor, sendo ele o responsável por gerenciar as aulas do programa, o que inclui o registro dos planos de aulas, a atribuição das faltas/frequências e avaliações dos estudantes, sendo possível visualizar, alterar ou excluir as informações. Sendo um usuário do tipo instrutor também seria possível a visualização das turmas ao qual está vinculado, dando a possibilidade de visualizar a relação dos estudantes e seus perfis.

Figura 3: Diagrama de caso de uso do instrutor



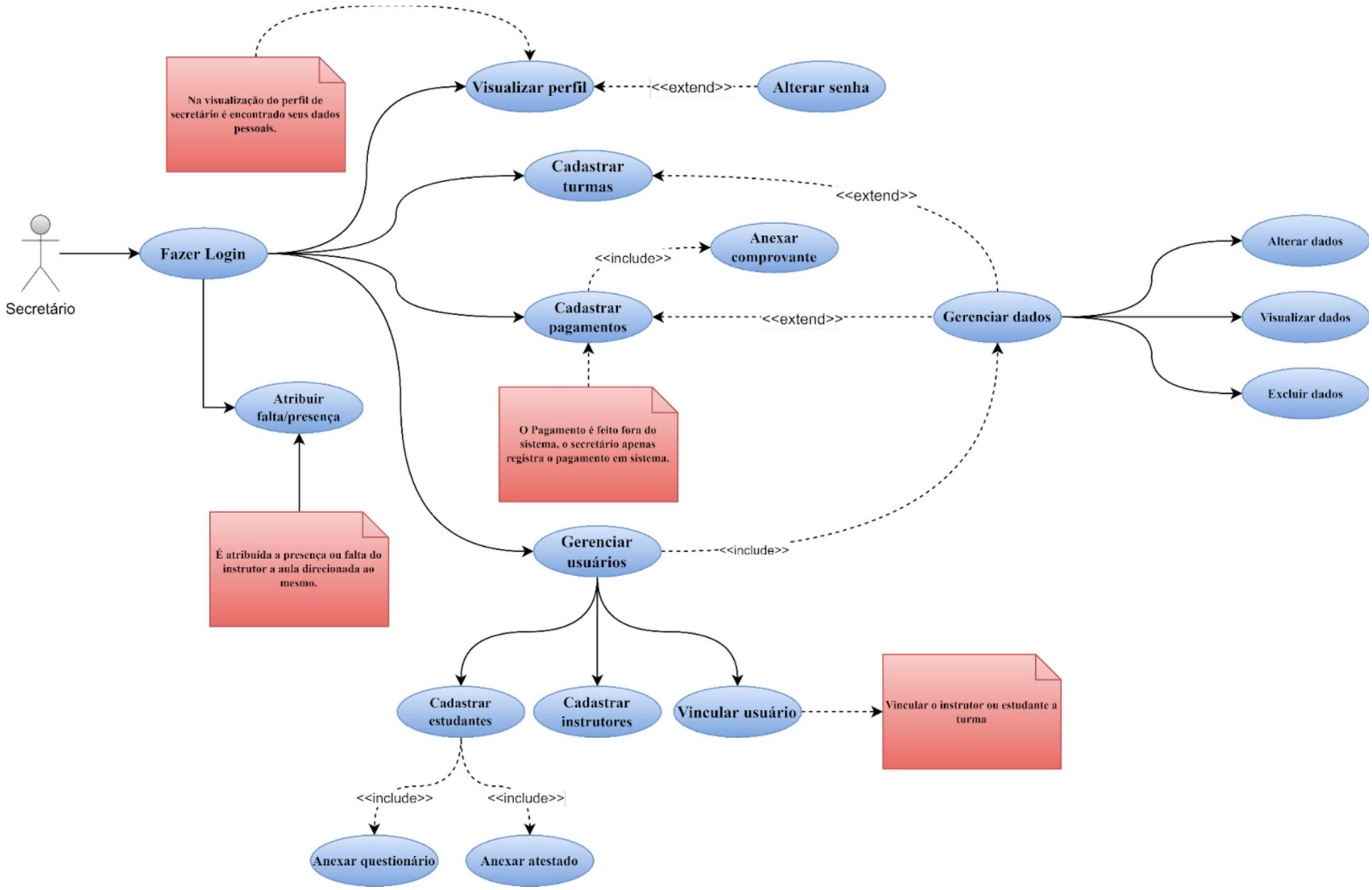
Fonte: Autoria Própria (2022)

Detalhando o caso de uso de secretário esboçado na Figura 4, é apresentado quais as funções para o usuário secretário. É um usuário com o cargo administrativo no sistema, não possuindo privilégio para gerenciar usuários de mesma hierarquia, atuando na gestão de turmas, capaz de excluir, atualizar, criar e visualizar os dados. O que lhe permite as funcionalidades de atribuir falta/presença no instrutor, o gerenciamento de turmas, gerenciamento de instrutores e estudantes, ao qual no cadastro de estudantes é o responsável pelo anexo dos atestados e questionários no ato da matrícula, além do papel de vincular os estudantes e instrutores as turmas, também responsável pelo registro dos pagamentos feitos ao programa, no registro de pagamento é realizado o envio dos comprovantes semelhante ao que é realizado no cadastro de estudante.

Finalizando os casos de uso, é observado o caso de uso do ator administrador na Figura 5, é o usuário com maior privilégio no sistema, ficou decidido que existiria apenas um usuário deste tipo, o qual teria acesso a todas as funcionalidades descritas nos casos de usos anteriores, incluindo as funcionalidades do secretário e instrutor. Este usuário pode gerenciar todos os outros usuários, incluindo o gerenciamento de secretário de forma exclusiva, além da funcionalidade de redefinir a senha de algum usuário específico, retornando a senha padrão criada no ato do cadastro.

