UNIVERSIDADE ESTADUAL DO MARANHÃO
CENTRO DE CIÊNCIAS TECNOLÓGICAS
CURSO DE ENGENHARIA DE COMPUTAÇÃO

HIGOR GABRIEL RODRIGUES LIMA

PROJETO E DESENVOLVIMENTO DE SISTEMA WEB PARA GESTÃO DE UM CATÁLOGO DE ESPÉCIES DE ANIMAIS DO CURSO DE CIÊNCIAS BIOLÓGICAS DO CECEN/UEMA

São Luís
2021

HIGOR GABRIEL RODRIGUES LIMA

PROJETO E DESENVOLVIMENTO DE SISTEMA WEB PARA GESTÃO DE UM CATÁLOGO DE ESPÉCIES DE ANIMAIS DO CURSO DE CIÊNCIAS BIOLÓGICAS DO CECEN/UEMA

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia da Computação da Universidade Estadual do Maranhão, como requisito para obtenção do grau de Bacharel em Engenharia de Computação.

Orientador: Prof. Me. Pedro Brandão Neto.

São Luís
2021

Lima, Higor Gabriel Rodrigues.
Projeto e desenvolvimento de sistema web para gestão de um catálogo de espécies de animais do curso de ciências biológicas do CECEN/UEMA / Higor Gabriel Rodrigues Lima. – São Luís, 2021.

101 f

Monografia (Graduação) – Curso de Engenharia da Computação, Universidade Estadual do Maranhão, 2021.

Orientador: Prof. Me. Pedro Brandão Neto.

1.Desenvolvimento. 2.Sistema web. 3.Modelagem. I.Título.

CDU: 004.415:57(812.1)

Elaborado por Giselle Frazão Tavares - CRB 13/665

HIGOR GABRIEL RODRIGUES LIMA

PROJETO E DESENVOLVIMENTO DE SISTEMA WEB PARA GESTÃO DE UM CATÁLOGO DE ESPÉCIES DE ANIMAIS DO CURSO DE CIÊNCIAS BIOLÓGICAS DO CECEN/UEMA

Monografia apresentada ao Curso de Engenharia de Computação da Universidade Estadual do Maranhão, como registro para obtenção do grau de Bacharel em Engenharia de Computação.

Trabalho aprovado. São Luís – MA, 9 de Abril de 2021.

Prof. Mestre Pedro Brandão Neto
Orientador

Prof. Doutor Luís Carlos Costa Fonseca
Primeiro Membro

Prof. Doutor Reinaldo de Jesus da Silva
Segundo Membro

AGRADECIMENTOS

Agradeço a Deus em primeiro lugar, por sem Sua ajuda e motivação, não teria chegado onde cheguei.

A minha família, principalmente a minha mãe Sonia, a meu pai José e a minha irmã Gláucia por todos os sacrifícios, esforço, apoio e confiança que fizeram e tiveram para que eu pudesse enfim concluir a graduação.

Aos amigos feitos durante o curso pois sem sua ajuda teria sido muito mais difícil trilhar e suportar essa jornada. Entre esses amigos destaco, Antônio Luís pela essencial ajuda no desenvolvimento do sistema assim como suporte na elaboração dos documentos da modelagem, a Andrelina Martins, Cícero Guilherme, Larissa Ferreira, Lipe, Luan Sousa, Lucas Raphael, Philipe Manoel e Ricardo Gonçalves, agradeço ao auxilio nos estudos, na realização dos trabalhos acadêmicos, nas caronas, aos lanches dentre vários outros aspetos que foram cruciais nesse período. Aos demais amigos que não foram citados, saibam que vocês também foram importantes neste processo e por isso deixo meu mais sincero obrigado.

Agradeço ao meu orientador Pedro pelo suporte e orientação prestados na elaboração deste projeto.

Aos professores e alunos do projeto CoFauMa pela colaboração no desenvolvimento do sistema, assim como seu aperfeiçoamento ao longo da fase de modelagem.

"Não existe triunfo sem perda, não há vitória sem sofrimento, não há liberdade sem sacrifício."

Aragorn - O Senhor dos Anéis – O Retorno do Rei (2003)

RESUMO

Este trabalho tem como objetivo apresentar a modelagem e o desenvolvimento de um sistema web para gestão de um catálogo de espécies do Curso de Ciências Biológicas da Universidade Estadual do Maranhão. A partir do levantamento de requisitos e análise das regras de negócio implementadas até o presente momento, foi observado que o modelo atual de construção do catálogo apresenta uma série de características que tornam todo o processo de catalogação das espécies demorado e complexo, nesse quesito, foram utilizados os conhecimentos de Engenharia de Software, Modelagem de Sistemas, Banco de Dados Relacional, Linguagem de Modelagem Unificada dentre outros em conjunto com os benefícios da utilização do framework Laravel para desenvolver a modelagem e o desenvolvimento de um sistema web que faça o melhor uso possível das tecnologias atuais para solucionar essas questões e no final automatizar todo o processo de coleta, armazenagem e disponibilização das informações obtidas através das amostras analisadas.

Palavras-Chave: Desenvolvimento, Sistema Web, Modelagem.

ABSTRACT

This work aims to present the modeling and development of a web system for managing a catalog of species in the Biological Sciences Course at the State University of Maranhão. From the requirements survey and analysis of the business rules implemented up to the present moment, it was observed that the current model of construction of the catalog has a series of characteristics that make the whole process of cataloging species time-consuming and complex. Knowledge of Software Engineering, Systems Modeling, Relational Database, Unified Modeling Language among others, together with the benefits of using the Laravel framework to develop the modeling and development of a web system that makes the best possible use of current technologies to solve these issues and, in the end, automate the entire process of collecting, storing and making available the information obtained through the analyzed samples.

***Keywords**: Development, Web System, Modeling.*

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface (Interface de Programação de Aplicativos)
CoFauMa	Coleção de Tecidos e DNA da Fauna Maranhense
CLI	Comand Line Interface (Interface de Linha de Comando)
CSS	Cascading Style Sheet (Folhas de Estilos em Cascata)
DNA	Ácido Desoxirribonucleico
HTML	HyperText Markup Language (Linguagem de Marcação de HiperTexto)
MVC	Model-View-Controller
PDF	Portable Document Format (Formato de documento portátil)
PHP	Hypertext Preprocessor
SGBD	Sistema Gerenciador de Banco de Dados
TI	Tecnologia da Informação
UML	Unified Modeling Language (Linguagem de Modelagem Unificada)

SUMÁRIO

1 INTRODUÇÃO

Os avanços constantes nas áreas da computação e eletrônica proporcionaram o nascimento de um ambiente em constante evolução para o campo da Tecnologia da Informação (TI) na forma do aumento da capacidade de processamento dos hardwares dos computadores. Desse aspecto surgiu a necessidade de empregar um meio que aproveitasse essa capacidade computacional elevada, sendo esta apresentada no uso de softwares.

Os softwares permitiram uma maior gama de aplicação para o uso dos computadores em um aspecto mais geral, pois eles podiam se ramificar a partir dos requisitos de determinado problema e assim aproveitar todo o potencial do hardware disponível. A partir dessa adequação, o papel do software foi amplamente amplificado na sociedade atual, seja na otimização de processos, melhor aproveitamento do tempo e/ou diminuição de gastos.

Dito isto, pode-se citar uma área específica onde o uso de softwares foi de suma importância, a gestão de arquivos. O uso de softwares nessa área modernizou o conceito de armazenamento e disponibilidade dos dados, isto é, proporcionou uma maneira mais eficaz para garantir a segurança, confiabilidade e autenticidade das informações, assim como garantiu a facilidade de acesso e a disponibilização dessas informações.

Tendo em vista esses fatores, observa-se o estudo da situação que ocasionou a problemática deste trabalho: o Curso de Ciências Biológicas da Universidade Estadual do Maranhão desenvolveu um projeto cujo objetivo era construir um catálogo de espécies da fauna brasileira: O CoFauMa (Coleção de Tecidos e DNA da Fauna Maranhense), onde o intuito era construir um acervo com esses dados e o disponibilizar para outras instituições de ensino, dessa maneira proporcionando um ambiente de aprendizagem por meio da troca de informações.

Portanto, a proposta neste trabalho pretende levar em conta os benefícios que o software proporciona e assim migrar o banco de dados físico para um meio digital, de tal forma que este obtenha uma melhor gestão e facilite a disposição do mesmo, além de elaborar uma interface gráfica que ofereça uma plataforma confiável e de fácil utilização para os membros do projeto concluir suas atividades. Nesse aspecto vale ressaltar que o sistema web proposto tem como o foco de realizar e desempenhar a função de “ponte” entre as informações armazenadas e facilitar o acesso a essas informações para os usuários por meio da interface.

1.1 Problema a Ser Resolvido

O CoFauMa em seus aspectos mais simplificados se resume na coleta de informações pertinentes a uma dada espécie para assim gerar uma ficha de identificação da mesma, onde essa ficha é armazenada em cadernos escritos a mão. Tendo esse panorama, no

decorrer do tempo em que o projeto foi estabelecido, surgiram várias situações onde haviam falhas na maneira em que as informações foram armazenadas, por exemplo, fichas repetidas e/ou incompletas e perda de alguns dados por falta de uma melhor gestão do banco de dados, além do que o acesso a essas informações é muito inconveniente, já que exige o fator presencial no local em que fica armazenado.

Nesse cenário este trabalho apresenta um sistema web que irá proporcionar uma solução para as dificuldades em que se encontra o projeto, desde melhorar a gestão dos arquivos a desenvolver uma interface que facilite o acesso a esses dados por meio da internet.

1.2 Justificativa

Atualmente, conforme a sociedade atual avança cada vez mais tecnologicamente, o ganho de tempo, desempenho e acessibilidade, se tornaram os principais aspectos que fundamenta o emprego de softwares, pois em um mundo cada vez mais globalizado é de suma importância que se garanta a segurança daquilo que é disponibilizado na rede de computadores, já que qualquer informação incorreta pode acarretar em sérias repercussões, principalmente se essas mesmas informações estiverem ligadas ao âmbito educacional, por exemplo, objetos de estudos de cursos de graduação.

Outro ponto importante, é a elaboração de uma boa gestão de arquivos, pois observando o objeto de estudo deste trabalho, como se trata de milhares de registros no acervo, isto é, inserções de fichas no banco de dados, é preciso estabelecer normas e procedimentos a serem seguidos para que conforme ocorra a alimentação de dados, os mesmos atendam a um conjunto de regrais iguais, de tal maneira que se possa identificar uma determinada ficha em meio à muitas.

Levando em conta os muitos problemas que o projeto sofre e comparando com os pontos mencionados nos parágrafos anteriores, se faz necessário rever todo o “esqueleto” em que o projeto se sustenta, ou seja, rever os procedimentos de garantia e disponibilidade das informações prestadas.

Portanto, a implementação de um sistema web seguindo as diretrizes estabelecidas na engenharia de software sanaria esses muitos aspectos que impossibilitam o projeto avançar tecnologicamente, além de que usufruiria dos muitos benefícios que uso de um software traz, assegurando todo o aporte para futuras implementações tanto na melhoria da apresentação como na possível implementação desses dados em outras áreas de interesse.

1.3 Objetivos

1.3.1 Objetivo Geral

Observando o estado em que se encontra o processo de coleta e armazenamento de dados do projeto CoFauMa, realizado pelo Curso de Ciências Biológicas da Universidade Estadual do Maranhão, este trabalho tem como objetivo principal desenvolver e implementar um sistema web que facilite o processo de catalogação de espécies assim como sua disponibilização, depois de tratado os dados, para alunos de outras instituições.

1.3.2 Objetivos Específicos

O trabalho proposto tem os seguintes objetivos específicos:

- Modelar os requisitos do sistema;
- Construir os diagramas correspondentes a modelagem;
- Definição das entidade-relacionamento;
- Otimizar o processo de cadastro das informações do catálogo de espécies;
- Implementar níveis de acesso do usuário em relação às funções que o sistema desempenhará;
- Garantir a confidencialidade, integridade e disponibilidade das informações dispostas;
- Garantir que o sistema seja o mais didático possível, onde todas as funções que o sistema terá serão de fácil utilização por parte de seus respectivos usuários.

1.4 Metodologia

Este trabalho de conclusão de curso se fundamenta nas dificuldades encontradas na construção e manutenção do catálogo de espécies do CoFauMa, mais especificamente na gestão dos dados, na conversão destes em informações utilizáveis e a sua posterior divulgação.

Observando esse panorama, será feita primeiramente um levantamento de requisitos funcionais e não funcionais através da análise do atual modelo de implementação do projeto CoFauMa com o intuito de estabelecer as regras de negócio que deve ser implementada no sistema. Posteriormente será realizada a modelagem do sistema onde será empregada a Linguagem de Modelagem Unificada – UML para construção dos diagramas e das tabelas correspondentes ao modelo de desenvolvimento que o sistema deve seguir no decorrer de sua implementação.

Em seguida, com o ambiente estabelecido através da modelagem, será realizado o desenvolvimento do sistema. Utilizando os meios fornecidos através dos frameworks Laravel e Bootstrap o sistema será composto de uma única aplicação, isto é, tanto o *back-end* quanto o *front-end* serão integrados em uma única aplicação sem o uso de uma API, de tal forma que possa lidar com os aspectos de hardware do ambiente em que será empregado.

1.5 Estrutura do trabalho

Este trabalho será estruturado da seguinte maneira: No capítulo 2 será apresentada a fundamentação teórica contendo as informações das tecnologias e conceitos utilizados para o desenvolvimento do sistema. O capítulo 3 apresenta a proposta do sistema assim como a sua modelagem. O capítulo 4 irá apresentar o desenvolvimento e a implementação do sistema como um todo, e por último no capítulo 5 serão apresentados as conclusões e objetivos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Conceitos de Engenharia de Software

Os softwares constituem uma parte integrante da sociedade atual, a esse aspecto Sommerville (2011, p.2) comenta que:

> O mundo moderno não poderia existir sem o software. Infraestruturas e serviços nacionais são controlados por sistemas computacionais, e a maioria dos produtos elétricos inclui um computador e um software que o controla. A manufatura e a distribuição industriais são totalmente informatizadas, assim como o sistema financeiro. A área de entretenimento, incluindo a indústria da música, jogos de computador, cinema e televisão, faz uso intensivo de software. Portanto, a engenharia de software é essencial para o funcionamento de sociedades nacionais e internacionais.

Nesse aspecto, conforme a sociedade se tornou cada vez mais centrada na obtenção de soluções em um curto prazo de tempo o papel do software a cada dia foi tomando o lugar de uma peça fundamental nesse processo, pois segundo Pressman (2011, p.31):

> O software distribui o produto mais importante de nossa era — a informação. Ele transforma dados pessoais (por exemplo, transações financeiras de um indivíduo) de modo que possam ser mais úteis num determinado contexto; gerencia informações comerciais para aumentar a competitividade; fornece um portal para redes mundiais de informação (Internet) e os meios para obter informações sob todas as suas formas.

Utilizando o contexto elaborado a partir das observações acima, pode-se notar a importância que o software tem na sociedade como um todo, seguindo esse aspecto relacionado à engenharia de software ela desenvolveu um conjunto de métodos a fim de construir um ambiente onde se pudesse alcançar certa uniformidade no processo de utilização de um software, sobre os métodos empregados pela engenharia de software Pressman (2011, p.40) diz que: “Os métodos da engenharia de software fornecem as informações técnicas para desenvolver softwares. Os métodos envolvem uma ampla gama de tarefas, que incluem: comunicação, análise de requisitos, modelagem de projeto, construção de programa, testes e suporte.”.

Sendo assim, o trabalho proposto seguirá as normas elaboradas pela engenharia de software com o intuito de atender aos seus padrões de projeto afim facilitar o uso da documentação que será demonstrada no decorrer deste trabalho.

2.2 Framework

O desenvolvimento de sistemas envolve inúmeras etapas e demanda vários processos que devem ser levados em consideração no momento de elaboração e construção, sendo assim um desses principais pontos que se deve ressaltar é tempo de codificação de um software. Por causa de sua complexidade mesmo um software de pequena escala, demanda certa quantia de tempo para ser processado e codificado, nesse intuito pode-se abordar o uso de frameworks.

Os frameworks podem ser entendidos como um ambiente de programação voltado para a gestão e criação de sistemas, em outras palavras, os frameworks funcionam como uma plataforma onde o desenvolvedor dispõem de ferramentas que o irão auxiliar na programação, na forma de reutilização de códigos embutidos, utilização de bibliotecas, conexão com o banco de dados entre outras. Vale ressaltar que ao optar por dado framework o desenvolvedor terá que seguir, com alguma margem de manobra, a arquitetura constituinte do mesmo, pois é ela que irá governar o processo de construção e execução do sistema.

A importância dos frameworks só cresceu ao longo dos últimos anos e conforme seu avanço possibilitou inúmeras facilidades para os desenvolvedores, por exemplo, certos frameworks já possuem diversas funções já implementadas e prontas para serem usadas de imediato nas aplicações.

Observando esses pontos, e levando em consideração as especificações do servidor ao qual irá ser alocado o sistema, no caso o servidor da própria instituição, foi empregado o Laravel, um framework de desenvolvimento que utiliza a linguagem de programação PHP e o Bootstrap um framework de desenvolvimento que possui várias ferramentas gráficas para a construção de interfaces.

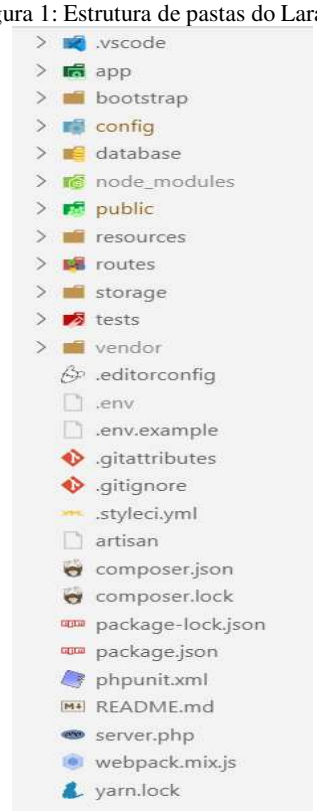
2.2.1 Laravel e Bootstrap

O Laravel é um framework de código aberto baseado na linguagem de programação PHP (um acrônimo recursivo para PHP: Hypertext Preprocessor), tendo como suas principais características, a facilidade na construção dos códigos enxutos e limpos, ou seja, usando o mínimo de código possível para implementar uma dada função. Este framework consegue realizar tal tarefa, pois o Laravel utiliza a arquitetura MVC (Model-View-Controller), a ideia por trás desse modelo é separar as regras de negócio da aplicação em três camadas distintas, cada uma responsável por uma determinada parte, segundo Turini (2015, p.20), essas três camadas seriam definidas como:

> **Model** é a camada onde ficam nossas regras de negócio, nossas identidades e classes de acesso ao banco de dados.
>
> **View** é a responsável por apresentar as páginas e outros tipos de resultado para o usuário (ou mesmo para outros sistemas, que se comunicam). É a resposta que o framework envia para o navegador, que normalmente é um HTML.
>
> **Controller** é quem cuida de receber as requisições web e decidir o que fazer com elas. Nessa camada definimos quais **model**os devem ser executados para determinada ação e para qual **view** vamos encaminhar a resposta. Em outras palavras, essa camada quem faz o link entre todas as outras. (grifo do autor).

O Laravel não é o único framework que utiliza esse modelo de arquitetura, nesse aspecto pode-se citar o CodeIgniter, Symfony, Zend, mas o que distingue e o levou a ser escolhido em relação aos demais foram dois aspectos: o primeiro é que o Laravel atualmente é um dos frameworks mais utilizados para a criação de aplicações web no mundo e o segundo leva em consideração a sua estrutura de pastas (Figura 1) que fornecem um ambiente de suporte para a arquitetura em si. Um dos pontos que vale ressaltar é que o sistema utiliza a versão 6 do Laravel, pois como o framework se encontra em constante evolução, sua estrutura de pastas sofre diversas modificações de uma versão para a outra, algumas ocasionando depreciação de funções e/ou mudança da usabilidade das mesmas.

Figura 1: Estrutura de pastas do Laravel



Fonte: Autor

Na Figura pode-se observar a estrutura de pastas do Laravel, sendo os principais:

- **app**: esta pasta contém os arquivos utilizados para construção da parte lógica da aplicação que será desenvolvida;
- **bootstrap**: esta pasta contém os arquivos necessários para inicialização do framework;
- **config**: esta pasta contém os arquivos de configuração do framework;
- **database**: esta pasta contém os arquivos de criação e gestão do banco de Dados;
- **public**: esta pasta contém a raiz da aplicação;
- **resources**: esta pasta contém os arquivos utilizados na criação da interface gráfica da aplicação;
- **routes**: esta pasta contém os arquivos utilizados na criação das rotas da aplicação;
- **vendor**: esta pasta contém os pacotes de autoload do composer assim como arquivos de inicialização de templates utilizados de outras fontes;
- **.env**: arquivo onde ficam armazenadas as variáveis de ambiente do sistema;
- **artisan**: arquivo responsável pela interface CLI (Comand Line Interface), nesta interface pode-se rodar diversos comandos cuja finalidade é automatizar a criação de alguns componentes da aplicação como models, migrations e controllers;
- **composer.json**: arquivo responsável pelo armazenamento das dependências externas do projeto;
- **package.json**: arquivos responsável pelo armazenamento das dependências externas dos pacotes Javascript do projeto;
- **webpack.mix.js**: arquivo responsável pelo armazenamento das configurações do webpack;

Como pode ser visto na própria estrutura de pastas, o Laravel está intimamente ligado com o Bootstrap, um framework que pode ser definido como um kit de ferramentas muito utilizado para aplicações web, sendo de código livre e é desenvolvido empregando diversos tipos de tecnologia como HTML, CSS e JavaScript. Neste sistema ele foi empregado como auxiliar para a criação da interface gráfica da aplicação.

2.3 Modelagem do Sistema e Banco de Dados

2.3.1 A Linguagem de Modelagem Unificada - UML

Segundo Eduardo Bezerra (2007, p.15).

> A UML é uma *linguagem visual* para modelar sistemas orientados a objetos. Isso quer dizer que a UML é uma linguagem que define elementos gráficos (visuais) que podem ser utilizados na modelagem de sistemas. Esses elementos permitem representar os conceitos do paradigma da orientação a objetos. Através dos elementos gráficos definidos nesta linguagem pode-se construir diagramas que representam diversas perspectivas de um sistema. (grifo do autor).

Portanto, o sistema proposto será modelado com bases nas especificações encontradas na linguagem UML, de tal forma que atenda as normas a serem seguidas no desenvolvimento de um sistema, de acordo a Engenharia de Software.

2.3.2 Banco de Dados e Sistemas Gerenciador de Banco de Dados

Primeiramente é necessário uma definição para um banco de dados e para isso podemos utilizar a visão de (DATE, 2004) quando ele diz: “Um banco de dados é uma coleção de dados persistentes, usada pelos sistemas de aplicação de uma determinada empresa”. Partindo desse princípio pode observar que o Banco de Dados é uma estrutura simples mais ao mesmo tempo complexa no sentido em que oferece diversas metodologias estruturais para se adequar as mais diversas aplicações. Nesse âmbito para lidar com tamanha estrutura surgiram os Sistemas Gerenciadores de Banco de Dados, que diferente um sistema voltado mais para o usuário, este por sua vez é mais relacionado com o próprio banco, fato esse observado quando Elmasri e Navathe (2005, p.4) diz:

> Um sistema gerenciador de banco de dados (SGBD) é uma coleção de programas que permite aos usuários criar e manter um banco de dados. O SGBD é, portanto, um sistema de software de propósito geral que facilita os processos de definição, construção, manipulação e compartilhamento de bancos de dados entre vários usuários e aplicações.

Existem diversos Banco de Dados no mercado como o Oracle Database, MySQL, Midrosoft SQL Server, Postgres dentre outros. Foi escolhido o Postgres como Banco de Dados a ser utilizado pelo sistema em decorrência do servidor utilizado para alocar a aplicação, pois como este é o próprio servidor da instituição e como ele executa o Postgres como Banco de Dados, a fim de evitar incompatibilidades entre aplicação e o servidor o mesmo foi optado, na figura 2 pode-se observar o SGBD do Postgres: o PgAdmin.

