Gustavo Epichin Monjardim

Portal de Biodiversidade de Chalcidoidea

Vitória, ES

2019

Gustavo Epichin Monjardim

Portal de Biodiversidade de Chalcidoidea

Monografia apresentada ao Curso de Ciência da Computação do Departamento de Informática da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Bacharel em Ciência da Computação.

Universidade Federal do Espírito Santo – UFES

Centro Tecnológico

Departamento de Informática

Orientador: Prof. Dr. Vítor E. Silva Souza

Coorientador: Dra. Ana Dal Molin

Vitória, ES

2019

Gustavo Epichin Monjardim
Portal de Biodiversidade de Chalcidoidea/ Gustavo Epichin Monjardim. – Vitória, ES, 2019-
109 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Vítor E. Silva Souza

Coorientador: Dra. Ana Dal Molin

Monografia (PG) – Universidade Federal do Espírito Santo – UFES
Centro Tecnológico
Departamento de Informática, 2019.

1. Desenvolvimento Web. 2. Engenharia de Software. 3. Express.js. 4. Node.js. 5. JavaScript. 6. FrameWeb. 7. Chalcidoidea. 8. Chaves de Identificação. I. Souza, Vítor Estêvão Silva. II. Dal Molin, Ana. III. Universidade Federal do Espírito Santo. IV. Portal de Biodiversidade de Chalcidoidea

CDU 02:141:005.7

Gustavo Epichin Monjardim

Portal de Biodiversidade de Chalcidoidea

Monografia apresentada ao Curso de Ciência da Computação do Departamento de Informática da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Bacharel em Ciência da Computação.

Trabalho aprovado. Vitória, ES, 25 de setembro de 2019:

Prof. Dr. Vítor E. Silva Souza
Orientador

Dra. Ana Dal Molin
Coorientadora

Prof. Dr. João Paulo Andrade Almeida
Universidade Federal do Espírito Santo

Silas Louzada Campos
Universidade Federal do Espírito Santo

Vitória, ES
2019

Agradecimentos

Agradeço aos meus pais, Paulo e Elaine, por sempre me auxiliarem em tudo que precisei ao longo de todos esses anos de curso. Ao meu irmão Gabriel, que também optou por seguir os mesmos passos que o irmão mais velho. A todos os meus avós, pelo investimento na minha educação e por ajudarem a definir quem eu sou. A toda minha família, meu muito obrigado.

À minha namorada, melhor amiga e amor da minha vida, Thamyris, por ser a melhor companheira de todas e estar ao meu lado em todos os momentos, por todo o apoio e incentivo que me deu que foram essenciais para eu superar todos os desafios que enfrentei. Agradeço também a sua mãe e suas irmãs, por me acolherem e serem minha segunda família nesses últimos anos.

Agradeço ao professor Vítor, pela orientação neste trabalho e em projetos passados no laboratório NINFA, além de ter sido responsável por algumas de minhas disciplinas favoritas do curso. Agradeço também a todos os professores do Departamento de Informática da UFES, por todas as contribuições que fizeram para o meu aprendizado.

À Dra. Ana Dal Molin, que coorientou este trabalho e me ofereceu a oportunidade de desenvolver um projeto na área de Biologia, onde estive em contato com diversos conceitos que vão muito além da minha área de estudo, e que resultou no meu projeto de graduação. Agradeço também ao professor Marcelo Tavares e a todos do Laboratório de Biodiversidade de Insetos da UFES pela oportunidade de trabalhar nesse projeto,

Por fim, agradeço a todos os meus amigos, sejam os amigos que fiz durante a graduação, os amigos de longa data, companheiros de estágio. Todos contribuíram para que eu chegasse até aqui e fizeram minha graduação ser muito mais divertida.

Muito obrigado a todos!

Resumo

O desenvolvimento de chaves de identificação taxonômica tem um papel fundamental para a pesquisa científica em Biologia, uma vez que permitem que o fruto da pesquisa básica seja disponibilizado para que o não-especialista possa proceder à identificação de espécies. Uma das formas mais amigáveis para possibilitar esse tipo de uso são chaves digitais multi-entrada. Para o desenvolvimento dessas chaves, uma série de softwares já se encontram disponíveis. Esses softwares, porém, por via de regra, possuem código fechado, e só permitem a exportação das chaves desenvolvidas em formatos pouco amigáveis e com baixa portabilidade. Dessa forma, o Portal de Biodiversidade de Chalcidoidea foi concebido com o intuito de oferecer uma forma simples para taxônomos disponibilizarem em páginas Web as chaves de identificação desenvolvidas, de forma que as mesmas possam ser acessadas por usuários por meio de computadores e celulares. Além disso, o portal também oferece uma série de recursos sobre biodiversidade e taxonomia de Chalcidoidea, suas interações ecológicas e importância econômica.