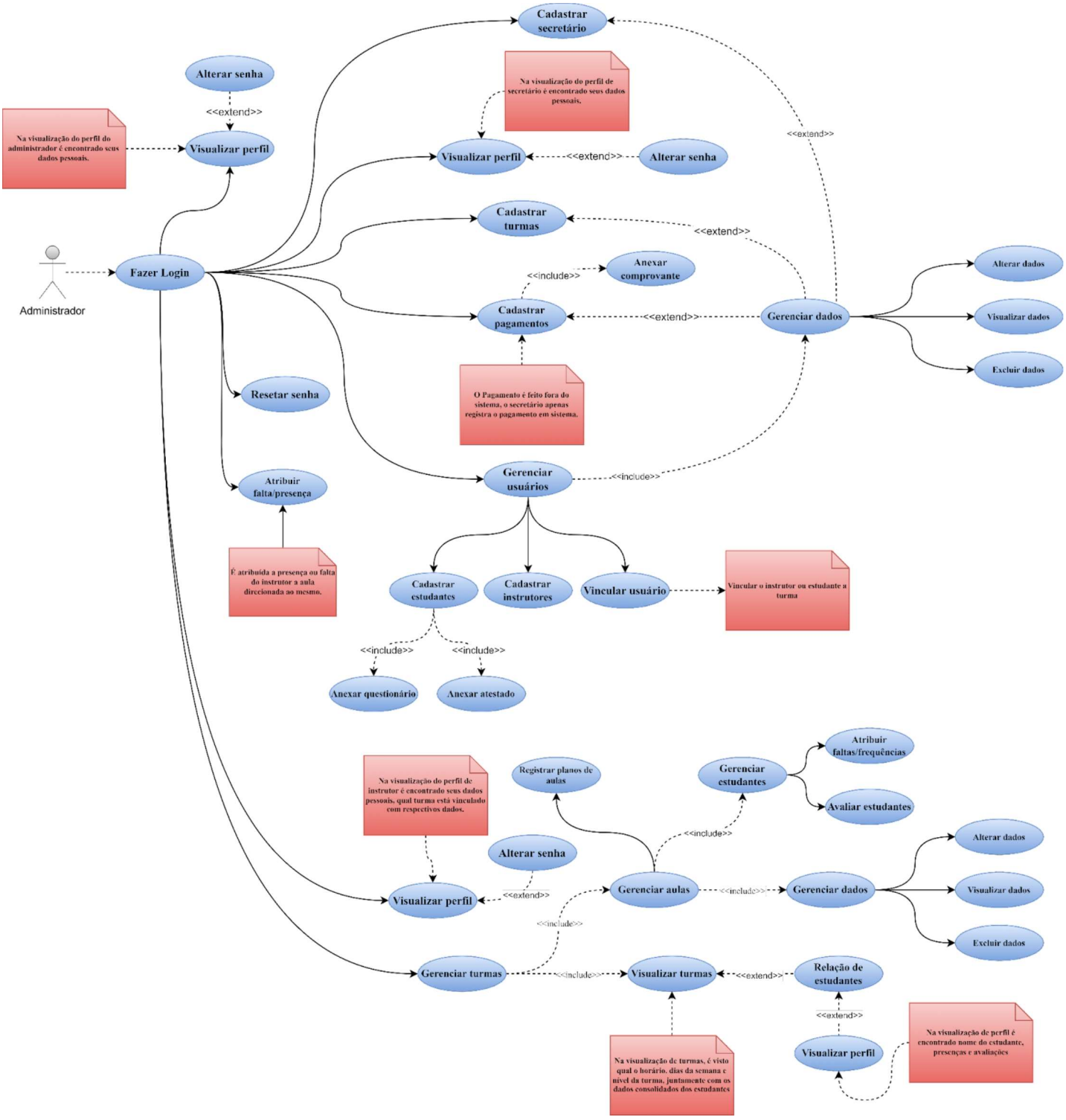
Os diagramas de casos de uso, foram bastante cruciais no desenvolvimento do sistema, sendo utilizado desde a sua criação até a finalização do trabalho.

Figura 4:Diagrama de caso de uso do secretário



Fonte: Autoria Própria (2022)

Figura 5: Diagrama de caso de uso do administrador



Fonte: Autoria Própria (2022)

2.2 DIAGRAMA DE CLASSES

Foi utilizado o diagrama de classes, que segundo Sommerville (2011, p. 90): “... são usados no desenvolvimento de um modelo de sistema orientado a objetos para mostrar as classes de um sistema e as associações entre essas classes", para estabelecer um modelo estrutural das classes e mostrar as conexões entre elas, que de acordo com Sommerville (2011, p. 100): “Os modelos estruturais mostram a organização e a arquitetura de um sistema.”

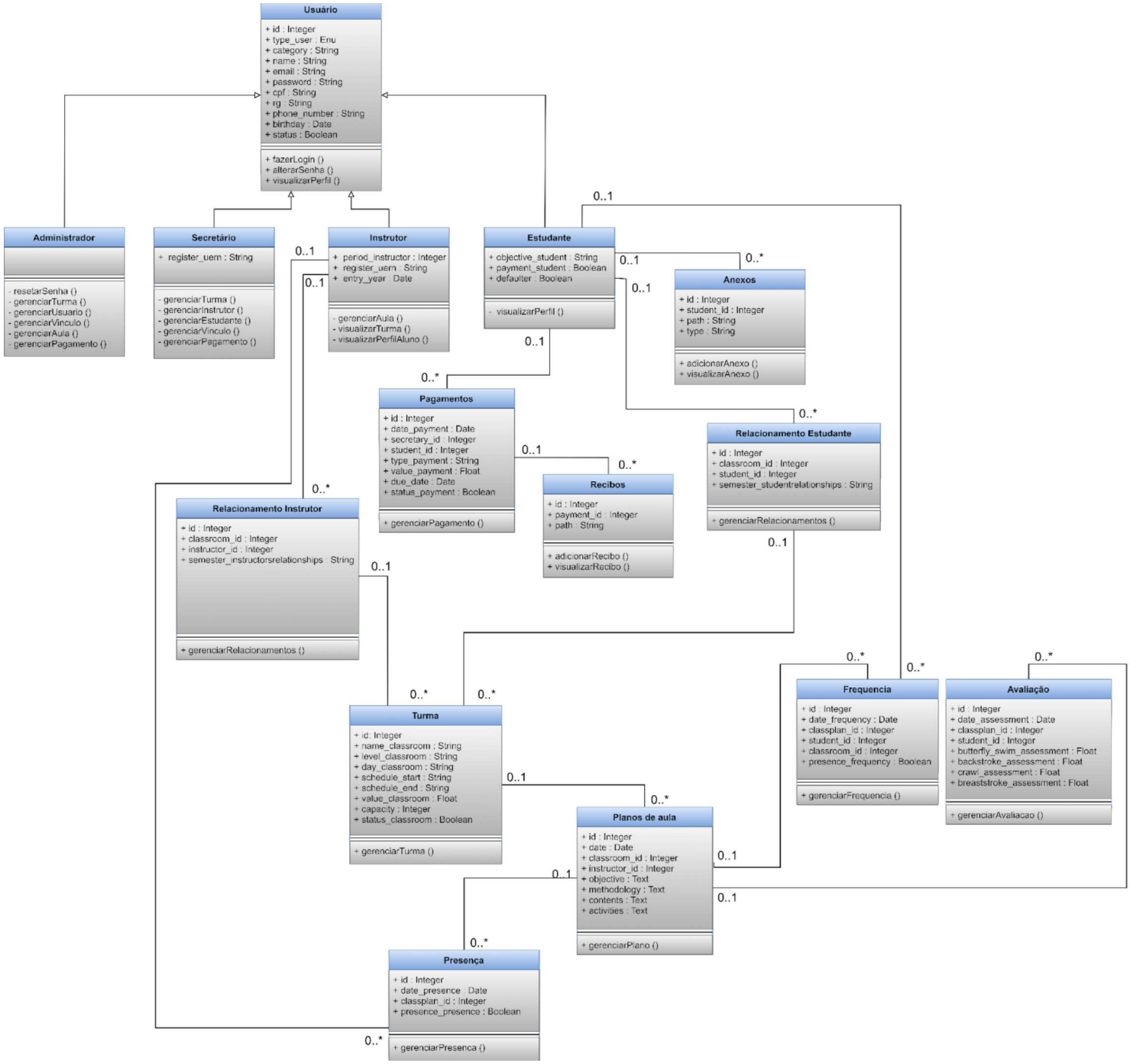
Na Figura 6, é ilustrado o diagrama de classe, nela há uma classe Usuário com os dados comuns a todos os usuários do sistema, com quatro subclasses representando o administrador do sistema, o secretário, instrutores e os estudantes, cada uma dessas subclasses têm seus dados particulares e métodos.