Figura 2: Interface do SGBD PgAdmin

Fonte: Autor

2.3.3 Modelo Entidade-Relacionamento

O Modelo Entidade Relacionamento (também chamado Modelo ER, ou simplesmente MER), é um modelo conceitual utilizado na Engenharia de Software proposto por Peter Chen em 1976, para descrever como os objetos (entidades) de um domínio, com suas próprias características (atributos) se relacionam (relacionamentos) (RODRIGUES, 2014).

Sobre entidades e atributos Elmasri e Navathe (2005, p.39) comentam que:

> Entidades e Seus Atributos. O objeto básico que o modelo ER representa é uma entidade, 'algo' do mundo real, com uma existência independente. Uma entidade pode ser um objeto com uma existência física (por exemplo, uma pessoa, um carro, uma casa ou um funcionário) ou um objeto com uma existência conceitual (por exemplo, uma empresa, um trabalho ou um curso universitário).
>
> Cada entidade tem atributos — propriedades particulares que a descrevem. Por exemplo, uma entidade empregado pode ser descrita pelo nome do empregado, idade, endereço, salário e trabalho (função). Uma dada entidade terá um valor para cada um de seus atributos. Os valores dos atributos que descrevem cada entidade se tornarão a maior parte dos dados armazenados no banco de dados.

Ainda sobre as entidades podemos classificá-las como:

- **Entidades fortes**: não dependem de outras entidades para existirem;
- **Entidades fracas**: dependem de outras entidades pra existirem;
- **Entidades associativas**: é utilizada quando surge a necessidade de ligar uma entidade a relacionamento existente.

Os relacionamentos, como o próprio nome se refere, ocorre quando uma entidade se relaciona com outra entidade, conforme os números de objetos envolvidos no relacionamento, estes podem classificados como:

> **Relacionamento 1..1 (um para um):** cada uma das duas entidades envolvidas referenciam obrigatoriamente apenas uma unidade da outra. Por exemplo, em um banco de dados de currículos, cada usuário cadastrado pode possuir apenas um currículo na base, ao mesmo tempo em que cada currículo só pertence a um único usuário cadastrado.
>
> **Relacionamento 1..n ou 1..* (um para muitos):** uma das entidades envolvidas pode referenciar várias unidades da outra, porém, do outro lado cada uma das várias unidades referenciadas só pode estar ligada uma unidade da outra entidade. Por exemplo, em um sistema de plano de saúde, um usuário pode ter vários dependentes, mas cada dependente só pode estar ligado a um usuário principal. Note que temos apenas duas entidades envolvidas: usuário e dependente. O que muda é a quantidade de unidades/exemplares envolvidas de cada lado.
>
> **Relacionamento n..n ou *..* (muitos para muitos):** neste tipo de relacionamento cada entidade, de ambos os lados, podem referenciar múltiplas unidades da outra. Por exemplo, em um sistema de biblioteca, um título pode ser escrito por vários autores, ao mesmo tempo em que um autor pode escrever vários títulos. Assim, um objeto do tipo autor pode referenciar múltiplos objetos do tipo título, e vice versa (RODRIGUES, 2014). (grifo do autor)

O diagrama referente ao modelo ER será apresentado no decorrer do trabalho.

2.4 Conceitos de Biologia

2.4.1 Classificação dos seres vivos

A taxonomia é o ramo da biologia responsável pela nomenclatura e classificação dos seres vivos, dessa forma utilizando o sistema proposto Carl von Linné (1707-1778) ou Lineu (em português) foi possível elaborar um sistema de classificação das espécies, com a ressalva de que este com o passar do tempo sofreu algumas modificações para melhor classificar as espécies tendo em vista sua ampla gama de diversidade.

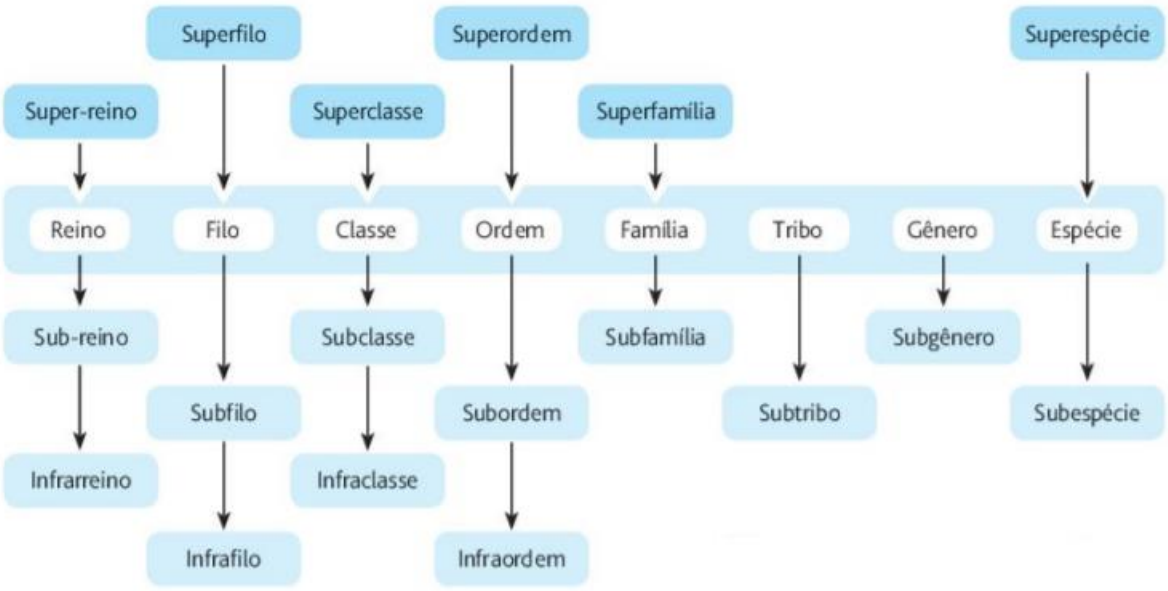
Sobre o sistema elaborado por Lineu, Lopes (2006, p.180) diz:

> No sistema de Lineu a unidade básica de classificação é a **espécie**. Espécies semelhantes são agrupadas em um mesmo **gênero**. Gêneros semelhantes são agrupados em uma mesma **família**. Famílias são agrupadas em **ordens**, que são agrupadas em **classes**, que são agrupadas em **filos** ou **divisões**, que são agrupados em **reinos**. (grifo do autor)

A respeito das modificações sofridas pelo sistema de Lineu em decorrência do tempo Lopes (2006, p.181) diz "Além dessas, muitas vezes utilizam-se categorias intermediárias e não-obrigatórias, como **subfilo**, **infraclasse**, **superordem**, **superfamília**, **subfamília** e **subgênero**." (grifo do autor).

Na imagem abaixo foi apresentado o sistema de classificação das espécies.

Figura 3: Esquema de classificação das espécies



Fonte: Amabis e Martho (2013), p.14.

2.4.2 Regras de Nomenclatura

Sobre estas regras, Linhares e Gewandsznajder (2013, p.15) comentam que:

> Todos os nomes científicos devem ser escritos em latim; se foram derivados de outra língua, deverão ser latinizados. A justificativa é que o latim não sofre modificações ao longo do tempo e, além disso, a nomenclatura passa a não privilegiar nenhuma nação atual.
>
> Os termos que indicam gênero até reino devem ter inicial maiúscula; o gênero deve ser escrito em *itálico*, quando em texto impresso, ou sublinhado, quando escrito à mão.
>
> O nome das espécies é duplo (binomial) e escrito em *itálico*, quando em texto impresso, ou sublinhado, quando escrito à mão: *Homo sapiens* (ser humano), *Felis catus* (gato doméstico), *Musca domestica* (mosca). A primeira palavra indica o gênero, e a segunda, o termo específico (ou epíteto específico) escrito com inicial minúscula. Em um texto, a partir da segunda ocorrência, o nome da espécie pode ser abreviado. Por exemplo, neste texto, uma nova ocorrência do nome *Musca domestica* poderia aparecer assim: *M. domestica*.
>
> A nomenclatura de uma subespécie (populações da mesma espécie geograficamente isoladas, que podem, no futuro, formar novas espécies) é trinominal (ou trinomial): *Crotalus durissus terrificus* (cascavel encontrada na Amazônia), *Crotalus durissus durissus* (cascavel encontrada na Guiana). (grifo do autor)

Este conjunto de regras foi convencionado para que a nomenclatura de uma espécie fosse uniforme para todos independente da região do planeta.

3 PROPOSTA DO DESENVOLVIMENTO DO SISTEMA WEB

3.1 Proposta de Sistema (Visão Geral)

O sistema proposto neste trabalho de conclusão de curso foi elaborado com o intuito de dar suporte ao Projeto CoFauMa (Coleção de Tecidos e DNA da Fauna Maranhense), projeto este criado e mantido por professores e alunos do Curso de Ciências Biológicas da Universidade Estadual do Maranhão.

O suporte será realizado na forma da construção de banco de dados digital em um servidor online e disponibilizá-lo por meio de uma interface gráfica utilizando a internet como meio de acesso. Além de prover uma estrutura que possa realizar as seguintes atribuições:

- Cadastrar os dados relevantes de uma espécie coletada e eventualmente construir um catálogo contendo as informações obtidas do estudo do estudo dos dados;
- Identificar o local e o coletor;
- Gerar uma identificação própria para aquela espécie, tendo em vista a diferenciação entre instituições que posteriormente pudessem vir a integrar o projeto;
- Elaborar um processo de verificação dos dados coletados através da sua validação por um profissional capacitado da área;
- Emitir um relatório contendo as informações de qualquer espécie do catálogo.

O sistema terá três tipos de usuários: master, administrador e comum, onde o master e o administrador desempenham todas as funções do sistema, a diferença entre eles é que o usuário master é uma conta inicial quando o sistema entra em produção, ambos só podem ser atribuídos a professores responsáveis pelo projeto e o usuário comum detém apenas algumas funcionalidades, sendo este representado por alunos participantes do projeto.

A intenção com essa proposta de sistema é construir um ambiente que possibilite uma maior qualidade na gestão das atividades do projeto, isto é, prover com os recursos que a tecnologia tem ao alcance um ambiente informatizado que possa gerir todo o processo de elaboração dos objetivos que o projeto CoFauMa pretende alcançar.

3.2 Modelagem de Sistema

3.2.1 Requisitos Funcionais

Os requisitos funcionais correspondem às funcionalidades do sistema, ou seja, as funções que o cliente quer que o produto tenha ao final do processo de desenvolvimento.

Abaixo a lista dos requisitos funcionais que o sistema proposto deve atender.

- **RF1.** O sistema deve conter níveis de acesso quanto à disponibilidade de suas funções;
- **RF2.** O sistema deve ser capaz de registrar novas contas de usuário, podendo alterá-las e posteriormente deletá las a partir de uma conta de administrador;
- **RF3.** O sistema deve ser capaz de registrar, alterar, excluir, validar e restaurar um formulário de uma espécie;
- **RF4.** O sistema deve permitir a uma conta de administrador trocar o nível de acesso às funções do sistema de uma outra conta;
- **RF5.** O sistema deve listar todas as espécies não validadas, validadas, excluídas, cadastradas e deletadas, além de posteriormente exibi-las para os usuários de acordo com seu nível de acesso;
- **RF6.** O sistema deve ser capaz de gerar um PDF contendo as informações dos dados de uma espécie cadastrada, assim como os logs de validação da mesma;
- **RF7.** O sistema deve ser capaz de listar todos os usuários cadastrados, assim como o seu nível de acesso;
- **RF8.** O sistema deve conter uma restrição quanto a quem pode realizar a validação de uma espécie para a confirmação das informações a serem disponibilizadas aos demais usuários.

3.2.2 Requisitos Não-Funcionais

Segundo Guedes (2009, p.22), ”[...] os requisitos não-funcionais correspondem às restrições, condições, consistências, validações que devem ser levadas a efeito sobre os requisitos funcionais [...]”, utilizando essa visão, os requisitos não-funcionais do sistemas são:

- **RNF1.** O sistema deve utilizar o PostgreSQL com banco de dados;
- **RFN2.** O sistema deve ser portável, podendo ser acessado de qualquer lugar com acesso à internet e com o hardware mínimo necessário;
- **RFN3**. O sistema deve garantir a segurança e confiabilidade dos dados, desde o processo da coleta e inserção dos mesmos no banco de dados até a consulta por parte dos demais usuários;
- **RFN4.** O sistema deve conter uma interface de fácil usabilidade, ou seja, o usuário deve poder navegar entre as funções do sistema de maneira rápida e prática de forma simples;
- **RFN5.** O sistema precisa estar disponível e online em pelo menos 99% do tempo.

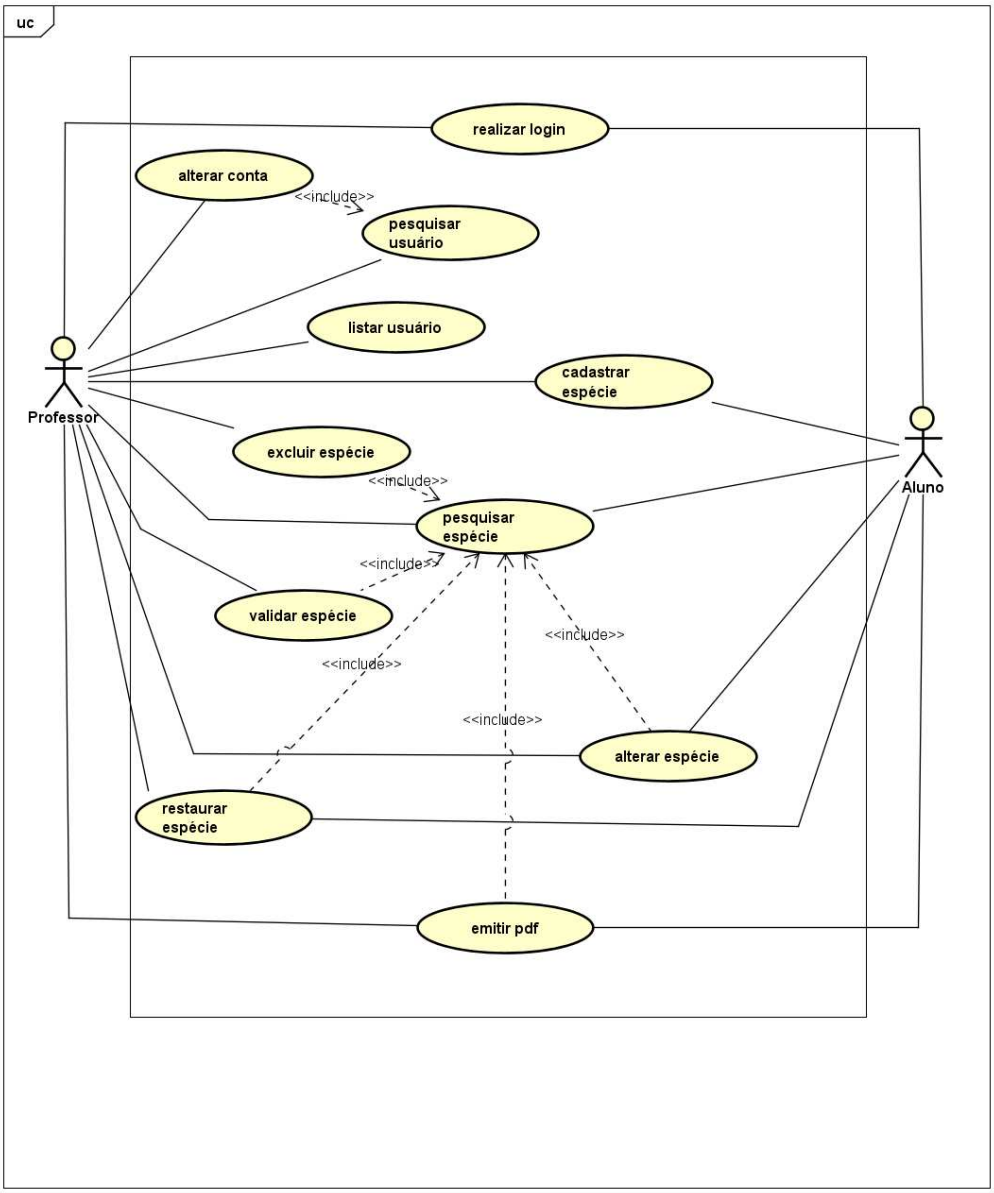
3.3 Diagramas

Os diagramas são ferramentas essenciais para a organização dos sistemas, pois a demonstração visual facilita o processo de análise e concepção das ideias referentes ao desenvolvimento.

3.3.1 Diagrama de Casos de Uso

O diagrama de caso de uso segundo Guedes (2009, p.30), "[...] Apresenta uma linguagem simples e de fácil compreensão para que os usuários possam ter uma ideia geral de como o sistema irá se comportar [...]". Atentando a esse aspecto chegou-se ao diagrama de casos de uso observado na figura 4.

Figura 4: Diagrama de Casos de Uso



Fonte: Autor

Da figura 4 podemos observar os atores do sistema: o professor e o aluno, onde o professor tem acesso às funções do sistema em sua totalidade e o aluno apenas exerce acesso parcial as mesmas.

Abaixo serão documentados os casos de uso do sistema com base no diagrama observado anteriormente.

Tabela 1: Realizar login

Nome do Caso de Uso	Realizar login
Atores	Professor, aluno
Resumo	Este caso de uso descreve os procedimentos necessários para efetuar o login no sistema.
Pré-Condições	O professor/aluno precisa estar cadastrado no sistema e possuir um nome e senha válidos reconhecidos pelo mesmo.
Pós-Condições	O login é realizado com sucesso.
Fluxo Principal	
1. O professor/aluno navega até o menu de login do sistema e insere seu nome de usuário e senha; 2. O sistema valida a entrada dos dados fornecidos; 3. O login é efetuado.	
Fluxo Alternativo	
1. O sistema informa que os dados fornecidos estão incorretos e retorna para a tela de login informando se o nome de usuário e/ou senha inseridos estão incorretos.	

Fonte: Autor

Tabela 2: Alterar conta

Nome do Caso de Uso	Alterar conta
Atores	Professor
Resumo	Este caso de uso descreve os procedimentos para mudança do tipo da conta em relação aos níveis de acesso às funções do sistema.
Pré-Condições	O professor com permissão para efetuar a operação de mudança precisa ter realizado login no sistema.
Pós-Condições	Alteração do tipo de conta realizada com sucesso.

Fluxo Principal
1. O professor pesquisa os usuários do sistema através da função correspondente; 2. O sistema lista os usuários cadastrados; 3. O professor seleciona o usuário; 4. O sistema exibe as informações do usuário a ser concedida a troca, assim como as opções de níveis de acesso; 5. O administrador seleciona o nível pretendido e confirma a escolha; 6. O sistema efetua a troca.
Fluxo Alternativo
1. Usuário não encontrado.

Fonte: Autor

Tabela 3: Pesquisar usuário

Nome do Caso de Uso	Pesquisar usuário
Atores	Professor
Resumo	Este caso de uso descreve os procedimentos para pesquisar um usuário cadastrado no sistema.
Pré-Condições	O professor precisa ter efetuado o login no sistema.
Pós-Condições	O professor encontrou o usuário cadastrado.
Fluxo Principal	
1. O professor seleciona a função de pesquisar usuário no menu do sistema; 2. O sistema exibe uma tela contendo uma lista dos usuários cadastrados; 3. O professor seleciona o usuário na lista; 4. O sistema exibe as informações do usuário selecionado;	
Fluxo Alternativo	

Fonte: Autor

Tabela 4: Cadastrar espécie

Nome do Caso de Uso	Cadastrar espécie
Atores	Professor, aluno
Resumo	Este caso de uso descreve os procedimentos referentes ao processo de cadastro de uma espécie no sistema.

Pré-Condições	O professor/aluno deve ter realizado login no sistema.
Pós-Condições	O cadastro da espécie foi realizado com sucesso.
Fluxo Principal	
1. O professor/aluno acessa a tela de cadastro no sistema; 2. O sistema exibe a tela como os campos a serem preenchidos pelo usuário; 3. O professor/aluno efetua o preenchimento dos dados e clica no botão de salvar para concluir o cadastro; 4. O sistema exibe a tela de confirmação de cadastro, assim como as informações referentes a quem executa tal função.	
Fluxo Alternativo	
1. O professor/aluno preencheu de forma incorreta os campos solicitados; 2. O sistema retorna a tela de cadastro informando quais campos foram preenchidos incorretamente.	

Fonte: Autor

Tabela 5: Validar espécie

Nome do Caso de Uso	Validar espécie
Atores	Professor
Resumo	Este caso de uso descreve os procedimentos a serem realizados para efetuar o processo de validação de cadastro de uma espécie.
Pré-Condições	O professor deve ter efetuado login no sistema e deve estar disponível para ele uma lista com as espécies a serem validadas.
Pós-Condições	O cadastro foi validado com sucesso.
Fluxo Principal	
1. O sistema exibe a lista de espécies a serem validadas. 2. O professor escolhe na lista uma espécie para ser validada; 3. O usuário examina as informações contidas na folha de cadastro e avalia a veracidade destas; 4. O usuário aperta o botão de validar e confirmar o processo; 5. O sistema exibe a tela de confirmação do processo de validação, assim como o histórico de usuários que manipularam a ficha.	
Fluxo Alternativo	

1. O usuário encontra erros no processo de avaliação das informações e conclui que a mesma não pode ser validada; 2. O sistema mantém a ficha na lista de espécies não validadas aguardando as instruções do usuário.

Fonte: Autor

Tabela 6: Pesquisar espécie

Nome do Caso de Uso	Pesquisar espécie
Atores	Professor, alunos
Resumo	Este caso de uso descreve os procedimentos para pesquisar uma espécie cadastrada no sistema.
Pré-Condições	O professor/aluno precisa ter efetuado o login no sistema.
Pós-Condições	O professor/aluno encontrou a espécie cadastrada.
Fluxo Principal	
1. O sistema através da interface exibe menus que filtram as espécies para facilitar a pesquisa de uma espécie específica; 2. O professor/aluno navega pelos filtros disponibilizados em cada lista até encontrar aquela ao qual deseja verificar as informações; 3. O sistema exibe uma tela contendo as informações cadastradas referente a espécie pesquisada e selecionada.	
Fluxo Alternativo	
1. O professor/aluno não conseguiu encontrar a espécie que desejava.	

Fonte: Autor

Tabela 7: Excluir espécie

Nome do Caso de Uso	Excluir espécie
Atores	Professor
Resumo	Este caso de uso descreve os procedimentos para excluir uma espécie cadastrada no sistema.
Pré-Condições	O professor deve ter realizado login no sistema e a espécie estar cadastrada no sistema.
Pós-Condições	A espécie foi excluída com sucesso.
Fluxo Principal	

1. O professor navega pelo menu até a função de exibir a lista das espécies cadastradas no sistema; 2. O sistema exibe listas com seus respectivos filtros das espécies cadastradas; 3. O professor navega pelos filtros disponibilizados em cada lista até encontrar a espécie ao qual deseja excluir e a seleciona; 4. O sistema exibe uma tela contendo as informações cadastradas referentes à espécie pesquisada e selecionada; 5. O professor clica no botão com a função de excluir; 6. O sistema exibe uma tela informando que a exclusão da espécie foi realizada com sucesso.
Fluxo Alternativo

Fonte: Autor

Tabela 8: Restaurar espécie

Nome do Caso de Uso	Restaurar espécie
Atores	Professor
Resumo	Este caso de uso descreve os procedimentos para restaurar uma espécie excluída no sistema.
Pré-Condições	O professor deve ter realizado login no sistema e a espécie estar cadastrada no sistema e excluída do mesmo.
Pós-Condições	A espécie foi restaurada com sucesso.
Fluxo Principal	
1. O professor navega pelo menu até a função de exibir a lista das espécies excluídas; 2. O sistema exibe listas com seus respectivos filtros das espécies excluídas; 3. O professor navega em cada lista até encontrar aquela a espécie ao qual deseja restaurar e a seleciona; 4. O sistema exibe uma tela contendo as informações cadastradas referentes à espécie pesquisada e selecionada; 5. O professor clica no botão com a função de restaurar a espécie; 6. O sistema exibe uma tela informando que a restauração da espécie foi realizada com sucesso.	