Para a construção do sistema, seguiu-se um processo de Engenharia de Software, realizando as etapas de levantamento de requisitos, especificação e análise de requisitos, definição da arquitetura do sistema, implementação e testes. Também foram utilizados métodos e técnicas de modelagem e desenvolvimento orientado a objetos, em particular o método FrameWeb para projeto de aplicações Web baseadas em *frameworks*, que para esse projeto foi adaptado para o *framework* Express.js.

Palavras-chaves: Aplicação Web, Engenharia de Software, Express.js, Node.js, JavaScript, FrameWeb, Hymenoptera, Chaves de Identificação, Chave Multiacesso, Policlave, Taxonomia Biológica, Descrições de Espécies

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

AJAX	Javascript Assíncrono e XML, do inglês Asynchronous Javascript and XML
API	Interface de Programação de Aplicação, do inglês Application Programming Interface
CRUD	Create, Read, Update and Retrieve
HTML	Linguagem de Marcação de Hipertexto, do inglês HyperText Markup Language
HTTP	Protocolo de Transferência de Hipertexto, do inglês HyperText Transfer Protocol
JSON	JavaScript Object Notation
SDD	Structured Descriptive Data
UML	Unified Modeling Language
URL	Localizador Uniforme de Recursos, do inglês Uniform Resource Locator
XML	Extensible Markup Language
ORM	Mapeamento Objeto-Relacional, do inglês Object-Relational Mapping
CSS	Folha de Estilo em Cascatas, do inglês Cascading Style Sheets
MVC	Modelo-Visão-Controlador, do inglês Model-View-Controller

Sumário

1 Introdução

A identificação correta de organismos é um requerimento básico para o desenvolvimento de pesquisa científica em Biologia. Toda a informação fundamental associada à espécie é dependente de sua identificação taxonômica: inferências sobre ciclo de vida, relações evolutivas, impacto econômico, papel no ecossistema, particularidades fisiológicas e ecológicas.

Chacidoidea é uma das várias superfamílias de vespas parasitoides, que são majoritariamente conhecidas como inimigos naturais que ajudam a manter a estabilidade de populações de insetos que podem se tornar pragas agrícolas e florestais. Nesse contexto, a identificação de espécies também tem implicações econômicas, pois é necessário o uso das espécies e linhagens corretas para obter a eficácia desejada e prevenir, dentro do possível, efeitos colaterais. O uso de espécies identificadas erroneamente no passado resultou em prejuízos financeiros significativos e na falta de sucesso de programas de controle biológico.

O desenvolvimento de chaves de identificação taxonômicas é uma maneira do especialista no grupo de organismos possibilitar ao não-especialista que realize ao menos parte do trabalho de identificação. Para o desenvolvimento dessas chaves, uma série de softwares já se encontram disponíveis. Um dos softwares mais utilizados para este fim é o Lucid.[1] Outra ferramenta desenvolvida para estes fins desde os anos 80 é a plataforma Delta.[2]

A dificuldade atualmente encontrada com Lucid é que este é um programa de código fechado, que só permite a exportação de chaves no formato Java, o que dificulta seu uso, por exemplo, em celulares. Além disso, a conversão de chaves desenvolvidas em Lucid para o formato Web e como aplicativo de celular envolve alto custo financeiro. No entanto, com o avanço da pressão para que se produzam ferramentas científicas em formatos abertos, as chaves podem ser exportadas em um formato XML simples, chamado SDD.[3]

As convenções associadas ao formato SDD e outros baseados em XML são mantidas por uma equipe internacional envolvendo profissionais da Biologia e da Informática, por meio do *Taxonomic Database Working Group* (TDWG). Além de serem mais acessíveis, o fato destes formatos serem abertos e independentes de plataformas de software aumenta a estabilidade do produto e diminui os riscos associados à defasagem da versão dos programas usados. No entanto, não há uma ferramenta disponível que faça a conversão automática

[1] <http://www.lucidcentral.com/>.
[2] <https://www.delta-intkey.com/>.
[3] <https://www.github.com/tdwg/sdd/>.

dos formatos exportados por programas como Lucid e Delta para uma página Web de fácil navegação que possa ser usada por um usuário leigo.