A classe Estudante contém um conexão quase que direta com a classe Anexos, pelo fato da mesma representar os anexos que são associados ao estudante no ato do cadastro, que são os questionário e atestado médico, também se relaciona com a classe Pagamentos, simbolizando os pagamentos realizados pelos estudantes ao programa, a classe Pagamentos tem uma ligação com a classe Recibos, representando todos os comprovantes dos pagamentos registrados pelo secretário.

As classes Relacionamento Instrutor e Relacionamento Estudante, representam os vínculos que os instrutores e estudantes têm com as turmas simbolizada na classe Turma, dito isto, cada turma tem suas aulas, possuindo um plano de aula associado a ela (relação representada pela classe Plano de aula), na qual cada estudante tem registrado sua frequência representada pela classe Frequência, cada aula ainda tem associada a presença do instrutor na classe Presença.

Em todo o diagrama são indicadas todas as conexões entre as classes, apresentando também os atributos e valores permitidos que foram utilizados na criação do banco de dados.

Figura 6: Diagrama de classes



Fonte: Autoria Própria (2022)

2.3 PRINCIPAIS REQUISITOS FUNCIONAIS

Logo após definir os casos de uso e diagrama de classes, se fez necessário para definir os requisitos funcionais do sistema, sendo tratados com três prioridades, a que seria essencial para funcionamento do sistema, que seria importante e que seria seria um requisito desejável, é mostrado nos Quadros 1 ao 7 exemplos dos

requisitos funcionais projetados para o sistema, no **apêndice A** são listados todos os requisitos funcionais elaborados.

Quadro 1: Requisito funcional 01

[RF-01] Login do Usuário
Descrição de caso de uso: Este caso de uso permite que o usuário seja validado e tenha acesso ao sistema.
Prioridade: [X] Essencial [ ] Importante [ ] Desejável
Entradas e pré-condições: Tipo do acesso, email e senha; Usuário possuir cadastro no sistema.
Saídas e pós-condições: Tipo da autenticação, token, tipo do acesso, id e nome; Usuário logado.
Fluxo básico: 1. Usuário desejar logar no sistema 2. Usuário escolher o tipo de acesso 3. Usuário preencher email e senha 4. Autenticação é bem sucedida 5. Usuário é logado no sistema
Fluxo alternativo: 1. Usuário desejar logar no sistema 2. Usuário escolher o tipo de acesso 3. Usuário preencher email e senha 4. Autenticação é mal sucedida 5. Retorna ao passo 3.

Fonte: Autoria Própria (2022)

Quadro 2: Requisito funcional 04

[RF-04] Cadastro de Secretário
Descrição de caso de uso: Este caso de uso permite que o usuário tipo secretário seja cadastrado no sistema.
Prioridade: [X] Essencial [ ] Importante [ ] Desejável
Entradas e pré-condições: Categoria, nome, email, CPF, RG, telefone, data de nascimento e registro da UERN; Administrador logado no sistema.
Saídas e pós-condições: Usuário secretário cadastrado.

Fluxo básico: 1. Administrador logar no sistema 2. Administrador escolher cadastrar secretário 3. Administrador preencher o cadastro 4. Cadastro é efetuado com sucesso

Fonte: Autoria Própria (2022)

Quadro 3: Requisito funcional 05

[RF-05] Cadastro de Turma
Descrição de caso de uso: Este caso de uso permite que a turma seja cadastrada no sistema
Prioridade: [X] Essencial [ ] Importante [ ] Desejável
Entradas e pré-condições: Nome, nivel, dias da semana, horário de início, horário de término, valor, capacidade e status; Administrador ou secretário logado no sistema.
Saídas e pós-condições: Turma cadastrada.
Fluxo básico: 1. Administrador ou secretário logar no sistema 2. Administrador ou secretário escolher cadastrar turma 3. Administrador ou secretário preencher o cadastro 4. Cadastro é efetuado com sucesso

Fonte: Autoria Própria (2022)

Quadro 4: Requisito funcional 06

[RF-06] Cadastro de Instrutor
Descrição de caso de uso: Este caso de uso permite que o instrutor seja cadastrado no sistema.
Prioridade: [X] Essencial [ ] Importante [ ] Desejável
Entradas e pré-condições: Categoria, nome, email, CPF, RG, telefone, data de nascimento, período no curso, registro da UERN e data de entrada; Administrador ou secretário logado no sistema.
Saídas e pós-condições: Usuário instrutor cadastrado.
Fluxo básico: 1. Administrador ou secretário logar no sistema

2. Administrador ou secretário escolher cadastrar instrutor 3. Administrador ou secretário preencher o cadastro 4. Cadastro é efetuado com sucesso

Fonte: Autoria Própria (2022)

Quadro 5: Requisito funcional 07

[RF-07] Vincular Usuário
Descrição de caso de uso: Este caso de uso permite que o usuário instrutor ou estudante seja vinculado a uma turma.
Prioridade: [ ] Essencial [X] Importante [ ] Desejável
Entradas e pré-condições: ID da turma, ID do instrutor ou estudante e semestre; Administrador ou secretário logado no sistema, possuir turma cadastrada no sistema e possuir instrutor ou estudante cadastrado no sistema.
Saídas e pós-condições: Vínculo cadastrado.
Fluxo básico: 1. Administrador ou secretário logar no sistema 2. Administrador ou secretário decidir vincular estudante 3. Administrador ou secretário preencher o cadastro 4. Vínculo é efetuado com sucesso
Fluxo alternativo: 1. Administrador ou secretário logar no sistema 2. Administrador ou secretário decidir vincular instrutor 3. Administrador ou secretário preencher o cadastro 4. Vínculo é efetuado com sucesso

Fonte: Autoria Própria (2022)

Quadro 6: Requisito funcional 08

[RF-08] Cadastro de Estudante
Descrição de caso de uso: Este caso de uso permite que o estudante seja cadastrado no sistema.
Prioridade: [X] Essencial [ ] Importante [ ] Desejável
Entradas e pré-condições: Categoria, nome, email, CPF, RG, telefone, data de nascimento, objetivo e gratuidade; Administrador ou secretário logado no sistema, questionário preenchido e atestado.
Saídas e pós-condições: Usuário estudante cadastrado.