Fluxo Alternativo

Fonte: Autor

Tabela 9: Emitir Relatório

Nome do Caso de Uso	Emitir Relatório
Atores	Professor, aluno
Resumo	Este caso de uso descreve os procedimentos necessários para a emissão de um relatório contendo as informações de uma espécie cadastrada.
Pré-Condições	O professor/aluno deve ter realizado login no sistema.
Pós-Condições	A emissão do relatório foi realizada com sucesso.
Fluxo Principal	
1. O professor/aluno navega pela lista de espécies e escolhe aquela ao qual deseja emitir o PDF; 2. O sistema exibe a tela contendo os dados de cadastro e as opções referentes a mesma; 3. O professor/aluno clica no botão de emitir relatório; 4. O Sistema abre uma nova aba do navegador contendo o relatório emitido.	
Fluxo Alternativo	

Fonte: Autor

3.3.2 Diagrama de Classes

O diagrama de caso de classes Segundo Guedes (2009, p.31), "[...] define a estrutura das classes utilizadas pelo sistema, determinando os atributos e métodos que cada classe tem, além de estabelecer como as classes se relacionam e trocam informações entre si [...]". Dito isto, foram identificadas as seguintes classes: Usuário, Professor, Aluno e Espécie.

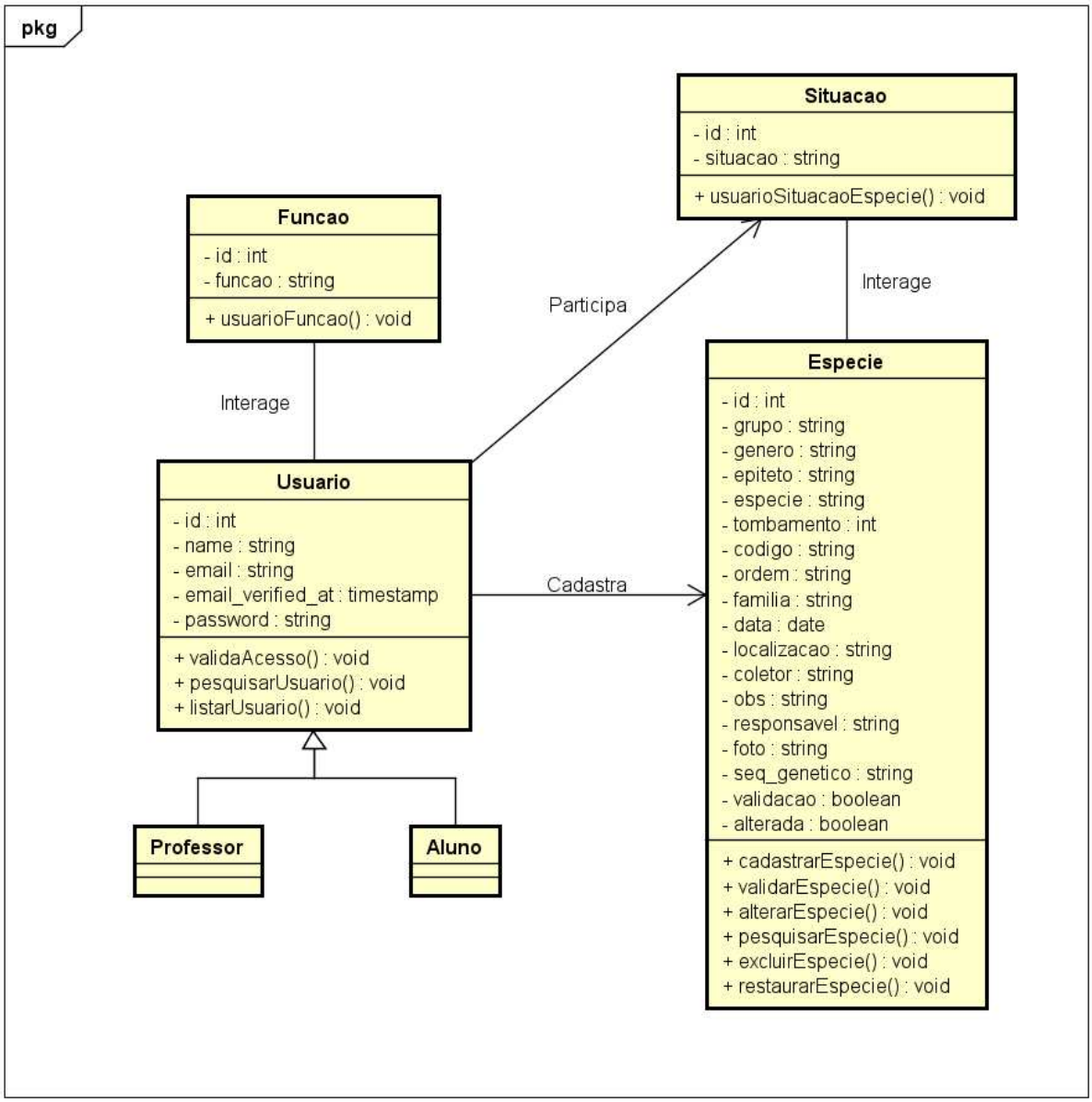
A classe Usuario irá representar a pessoa que utiliza o sistema, sendo este diferenciado apenas pelo nível de acesso às funcionalidades que o sistema possui, com isso podem ser observadas duas subclasses a partir desta: Professor e Aluno.

A classe Especie irá representar o que será cadastrado e manipulado no catalogo de espécie, foco principal do sistema.

A classe Funcao irá representar o nível de acesso do usuário as funcionalidades do sistema.

A classe Situacao irá representar em qual estado se encontra um dado registro de um cadastro de uma espécie no sistema.

Figura 5: Diagrama de Classes



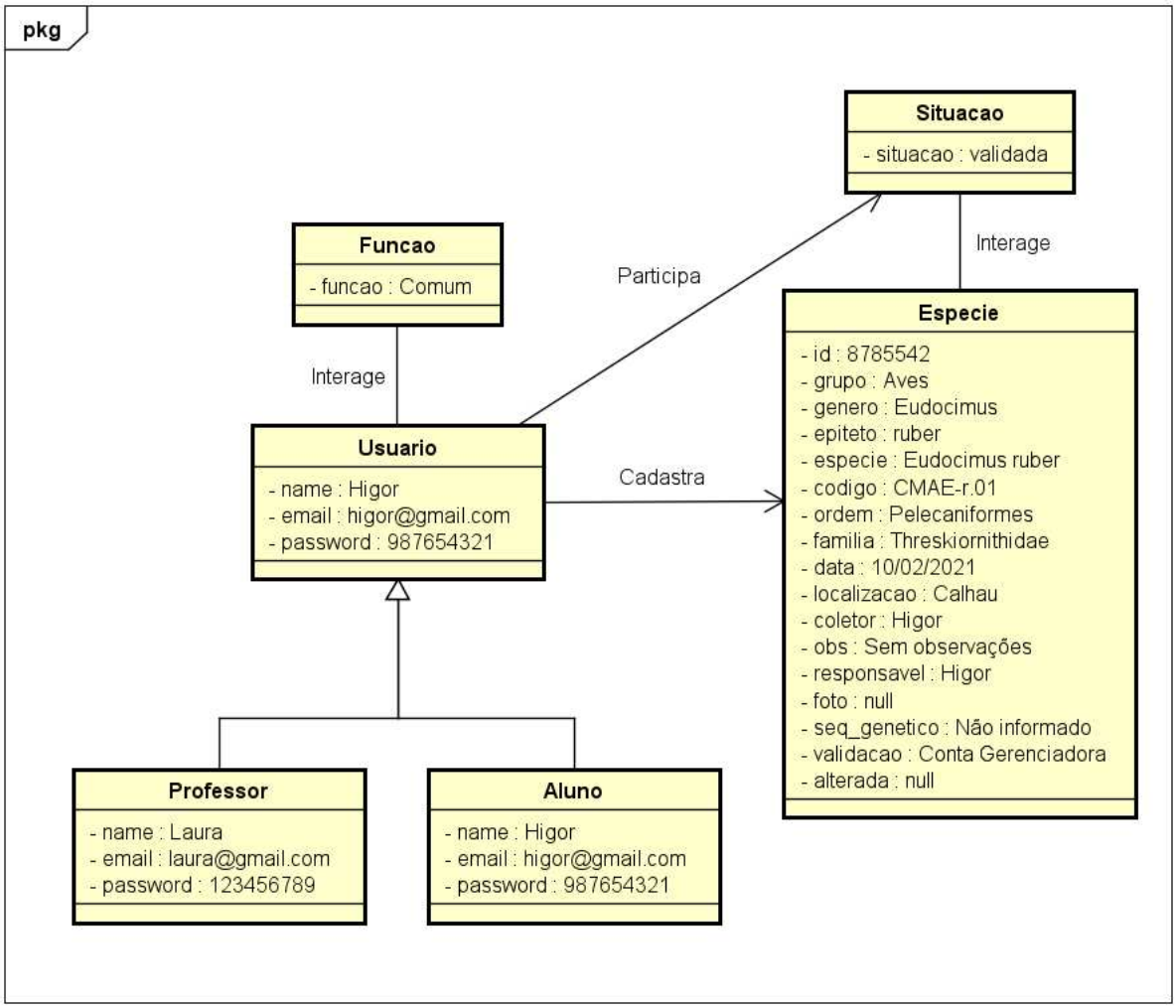
Fonte: Autor

3.3.3 Diagrama de Objetos

O diagrama de objetos segundo Guedes (2009, p.32), “[...] fornece uma visão dos valores armazenados pelos objetos de um diagrama de classes em um determinado momento da execução de um processo do software [...]”. Sendo assim, pode-se utilizar a seguinte abordagem para demonstrar como este diagrama foi empregado.

Tomando a situação em que no momento estão cadastrados um aluno e um professor, o professor irá efetuar o cadastro de uma espécie no sistema, dessa forma a classe Usuário irá naquele momento representar a classe Professor, possuindo assim os mesmos atributos.

Figura 6: Diagrama de Objetos



Fonte: Autor

3.3.4 Diagrama de Sequência

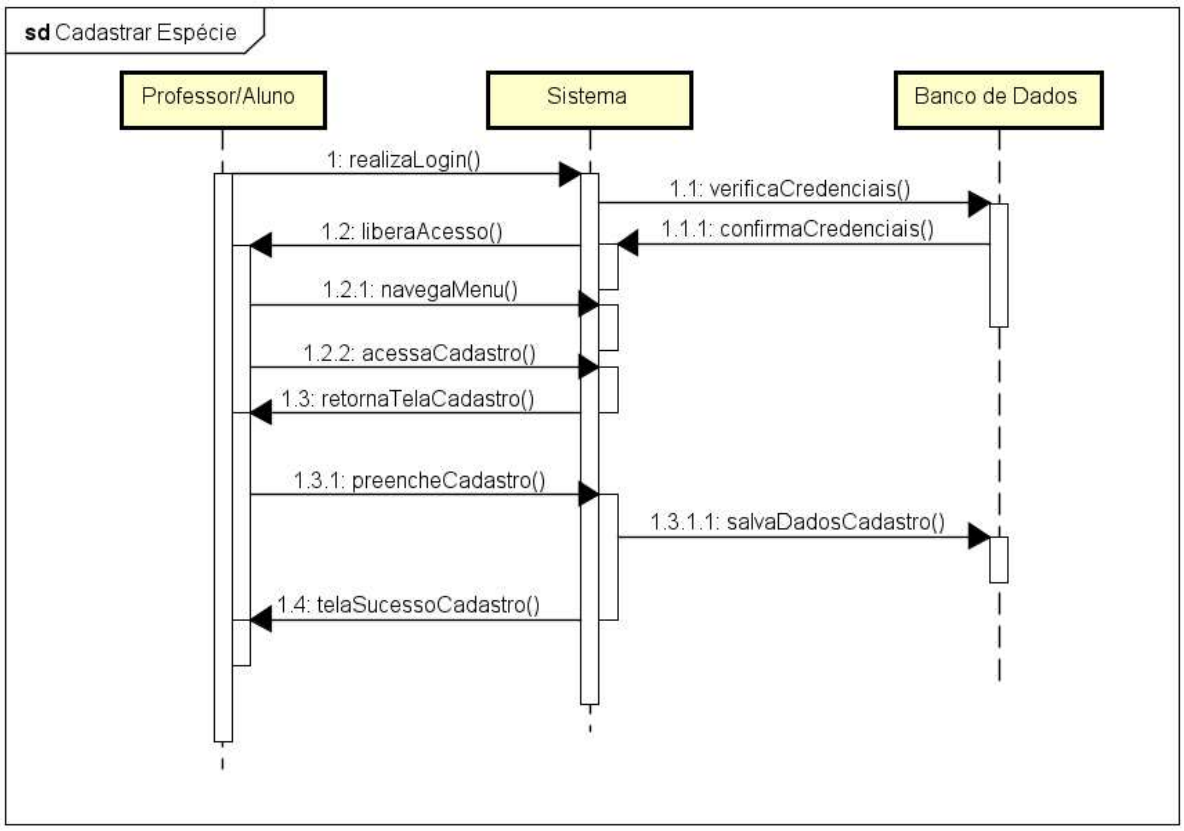
O diagrama de sequência segundo Guedes (2009, p.33), "[...] é um diagrama comportamental que preocupa-se com a ordem temporal em que as mensagens são trocadas entre os objetos envolvidos em um determinado processo [...]". Portanto, seguindo esse conceito temos os seguintes diagramas abaixo, demonstrando o processo de cadastro de uma espécie e validação de uma espécie.

3.3.4.1 Cadastro de uma espécie

A troca de mensagens será entre os objetos: Professor/Aluno, Sistema e banco de dados, e ocorrerá da seguinte maneira:

1. O Professor/Aluno realiza o login no sistema;
2. O sistema consulta o Banco de Dados para verificar se as credenciais de acesso estão corretas;
3. O Banco de Dados retorna à confirmação positiva das credenciais;
4. O sistema retorna a tela de acesso comum a ambos após o processo de validação;
5. O Professor/Aluno navega pelo menu até a função de cadastro de uma nova espécie;
6. O sistema retorna a tela de cadastro;
7. O Professor/Aluno preenche os dados requisitados para um novo cadastro;
8. O sistema salva os dados no Banco de Dados e retorna a confirmação de que os mesmos foram cadastrados com sucesso.

Figura 7: Diagrama de sequência – Cadastro de Espécie



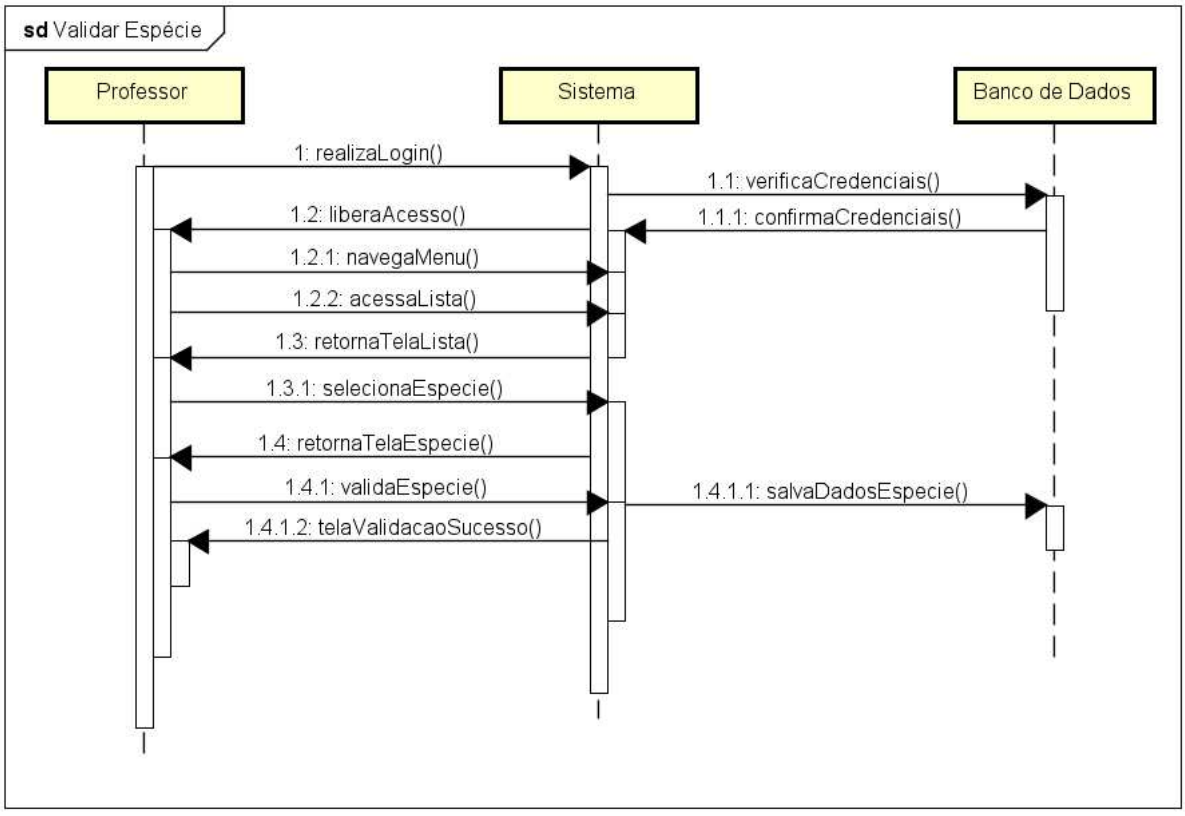
Fonte: Autor

3.3.4.2 Validação de uma espécie

A troca de mensagens será entre os objetos: Professor, Sistema e Banco de Dados, e ocorrerá da seguinte maneira:

1. O Professor realiza o login no sistema;
2. O sistema consulta o Banco de Dados para verificar se as credenciais de acesso estão corretas;
3. O Banco de Dados retorna a confirmação positiva das credenciais;
4. O sistema retorna a tela de acesso própria do usuário professor após o processo de validação;
5. O Professor navega pelo menu até a lista de espécies não validadas;
6. O sistema retorna a tela exibindo a lista;
7. O Professor seleciona uma espécie para começar o processo de validação;
8. O sistema retorna uma tela exibindo as informações da espécie;
9. O professor valida a espécie;
10. O sistema salva os dados no Banco de Dados e retorna a confirmação de que o cadastro foi validado com sucesso.

Figura 8: Diagrama de sequência – Validar Espécie



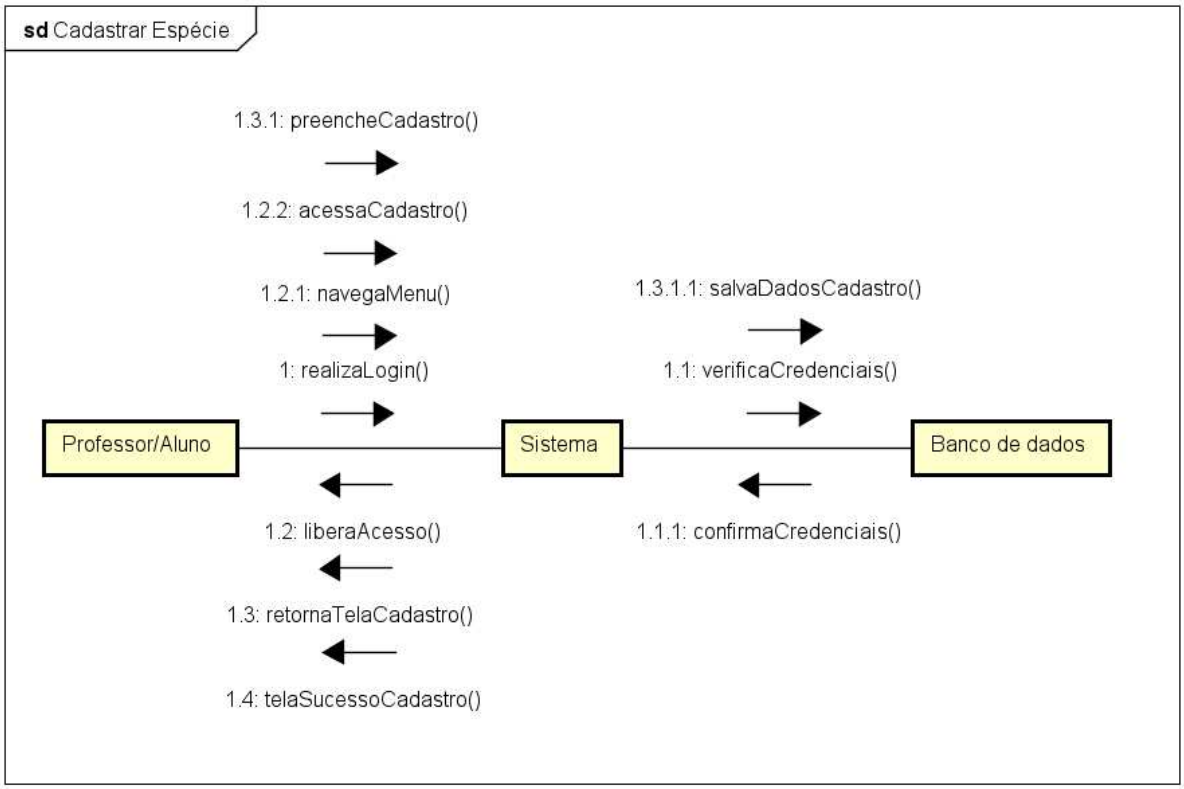
Fonte: Autor

3.3.5 Diagrama de Comunicação

O diagrama de comunicação segundo Guedes (2009, p.35), "[...] Está amplamente associado ao diagrama de sequência: na verdade, um complementa o outro [...]". Pois neste em comparação com o diagrama de sequência, o foco é distinto na questão em como os elementos estão vinculados e em como ocorre a troca de mensagens entre si. Tomando esse fator como base será apresentada dois diagramas correspondendo aos respectivos processos de cadastro e validação de uma espécie.

1. Cadastro de uma espécie.

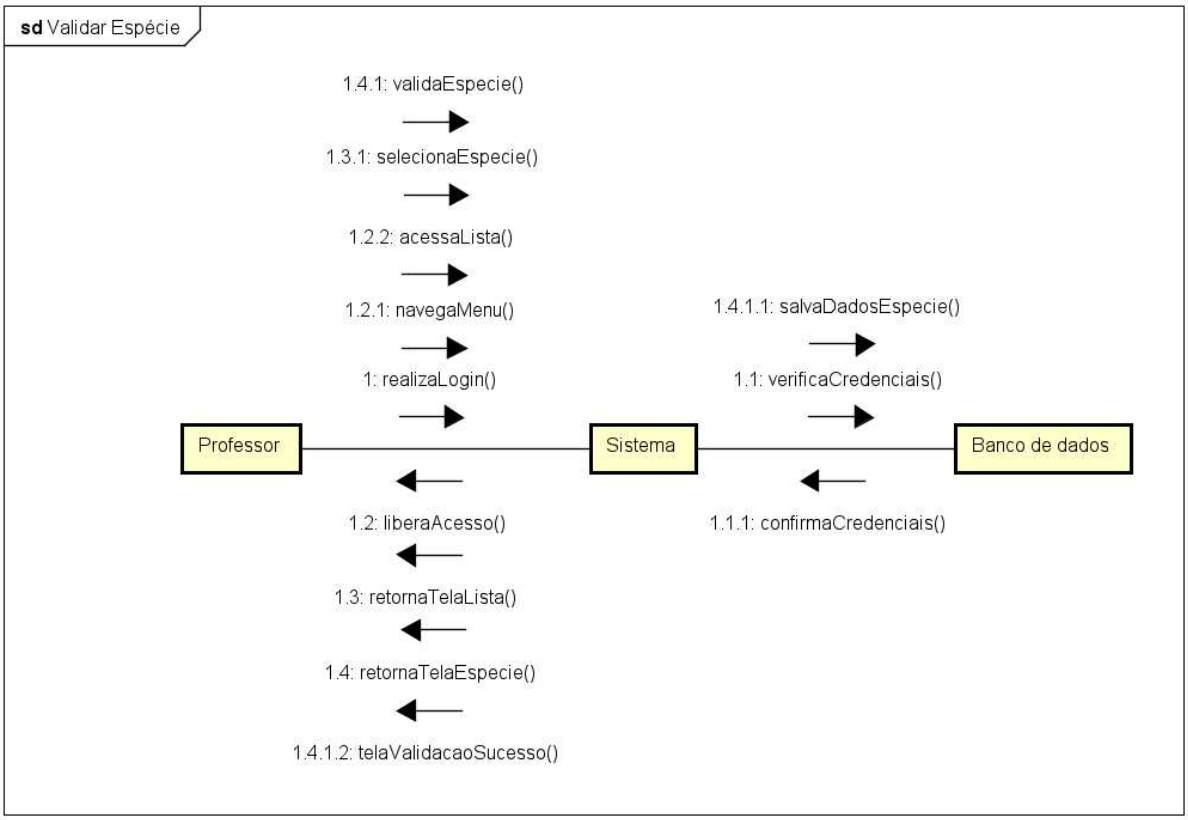
Figura 9: Diagrama de comunicação – Cadastro de Espécie



Fonte: Autor

2. Validação de uma espécie.

Figura 10: Diagrama de comunicação – Validar Espécie



Fonte: Autor

3.3.6 Diagrama de Atividade

O diagrama de atividade segundo Guedes (2009, p.35), "[...] preocupa-se em descrever os passos a serem percorridos para a conclusão de uma atividade específica, podendo esta ser representada por um método com certo grau de complexidade, um algoritmo, ou mesmo por um processo completo [...]. Assim, serão apresentados dois diagramas correspondentes aos fluxos encontrados no sistema.