Desta maneira, este trabalho tem o objetivo de desenvolver uma aplicação Web que permita ao taxônomo compartilhar chaves de identificação a partir de um arquivo em formato matricial ou XML. A chave disponibilizada dessa maneira poderá então ser utilizada pelo público em computador ou celular sem a necessidade do software originalmente utilizado para o preparo da chave. Além disso, o sistema permitirá ao taxônomo compartilhar textos, de modo a disponibilizar no portal informações sobre biodiversidade e taxonomia de Chalcidoidea, suas interações ecológicas e importância econômica, bem como a listagem de grupos de pesquisa, colaboradores e artigos publicados.

1.1 Objetivos

O objetivo geral do trabalho foi desenvolver um sistema Web que possibilite que informações geradas por taxônomos possam ser compartilhadas e acessadas facilmente. Dessa forma, o taxônomo é capaz de compartilhar ferramentas de identificação taxonômica, bem como informações sobre biodiversidade e taxonomia de Chalcidoidea, suas interações ecológicas e importância econômica.

Tal objetivo geral pode ser dividido nos seguintes objetivos específicos:

- Realizar o levantamento e análise de requisitos necessários, de modo a produzir documentos de especificação e análise de requisitos para compor a documentação do sistema;
- Definir a arquitetura do sistema utilizando o método FrameWeb (SOUZA, 2007), de modo a não só complementar a documentação do sistema com o documento de projeto arquitetural, mas também aplicar o método FrameWeb visando contribuir com sua pesquisa e desenvolvimento, atualmente em andamento;
- Implementar o sistema de acordo com a documentação produzida, utilizando tecnologias e *frameworks* modernos para o desenvolvimento Web;
- Hospedar a aplicação em um servidor para que fique disponível para o público por meio de uma URL de forma segura e eficiente.

1.2 Método de Desenvolvimento do Trabalho

O método utilizado para desenvolver este trabalho foi composto pelas seguintes atividades:

1. Revisão bibliográfica: estudo de boas práticas de Engenharia de Software (FALBO, 2014) e Requisitos (FALBO, 2017), Padrões de Projetos de Sistemas (FALBO, 2016; FOWLER, 2002), Programação Orientada a Objetos, uso e projeto de Banco de Dados Orientado a Documentos, entre outros;

2. Elaboração da documentação dos requisitos sistema: produção do Documento de Requisitos, apresentando uma descrição do sistema, de seu minimundo, definição dos requisitos funcionais e não funcionais, além das regras de negócio; e do Documento de Especificação de Requisitos, apresentando a identificação dos subsistemas, modelos de casos de uso, modelo estrutural, modelo dinâmico e glossário do projeto. Nesta atividade foram utilizados conceitos das disciplinas de Engenharia de Software e, em particular, Engenharia de Requisitos;

3. Elaboração da documentação da arquitetura do sistema: produção do Documento de Projeto de Sistema, apresentando a plataforma de desenvolvimento, atributos de qualidade e táticas, a arquitetura do software e projeto detalhado de cada um dos seus componentes, seguindo a abordagem FrameWeb (SOUZA, 2007). Esta atividade relaciona-se com as disciplinas de Projeto de Sistemas e Desenvolvimento Web e Web Semântica (optativa);

4. Estudo das tecnologias: estudo das tecnologias utilizadas para o desenvolvimento de APIs e interfaces para Web, tais como a linguagem de programação JavaScript, a plataforma Node.js, o *framework* Express.js, o banco de dados MongoDB, dentre outras;

5. Implementação do sistema e testes: desenvolvimento da aplicação Web acompanhando de testes, realizados sempre que uma nova funcionalidade for implementada, de modo a encontrar e corrigir possíveis erros. Nesta atividade exercitou-se conceitos das disciplinas de Programação, Linguagens de Programação, Banco de Dados e Desenvolvimento Web;

6. Implantação: hospedagem da aplicação em um servidor Web para ficar disponível para acesso ao público geral por meio de uma URL de forma segura e eficiente;

7. Redação da monografia: escrita da monografia em LaTeX[4] utilizando o editor TexStudio[5] e o template abnTeX[6] que atende os requisitos das normas da ABNT (Associação Brasileira de Normas Técnicas) para elaboração de documentos técnicos e científicos brasileiros.

[4] <https://www.latex-project.org/>.
[5] <https://www.texstudio.org/>.
[6] <https://www.abntex.net.br/>.