Fluxo básico: 1. Administrador ou secretário logar no sistema 2. Administrador ou secretário escolher cadastrar estudante 3. Administrador ou secretário preencher o cadastro 4. Administrador ou secretário anexa o questionário e atestado ao cadastro. 5. Cadastro é finalizado com sucesso

Fonte: Autoria Própria (2022)

Quadro 7: Requisito funcional 14

[RF-14] Registro de Plano de Aula
Descrição de caso de uso: Este caso de uso permite o registro de plano de aula no sistema.
Prioridade: [ ] Essencial [X] Importante [ ] Desejável
Entradas e pré-condições: Data da aula, ID da turma, ID do instrutor, objetivo, metodologia, conteúdo e atividades; Administrador ou instrutor logado no sistema e possuir turma cadastrada no sistema.
Saídas e pós-condições: Plano de aula cadastrado e criação da frequência dos estudantes vinculados.
Fluxo básico: 1. Administrador ou instrutor logar no sistema 2. Administrador ou instrutor escolhe a turma a registrar o plano de aula 3. Administrador ou instrutor desejar registrar plano de aula 4. Administrador ou instrutor preencher o cadastro 5. Plano de aula é cadastrado com sucesso

Fonte: Autoria Própria (2022)

3 IMPLEMENTAÇÃO

Nesta Seção, será abordado mais sobre a implementação desenvolvida. Tendo isso em vista, será falado sobre as tecnologias utilizadas e sobre a metodologia de desenvolvimento.

3.1 TECNOLOGIAS UTILIZADAS

3.1.1 Adonis JS

O sistema foi desenvolvido utilizando do framework AdonisJS, que para Fernandes (2018): “...é um *framework* mais robusto com uma série de funcionalidades prontas, extremamente baseado em *frameworks* famosos de outras linguagens como o Laravel, Rails ou Django.”. Outro critério utilizado foi que assim que instalado, ele já vem com uma estrutura pronta, além de já possuir uma série de funcionalidades pré-implementadas, como a autenticação (FERNANDES, 2018).

3.1.2 PostgreSQL

Para o registro de todas as informações foi utilizado o gerenciador de banco de dados objeto-relacional PostgreSQL, que conforme DevMedia ([202?]): “é um dos cinco SGBDs relacionais mais utilizados no mercado", por ser projetado para ser portátil, atuando em várias plataformas e é um software de código aberto e de baixo custo.(DEVMEDIA, [202?]).

3.1.3 Insomnia

Para o auxiliar o desenvolvimento e visualização das consultas do banco de dados, foi utilizado o Insomnia que é uma ferramenta cliente de API REST (do inglês, *Application Programming Interface, Representational State Transfer*) que para Noleto (2022) : “...é uma abstração de arquitetura de software que fornece dados em um formato padronizado para modelos de requisições HTTP.” , por permitir uma fácil organização das solicitações HTTP, podendo agrupá-las de acordo com o necessário. Também por permitir o uso de vários ambientes para estas requisições.

3.1.4 Heroku

Para integração do sistema foi utilizado a plataforma Heroku, que para Heroku ([202?], tradução nossa) :“... é uma plataforma em nuvem que permite que as empresas criem, entreguem, monitorem e dimensionem aplicativos.”

3.1.5 Visual Studio Code

Para desenvolvimento da implementação foi utilizado o Visual Studio Code, que Microsoft ([202?], tradução nossa) define da seguinte forma:

> O Visual Studio Code é um editor de código-fonte leve, mas poderoso, que é executado em sua área de trabalho e está disponível para Windows, macOS e Linux. Ele vem com suporte integrado para JavaScript, TypeScript e Node.js e possui um rico ecossistema de extensões para outras linguagens (como C++, C#, Java, Python, PHP, Go) e ambientes de execução (como .NET e Unity).

Utilizado também por fornecer inúmeros atalhos de teclado, assim aumentando a eficiência e produtividade, também por sempre se manter atualizado, por sua política de ser atualizado sempre automaticamente.

3.2 METODOLOGIA DE DESENVOLVIMENTO

O desenvolvimento deste projeto, foi dividido em sete fases de desenvolvimento apresentadas no Gráfico 1, que serão descritas ao decorrer deste tópico.

Gráfico 1: Fases de desenvolvimento

Fonte: Autoria Própria (2022)

3.2.1 Levantamento de requisitos

Nesta fase, foi realizada uma reunião inicial com o intuito de levantar, toda informação necessária do que o sistema iria precisar para ser construído, informações estas de como até então funcionava o programa Pra Nadar, como eram feitos os cadastros de instrutores e praticantes, de como era a organização de matrículas em turmas entre outras informações.

3.2.2 Desenvolvimento das especificações do sistema

Com base nas informações levantadas, o primeiro passo foi definir quais seriam as ferramentas a serem utilizadas e funções a serem implementadas, diante disto foi elaborado os diagramas de caso de uso, diagrama de classe, indicando quais seriam as funções a serem implementadas, os diagramas podem ser conferidos nas Figuras 2, 3, 4, 5 e 6 presentes na Seção 2.

3.2.3 Desenvolvimento do banco de dados

Com a especificação do sistema montada, foi dado início a construção do banco de dados para ser consumido pela API, a princípio foi elaborado o modelo lógico de base de dados (**apêndice B)** para em seguida ser a base para a construção das *migrations*, que segundo AdonisJS ([2021?], tradução nossa): "... são mutações de banco de dados documentadas, criadas ao longo do ciclo de vida de desenvolvimento do seu aplicativo que você pode reverter ou executar novamente a qualquer momento", é mostrado um exemplo de *migration* para geração da tabela *Student* no Quadro 8.

Quadro 8: *Migration* da tabela students

Migration: StudentSchema

```
1  'use strict'
2
3  /** @type {import('@adonisjs/lucid/src/Schema')} */
4  const Schema = use('Schema');
5
6  class StudentSchema extends Schema {
7    up () {
8      this.create('students', (table) => {
```

```
9         table.inherits('users');
10        table.string('email', 50).notNullable().unique();
11        table.string('cpf', 14).notNullable().unique();
12        table.string('rg', 14).notNullable().unique();
13        table.primary(['id']);
14        table.string('objective_student',
15 25).notNullable();
16        table.boolean('payment_student').notNullable();
17        table.boolean('status').defaultTo(false);
18        table.boolean('defaulter').defaultTo(false);
19        table.enu('type_user',['Administrador',
20 'Secretario', 'Instrutor',
21 'Estudante']).defaultTo('Estudante')
22      })
23    }
24
25    down () {
26      this.drop('students');
27    }
28 }
29
30 module.exports = StudentSchema
31
```

Fonte: Autoria Própria (2022)

No Quadro 8, é apresentado nas linhas 7 a 24, o método *up* que é usado para realizar alguma ação em uma tabela, seja criando ou alterando uma tabela. Ainda neste método é indicado quais são os atributos, com seu tipo, tamanho e características que foram criadas na tabela *students.* Enquanto que das linhas 26 a 28 é mostrado o método *down*, responsável por reverter quaisquer ações feitas no método *up*.