3.3.6.1 Cadastro de uma espécie

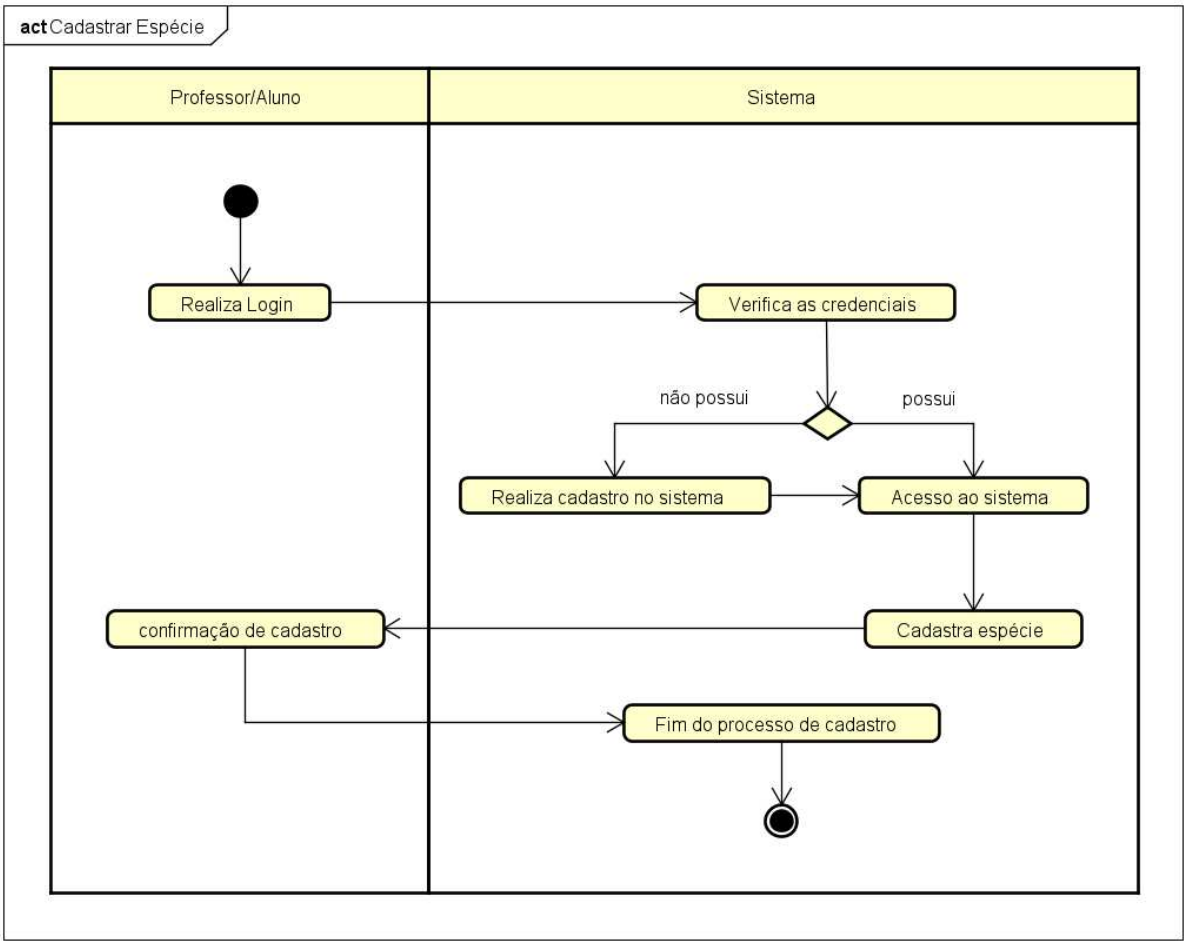
O diagrama abaixo irá representar o fluxo correspondente ao processo de cadastro de uma espécie no sistema, sendo descrito da seguinte maneira.

1. Professor/Aluno realizar login no sistema;
2. O sistema verifica as credenciais de acesso;
    2.1. Caso possua cadastro no sistema é liberado a tela inicial;

2.2. Caso não possua cadastro, é necessário realizá-lo para poder prosseguir com a operação.

3. O Professor/Aluno cadastra a espécie no sistema;
4. O sistema exibe a tela de confirmação de cadastro.

Figura 11: Diagrama de atividade – Cadastro de Espécie



Fonte: Autor

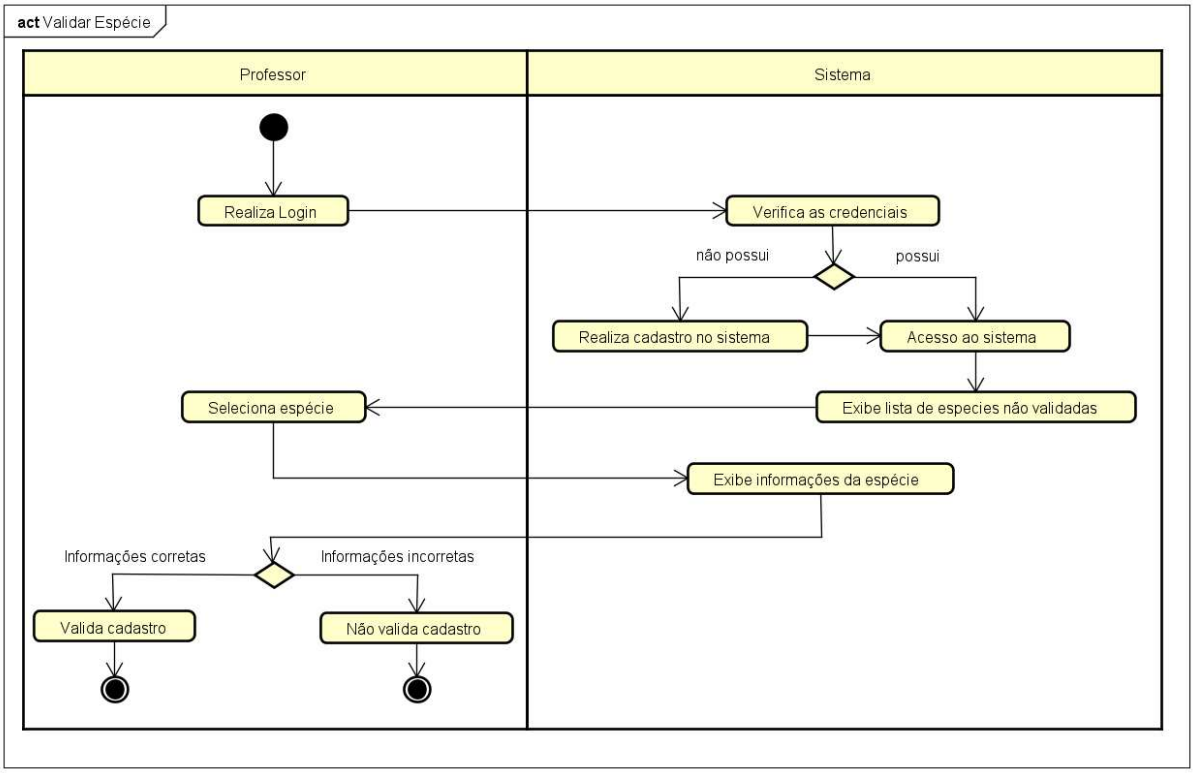
3.3.6.2 Validação de uma espécie

O diagrama abaixo irá apresentar o fluxo correspondente ao processo de validação de uma espécie cadastrada no sistema, sendo descrito da seguinte maneira.

1. Professor realizar login no sistema;
2. O sistema verifica as credenciais de acesso;
   2.1. Caso possua cadastro no sistema é liberado a tela inicial;
   2.2. Caso não possua cadastro, é necessário realizá-lo para poder prosseguir com a operação.
3. O professor verifica a lista de espécies não validadas;

4. O professor seleciona aquela que ele quer validar;
5. O sistema exibe as informações da espécie selecionada;
    5.1. Caso as informações estejam corretas, o professor valida o cadastro e termina o processo;
    5.2. Caso as informações estejam incorretas, o professor não valida o cadastro e termina o processo.

Figura 12: Diagrama de atividade – Validar Espécie

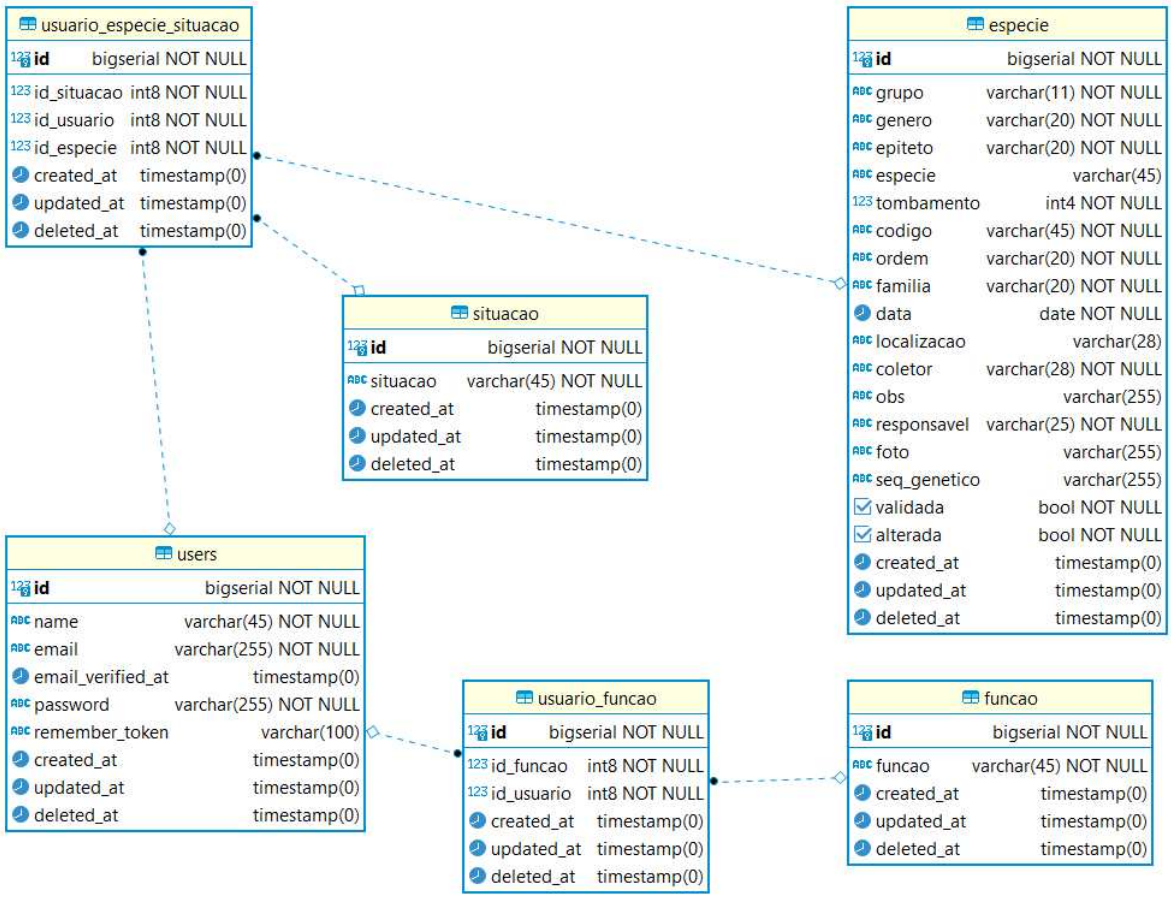


Fonte: Autor

3.4 Modelagem do Banco de Dados

Baseando nos estudos feitos nos capítulos anteriores, foi elaborado o modelo Entidade-Relacionamento do Banco de Dados. Para facilitar a sua construção foi utilizado o software DBeaver, onde este importou o banco de dados do Postgres e posteriormente através de suas funções elaborou o Modelo ER correspondente, sendo este apresentado na imagem abaixo.

Figura 13: Modelo Entidade-Relacionamento



Fonte: Autor

Observando a figura 13, pode-se observas as seguintes características:

1. As entidades especie e users correspondem respectivamente aos cadastros de espécie e as contas de usuários do sistema. Elas possuem o relacionamento de um para muitos onde um usuário pode efetuar cadastrar várias espécies, mas o cadastro de uma espécie só pode ser realizado por um único usuário.

2. A entidade situacao corresponde a situação de um cadastro de uma espécie. Possui um relacionamento de um para muitos com a entidade associativa usuario_especie_situacao onde através dela é possível identificar que um cadastro de uma espécie realizado por um dado usuário só pode ter uma situação associada a seu cadastro, mas vários cadastros podem ter o mesmo tipo de situação.

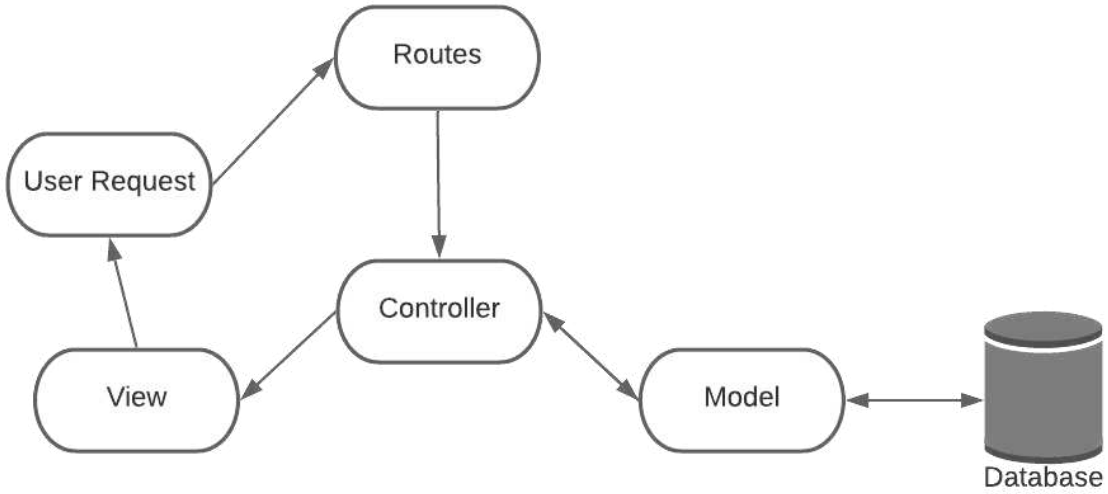
3. As entidades users e funcao correspondem as respectivamente as contas de usuários do sistema e os níveis de acesso das contas dos usuários as funcionalidades do sistema.

Possuem um relacionamento de um para muitos onde uma conta de usuário possui apenas um nível de acesso, mas um nível de acesso pode possui múltiplas contas de usuários, esse relacionamento é empregado utilizando a entidade usuario_funcao já que no sistema é possível efetuar a troca de nível de acesso, sendo assim necessária a utilização de uma outra entidade pra efetuar devidamente o relacionamento.

4 IMPLEMENTAÇÃO DO SISTEMA WEB PARA GESTÃO DE UM CATÁLOGO DE ESPÉCIES DE ANIMAIS

O desenvolvimento do sistema seguirá as características do framework selecionado para a codificação do mesmo, o Laravel, seguindo a estrutura MVC, aderindo aos requisitos levantados, o sistema foi desenvolvido inteiramente utilizando a estrutura de pastas do Laravel, sem a utilização de um API (Application Programming Interface). A configuração do back-end e front-end do sistema será composto de uma única aplicação, na imagem abaixo é possível observar como se dará o comportamento dos componentes principais do sistema, estes componentes serão explicados posteriormente no decorrer deste trabalho.

Figura 14: Esquema do modelo MVC no Laravel



Fonte: Autor

4.1 Integração e construção do Banco de Dados (Database)

Primeiramente, utilizando o arquivo *database.php* na pasta *config* é selecionado e configurado o Banco de Dados do sistema. Neste projeto em questão foi escolhido o Postgres, feito esse processo, a seguir será abordado as estruturas responsáveis pela gestão das tabelas e os relacionamentos das classes.

4.1.1 Migrations

As migrations na estrutura de pastas do Laravel são as responsáveis pela criação, alteração e exclusão das tabelas no Banco de Dados. Uma das características mais importantes dessa ferramenta é sua capacidade simplificada ao executar cada uma dessas ações.

Em cada migration podemos encontrar dois métodos muito importantes, o método *up* que instancia a tabela no banco e o método down que realiza a função de desfazê-la. Além deles foram utilizados o seguinte modificador, tipo de index e comandos respectivamente para gerar as colunas da tabela, utilizados da própria documentação do Laravel.

Tabela 10: Modificadores dos atributos

Modificador	Descrição
->nullable()	Permitir valores NULL para serem inseridos na coluna.

Fonte: Documentação do Laravel v6.x

Tabela 11: Indicadores dos atributos

Comando	Descrição
$table->unique('email');	Adiciona um index único.

Fonte: Documentação do Laravel v6.x

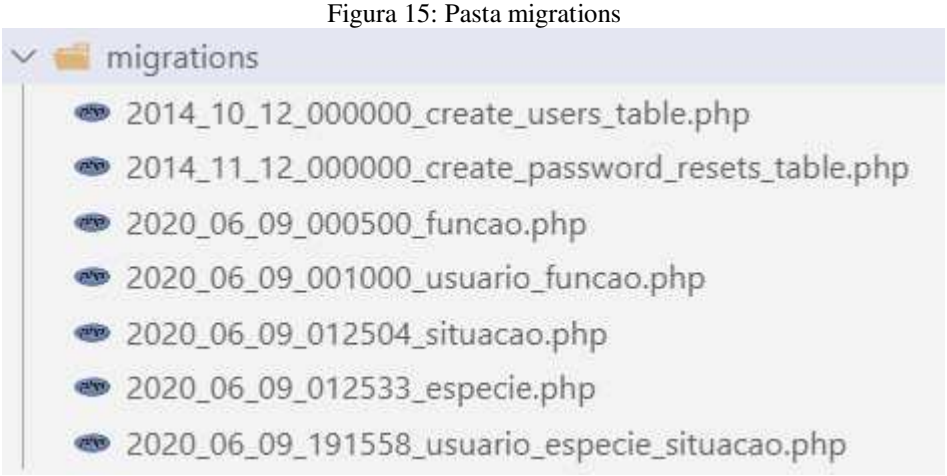
Tabela 12: Métodos dos atributos

Comando	Descrição
$table->bigIncrements('id');	Incrementando ID usando um "big integer" equivalente.
$table->boolean('confirmed');	BOOLEAN equivalente para o banco de dados.
$table->date('created_at');	DATE equivalente para o banco de dados.
$table->rememberToken();	Adiciona um remember_token VARCHAR(100) NULL.
$table->softDeletes();	Adiciona uma coluna deleted_at para soft deletes.
$table->string('email');	VARCHAR coluna equivalente.
$table->string('name', 100);	VARCHAR com um comprimento equivalente.
$table->timestamp('added_on');	TIMESTAMP equivalente para o banco de dados.
$table->timestamps();	Adiciona uma coluna created_at e updated_at.

Fonte: Documentação do Laravel v6.x

Dito isto, na imagem abaixo se tem as migrations utilizadas no sistema.

Figura 15: Pasta migrations



Fonte: Autor

Detalhando cada uma delas, tem-se:

1. 2014_10_12_000000_create_users_table.php é a migration responsável pela criação e gestão da tabela de usuários do sistema no Banco de Dados.

Figura 16: Migration 2014_10_12_000000_create_users_table.php

2014_10_12_000000_create_users_table.php

database > migrations > 2014_10_12_000000_create_users_table.php

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class CreateUsersTable extends Migration
{

    public function up()
    {
        Schema::create('users', function (Blueprint $table) {
            $table->bigIncrements('id');
            $table->string('name', 45);
            $table->string('email')->unique();
            $table->timestamp('email_verified_at')->nullable();
            $table->string('password');
            $table->rememberToken();
            $table->timestamps();
        });
        Schema::table('users', function (Blueprint $table) {
            $table->softDeletes();
        });
    }

    public function down()
    {
        Schema::dropIfExists('users');
    }
}

```

Fonte: Autor

Nessa migration podemos identificar os seguintes atributos:

- **Atributo id**: identificador do usuário, do tipo bigIcrements;
- **Atributo name**: nome do usuário, do tipo string, obrigatório, com tamanho máximo de 45 caracteres;
- **Atributo email**: e-mail do usuário, do tipo string, obrigatório e único por usuário do sistema;
- **Atributo email_verified_at**: do tipo timestamp('added_on'), adiciona um TIMESTAMP no banco pra esse campo;
- **Atributo password**: senha de acesso do usuário no sistema, do tipo string, obrigatório;
- **Atributo remenberToken**: auxiliar da estrutura de login, funciona como uma chave de acesso para o usuário quando este inicia uma sessão no sistema;
- **Atributo timestamps**: vem implementado por padrão quando criamos uma nova migration, segue a função original do método.

2. 2014_11_12_000000_create_password_resets_table.php é a migration responsável pela alteração da senha de acesso do usuário ao sistema.

Figura 17: Migration 2014_11_12_000000_create_password_resets_table.php

2014_11_12_000000_create_password_resets_table.php

database > migrations > 2014_11_12_000000_create_password_resets_table.php

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class CreatePasswordResetsTable extends Migration
{

    public function up()
    {
        Schema::create('password_resets', function (Blueprint $table) {
            $table->string('email')->index();
            $table->string('token');
            $table->timestamp('created_at')->nullable();
        });
    }

    public function down()
    {
        Schema::dropIfExists('password_resets');
    }
}

```

Fonte: Autor

Nessa migration podemos identificar os seguintes atributos:

- **Atributo email**: e-mail do usuário, do tipo string, obrigatório e único por usuário;
- **Atributo token**: funciona como uma chave de segurança para o usuário quando este inicia uma sessão, do tipo string, obrigatório e inserido automaticamente quando o token da sessão é gerado ao logar no sistema;
- **Atributo created_at**: do tipo timestamp('added_on'), adiciona um TIMESTAMP no banco pra esse campo.

3. 2020_06_09_000500_funcao.php é a migration responsável pelo nível de acesso ao qual o usuário tem sobre as funções do sistema;

Figura 18: Migration 2020_06_09_000500_funcao.php

2020_06_09_000500_funcao.php

database > migrations > 2020_06_09_000500_funcao.php

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class Funcao extends Migration
{

    public function up()
    {
        Schema::create('funcao', function (Blueprint $table) {
            $table->bigIncrements('id');
            $table->string('funcao', 45);
            $table->timestamps();
        });
        Schema::table('funcao', function (Blueprint $table) {
            $table->softDeletes();
        });
    }

    public function down()
    {
        Schema::dropIfExists('funcao');
    }
}

```

Fonte: Autor

Nessa migration podemos identificar os seguintes atributos:

- **Atributo id**: identificador do nível de acesso, do tipo bigIcrements;
- **Atributo funcao**: nome do nível de acesso, do tipo string, obrigatório e tamanho máximo de 45 caracteres;
- **Atributo timestamps**: vem implementado por padrão quando criamos uma nova migration, segue a função original do método.

4. 2020_06_09_001000_usuario_funcao.php é a migration responsável por estabelecer e manter o relacionamento entre os níveis de acesso ao sistema e os usuários;

Figura 19: Migration 2020_06_09_001000_usuario_funcao.php

2020_06_09_001000_usuario_funcao.php

database > migrations > 2020_06_09_001000_usuario_funcao.php

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class UsuarioFuncao extends Migration
{

    public function up()
    {
        Schema::create('usuario_funcao', function (Blueprint $table) {
            $table->bigIncrements('id');
            $table->unsignedBigInteger('id_funcao');
            $table->unsignedBigInteger('id_usuario');
            $table->timestamps();
        });
        Schema::table('usuario_funcao', function (Blueprint $table) {
            $table->softDeletes();
            $table->foreign('id_funcao')->references('id')->on('funcao');
            $table->foreign('id_usuario')->references('id')->on('users');
        });
    }

    public function down()
    {
        Schema::dropIfExists('usuario_funcao');
    }
}

```

Fonte: Autor

Nessa migration podemos identificar os seguintes atributos:

- **Atributo id**: identificador da relação entre usuário e o nível de acesso ao qual foi designado, do tipo *bigIcrements*;
- **Atributo id_funcao**: este atributo indica qual o nível de acesso que irá estar associado a um dado usuário;
- **Atributo id_usuario**: este atributo indica qual o usuário está associado a qual nível de acesso.
- **Atributo timestamps**: vem implementado por padrão quando criamos uma nova migration, segue a função original do método.