1.3 Organização do Texto

Esta monografia é estruturada em cinco partes e contém, além da presente introdução, os seguintes capítulos:

- **Capítulo 2** – Referencial Teórico: apresenta uma revisão da literatura acerca de temas relevantes ao contexto deste trabalho, a saber: Engenharia de Software, FrameWeb e desenvolvimento Web;
- **Capítulo 3** – Especificação de Requisitos: apresenta a especificação de requisitos do sistema, descrevendo o minimundo e exibindo os seus diagramas de classes e casos de uso;
- **Capítulo 4** – Projeto Arquitetural do sistema: apresenta a arquitetura do sistema, exibindo os modelos elaborados para cada camada do projeto utilizando o método FrameWeb;
- **Capítulo 5** – Implementação e Apresentação do sistema: apresenta detalhes da implementação do sistema, bem como capturas de tela do sistema em funcionamento;
- **Capítulo 6** – Considerações Finais: apresenta as conclusões do trabalho, dificuldades encontradas, limitações e propostas de trabalhos futuros;
- **Apêndices** – Apresenta os documentos gerados nos processos de Engenharia de Software (Documento de Requisitos, Documento de Projetos, etc.)

2 Referencial Teórico

Este capítulo apresenta os principais conceitos teóricos que fundamentaram o desenvolvimento do Portal de Biodiversidade de Chalcidoidea e está organizado em 4 seções. A Seção 2.1 apresenta a superfamília Chalcidoidea. A Seção 2.2 define o que são chaves de identificação. A Seção 2.3 aborda os principais conceitos e processos utilizados na Engenharia de Software. A Seção 2.4 apresenta conceitos importantes de desenvolvimento Web. Por fim, a Seção 2.5 apresenta o método FrameWeb.

2.1 Chalcidoidea

Chalcidoidea é uma das várias superfamílias de vespas conhecidas como "parasitoides", ou seja, cujo ciclo de vida inclui uma fase larval que se desenvolve dentro de ou sobre outro organismo, normalmente outro inseto, que é consumido para a emergência da vespa adulta. Ao contrário das vespas mais popularmente conhecidas, elas não constroem ninhos nem possuem ferrão. Como pode ser visto na Figura 1, são extremamente diversas morfologicamente, a maioria com 0,5 a 10mm de comprimento, sendo que este grupo inclui os menores insetos conhecidos (0,13mm). Estima-se que existam mais de 500.000 espécies, das quais, somente 22.506 já foram descritas (HERATY et al., 2013).

Chalcidoidea representa uma porção significante da diversidade biológica encontrada em ecossistemas terrestres e desempenham um papel fundamental na regulagem da população de insetos terrestres. A maioria dessas vespas são parasitas de outros insetos e são extensivamente usadas no controle biológico de pragas de insetos. De fato, a maioria dos projetos de controle biológico de pragas de insetos bem sucedidos utilizaram vespas Chalcidoidea para obter controle substancial ou completo (GIBSON et al., 1997). Assim, estas espécies possuem uma importância ecológica e econômica muito grande.

Devido a sua abundância, distribuição mundial, interações biológicas complexas, diversidade morfológica, e a pequena quantidade de taxonomistas que a estudam, ainda há muita pesquisa a ser desenvolvida sobre essa superfamília.

2.2 Chaves de Identificação

Uma chave taxonômica é basicamente uma série de perguntas sobre as características de um organismo desconhecido, essas perguntas são chamadas de **caracteres** da chave de identificação. Cada caractere oferece uma lista de alternativas que o usuário pode selecionar como resposta à pergunta em questão, sendo tais alternativas conhecidas como

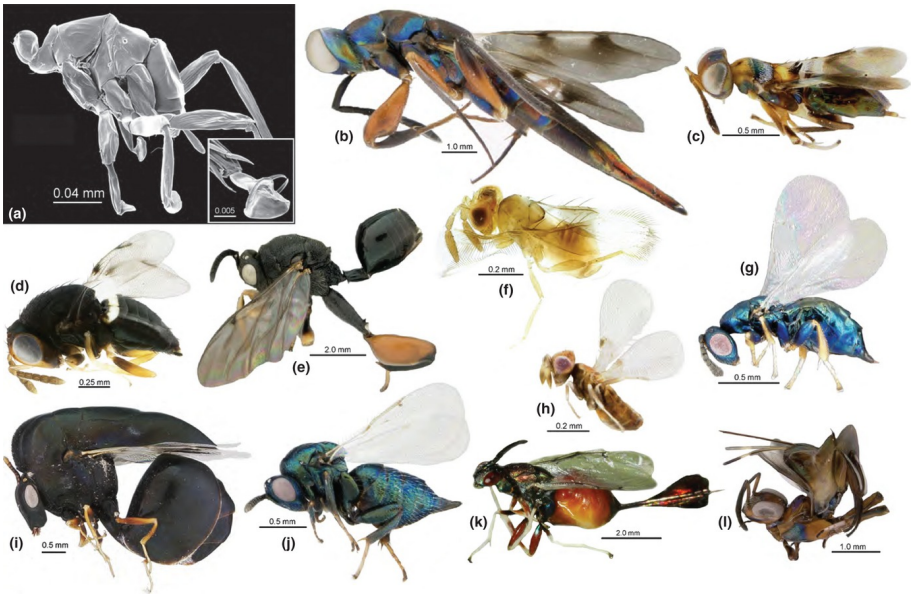

Figura 1 – Diversidade morfológica de Chalcidoidea (HERATY et al., 2013).