3.2.4 Reuniões de equipe

A quarta fase foi executada iterativamente e praticamente durante todo o decorrer do projeto, nela foi realizada reuniões sempre que necessário entre a equipe, com o intuito de ter uma melhor comunicação entre API, *Mobile* e *Web*, o resultado destas reuniões foram algumas correções de incoerências e elaborações de alguns métodos ligados ao banco de dados, a fim de melhorar a usabilidade do sistema, alguns dos métodos mais importantes foram a criação automática das frequências por plano de aula, o desvinculamento automático por inatividade do

usuário e atualização dos status de inadimplência, estes métodos podem ser visualizados nas Quadros 9, 10 e 11.

Quadro 9: Método *create* do ClassPlanController

Method:	ClassPlanController.create

```
async create({ request, response }) {
        try{
        const data = request.all([]);

        const autofrequency = await
ClassPlan.create(data)

        const listId = await
Database.from('student_relationships')
        .join('students',
'students.id','=','student_relationships.student_id')
        .select('student_relationships.student_id as
student_id')
        .orderBy('name','asc')

.whereIn('students.id',[Database.select('student_id').fr
om('student_relationships')])

.where('classroom_id','=',request.input('classroom_id'))

        listId.map(async (student) => {
            await Database
            .insert({date_frequency:
autofrequency['$attributes'].date,
                    classroom_id:
autofrequency['$attributes'].classroom_id,
                    classplan_id:
autofrequency['$attributes'].id,
                    student_id: student.student_id,
                    created_at:
autofrequency['$attributes'].created_at,
                    updated_at:
autofrequency['$attributes'].updated_at,})
            .into('frequencies')
        });
        return await autofrequency['$attributes'];
        }catch(error){
            return response.status(500).json({message:
error })
        }};
```

Fonte: Autoria Própria (2022)

No Quadro 9, nas linhas 8 a 19 é realizada uma consulta no banco de dados para encontrar a relação de todos os ID dos alunos vinculados a turma específica no momento da criação do plano de aula, com a relação dos ID, é inserido no banco de dados todas as frequências, com os dados sendo indicados na função da linha 21 a 35, a função funciona na incrementação, pegando o 1º ID da relação, inserindo os dados na tabela frequencies e em seguida vai para o 2º ID até passar por todos os ID da relação.

Quadro 10: Método *update* do StudentController

Method: StudentController.update

```
1  async update({ params, request, response }) {
2  try{
3  const student = await Student.findOrFail(params.id);
4
5  const data = request.all([]);
6
7  student.merge(data);
8
9  if ( request.input('status') == false ){
10      response.status(202).json({
11          message: 'Essa ação faz com que este estudante
12 seja desvinculado de todas as turmas'
13     });
14     await
15 Database.from('student_relationships').where('student_id
16 ','=',params.id).delete()
17     return student.save();
18 }
19 else{
20     return student.save();
21 }
22 }catch(error){
23     return response.status(500).json({message: error })
24 }
25 };
```

Fonte: Autoria Própria (2022)

No Quadro 10, sendo responsável pelo desvinculamento automático por inatividade do usuário, nas linhas 9 a 21 temos um bloco de *if*, *else*, onde no *if* se mudarmos o campo *status* para *false* ele vai ser satisfeito e nas linhas 14 a 16 vai ser realizada a busca de todos os vínculos deste estudante específico e apagá las

da tabela student_relationships e no caso do *else*, o método *update* funciona normalmente.

Quadro 11: Método *index* do StudentController

Method: StudentController.index

```
1  async index({ response }){
2  try{
3  const listId = await
4  Database.from('payments').select('student_id').distinct(
5  'student_id');
6
7  listId.map(async (student) => {
8     const verification = await Database.from('students')
9     .join('payments',
10 'payments.student_id','=','students.id')
11    .where('students.id','=',student.student_id)
12    .where('payments.due_date','<',new Date())
13    .where('payments.status_payment','=',false);
14
15    if(verification.length > 0){
16        await Database.from('students')
17        .where('students.id','=',student.student_id)
18        .update('defaulter',true);
19    }else{
20        await Database.from('students')
21        .where('students.id','=',student.student_id)
22        .update('defaulter',false);
23     };
24 });
25 const student = await Student.query().orderBy('name',
26 'asc').fetch();
27
28 return student;
29 }catch(error){
30     return response.status(404).json({message: error })
31 }
32 };
```

Fonte: Autoria Própria (2022)

No Quadro 11, sendo ele responsável pela atualização dos *status* de inadimplência, nas linhas 3 a 5 é feita a consulta no banco, para retornar uma relação de todos os estudantes com pagamentos no sistema, com essa relação é feita a função apresentada nas linhas 7 a 24, na função é realiza uma verificação nas linhas 8 a 13, se existe algum pagamento em atraso, em caso de afirmativa é atualizado o campo *defaulter* para *true* dentro do *if* visto nas linhas 15 a 19,

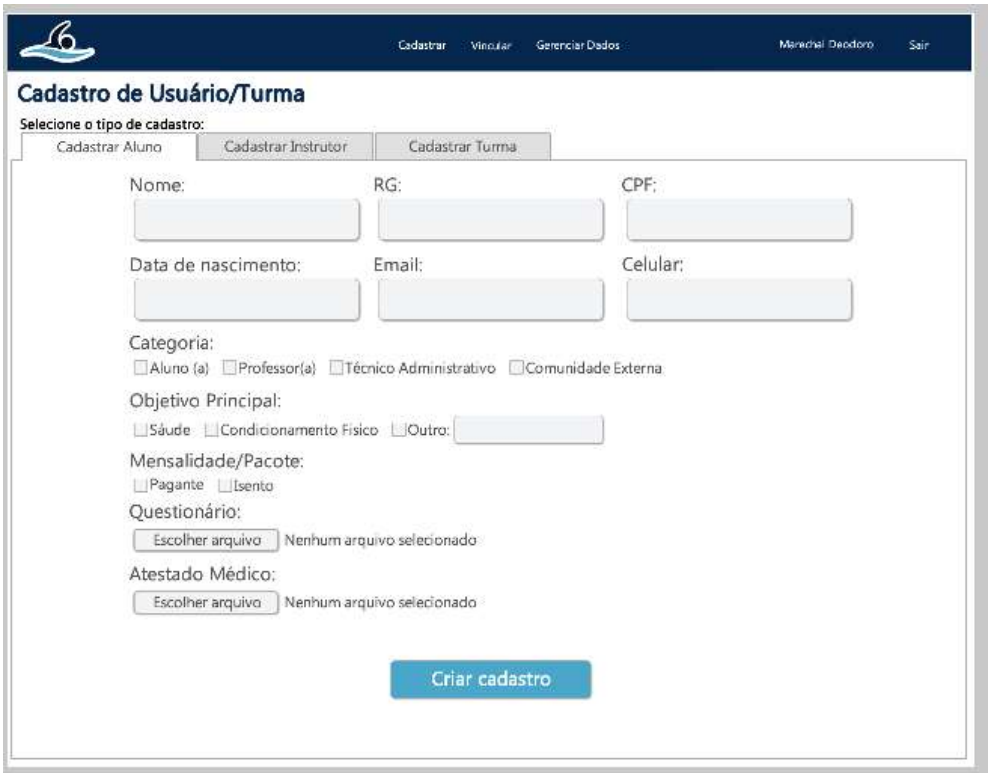
enquanto no *else* é atualizado para *false* quando não se tem nenhum pagamento em atraso.