5. 2020_06_09_012504_situacao.php é a migration responsável pela tabela situação no Banco de Dados.

Figura 20: Migration 2020_06_09_012504_situacao.php

2020_06_09_012504_situacao.php

database > migrations > 2020_06_09_012504_situacao.php

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class Situacao extends Migration
{

    public function up()
    {
        Schema::create('situacao', function (Blueprint $table) {
            $table->bigIncrements('id');
            $table->string('situacao', 45);
            $table->timestamps();
        });
        Schema::table('situacao', function (Blueprint $table) {
            $table->softDeletes();
        });
    }

    public function down()
    {
        Schema::dropIfExists('situacao');
    }
}

```

Fonte: Autor

Nessa migration podemos identificar os seguintes atributos:

- **Atributo id**: identificador da situação, do tipo bigIcrements responsável por incrementar automaticamente cada novo nível no banco de dados do sistema;
- **Atributo situacao**: nome da situação em que o cadastro da espécie se encontra, do tipo string, obrigatório e tamanho máximo de 45 caracteres;

6. 2020_06_09_012533_especie.php é a migration responsável pela tabela de especie no Banco de Dados.

Figura 21: Migration 2020_06_09_012533_especie.php

2020_06_09_012533_especie.php

database > migrations > 2020_06_09_012533_especie.php

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class Especie extends Migration
{

    public function up()
    {
        Schema::create('especie', function (Blueprint $table) {
            $table->bigIncrements('id');
            $table->string('grupo',11);
            $table->string('genero',20);
            $table->string('epiteto',20);
            $table->string('especie',45)->nullable();
            $table->integer('tombamento');
            $table->string('codigo',45);
            $table->string('ordem', 20);
            $table->string('familia',20);
            $table->date('data');
            $table->string('localizacao', 28)->nullable();
            $table->string('coletor', 28);
            $table->string('obs')->nullable();
            $table->string('responsavel', 25);
            $table->string('foto',255)->nullable();
            $table->string('seq_genetico')->nullable();
            $table->boolean('validada');
            $table->boolean('alterada');
            $table->timestamps();
        });
        Schema::table('especie', function (Blueprint $table) {
            $table->softDeletes();
        });
    }

    public function down()
    {
        Schema::dropIfExists('especie');
    }
}

```

Fonte: Autor

Nessa migration podemos identificar os seguintes atributos:

- **Atributo id**: identificador da espécie cadastrada, do tipo bigIcrements responsável por incrementar automaticamente cada novo cadastro de espécie no banco de dados do sistema;
- **Atributo grupo**: nome do grupo ao qual a espécie faz parte, do tipo string, obrigatório e com tamanho máximo de 11 caracteres;
- **Atributo genero**: nome do gênero ao qual a espécie pertence, do tipo string, obrigatório e tamanho máximo de 20 caracteres;
- **Atributo epiteto**: nome do epiteto da espécie, do tipo string, obrigatório e tamanho máximo de 20 caracteres;
- **Atributo especie**: nome da espécie propriamente dito, do tipo string, não obrigatório e tamanho máximo de 45 caracteres;
- **Atributo tombamento**: gera um número que será utilizado na regra de negócio referente a etiqueta de identificação da espécie no sistema;
- **Atributo codigo**: identificação visual do usuário da espécie cadastrada no banco de dados, diferente do componente id, o código funciona como um meio de controle para os usuários do sistema caso uma espécie com características semelhantes seja cadastrada no sistema, do tipo string e tamanho máximo de 45 caracteres;
- **Atributo ordem**: nome da ordem da espécie, do tipo string, obrigatório e tamanho máximo de 20 caracteres;
- **Atributo familia**: nome da família ao qual a espécie pertence, do tipo string, obrigatório e tamanho máximo de 20 caracteres;
- **Atributo data**: data em que a amostra da espécie foi coletada, do tipo *date*, obrigatório;
- **Atributo localizacao**: Local em que a espécie foi coletada, do tipo string, não obrigatório e tamanho máximo de 28 caracteres;
- **Atributo coletor**: Nome da pessoa que realizou a coleta da espécie, do tipo string, obrigatório e tamanho máximo de 28 caracteres;
- **Atributo obs**: Atributo responsável pelo registro de alguma observação da espécie cadastrada, do tipo string, não obrigatório;
- **Atributo responsavel**: Nome do responsável pelo cadastrado da espécie no sistema, do tipo string, obrigatório e tamanho máximo de 25 caracteres;

- **Atributo foto**: atributo responsável pelo armazenamento de uma possível imagem da especie, do tipo string, não obrigatório;
- **Atributo seq_genetico**: Atributo responsável por guardar o sequenciamento genético de uma espécie, esta elaborada pelos integrantes do projeto, do tipo string, não obrigatório;
- **Atributo validada**: Atributo responsável por confirmar se uma espécie passou pelo processo de validação ou não, do tipo boolean, obrigatório;
- **Atributo alterada**: Atributo responsável por confirmar se uma espécie sofreu alguma alteração em suas informações ou não, do tipo boolean, obrigatório.

7. 2020_06_09_191558_usuario_especie_situacao.php é a migration responsável por manter e estabelecer o relacionamento entre a situação da espécie, a tabela usuario e a tabela espécie.

Figura 22: Migration 2020_06_09_191558_usuario_especie_situacao.php

2020_06_09_191558_usuario_especie_situacao.php

database > migrations > 2020_06_09_191558_usuario_especie_situacao.php

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class UsuarioEspecieSituacao extends Migration
{

    public function up()
    {
        Schema::create('usuario_especie_situacao', function (Blueprint $table) {
            $table->bigIncrements('id');
            $table->unsignedBigInteger('id_situacao');
            $table->unsignedBigInteger('id_usuario');
            $table->unsignedBigInteger('id_especie');
            $table->timestamps();
        });
        Schema::table('usuario_especie_situacao', function (Blueprint $table) {
            $table->softDeletes();
            $table->foreign('id_situacao')->references('id')->on('situacao');
            $table->foreign('id_usuario')->references('id')->on('users');
            $table->foreign('id_especie')->references('id')->on('especie');
        });
    }

    public function down()
    {
        Schema::dropIfExists('usuario_especie_situacao');
    }
}

```

Fonte: Autor

Nessa migration os atributos indicam:

- **Atributo id_situacao**: situação em que se encontra o cadastro da especie, isto é, se passou pelo processo de validação ou não;
- **Atributo id_usuario**: Este relacionamento indica qual usuário irá realizar o processo de validação;
- **Atributo id_especie**: Este atributo indica em qual cadastrão de espécie será realizado o processo;

4.1.2 Seeder

O seeder é uma classe que tem a função de popular o banco de dados com arquivos de dados pré-configurados, no caso do sistema, o arquivo irá conter um login provisório para a conta master para a professora responsável pelo projeto, sendo estas credenciais de acesso alteradas posteriormente para que somente a professora tenha conhecimento dessas credenciais, além de popular a tabelas de *funcao* e *situação*, está sendo detalhados a seguir.

1. **Tabela funcao**: o código implementador demonstrado na imagem abaixo corresponde às entradas dos nomes dos níveis de acesso que os usuários podem exercer, sendo elas: Comum, Administrador e Master.

Figura 23: Seeder da tabela funcao

```
$funcao = [
    'funcao'=> 'Comum',
    'created_at' => $hoje->add($intervalo)
];
DB::table('funcao')->insert($funcao);
$funcao = [
    'funcao'=> 'Administrador',
    'created_at' => $hoje->add($intervalo)
];
DB::table('funcao')->insert($funcao);
$funcao = [
    'funcao'=> 'Master',
    'created_at' => $hoje->add($intervalo)
];
DB::table('funcao')->insert($funcao);
```

Fonte: Autor

2. **Tabela situacao**: o código implementador demonstrado na imagem abaixo corresponde às entradas dos nomes da possível situação em que se encontra um cadastro de uma

espécie qualquer, sendo elas: Cadastrada, Alterada, Removida, Validada, Desvalidada e Restaurada.

Figura 24: Seeder da tabela situacao

```
$situacao = [
    'situacao'=> 'Cadastrada',
    'created_at' => $hoje->add($intervalo)
];
DB::table('situacao')->insert($situacao);
$situacao = [
    'situacao'=> 'Alterada',
    'created_at' => $hoje->add($intervalo)
];
DB::table('situacao')->insert($situacao);
$situacao = [
    'situacao'=> 'Removida',
    'created_at' => $hoje->add($intervalo)
];
DB::table('situacao')->insert($situacao);
$situacao = [
    'situacao'=> 'Validada',
    'created_at' => $hoje->add($intervalo)
];
DB::table('situacao')->insert($situacao);
$situacao = [
    'situacao'=> 'Desvalidada',
    'created_at' => $hoje->add($intervalo)
];
DB::table('situacao')->insert($situacao);
$situacao = [
    'situacao'=> 'Restaurada',
    'created_at' => $hoje->add($intervalo)
];
DB::table('situacao')->insert($situacao);
```

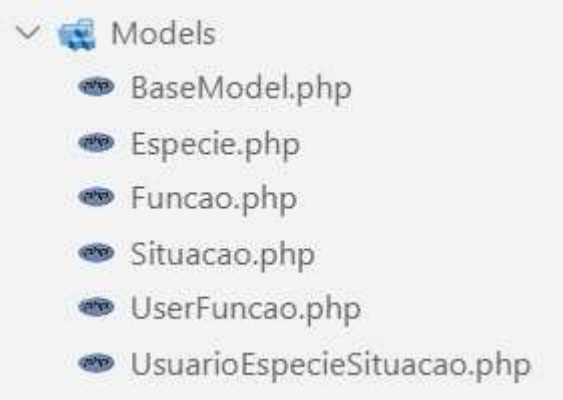
Fonte: Autor

4.2 Models

O Laravel utiliza o Eloquent que é ORM (Object Relational Mapping) nativo do próprio framework, sua função é facilitar o relacionamento entre as tabelas do banco de dados, pra alcançar tal objetivo o eloquent utiliza de vários métodos já prontos para utilização imediata do desenvolvedor e com isso acelerar o processo de conexão entre a aplicação e o banco de dado em questão.

As models são responsáveis pela elaboração das regras de negócio do sistema em questão, em outras palavras é o coração da arquitetura MV, cada tabela no banco de dados possui uma model correspondente onde esta tem a função de realizar uma consulta no banco e/ou a inserção de um novo registro no mesmo. Na imagem a seguir encontram-se as models que correspondem às tabelas criadas nas migrations.

Figura 25: Pasta Models



Fonte: Autor

É possível observar na estrutura de pastas do Laravel como ficam organizados os arquivos de models do sistema, dito isto, cada model tem a função de gerenciar sua respectiva tabela no Database, assim configurar a gestão dos mesmos, isto é, como será feito o controle da inserção dos registros. A fim de obter uma melhor visão dessas estruturas, a seguir serão detalhadas as models observadas na imagem anterior:

1. **BaseModel.php**: este arquivo contém funções padrões que podem ser utilizadas para executar determinadas ações no Banco de Dados, sendo assim cada nova model instancia o BaseModel e utiliza uma dessas funções sem precisar elaborar novamente. Nas models listadas anterior foram estendidas as seguintes funções:
    - **use SoftDeletes**: cria uma coluna na tabela 'deleted_at', essa coluna irá conter as informações referentes aquando determinado registro foi deletado, mas com a ressalva de que o mesmo não será excluído permanentemente e sim fará com que o eloquent não possa mais visualizá-lo;
    - **public function searchable()**: função que corresponde ao método de pesquisa de índices dos registros nas models, por padrão ela é instanciada com o índice correspondente ao nome da tabela, mas este pode ser alterado posteriormente ao descrever quais índices devem pesquisados;

- **public static function boot()**: função que estende o comportamento do BaseModel para as demais models sem ter que ficar sobrescrevendo o método boot;
- **public static function create(array $attributes = [])**: função que cria um novo registro no banco de dados;
- **public function update(array $attributes = [], array $options = [])**: função que altera um registro no banco de dados.

2. **Especie.php**: este arquivo é responsável pela gestão da tabela especie no Banco de Dados. Nesse model foram utilizadas as seguintes funções:
   - **protected $table**: especifica um nome para a tabela;
   - **protected $primaryKey**: especifica a chave a primária da tabela;
   - **protected $fillable**: estendida da BaseModel;
   - **protected $searchable**: estendida da BaseModel;
   - **public function usuarios()**: função utilizada para estabelecer o relacionamento entre a tabela usuario;
   - **public function situacoes()**: função utilizada para estabelecer o relacionamento entre a tabela situacao;
   - **public function usuarioEspecieSituacao()**: função utilizada para estabelecer o relacionamento entre a tabela usuario_especie_situacao.

3. **Funcao.php**: este arquivo é responsável pela gestão da tabela funcao no Banco de Dados. Nesse model foram utilizadas as seguintes funções:
   - **protected $table**: especifica um nome para a tabela;
   - **protected $primaryKey**: especifica a chave a primária da tabela;
   - **protected $fillable**: estendida da BaseModel;
   - **protected $searchable**: estendida da BaseModel;
   - **public function usuarios()**: função utilizada para estabelecer o relacionamento entre a tabela usuario;
   - **public function usuarioFuncao()**: função utilizada para estabelecer o relacionamento entre a tabela usuario_funcao.

4. **Situacao.php**: Este arquivo é responsável pela gestão da tabela situacao no Banco de Dados. Nesse model foram utilizadas as seguintes funções:
    - **protected $table**: especifica um nome para a tabela;
    - **protected $primaryKey**: especifica a chave a primária da tabela;
    - **protected $fillable**: estendida da BaseModel;
    - **protected $searchable**: estendida da BaseModel;
    - **public function usuarios()**: função utilizada para estabelecer o relacionamento entre a tabela usuario;
    - **public function especies()**: função utilizada para estabelecer o relacionamento entre a tabela especie;
    - **public function usuarioEspecieSituacao()**: função utilizada para estabelecer o relacionamento entre a tabela usuario_especie_situacao.

5. **UserFuncao.php**: este arquivo é responsável pela gestão da tabela usuario_funcao no Banco de Dados. Nesse model foram utilizadas as seguintes funções:
    - **protected $table**: especifica um nome para a tabela;
    - **protected $primaryKey**: especifica a chave primária da tabela;
    - **protected $fillable**: estendida da BaseModel;
    - **protected $searchable**: estendida da BaseModel;
    - **public function usuarios()**: função utilizada para estabelecer o relacionamento entre a tabela usuario;
    - **public function funcao()**: função utilizada para estabelecer o relacionamento entre a tabela função.

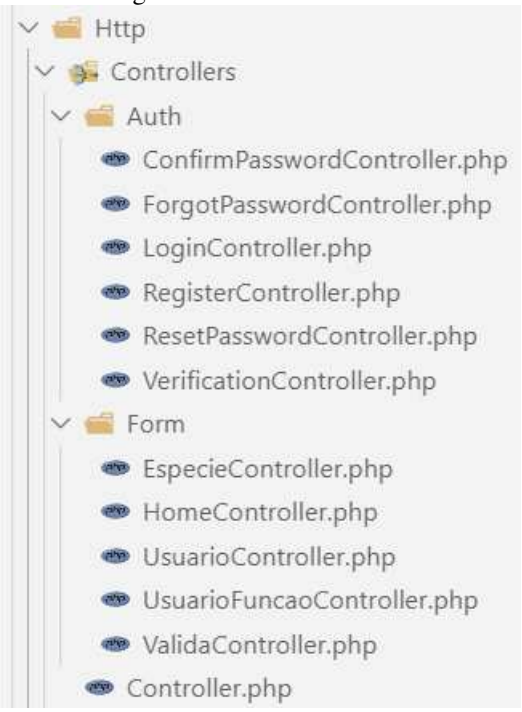
6. **UsuarioEspecieSituacao.php**: este arquivo é responsável pela gestão da tabela usuario_especie_situacao no Banco de Dados. Nesse model foram utilizadas as seguintes funções:
    - **protected $table**: especifica um nome para a tabela;
    - **protected $primaryKey**: especifica a chave primária da tabela;
    - **protected $fillable**: estendida da BaseModel;
    - **protected $searchable**: estendida da BaseModel;
    - **public function usuarios()**: função utilizada para estabelecer o relacionamento entre a tabela usuario;

- **public function especies()**: função utilizada para estabelecer o relacionamento entre a tabela especie;
- **public function funcao()**: função utilizada para estabelecer o relacionamento entre a tabela função.

4.3 Controllers

Os controllers na estrutura de pastas de laravel tem a função de organizar e gerir as solicitações de entrada da classe ao qual está se referenciando, incluindo, por exemplo, criação, exclusão e/ou alteração da classe em questão. Essa conexão é realizada entre as views que compõem o sistema e as models correspondentes, em outras palavras, cada controller gerencia as requisições realizadas pelo usuário na interface gráfica e as corresponde com o banco de dados através das models.

Figura 26: Pasta Controllers



Fonte: Autor

Na imagem anterior é possível observar todos os controllers do sistema, em específico:

1. **Controller.php**: este arquivo contém as funções básicas que uma controller deve ter para que esta possa ser executada com o mínimo de funções essenciais;

2. **Pasta Auth**: nela estão contidos os controllers responsáveis pelo sistema de login e registro do usuário, assim como a alteração dos mesmos, as funções, métodos e demais estruturas contidas nelas são padrões do próprio framework, para fins de explicação e para facilitar o entendimento, não será abordado cada arquivo já que em sua totalidade a sua principal função é gerenciar como será realizada o sistema de níveis de acesso, pois este está intimamente ligado com a tabela de usuários.

3. **Pasta Form**: nela estão organizados os controllers das classes do escopo principal do sistema, em outras palavras os arquivos responsáveis por gerenciar o núcleo do que o sistema realiza, sendo assim será explicado a seguir as principais funções empregadas para realizar essa gestão.

    3.1. **EspecieController.php**: é o controller responsável pela gestão das solicitações da classe especie realizadas através das views da pasta Especie.

    - **public function index(Request $request)**: função responsável por gerenciar as solicitações através da view index.blade.php;
    - **public function show(Especie $especie)**: função responsável por gerenciar as solicitações através da view show.blade.php;
    - **public function create()**: função responsável por gerenciar as solicitações através da view create.blade.php;
    - **public function store(EspecieRequest $request)**: função responsável por gerenciar a alocação do repositório onde será armazenado as imagens que podem vir agregadas no cadastro de uma espécie;
    - **public function edit(Especie $especie)**: função responsável por gerenciar as solicitações através da view edit.blade.php;
    - **public function update(Especie $especie, EspecieRequest $request)**: função responsável por gerenciar a alteração das imagens que podem vir agregadas no cadastro de uma espécie;
    - **public function delete(Especie $especie, Request $request)**: função responsável por gerir o ato de deletar uma dada espécie;
    - **public function deletadas(Request $request)**: função responsável por gerenciar as solicitações através da view deletadas.blade.php;
    - **public function restore($especie, Request $request)**: função responsável por gerir o ato de restaurar uma dada espécie da lista de deletadas;

- **function pdf(Especie $especie)**: função responsável por emitir uma solicitação para a emissão de um relatório em PDF da espécie selecionada;
- **public function seusCadastros(Request $request)**: função responsável por gerenciar as solicitações através da view seusCadastros.blade.php;
- **public function suasValidacoes(Request $request)**: função responsável por gerenciar as solicitações através da view suasValidacoes.blade.php;
- **public function buscar(Request $request)**: função responsável por gerenciar as solicitações através da view buscadas.blade.php;

3.2. **HomeController.php**: é o controller responsável pela gestão das solicitações da tela inicial do sistema;

- **public function home()**: função responsável por gerenciar as solicitações através da view home.blade.php;

3.3. **UsuarioController.php**: é o controller responsável pela gestão das solicitações da classe usuario realizadas através das views da pasta Usuario.

- **public function index(Request $request)**: função responsável por gerenciar as solicitações através da view index.blade.php;
- **public function perfil()**: função responsável por gerenciar as solicitações de visualização do perfil do usuário de acordo com o nível de acesso através da view show.blade.php;
- **public function show(User $user)**: função responsável por gerenciar as solicitações de caráter geral através da view show.blade.php;
- **public function edit(User $user)**: função responsável por gerenciar as solicitações através da view edit.blade.php;
- **public function update(User $user, UserRequest $request)**: função responsável por gerenciar o processo de troca de senha do usuário;
- **public function delete(User $user)**: função responsável por gerenciar o processo de exclusão das credenciais de acesso ao sistema do usuário;
- **public function deletados(Request $request)**: função responsável por gerenciar as solicitações através da view deletados.blade.php;
- **public function restore($user, Request $request)**: função responsável por gerenciar o processo de restauração das credenciais de acesso ao sistema do usuário.

3.4. **UsuarioFuncaoController.php**: é o controller responsável pela gestão das solicitações da classe funcao realizadas através das views da pasta funcao.

- **public function index(Request $request)**: função responsável por gerenciar as solicitações através da view index.blade.php;
- **public function edit(User $user)**: função responsável por gerenciar as solicitações através da view edit.blade.php;
- **public function update(UserFuncao $user, UserFuncaoRequest $request)**: função responsável por gerir o processo de alteração do nível de acesso da conta de um dado usuário.

3.5. **ValidaController.php**: é o controller responsável pela gestão das solicitações da classe situacao realizadas através das views restantes da pasta Especie.

- **public function validadas(Request $request)**: função responsável por gerenciar as solicitações através da view validadas.blade.php;
- **public function naoValidadas(Request $request)**: função responsável por gerenciar as solicitações através da view naoValidadas.blade.php;
- **public function validar(Especie $especie, Request $request)**: função responsável por gerenciar as solicitações através da opção de 'Validar' presente na view show.blade.php;
- **public function desvalidar(Especie $especie, Request $request)**: função responsável por gerenciar as solicitações através da opção de 'Desvalidar' presente na view show.blade.php.

4.4 Routes

As routes são responsáveis por interligar a interface web e o com o resto da aplicação, em outras palavras, elas são responsáveis por receber uma requisição em uma determinada URL e direcioná-las para o controller responsável por aquela view para determinar o próximo curso de ação.

Para uma melhor compreensão de como é construído o arquivo routes, iremos desmembrá-lo e detalhar os trechos de código correspondentes.

1. **Trecho 1**: responsável pelas URLs das telas iniciais do sistema;

Figura 27: Routes da página inicial

```
Route::get('/home', 'Form\\HomeController@home')->name('home');
Route::get('/', 'Form\\HomeController@home')->name('cofauma');
```

Fonte: Autor

2. **Trecho 2**: responsável pelas URLs relacionados aos processos envolvidos com o usuário;

Figura 28: Routes das ações do usuário