3.2.5 Prototipagem de tela

Esta fase foi bastante importante para a validação do projeto, pois nela foi feita a criação dos protótipos de telas do *front-end Web* e *Mobile*, apresentados ao cliente para um *feedback* e servir de base a ser seguido para a implementação do *Web* e *Mobile*. As telas apresentadas do *front-end Web* foram, a de cadastro, de alteração, de visualização e de vinculação, exemplos são vistos nesta ordem nas Figuras 7, 8, 9 e 10.

Na Figura 7, é ilustrada a tela apresentada ao cliente para o cadastro dos estudantes no programa, com os campos já definidos na etapa de desenvolvimento das especificações do sistema.

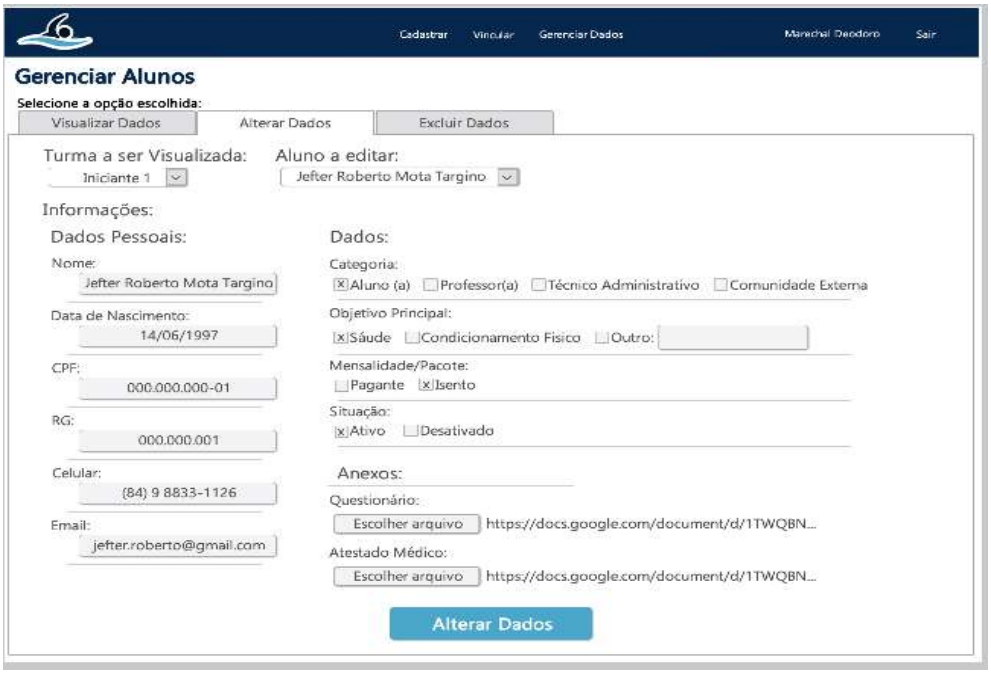
Figura 7: Protótipo da tela do cadastro do estudante



Fonte: Autoria Própria (2022)

É mostrada na Figura 8, a proposta inicial para a tela de alteração dos dados dos estudantes, com a prévia de como ficaria organizada as informações.

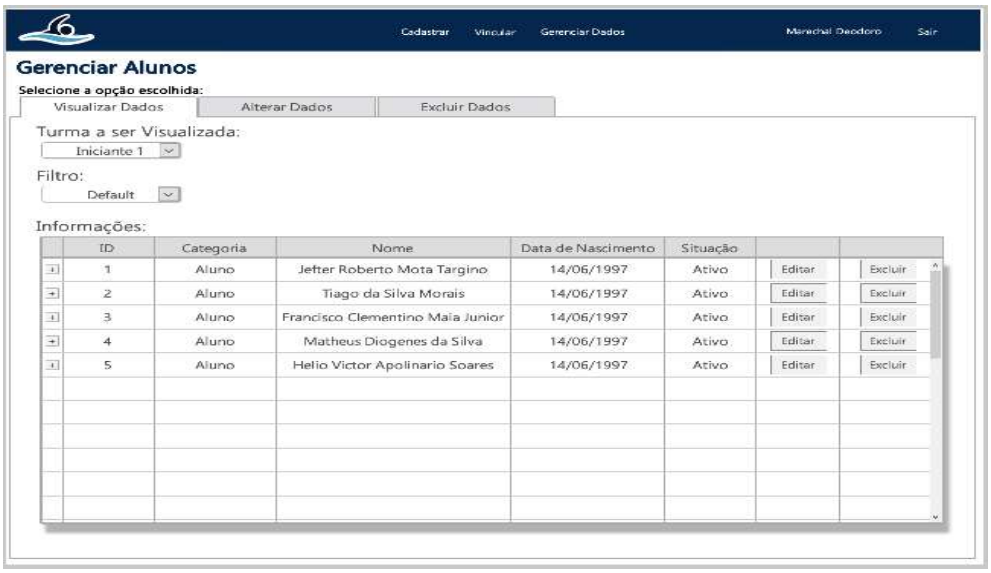
Figura 8: Protótipo da tela do alteração do estudante



Fonte: Autoria Própria (2022)

Pode ser visualizada na Figura 9, o protótipo da tela da tabela com a relação de todos os estudantes cadastrados no sistema, com a pré-visualização de alguns dados gerais do cadastro do estudante.

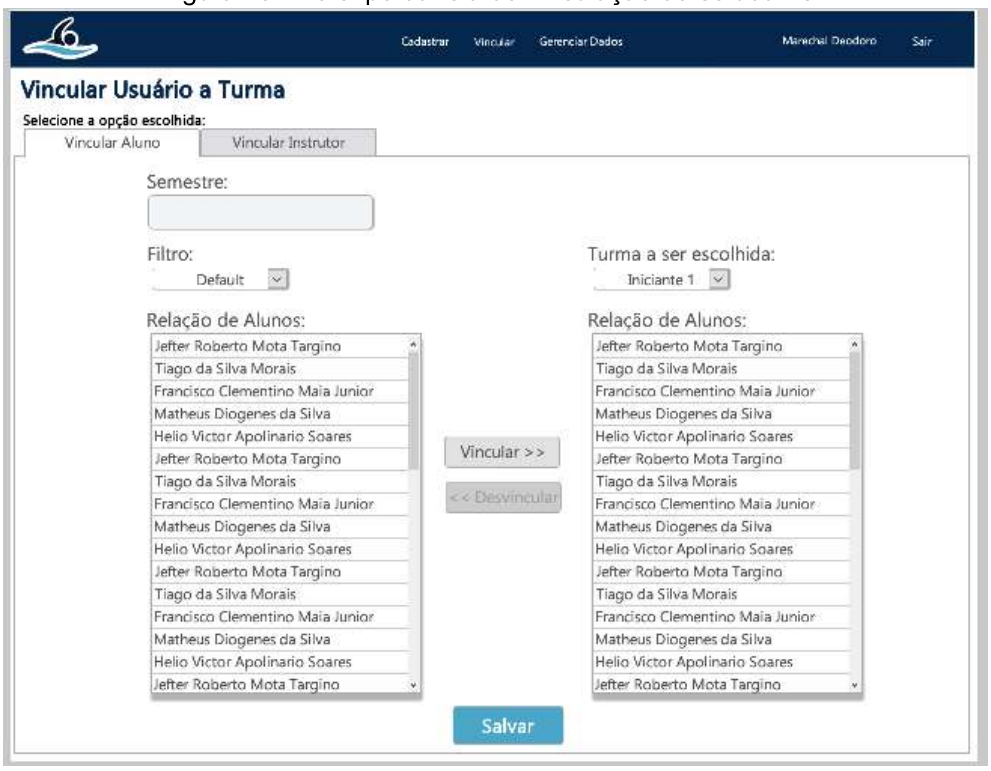
Figura 9: Protótipo da tela do visualização de estudantes



Fonte: Autoria Própria (2022)

É apresentado na Figura 10, o protótipo da tela de vinculação do estudante à turma, sendo uma das telas mais importantes na validação do projeto do sistema.

Figura 10: Protótipo da tela do vinculação do estudante

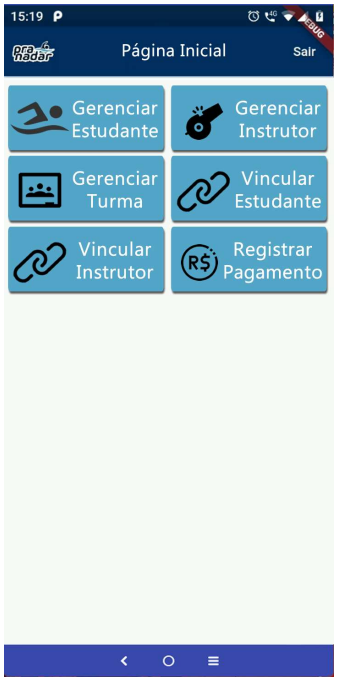


Fonte: Autoria Própria (2022)

Enquanto a respeito do *front-end Mobile*, foram apresentadas também tela inicial, cadastro, de alteração de dados, visualização e de vinculação, exemplos são vistas na seguinte ordem, Figura 11, 12, 13, 14 e 15.

O protótipo da tela inicial do *Mobile* é representado na Figura 11, exibindo as funcionalidades previstas para o mesmo.

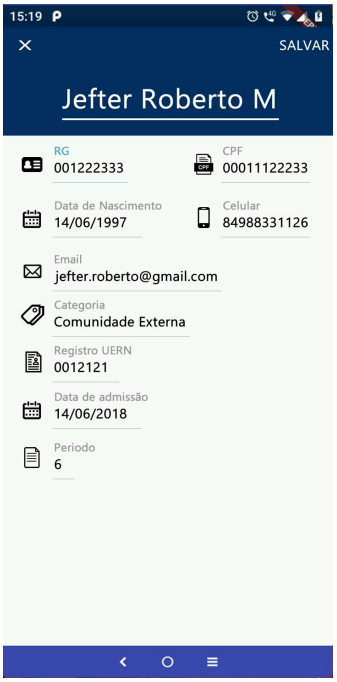
Figura 11: Protótipo da tela inicial do *Mobile*



Fonte: Autoria Própria (2022)

O protótipo da tela para cadastro do usuário tipo instrutor, que pode ser conferida na Figura 12, exibindo a maneira como ficaria organizada as informações e a disposição dos campos.

Figura 12: Protótipo da tela de cadastro de instrutor do *Mobile*



Fonte: Autoria Própria (2022)

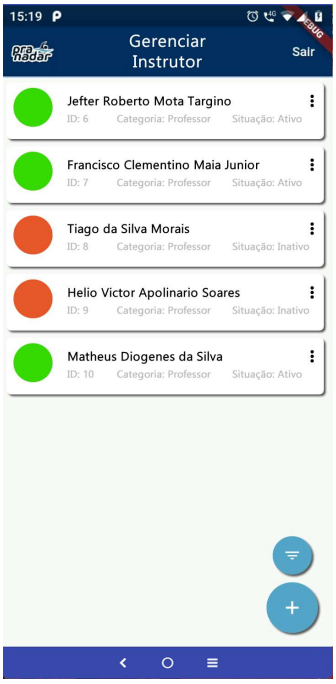
A Figura 13 ilustra o protótipo da tela de alteração de dados do usuário do tipo instrutor, sendo semelhante a tela de cadastro exibida na Figura 12, mas com o acréscimo do botão, para alterar a situação do usuário no sistema, se estaria ativo ou inativo.

Figura 13: Protótipo da tela de alteração de instrutor do *Mobile*

Fonte: Autoria Própria (2022)

O protótipo da tela de visualização geral dos instrutores pode ser conferida na Figura 14, contendo todos os instrutores, seus respectivos dados, a possibilidade de filtrar a busca e o botão para criar outro usuário.

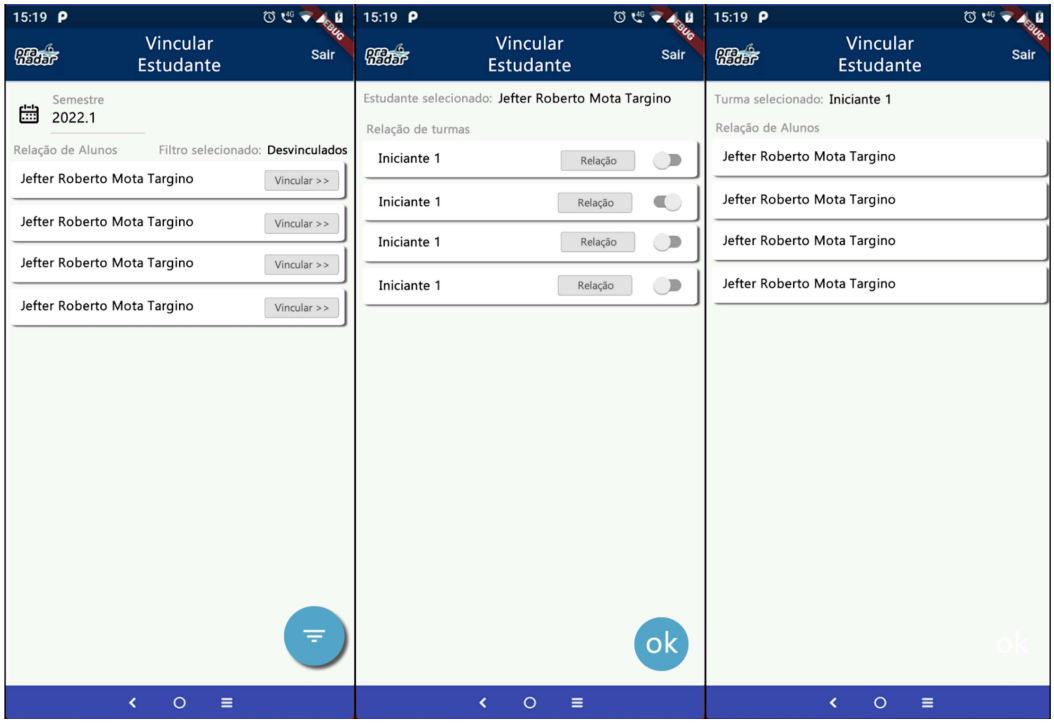
Figura 14: Protótipo da tela de visualização de instrutores do *Mobile*



Fonte: Autoria Própria (2022)

Enquanto na Figura 15, é ilustrado o protótipo do conjunto de telas para vincular os estudantes a uma turma, a primeira para escolher estudante, segunda para escolher a turma, enquanto a terceira é para visualizar os estudantes na turma.