```
// Açoes relacionadas ao user
Route::prefix('/usuario')->group(function () {
    Route::prefix('/lista')->group(function () {
        //listar
        Route::get('', 'Form\\UsuarioController@index')->name('usuario.index');
        //listar deletados
        Route::get('/deletados', 'Form\\UsuarioController@deletados')->name('usuario.deletados');
        //listas de cadastradas e validadas pelo user
        Route::get('/especie/seus/cadastrados', 'Form\\EspecieController@seusCadastros')->name('usuario.seusCadastros');
        Route::get('/especie/suas/validacoes', 'Form\\EspecieController@suasValidacoes')->name('usuario.suasValidacoes');
        //funcionalidades
        Route::get('/funcoes', 'Form\\UsuarioFuncaoController@index')->name('funcao.index');
    });
    // exibir
    Route::get('/detalhes/{user}','Form\\UsuarioController@show')->name('usuario.show');
    Route::get('/perfil','Form\\UsuarioController@perfil')->name('usuario');
    Route::prefix('/editar')->group(function () {
        //editar
        Route::get('{user}','Form\\UsuarioController@edit')->name('usuario.edit');
        Route::put('{user}/editar','Form\\UsuarioController@update')->name('usuario.update');
        //editar funcão no sistema
        Route::get('/{user}/funcao', 'Form\\UsuarioFuncaoController@edit')->name('funcao.edit');
        Route::put('/{user}/funcao/update', 'Form\\UsuarioFuncaoController@update')->name('funcao.update');
    });
    //apagar
    Route::delete('/apagar/{user}','Form\\UsuarioController@delete')->name('usuario.delete');
    //restaurar
    Route::post('/restaurar/{user}','Form\\UsuarioController@restore')->name('usuario.restore');
```

Fonte: Autor

3. **Trecho 3**: responsável pelas URLs relacionados aos processos envolvidos com o cadastro da espécie;

Figura 29: Routes das ações da espécie

```
// Açoes relacionadas a especie
Route::prefix('/especie')->group(function () {
    // cadastrar especie
    Route::get('/cadastrar','Form\\EspecieController@create')->name('especie.create');
    Route::post('/salvar','Form\\EspecieController@store')->name('especie.store');
    Route::prefix('/lista')->group(function () {
        // listar todas
        Route::get('', 'Form\\EspecieController@index')->name('especie.index');
        //listar deletadas
        Route::get('/deletadas', 'Form\\EspecieController@deletadas')->name('especie.deletadas');
        //listar validadas e não
        Route::get('/validadas', 'Form\\ValidaController@validadas')->name('especie.validadas');
        Route::get('/naoValidadas', 'Form\\ValidaController@naoValidadas')->name('especie.naoValidadas');
    });
    Route::get('/buscar', 'Form\\EspecieController@buscar')->name('especie.buscar');
    // exibir
    Route::get('/detalhes/{especie}','Form\\EspecieController@show')->name('especie.show');
    Route::prefix('/editar')->group(function () {
        // editar
        Route::get('/{especie}','Form\\EspecieController@edit')->name('especie.edit');
        Route::put('/{especie}/editar','Form\\EspecieController@update')->name('especie.update');
        //validar e desvalidar
        Route::post('/{especie}/validar', 'Form\\ValidaController@validar')->name('especie.validar');
        Route::post('/{especie}/desvalidar', 'Form\\ValidaController@desvalidar')->name('especie.desvalidar');
    });
    // apagar
    Route::delete('/apagar/{especie}','Form\\EspecieController@delete')->name('especie.delete');
    //restaurar
    Route::post('/restaurar/{especie}','Form\\EspecieController@restore')->name('especie.restore');
    // pdf
    Route::get('/pdf/{especie}', 'Form\\EspecieController@pdf')->name('especie.pdf');
```

Fonte: Autor

4.5 Requests

As Requests compõem uma parte essencial da arquitetura MVC.

> A Request ou requisição traduzindo diretamente para português, é o pedido que um cliente realiza a nosso servidor. Esse pedido contém uma série de dados que são usados para descrever exatamente o que o cliente precisa. Vamos pensar que um cliente precisa cadastrar um novo produto, ele deve passar todos os dados necessários para o cadastro acontecer de maneira correta, inclusive os dados que foram digitados pelo usuário em um formulário, no caso de uma aplicação web.
>
> No navegador toda vez que trocamos de página ou apertamos enter na barra de endereço uma nova request é feita. Independente se estamos apenas pedindo a exibição de uma página, cadastrando um novo recurso, atualizando ou excluindo (FONSECA, 2019).

Vale ressaltar os métodos HTTP utilizados para fazer essas requisições, são eles:

- **GET**: quando o cliente solicita recursos do servidor;
- **POST**: quando o cliente envia dados para o servidor;

Dessa forma, as request trabalham intimamente com as Routes, pois como é possível observar na implementação do código, cada route possui um método agregado responsável por realizar a solicitação daquela rota para a aplicação seguindo a arquitetura MVC. Na imagem abaixo é possível observar as requests observadas no sistema.

Figura 30: Pasta Requests

Fonte: Autor

Como cada arquivo apresenta as mesmas funções, estas serão especificadas de antemão e em sequência serão apresentados os códigos referentes a cada arquivo listado na imagem acima.

Em detalhes, possui as seguintes funções:

- **public function authorize()**: método de solicitação de formulários;
- **function rules()**: é especificado as regras de validação para cada atributo referente a classe especie. Nela podemos observar as seguintes validações utilizadas neste sistema:
    - **alpha**: o campo deve ser preenchido unicamente com caracteres alfabéticos;
    - **between:min,max**: o campo deve estar preenchido com caracteres dentro do intervalo min e max estabelecidos;
    - **date**: o campo deve ser preenchido com uma data válida;
    - **exists:table,column**: se o nome da coluna não for especificado, o nome do campo será usado;
    - **image**: o arquivo deve ser uma imagem (jpeg, png, bmp, gif, svg ou webp);
    - **nullable**: o campo pode ser *null*;
    - **required**: o campo deve estar preenchido e não deve estar vazio.
- **public funtion messages()**: utilizado para traduzir as mensagens de erros do inglês para o português dos arquivos de tradução;

A seguir estão presentes as imagens correspondentes aos arquivos.

1. **EspecieRequest.php**: request para a classe especie.

Figura 31: EspecieRequest.php

```php
<?php

namespace App\Http\Requests;

use Illuminate\Foundation\Http\FormRequest;

class EspecieRequest extends FormRequest
{

    public function authorize()
    {
        return true;
    }

    public function rules()
    {
        return [
            'genero' => 'required|alpha',
            'epiteto' => 'required|alpha',
            'grupo' => 'required|alpha',
            'combo' => 'required|alpha',
            'ordem' => 'required|alpha',
            'familia' => 'required|alpha',
            'local' => 'required|alpha',
            'coletor' => 'required|alpha',
            'data' => 'required|date',
            'responsavel' => 'alpha',
            'obs' => 'nullable',
            'seqgen' => 'nullable',
            'foto'=>'nullable|image',
        ];
    }

    public function messages() {
        return trans('validation');
    }
}
```

Fonte: Autor

2. **UserFuncaoRequest.php**:

Figura 32: UserFuncaoRequest.php

```php
<?php

namespace App\Http\Requests;

use Illuminate\Foundation\Http\FormRequest;

class UserFuncaoRequest extends FormRequest
{
    public function authorize()
    {
        return true;
    }

    public function rules()
    {
        return [
            'id_funcao' => 'required|exists:funcao,id',
        ];
    }

    public function messages() {
        return trans('validation');
    }
}
```

Fonte: Autor

3. **UserRequest.php**:

Figura 33: UserRequest.php

```php
<?php

namespace App\Http\Requests;

use Illuminate\Foundation\Http\FormRequest;

class UserRequest extends FormRequest
{
    public function authorize()
    {
        return true;
    }

    public function rules()
    {
        return [
            'name' => 'required|between:3,35',
            'senha_nova' => 'between:8,20|nullable',
            'password' => 'required|between:8,20',
        ];
    }