Figura 15: Protótipo das telas de vinculação de estudantes



Fonte: Autoria Própria (2022)

3.2.6 Reuniões de entrega

Nesta fase foram realizadas reuniões semanais, com o orientador e cliente, se utilizando da metodologia ágil, FDD (do inglês, *Feature Driven Development*), que de acordo com Rocha (2013): "… busca o desenvolvimento por funcionalidade, ou seja, por um requisito funcional do sistema". Também se utilizando de alguns princípios, valores e práticas da metodologia XP (do inglês, *Extreme Programming*), por ser uma metodologia que pode ser adotada por diferentes níveis de desenvolvedores e em qualquer tamanho de equipe, também ao fato de se adaptar a requisitos que às vezes podem mudar constantemente.(GUEDES, [2020?]). Nestas reuniões era apresentado o avanço da implementação ao decorrer do desenvolvimento.

3.2.7 Integração

A fase de integração foi responsável pela unificação do sistema, nela com o uso da plataforma Heroku, foi hospedado o banco de dados e API com o intuito de unificar *front-end Web* e *Mobile* em um sistema completo, possibilitando assim uma melhor integração na formação do sistema, viabilizando o cliente a realizar testes e utilizar o sistema como um todo.

4 CONSIDERAÇÕES FINAIS

Com a conclusão deste projeto, ciente da importância de uma boa análise de requisitos, por determinar quais seriam os requisitos fundamentais para elaboração do projeto, mostrando com clareza as necessidades do cliente, quais as melhores formas de desenvolvimentos dentre outros benefícios.

Em consequência disso, foi fornecida a base teórica necessária para implementação da API, *front-end Web* e *Mobile*, também fornecido o protótipo das telas e a entrega de toda estrutura de base de dados, deste modo a permitir ao programa Pra Nadar uma melhor organização de dados e gerenciamento de turmas e aulas ministradas, assim como se conclui que a integração dos sistemas foi satisfatória, entregando um sistema operante e pronto para uso, o sistema pode ser conferido em sua versão *Web* no Heroku[1], na versão Android na Play Store[2] e documentação *swagger* da API disponível no Heroku[3].

Como perspectivas futuras, se tem a possibilidade do reaproveitamento de dados, para atribuir outro cadastro para determinado email no sistema, pode-se citar também a possibilidade de exportação de dados para CSV ou XLS, sugerida pelo cliente, como também a adequação dos dados para adesão da Lei Geral de Proteção de Dados do Brasil (LGPD), a criação de termos de privacidade para os usuários do sistema, como preparação para elaboração de outros ciclos de desenvolvimento, com a descrição de novas funcionalidades e requisitos.

---

[1] O endereço da versão *web* desenvolvida está disponível no *link*: http://pranadar.herokuapp.com/
[2] O endereço da aplicação desenvolvida está disponível no *link*: http://encurtador.com.br/pABQ1
[3] A documentação da API está disponível no *link*: https://pranadarapi.herokuapp.com/docs/

REFERÊNCIAS

ADONISJS. **Migrations.** [S. l.], [2021?]. Disponível em: https://legacy.adonisjs.com/docs/4.1/migrations. Acesso em: 2 abr. 2022.

DEVMEDIA. **Tecnologia PostgreSQL.** [S. l.], [202?]. Disponível em: https://www.devmedia.com.br/guia/tecnologia-postgresql/34328. Acesso em: 29 mar. 2022.

FERNANDES, Diego. **AdonisJS vs ExpressJS: Quando utilizar cada um?.** [S. l.]: Blog da Rocketseat, 18 dez. 2018. Disponível em: https://blog.rocketseat.com.br/adonis-vs-express/. Acesso em: 29 mar. 2022.

GUEDES, Marylene. **O que é XP - Extreme Programming?.** [S. l.], [2020?]. Disponível em: https://www.treinaweb.com.br/blog/o-que-e-xp-extreme-programming. Acesso em: 29 mar. 2022.

HEROKU. **What is Heroku.** [S. l.], [202?]. Disponível em: https://www.heroku.com/what. Acesso em: 29 mar. 2022.

MICROSOFT. **Getting Started.** [S. l.], [202?]. Disponível em: https://code.visualstudio.com/docs. Acesso em: 29 mar. 2022.

NOLETO, Cairo. **API REST: o que é e como montar uma API sem complicação?.** [S. l.]: Blog da trybe, 4 jan. 2022. Disponível em: https://blog.betrybe.com/desenvolvimento-web/api-rest-tudo-sobre/. Acesso em: 31 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.

SOMMERVILLE, Ian. **Engenharia de SOFTWARE.** 9. ed. São Paulo: PEARSON, 2011.

ZUCHER, Vitor. **O que é padrão MVC? Entenda arquitetura de softwares!.** [S. l.], 17 jul. 2020. Disponível em: https://www.lewagon.com/pt-BR/blog/o-que-e-padrao-mvc. Acesso em: 2 abr. 2022.

APÊNDICE A – LISTA DE TODOS OS REQUISITOS FUNCIONAIS

[RF-01] Login do Usuário

[RF-02] Alterar Senha

[RF-03] Resetar Senha

[RF-04] Cadastro de Secretário

[RF-05] Cadastro de Turma

[RF-06] Cadastro de Instrutor

[RF-07] Vincular Usuário

[RF-08] Cadastro de Estudante

[RF-09] Gerenciar Secretários

[RF-10] Gerenciar Turmas

[RF-11] Gerenciar Instrutores

[RF-12] Gerenciar Vínculos

[RF-13] Gerenciar Estudantes

[RF-14] Registro de Plano de Aula

[RF-15] Atribuição de Falta/Frequência

[RF-16] Avaliar Estudantes

[RF-17] Gerenciar Planos de Aulas

[RF-18] Gerenciar Faltas/Frequências

[RF-19] Gerenciar Avaliações

[RF-20] Visualizar Turmas

[RF-21] Visualizar Estudantes

[RF-22] Visualizar Perfil Estudante

APÊNDICE B – MODELO DE DADOS LÓGICO