    public function messages() {
        return trans('validation');
    }
}
```

Fonte: Autor

4.6 Views

O laravel utiliza uma engine para gerenciar o template que é mostrado para o usuário final do sistema, neste caso o Blade. Ele é utilizado para a construção das views e é identificado utilizando a seguinte extensão (nome).blade.php, ao utilizar o blade, é permitido o uso de HTML/PHP é uma mesma página, isso ocasiona uma estrutura que dá a possibilidade de utilizar mais recursos do que somente uma página HTML pura.

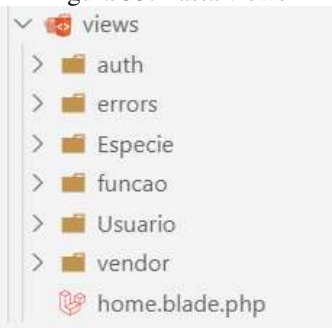
Observando o aspecto levantado anteriormente, nessa questão foi utilizado o framework Bootstrap para a elaboração da interface final em conjunto com o template gratuito AdminLTE v.2.0 onde foi utilizado os componentes para alcançar uma melhor fluidez entre todas as funcionalidades que o sistema oferece além de melhorar a qualidade gráfica do produto final.

Figura 34: Template AdminLTE v2.0

Fonte: Autor

As views na estrutura de pastas do laravel são responsáveis por armazenar os arquivos que correspondem ao front-end da aplicação. Na figura 35 pode observar como foi feita a divisão das blades por classe para facilitar o processo de desenvolvimento das telas.

Figura 35: Pasta views



Fonte: Autor

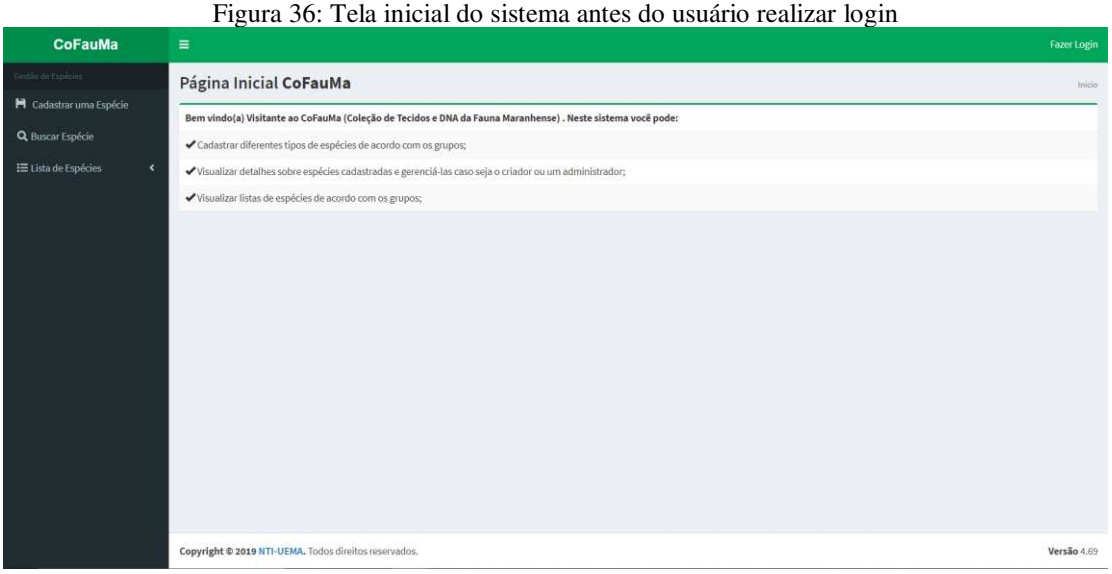
Como cada arquivo dentro da pasta views corresponde uma respectiva tela, cada pasta será apresentada sob a forma de um tópico afim obter uma distribuição destas ao longo do trabalho.

4.6.1 Arquivo home.blade.php

O arquivo *home.blade.php* é responsável pela tela Home do sistema, em outras palavras, a tela inicial quando o usuário acessa o link para o sistema CoFauMa, como existe um sistema de níveis no sistema, quando um usuário efetua login, ele é redirecionado para uma tela inicial condizente com o seu nível de acesso.

1. **Tela Home**: É a tela inicial do sistema apresentada a qualquer usuário. Nela é possível observar as funções comuns a todos os níveis de acesso, funções essas listadas no menu lateral esquerdo da imagem a seguir, assim como é apresentado através do dashboard uma lista informando ao usuário o que ele pode fazer no sistema.

Figura 36: Tela inicial do sistema antes do usuário realizar login



Fonte: Autor

Nessa tela foram desenvolvidas as seguintes funções:

- **Fazer Login**: redireciona o usuário para a tela de login do sistema.

2. Tela inicial dos níveis master e administrador: ambos os níveis possuem uma tela com as mesmas funções e configurações.

Figura 37: Tela inicial dos níveis master/administrador

Fonte: Autor

Nessa tela foram desenvolvidas as seguintes funções:

- **Lista de usuários**: exibe uma tela contendo uma lista com todos os usuários cadastrados no sistema;
- **Funções dos Usuários**: Exibe uma tela contendo uma lista com os usuários cadastrados e seus respectivos níveis de acesso;
- **Usuários Deletados**: exibe uma tela contendo uma lista com os usuários deletados do sistema;
- **Cadastrar uma Espécie**: exibe a tela de cadastro de uma espécie;
- **Buscar uma Espécie**: exibe a tela de pesquisa de uma espécie;
- **Lista de Espécies**: menu de filtros contendo a função de exibe uma tela contendo a lista de espécies cadastradas a partir dos filtros de seleção escolhido;
- **Espécies Não Validadas**: corresponde as mesmas funcionalidades de ‘Lista de Espécies’ com a diferença de que as espécies exibidas na lista serão aquelas com a situação de ‘Não validadas’;

- **Espécies Deletadas**: corresponde as mesmas funcionalidades de ‘Lista de Espécies’ com a diferença de que as espécies exibidas na lista serão aquelas com a situação de ‘Deletadas’;
- **Ver Lista**: implementado como meio de atalho rápido as listas em seus respectivos blocos do dashboard;
- **Indicadores Quantitativos**: indicam a quantidade exata de itens por lista de acordo com seus respectivos blocos no dashboard.

3. Tela inicial do nível comum.

Figura 38: Tela inicial do nível comum

Fonte: Autor

Nessa tela foram desenvolvidas as seguintes funções:

- **Cadastrar uma Espécie**: exibe a tela de cadastro de uma espécie;
- **Buscar uma Espécie**: exibe a tela de pesquisa de uma espécie;
- **Lista de Espécies**: menu de filtros contendo a função de exibe uma tela contendo a lista de espécies cadastradas a partir dos filtros de seleção escolhido;
- **Ver Lista**: implementado como meio de atalho rápido a lista em seu respectivo bloco do dashboard;
- **Sair**: executa o logout do sistema.

4.6.2 Pasta auth

A pasta auth contém os arquivos responsáveis pelas telas de login e registro do usuário no sistema.

Figura 39: Pasta auth

Fonte: Autor

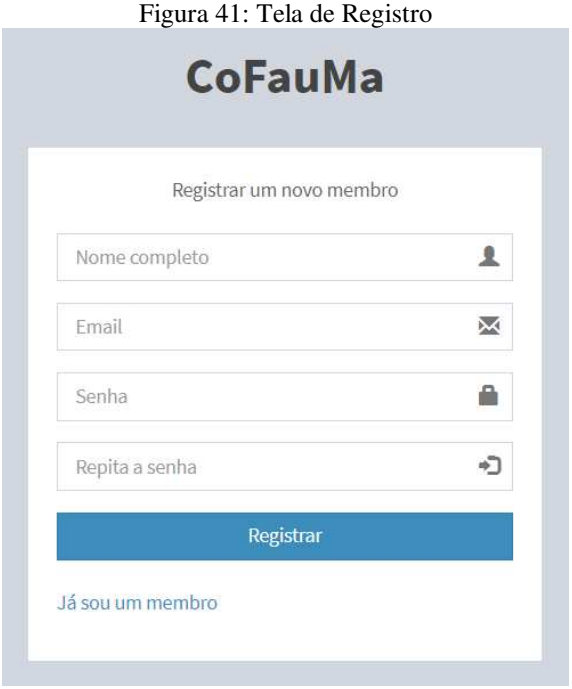
1. **login.blade.php**: Na tela de login o usuário informa suas credenciais de acesso, sendo estas, o e-mail e a senha previamente escolhidos no momento do registro do mesmo no sistema, essa tela possui um processo de validação informando caso as credenciais forem incorretas. Além de um link para a tela de registro caso o usuário não possua cadastro no sistema.

Figura 40: Tela de Login

Fonte: Autor

2. **registro.blade.php**: Na tela de registro, o usuário informa seu nome completo, e-mail e senha, para realizar o cadastro no sistema, caso ele tenha cadastro a mesma tela possui um link para a tela de login.

Figura 41: Tela de Registro

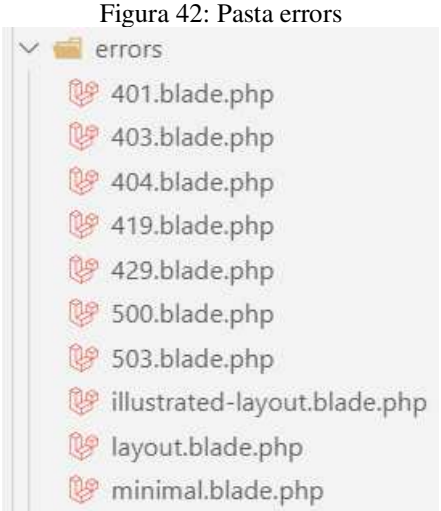


Fonte: Autor

4.6.3 Pasta errors

A pasta erros contém os arquivos responsáveis pelas mensagens de erros que podem aparecer dependendo do acesso do usuário a alguma funcionalidade do sistema.

Figura 42: Pasta errors



Fonte: Autor

Cada arquivo desta pasta representa uma tela semelhante, com a diferença de que na mensagem exibida será mostrado o número referente ao erro. Na imagem abaixo pode ser observada a tela de apresentação dos erros para os usuários.

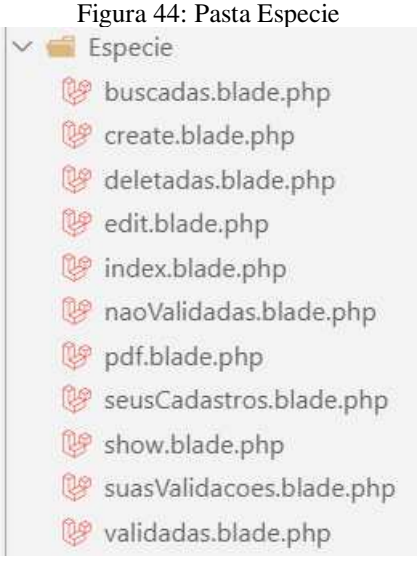
Figura 43: Tela de erro 403

Fonte: Autor

4.6.4 Pasta Especie

A pasta Especie contém os arquivos responsáveis pelas telas que o envolvem o processo de construção do catálogo de espécies e a visualização do mesmo para os demais usuários, assim como as operações especificadas no levantamento de requisitos e modelagem.

Figura 44: Pasta Especie



Fonte: Autor

1. **buscadas.blade.php**: Essa tela pode ser acessada pelos níveis master/administrador/comum, nela pode-se efetuar a pesquisa de uma espécie específica.

Figura 45: Tela Buscar Espécie

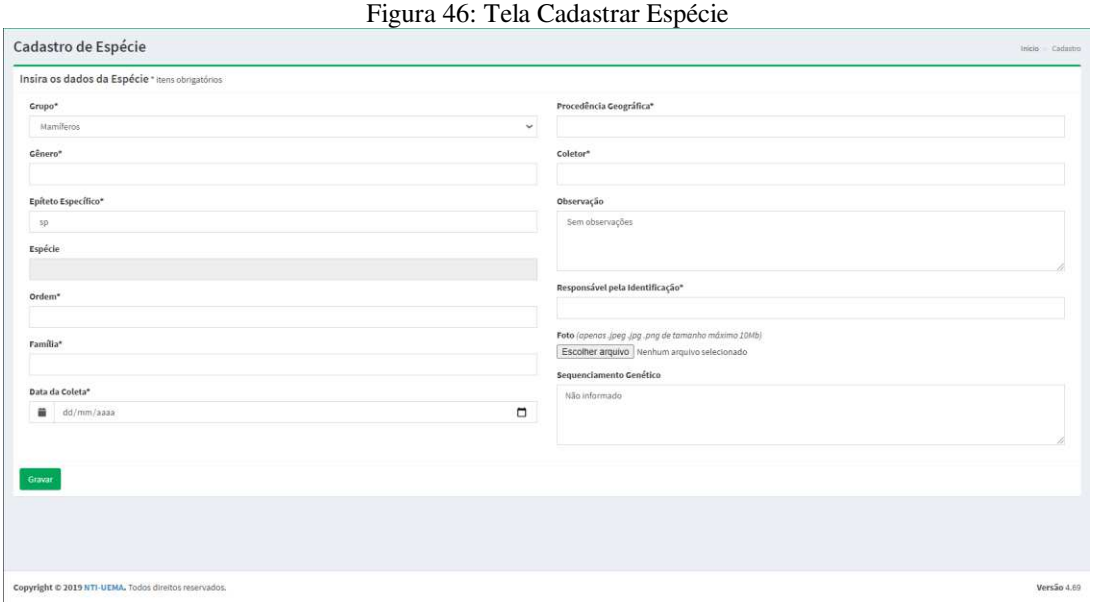
Fonte: Autor

Nessa tela foram desenvolvidas as seguintes funções:

- **Digite o Código**: pesquisa todas as espécies do banco de dados e retorna aquela em que corresponde aos dados fornecidos, caso contrário informa que foi possível encontrar nenhuma espécie;
- **Ordene sua busca**: foi implementada com intuito de auxiliar na organização da lista de usuário cadastrados no sistema, a ordenação pode ser realizada por espécie, código, grupo, ordem, família e data de criação.
- **Método**: foi implementado para organizar a lista de usuários em ordem ascendente ou descendente de acordo com o parâmetro escolhido na função 'Ordene sua Busca';
- **Itens por Lista**: foi implementado para escolher o número máximo de itens exibidos na tela, sendo eles: 25, 50, 100 e 200;
- **Listar**: Botão de confirmação de escolhas das configurações de exibição da lista de espécies cadastradas;
- **Zerar lista**: retorna à exibição original da lista de espécies cadastradas;
- **Visualizar**: redireciona para a tela 'Exibição da Espécie'.

2. **create.blade.php**: Nesta tela apresenta o formulário de cadastro de uma espécie. Este cadastro possui validações apontadas no processo de levantamento de requisitos e modelagem do sistema. Esta é uma tela comum a todos os níveis de acesso, em outras palavras qualquer usuário que utilize do sistema até a presente versão pode efetuar o cadastro de uma nova espécie no catálogo;

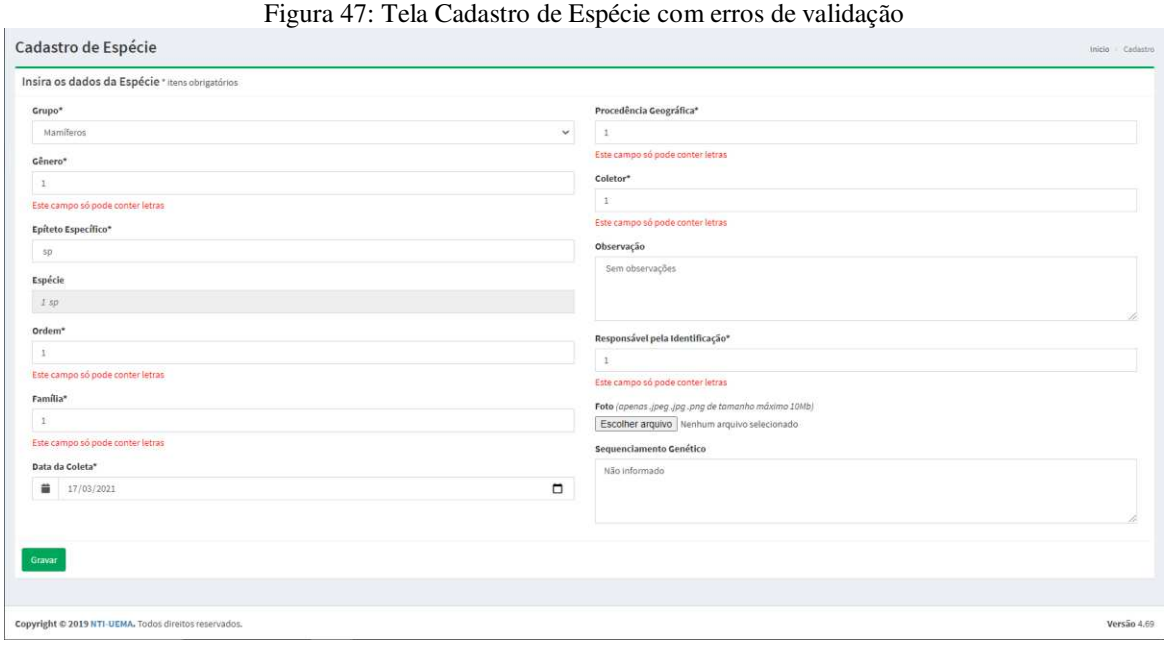
Figura 46: Tela Cadastrar Espécie



Fonte: Autor

Nessa tela todos os campos com o símbolo (*) são obrigatórios no momento do cadastro das informações e os demais itens não necessitam que sejam informados para a conclusão do processo. Na imagem a seguir é apresentado um exemplo do preenchimento incorreto dos campos do cadastro e a forma como são apresentadas as validações para o usuário.

Figura 47: Tela Cadastro de Espécie com erros de validação



Fonte: Autor

3. **deletadas.blade.php**: Essa tela pode ser acessada pelos níveis master/administrador/comum. É uma tela geral que lista todos os cadastros de espécies que estejam com situação ‘Deletada’ de acordo com o que foi estabelecido no levantamento de requisitos.

Figura 48: Tela Lista de Espécies Deletadas

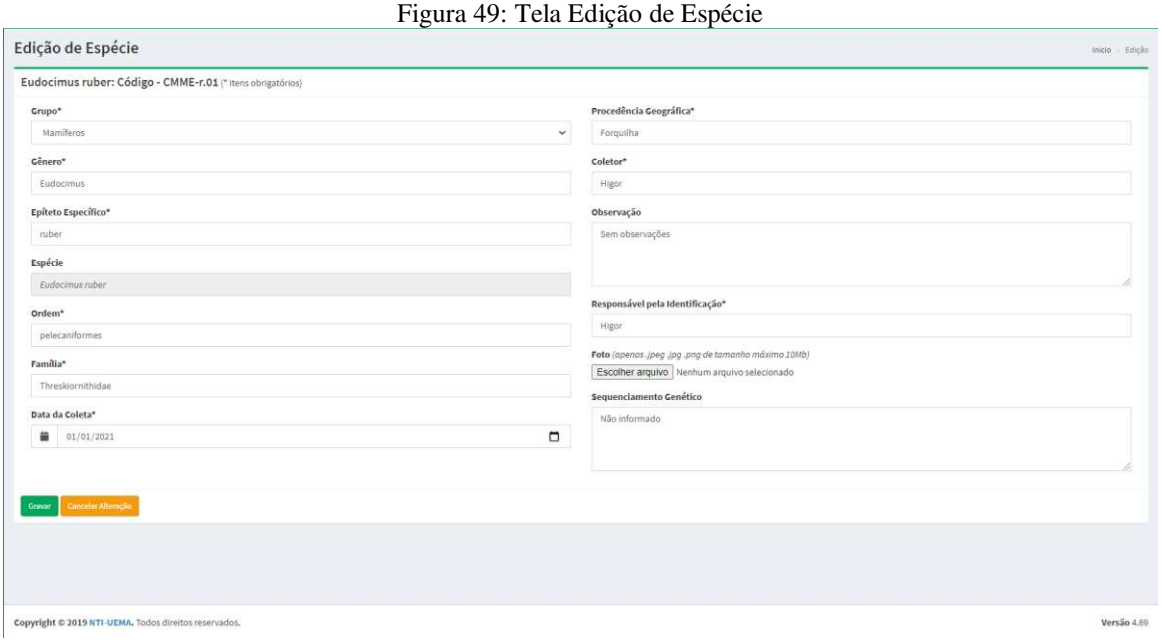
Fonte: Autor

Nessa tela foram desenvolvidas as seguintes funções:

- **Ordene sua busca**: foi implementada com intuito de auxiliar na organização da lista de usuário cadastrados no sistema, a ordenação pode ser realizada por espécie, código, grupo, ordem, família e data de criação.
- **Método**: foi implementado para organizar a lista de usuários em ordem ascendente ou descendente de acordo com o parâmetro escolhido na função 'Ordene sua Busca';
- **Itens por Lista**: foi implementado para escolher o número máximo de itens exibidos na tela, sendo eles: 25, 50, 100 e 200;
- **Listar**: Botão de confirmação de escolhas das configurações de exibição da lista de espécies cadastradas;
- **Zerar lista**: retorna à exibição original da lista de espécies cadastradas;
- **Restaurar**: restaura a espécie para a lista geral e elimina da lista de espécies deletadas.

4. **edit.blade.php**: Essa tela pode ser acessada pelos níveis master/administrador/comum, possui as mesmas características da tela de cadastro, sendo assim sujeita às mesmas validações, além de que no momento em que uma espécie sofre alteração o nome da conta é registrado no 'Histórico da Espécie' para fins de auditoria. Para finalizar o processo, é necessário confirmar as alterações ao confirmar em *Gravar*.

Figura 49: Tela Edição de Espécie



Fonte: Autor

5. **index.blade.php**: Essa tela pode ser acessada pelos níveis master/administrador/comum. É uma tela geral que lista todos os cadastros de espécies.

Figura 50: Tela Lista de Espécies

Fonte: Autor

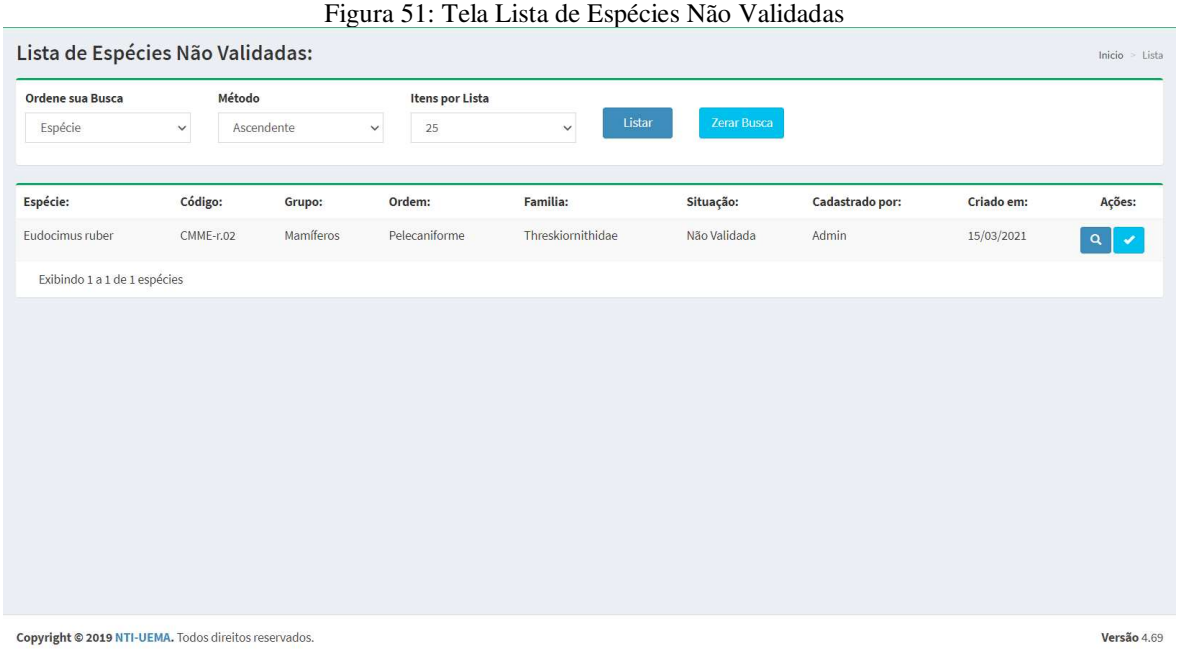
Nessa tela foram desenvolvidas as seguintes funções:

- **Ordene sua busca**: foi implementada com intuito de auxiliar na organização da lista de usuário cadastrados no sistema, a ordenação pode ser realizada por espécie, código, grupo, ordem, família e data de criação.

- **Método**: foi implementado para organizar a lista de usuários em ordem ascendente ou descendente de acordo com o parâmetro escolhido na função ‘Ordene sua Busca’;
- **Itens por Lista**: foi implementado para escolher o número máximo de itens exibidos na tela, sendo eles: 25, 50, 100 e 200;
- **Listar**: Botão de confirmação de escolhas das configurações de exibição da lista de espécies cadastrados;
- **Zerar lista**: retorna a exibição original da lista de espécies cadastrados;
- **Visualizar**: redireciona para a tela ‘Exibição da Espécie’.

6. **naoValidadas.blade.php**: Essa tela pode ser acessada pelos níveis master/administrador. É uma tela geral que lista todos os cadastros de espécies que estejam com situação ‘Não Validada’ de acordo com o que foi estabelecido no levantamento de requisitos.

Figura 51: Tela Lista de Espécies Não Validadas



Fonte: Autor

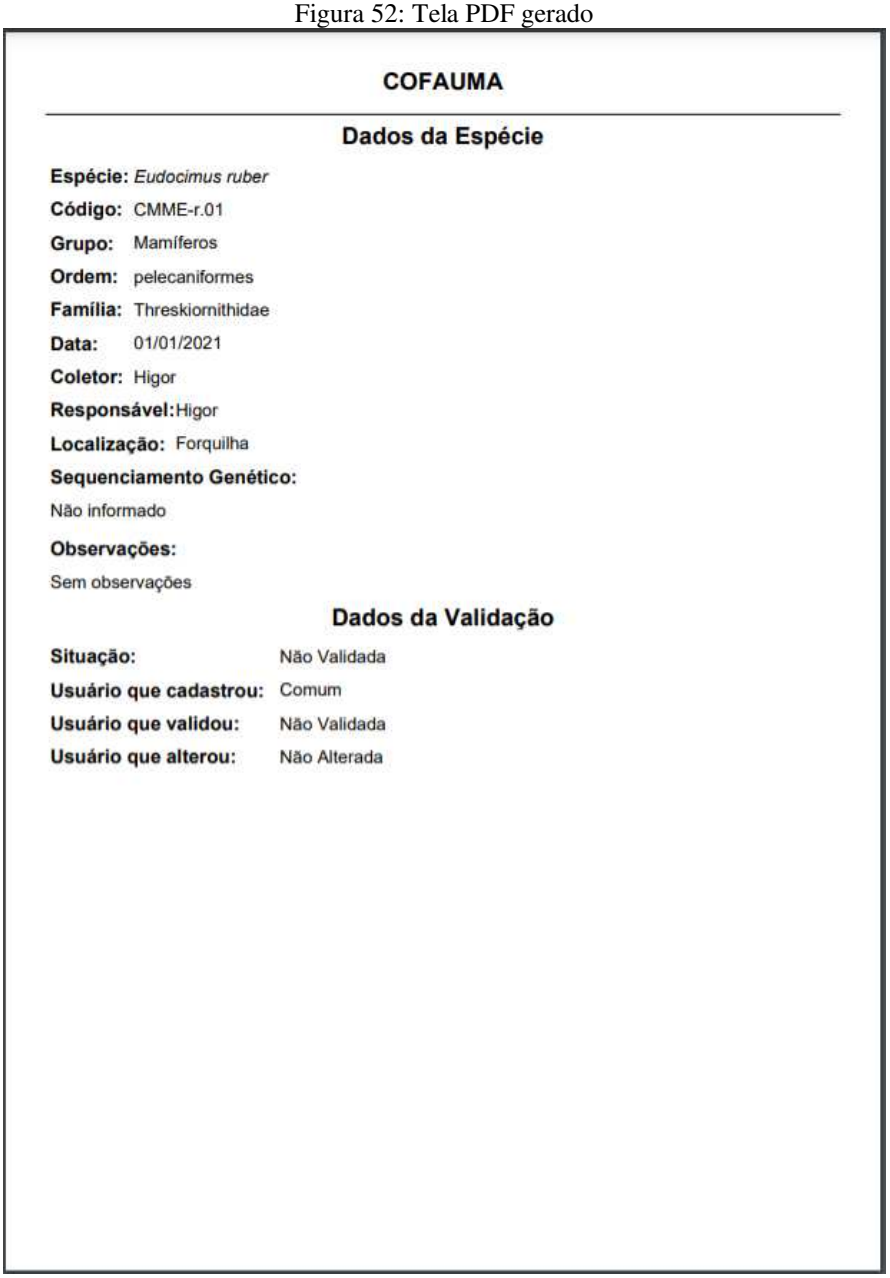
Nessa tela foram desenvolvidas as seguintes funções:

- **Ordene sua busca**: foi implementada com intuito de auxiliar na organização da lista de usuário cadastrados no sistema, a ordenação pode ser realizada por espécie, código, grupo, ordem, família e data de criação.
- **Método**: foi implementado para organizar a lista de usuários em ordem ascendente ou descendente de acordo com o parâmetro escolhido na função ‘Ordene sua Busca’;

- **Itens por Lista**: foi implementado para escolher o número máximo de itens exibidos na tela, sendo eles: 25, 50, 100 e 200;
- **Listar**: Botão de confirmação de escolhas das configurações de exibição da lista de espécies cadastrados;
- **Zerar lista**: retorna à exibição original da lista de usuários cadastrados;
- **Detalhes**: redireciona para a tela ‘Exibição da Espécie’;
- **Validar**: altera a situação para ‘Validada’ e move a espécie para a lista de espécies geral.

7. **pdf.blade.php**: Essa tela pode ser acessada pelos níveis master/administrador/comum. É disponibilizado para o usuário na sessão ‘Zona de perigo’ da tela de ‘Exibição de uma Espécie’ ao selecionar a função de ‘Gerar PDF’. Feito esse processo, seguindo a explicação da funcionalidade, é exibido uma nova aba no navegador de internet ao qual o usuário esta utilizando para acessar o sistema, e nessa aba um arquivo PDF é exibido contendo as informações das sessões de ‘Ficha de cadastro’ e ‘Histórico da Espécie’. Para um melhor entendimento, a imagem a seguir descreve um PDF gerado do exemplo de cadastro de uma espécie realizada no sistema.

Figura 52: Tela PDF gerado



COFAUMA

Dados da Espécie

Espécie: *Eudocimus ruber*
Código: CMME-r.01
Grupo: Mamíferos
Ordem: pelecaniformes
Família: Threskiornithidae
Data: 01/01/2021
Coletor: Higor
Responsável: Higor
Localização: Forquilha
Sequenciamento Genético:
Não informado
Observações:
Sem observações

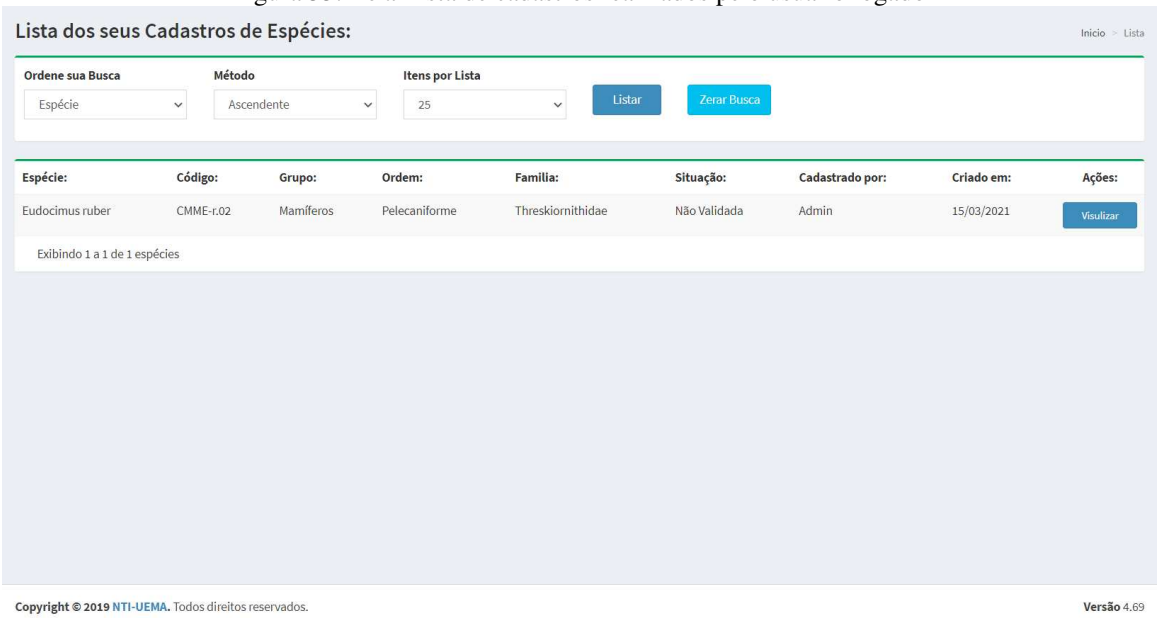
Dados da Validação

Situação: Não Validada
Usuário que cadastrou: Comum
Usuário que validou: Não Validada
Usuário que alterou: Não Alterada

Fonte: Autor

8. **seusCadastros.blade.php**: Essa tela pode ser acessada pelos níveis master/administrador. É uma tela geral que lista todos os cadastros de espécies realizados por aquele usuário.

Figura 53: Tela Lista de cadastros realizados pelo usuário logado



Fonte: Autor

Nessa tela foram desenvolvidas as seguintes funções:

- **Ordene sua busca**: foi implementada com intuito de auxiliar na organização da lista de usuário cadastrados no sistema, a ordenação pode ser realizada por espécie, código, grupo, ordem, família e data de criação.
- **Método**: foi implementado para organizar a lista de usuários em ordem ascendente ou descendente de acordo com o parâmetro escolhido na função 'Ordene sua Busca';
- **Itens por Lista**: foi implementado para escolher o número máximo de itens exibidos na tela, sendo eles: 25, 50, 100 e 200;
- **Listar**: Botão de confirmação de escolhas das configurações de exibição da lista de espécies cadastrados;
- **Zerar lista**: retorna à exibição original da lista de espécies cadastrados;
- **Visualizar**: redireciona para a tela 'Exibição da Espécie'.

9. **show.blade.php**: tela de exibição da espécie, onde informa os dados cadastrados e funções que podem ser realizadas com esse cadastro. Esta tela é dividida em três sessões distintas, sendo elas:

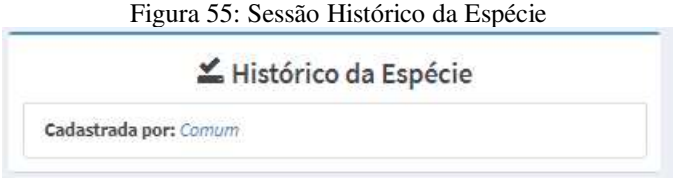
- **Ficha de cadastro**: Contém as informações cadastradas da espécie, assim como a data em que foi criada e a situação em que se encontra. Esta sessão é comum a todos os níveis de acesso;

Figura 54: Sessão de Exibição da Espécie

Fonte: Autor

- **Histórico da Espécie**: Contém os logs que declaram qual usuário realizou determinada função sobre a ficha.

Figura 55: Sessão Histórico da Espécie



Fonte: Autor

Nessa tela indica a situação da ficha de cadastro e qual usuário responsável por esta situação, no exemplo acima, o histórico indica que a ficha foi cadastrada por um usuário cujo nome é Comum.

- **Zona de Perigo**: Contêm funções específicas que podem ser realizadas com a ficha de cadastro, essas funções são para o usuário de acordo com o nível de acesso da conta.
    - Nível master ou administrador (Caso a situação da ficha seja ‘Não Validada’):

Figura 56: Sessão Zona de Perigo da Espécie Não Validada

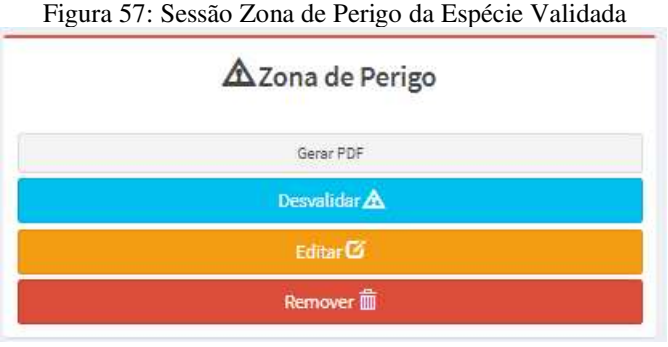
Fonte: Autor

Nessa sessão foram desenvolvidas as seguintes funções:

- **Gerar PDF**: emite um relatório em PDF contendo as informações da ficha de cadastro e o histórico da espécie;
- **Validar**: altera a situação da ficha pra ‘Validada’;
- **Editar**: exibe a tela de alteração das informações contidas na ficha de cadastro;
- **Remover**: Exclui a ficha de cadastro e altera a situação da mesma para ‘Deletada’.

    - Nível master ou administrador (Caso a situação da ficha seja ‘Validada’):

Figura 57: Sessão Zona de Perigo da Espécie Validada



Fonte: Autor

Nessa sessão foram desenvolvidas as seguintes funções:

- **Gerar PDF**: emite um relatório em PDF contendo as informações da ficha de cadastro e o histórico da espécie;
- **Desvalidar**: altera a situação da ficha pra ‘Não validada;
- **Editar**: exibe a tela de alteração das informações contidas na ficha de cadastro;
- **Remover**: Exclui a ficha de cadastro e altera a situação da mesma para ‘Deletada’.

- Nível comum:

Figura 58: Sessão Zona de Perigo da Espécie do usuário nível comum

Fonte: Autor

Nessa sessão foram desenvolvidas as seguintes funções:

- **Gerar PDF**: emite um relatório em PDF contendo as informações da ficha de cadastro e o histórico da espécie;
- **Editar**: exibe a tela de alteração das informações contidas na ficha de cadastro;
- **Remover**: Exclui a ficha de cadastro e altera a situação da mesma para ‘Deletada’.

Ao combinar as três sessões, é gerada a tela que é apresentada de fato para o usuário, novamente a composição das mesmas se diferem de acordo com o nível de acesso, nas imagens a seguir, estarão representadas as composições finais para os níveis master/administrador e comum respectivamente.

Figura 59: Tela Exibição da Espécie – nível master/administrador

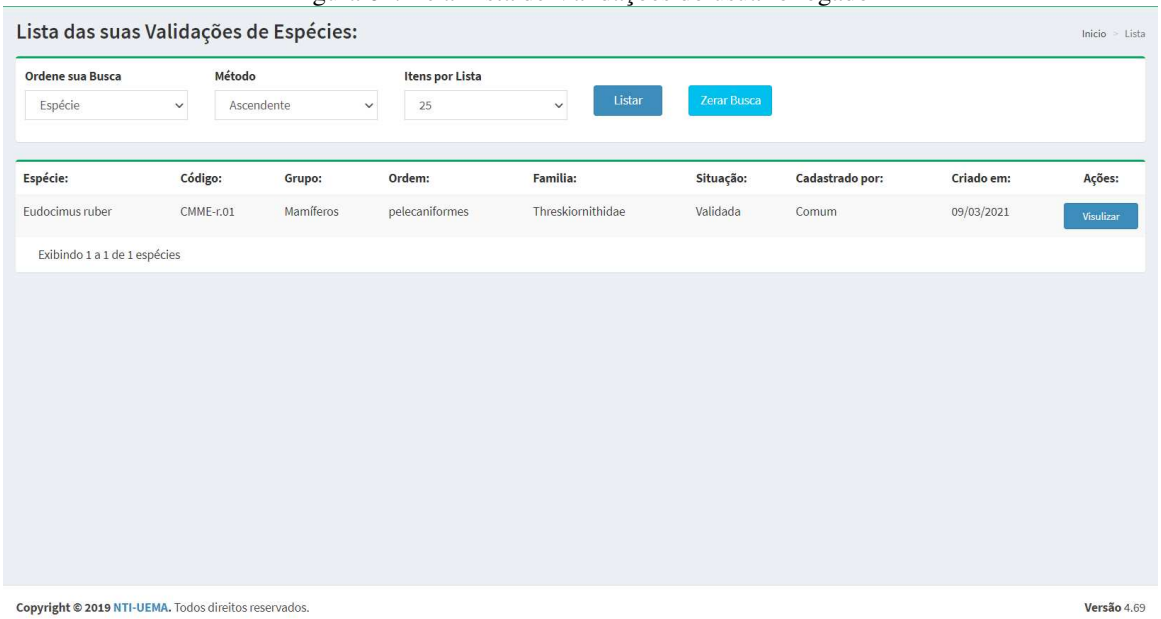
Fonte: Autor

Figura 60: Tela Exibição da Espécie – nível comum

Fonte: Autor

10. **suasValidacoes.blade.php**: Essa tela pode ser acessada pelos níveis master/administrador. É uma tela geral que lista todos as validações de cadastros realizados pelo usuário.

Figura 61: Tela Lista de Validações do usuário logado



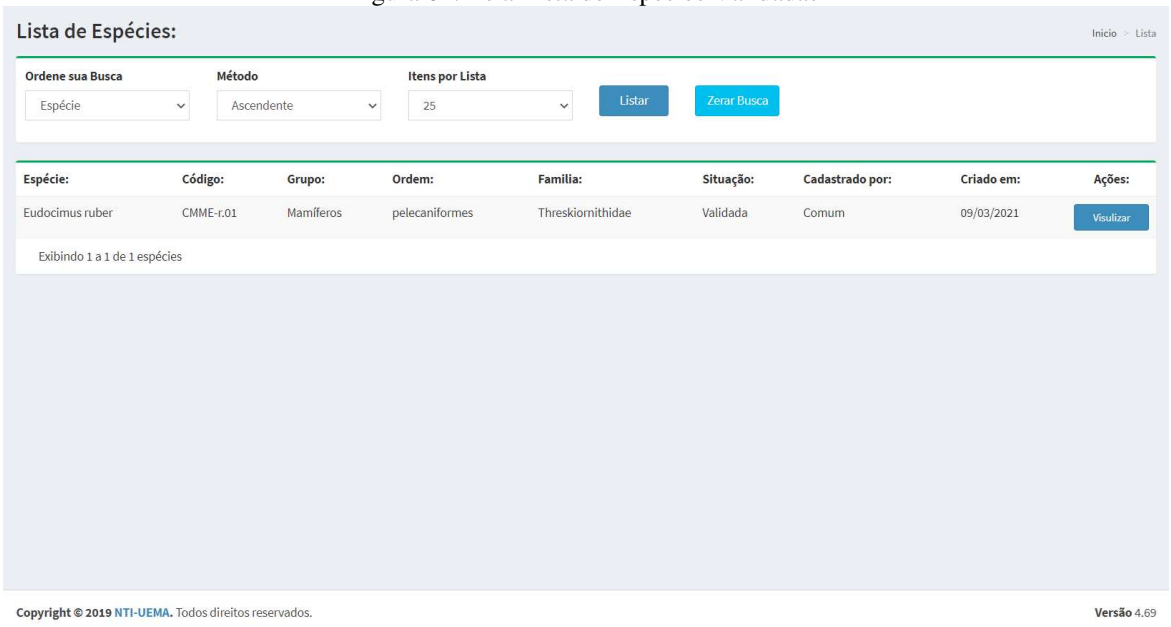
Fonte: Autor

Nessa tela foram desenvolvidas as seguintes funções:

- **Ordene sua busca**: foi implementada com intuito de auxiliar na organização da lista de usuário cadastrados no sistema, a ordenação pode ser realizada por espécie, código, grupo, ordem, família e data de criação.
- **Método**: foi implementado para organizar a lista de usuários em ordem ascendente ou descendente de acordo com o parâmetro escolhido na função 'Ordene sua Busca';
- **Itens por Lista**: foi implementado para escolher o número máximo de itens exibidos na tela, sendo eles: 25, 50, 100 e 200;
- **Listar**: Botão de confirmação de escolhas das configurações de exibição da lista de espécies cadastrados;
- **Zerar lista**: retorna à exibição original da lista de espécies cadastrados;
- **Visualizar**: redireciona para a tela 'Exibição da Espécie'.

11. **validadas.blade.php**: Essa tela pode ser acessada pelos níveis master/administrador. É uma tela geral que lista todos os cadastros de espécies que estejam com situação 'Validada' de acordo com o que foi estabelecido no levantamento de requisitos.

Figura 62: Tela Lista de Espécies Validadas



Fonte: Autor

Nessa tela foram desenvolvidas as seguintes funções:

- **Ordene sua busca**: foi implementada com intuito de auxiliar na organização da lista de usuário cadastrados no sistema, a ordenação pode ser realizada por espécie, código, grupo, ordem, família e data de criação.
- **Método**: foi implementado para organizar a lista de usuários em ordem ascendente ou descendente de acordo com o parâmetro escolhido na função 'Ordene sua Busca';
- **Itens por Lista**: foi implementado para escolher o número máximo de itens exibidos na tela, sendo eles: 25, 50, 100 e 200;
- **Listar**: Botão de confirmação de escolhas das configurações de exibição da lista de espécies cadastrados;
- **Zerar lista**: retorna à exibição original da lista de espécies cadastradas;
- **Visualizar**: redireciona para a tela 'Exibição da Espécie'.

4.6.5 Pasta funcao

A pasta função contém os arquivos responsáveis pelas telas de escolha do nível de acesso ao qual o usuário pode vir a ser vinculado.

Figura 63: Pasta funcao

Fonte: Autor

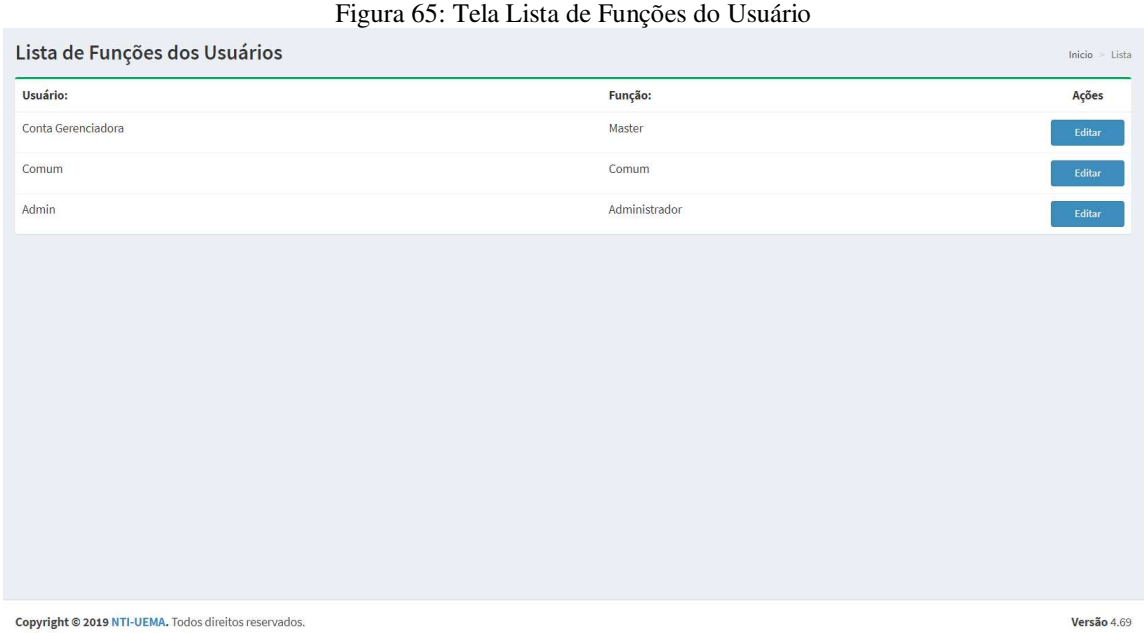
1. **edit.blade.php**: Nesta tela apresenta o formulário onde é possível escolher o nível de acesso (Comum, Administrador e Master) do usuário, assim como alterar posteriormente seguindo o mesmo procedimento. Para efetuar a escolha é necessário clicar no botão Gravar onde a escolha é submetida. Essa tela é pode ser acessada pelos níveis master e administrador.

Figura 64: Tela Edição de Função do Usuário

Fonte: Autor

2. **index.blade.php**: Essa tela pode ser acessada pelos níveis master/administrador. Nela pode ser observada a lista de todos os usuários cadastrados no sistema, assim como a função exercida por cada um.

Figura 65: Tela Lista de Funções do Usuário



Fonte: Autor

Nessa tela foram desenvolvidas as seguintes funções:

- **Visualizar**: exibe uma nova tela contendo as informações do usuário.

4.6.6 Pasta Usuario

A pasta Usuario contém os arquivos responsáveis pelas telas de funções referentes a gestão dos usuários no sistema.

Figura 66: Pasta Usuario

Fonte: Autor

1. **deletados.blade.php**: Essa tela pode ser acessada pelos níveis master/administrador. Nela pode ser observada a lista de todos os usuários que foram deletados do sistema, assim como algumas de suas informações, tais como o nome, o e-mail e a função.

Figura 67: Tela Lista de Usuários Deletados

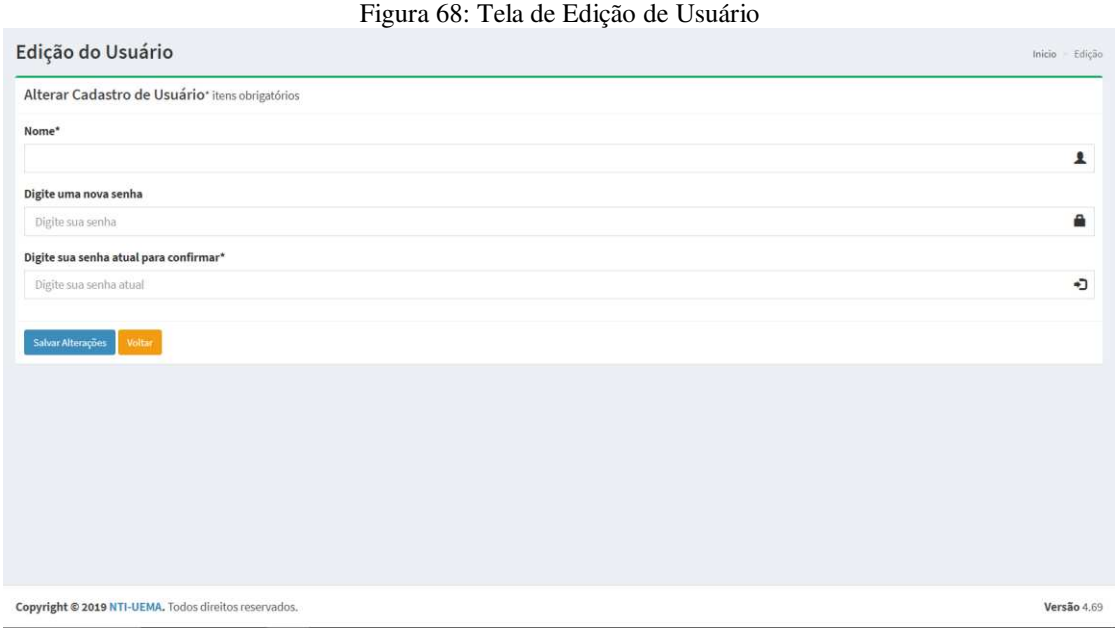
Fonte: Autor

Nessa tela foram desenvolvidas as seguintes funções:

- **Ordene sua busca**: foi implementada com intuito de auxiliar na organização da lista de usuário cadastrados no sistema, a ordenação pode ser realizada por nome, e-mail ou situação;
- **Método**: foi implementado para organizar a lista de usuários em ordem ascendente ou descendente de acordo com o parâmetro escolhido na função 'Ordene sua Busca';
- **Itens por Lista**: foi implementado para escolher o número máximo de itens exibidos na tela, sendo eles: 25, 50, 100 e 200;
- **Listar**: Botão de confirmação de escolhas das configurações de exibição da lista de usuários cadastrados;
- **Zerar lista**: retorna a exibição original da lista de usuários cadastrados;
- **Restaurar**: retorna as credenciais de acesso do usuário para o sistema como um todo.

2. **edit.blade.php**: tela onde o usuário pode alterar seu nome e/ou sua senha de acesso ao preencher o formulário e clicar em 'Salvar Alterações'.

Figura 68: Tela de Edição de Usuário



Fonte: Autor

3. **index.blade.php**: Essa tela pode ser acessada pelos níveis master/administrador, nela pode ser observada a lista de todos os usuários cadastrados no sistema, assim como algumas de suas informações, tais como o nome, o e-mail e a função.

Figura 69: Tela de Lista de Usuários

Fonte: Autor

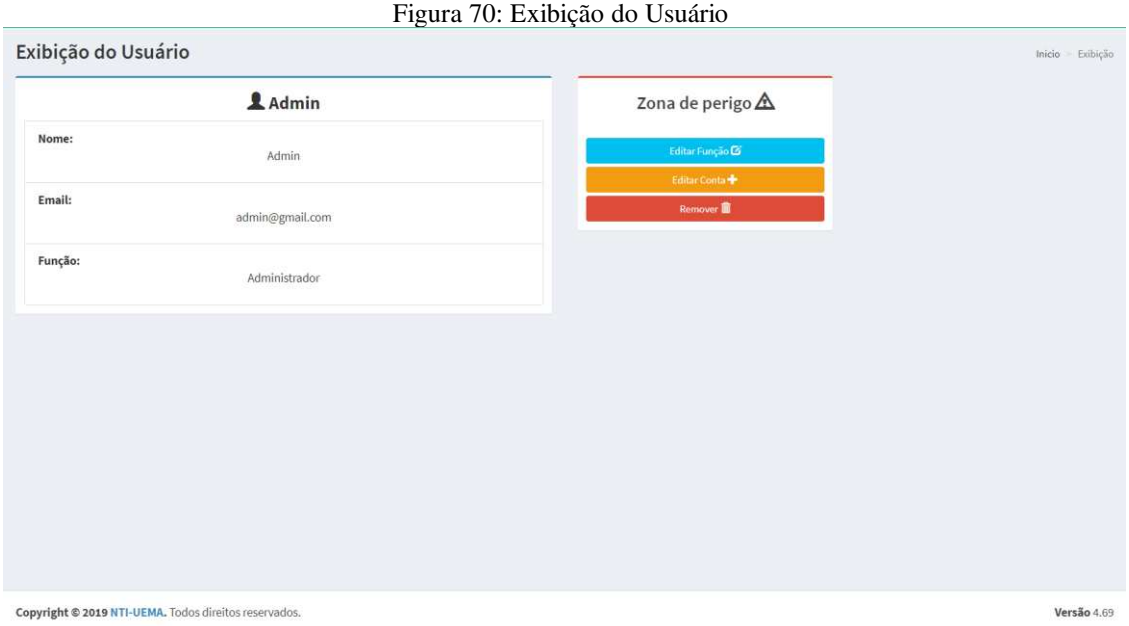
Nessa tela foram desenvolvidas as seguintes funções:

- **Ordene sua busca**: foi implementada com intuito de auxiliar na organização da lista de usuário cadastrados no sistema, a ordenação pode ser realizada por nome, e-mail ou situação;

- **Método**: foi implementado para organizar a lista de usuários em ordem ascendente ou descendente de acordo com o parâmetro escolhido na função ‘Ordene sua Busca’;
- **Itens por Lista**: foi implementado para escolher o número máximo de itens exibidos na tela, sendo eles: 25, 50, 100 e 200;
- **Listar**: Botão de confirmação de escolhas das configurações de exibição da lista de usuários cadastrados;
- **Zerar lista**: retorna a exibição original da lista de usuários cadastrados;
- **Visualizar**: exibe uma nova tela contendo as informações do usuário.

4. **show.blade.php**: tela onde o usuário pode acessar os seus dados cadastrados e sua função, assim como realizar determinadas ações.

Figura 70: Exibição do Usuário



Fonte: Autor

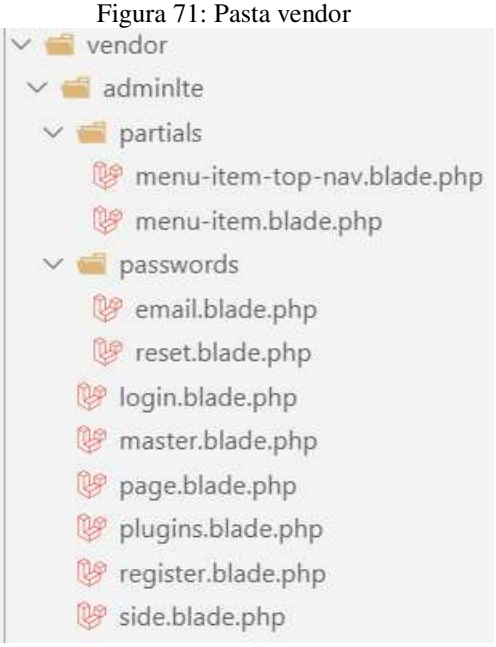
Nessa tela foram desenvolvidas as seguintes funções:

- **Editar Função**: exibe a tela onde é possível alterar o nível de acesso da conta o usuário;
- **Editar Conta**: redireciona para a tela de Edição do Usuário;
- **Remover**: excluir o usuário do sistema ao mesmo tempo em que o move para a lista de usuários deletados.

4.6.7 Pasta vendor

A pasta vendor contém os arquivos referentes ao template utilizado para a construção do frontend da aplicação, neste caso o AdminLTE v2.0, apresentado anteriormente. As views presentes nesta pasta correspondem especificamente aos componentes utilizados na construção das telas apresentadas anteriormente, tendo em vista isso não vem ao caso explicitar cada view.

Figura 71: Pasta vendor



Fonte: Autor

4.7 Segurança

A parte de segurança será realizada utilizando o próprio sistema de autenticação do laravel, pois este já implementado por padrão no framework dependendo de o desenvolvedor optar por utilizá-lo ou não. Dessa forma ao executar os comandos composer require laravel/ui "^1.0" –dev e php artisan ui vue –auth respectivamente, serão gerados os arquivos correspondentes do sistema de autenticação, ou seja, serão gerados as views para login e registro, as controllers e as routes correspondentes, assim como as funções para que possam ser utilizadas com os demais componentes da estrutura de pastas.

Vale ressaltar que as views, controllers e routes responsáveis pela segurança já foram devidamente explicados nos tópicos Database e Views.

4.7.1 Middleware

Segundo a documentação do Laravel v6.x:

> O middleware HTTP fornece um mecanismo conveniente para filtrar solicitações HTTP que entram em seu aplicativo. Por exemplo, o Laravel inclui um middleware que verifica se o usuário do seu aplicativo está autenticado. Se o usuário não estiver autenticado, o middleware redirecionará o usuário para a tela de login. No entanto, se o usuário for autenticado, o middleware permitirá que a solicitação prossiga no aplicativo.

Na imagem a seguir podemos observar quais middlewares foram utilizados no sistema de autenticação e proteção das rotas.

Figura 72: Pasta Middleware

Fonte: Autor

4.7.2 Policies

Segundo a documentação do Laravel v6.x, as policies são classes que organizam a lógica de autorização em torno de um modelo ou recurso específico. Sendo elas especificam as ações que um usuário está autorizado a executar. Na imagem a seguir estão presentes as policies utilizadas no sistema.

Figura 73: Pasta Policies

Fonte: Autor

- **EspeciePolicy.php**: especifica quais funções relacionadas ao processo de cadastro e gestão de uma espécie um usuário está autorizado a realizar.
- **UserPolicy.php**: especifica quais funções relacionadas ao processo de cadastro e gestão de um usuário, um dado usuário está autorizado a realizar sobre outro usuário.

5 CONCLUSÕES E TRABALHOS FUTUROS

O presente trabalho apresentou o desenvolvimento e implementação de um sistema para o projeto CoFauMa, do curso de Ciências Biológicas da Universidade Estadual do Maranhão. O foco principal deste trabalho, com a utilização dos conhecimentos adquiridos no decorrer da vivência estudantil como, por exemplo, Análise de Projetos, Modelagem de Sistemas, Levantamento de Requisitos, além de vários outros aprendizados durante a graduação, foi a modelagem e implementação dos módulos front-end e back-end.

O módulo front-end foi implementado utilizando os componentes extraídos do template AdminLTE v2.0 em conjunto com os recursos do Laravel e Bootstrap, com isso desenvolveu-se a interface gráfica do sistema. Já o módulo back-end, fez o uso da arquitetura MVC e estrutura de pastas do Laravel para implementar as regras de negócio e modelagem seguindo os preceitos da Engenharia de Software.

Portanto, ao analisar o conteúdo apresentado nos tópicos expostos, tanto a modelagem quanto o desenvolvimento do sistema atenderam aos objetivos e requisitos descritos no início deste trabalho. Assim é possível concluir que a concretização da proposta inicial do sistema e os processos de modelagem, neste caso os diagramas do sistema e modelo do Banco de Dados, foram criados e atendidos de acordo com as devidas normas e procedimentos a serem seguidos no desenvolvimento e implementação de um sistema e que este está pronto para ser utilizado pelo projeto e assim solucionar todas as situações problemáticas que levaram a elaboração do próprio sistema.

Para trabalhos futuros, pode-se realizar o acompanhamento da utilização do sistema por parte do projeto e assim coletar o feedback dos usuários afim implementar novas tecnologias e com isso implementar/otimizar funcionalidades com o intuito de obter ainda mais informações dos dados coletados. Um outro ponto a ser discutido nesse aspecto seria o desenvolvimento de aplicativo para melhorar o escopo de coleta das informações das espécies, este com o foco de ser offline, ou seja, o aplicativo poderia armazenar as informações de acordo com o modelo atual, tendo em vista que muitas áreas de coleta não possuem acesso a internet, e logo em seguida ao chegar em uma região com acesso realizar o upload das informações sem precisar ter que cadastrar novamente essas informações e com isso ganhar mais rapidez no processo de construção do catálogo.

REFERÊNCIAS

ADMINLTE. **AdminLTE Bootstrap Admin Dashboard Template**. Disponível em: https://adminlte.io/themes/AdminLTE/index2.html. Acesso: 18 de fev de 2021.

AMABIS, José Mariano; MARTHO, Gilberto Rodrigues. **Biologia Moderna: 2 Ensino Médio**. 1ª ed. São Paulo: Moderna, 2016.

BEZERRA, Eduardo. **Princípios de análise e projeto de sistemas com UML**. 2ª ed. Rio de Janeiro: Elsevier, 2007.

DATE, C. J. **Introdução a Sistemas de Banco de Dados**. 8ª ed. Rio de Janeiro: Elsevier, 2003.

ELMASRI, Ramez; NAVATHE, Shamkant B. **Sistemas de Banco de Dados**. 4ª ed. São Paulo: Pearson Education, 2005.

FERREIRA, Lucas Raphael Fernandes. **PROPOSTA DE IMPLEMENTAÇÃO DE SOFTWARE PARA GESTÃO DE DATASHOWS DO CENTRO DE CIÊNCIAS TECNOLÓGICAS DA UNIVERSIDADE ESTADUAL DO MARANHÃO**, 2019. Trabalho de Conclusão de Curso – Bacharel em Engenharia da Computação. Universidade Estadual do Maranhão, São Luís, 2019.

FONSECA, Elton. **O que é HTTP, Request, GET, POST, Response, 200, 404?** 2019. Disponível em: https://www.treinaweb.com.br/blog/o-que-e-http-request-get-post-response-200-404/. Acesso em: 12 de mar de 2021.

GUEDES, Gilleanes T. A. UML 2: Uma Abordagem Prática. 2ª ed. São Paulo: Novatec, 2011.

LARAVEL. **Documentation Laravel v6.x**. 2021. Disponível em: https://laravel.com/docs/6.x. Acesso em: 20 de fev de 2021.

LINHARES, Sérgio; GEWANDSNAJDER Fernando. Biologia Hoje: Os Seres Vivos. 2ª ed. São Paulo: Ática, 2013.

LOPES, Sônia. **BIO**. 1ª ed. São Paulo: Saraiva, 2004.

PRESSMAN, Roger S. **Engenharia de Software: Uma abordagem profissional**. 7ª ed. New York: McGraw-Hill, 2011.

RODRIGUES, Joel. **Modelo Entidade Relacionamento (MER) e Diagrama Entidade-Relacionamento (DER)**. 2014. Disponível em: https://www.devmedia.com.br/modelo-

entidade-relacionamento-mer-e-diagrama-entidade-relacionamento-der/14332. Acesso em: 1 de mar de 2021.

SOMMERVILLE, Ian. Engenharia de Software. 9ª ed. São Paulo: Pearson Education, 2011.

SOUSA, Luan F. A. **MODELAGEM DE UMA PROPOSTA DE SOFTWARE PARA GESTÃO DE APARELHOS DE DATASHOW NO CENTRO DE CIÊNCIAS TECNOLÓGICAS DA UNIVERSIDADE ESTADUALDO MARANHÃO**, 2019. Trabalho de Conclusão de Curso – Bacharel em Engenharia da Computação. Universidade Estadual do Maranhão, São Luís, 2019.

THE GROUP PHP. **O que é PHP?** 2021. Disponível em: https://www.php.net/manual/pt_BR/intro-whatis.php. Acesso em: 8 de mar de 2021.

TURINI, Rodrigo. **PHP e Laravel: Crie aplicações web como um verdadeiro artesão**. Casa do Código, 2015.