os **estados** desse caractere. Ao responder as perguntas, o usuário é levado à identificação do organismo desconhecido, encontrando o táxon ao qual ele pertence. Para isso, a chave de identificação também possui uma lista de todos os possíveis organismos que podem ser identificados por essa chave, esses organismos são chamados de **entidades** da chave de identificação.

Essas chaves são dividas em duas categorias: chaves de acesso único, também conhecidas como chaves sequenciais, e chaves de múltiplo acesso, também conhecidas como chaves de múltipla entrada ou policlaves (WINSTON, 1999).

Tradicionalmente, chaves de identificação em sua maior parte são desenvolvidas como chaves de acesso único. Nesse tipo de chave é oferecida uma sequência fixa de passos de identificação, cada um com múltiplas alternativas, sendo que a escolha de uma dessas alternativas leva ao próximo passo. Quando cada passo possui apenas duas alternativas para o usuário escolher, a chave é chamada de dicotômica, caso contrário, ela é uma chave politômica (WIKIPEDIA, 2018). Como nesse tipo de chave deve-se começar a partir do primeiro passo e seguir uma ordem determinada até chegar ao fim, o progresso pode ser interrompido quando o usuário não sabe a resposta em um dos passos. Dessa forma o resultado final não é obtido.

Por outro lado, as chaves de múltiplo acesso ou chaves interativas, permitem que o usuário escolha em qual passo de identificação deseja começar, podendo também pular os passos na qual a resposta não é conhecida.

Uma chave interativa é um programa de computador na qual o usuário informa os atributos (valores de estado-caractere) do organismo. Tipicamente, o programa elimina os táxons que possuem atributos diferentes dos atributos informados pelo usuário. O

processo continua até o usuário informar atributos suficientes para sobrar apenas um táxon, identificando assim o organismo desconhecido (DALLWITZ; PAINE; ZURCHER, 2018). As principais vantagens que chaves interativas possuem em relação às chaves convencionais são:

- Qualquer caractere pode ser acessado e seus valores podem ser modificados, em qualquer ordem;
- Uma identificação correta pode ser feita mesmo com a existência de erros do usuário ou dos dados;
- Caracteres numéricos podem ser usados diretamente, sem a necessidade de serem divididos em classes;
- O usuário pode expressar incerteza, selecionando mais de um estado para um caractere.

Na taxonomia, um caractere deve prover de forma clara e facilmente observável uma característica distinta que ajude a identificar um organismo. A escolha de caracteres na hora de elaborar uma chave de identificação deve ter como objetivo facilitar a identificação correta do organismo na menor quantidade de passos possível.

2.3 Engenharia de Software

O uso de softwares tornou-se um aspecto muito presente em todas as áreas de nossas vidas e, consequentemente, o número de pessoas interessadas nos recursos e nas funções oferecidas por uma determinada aplicação tem crescido significativamente. Quando uma aplicação está sendo desenvolvida, muitas pessoas devem ser ouvidas, e, no geral, cada uma delas possui uma ideia diferente de quais funções ou recursos o software deve oferecer. Dessa forma, deve-se dedicar um esforço concentrado para compreender o problema antes de desenvolver uma solução de software (PRESSMAN, 2011). Visando melhorar a qualidade dos produtos de software e aumentar a produtividade no processo de desenvolvimento, surgiu a Engenharia de Software (FALBO, 2014).

A Engenharia de Software é a área da Ciência da Computação voltada à especificação, desenvolvimento e manutenção de sistemas de software, com aplicação de tecnologias e práticas de gerência de projetos e outras disciplinas, visando organização, produtividade e qualidade no processo de software (FALBO, 2014).

No livro *How to Solve it*, Polya (1945) definiu quatro etapas para a solução de problemas. Consequentemente, essas 4 etapas são a essência da Engenharia de Software:

1. Compreender o problema (comunicação e análise);
2. Planejar uma solução (modelagem e projeto de software);
3. Executar o plano (codificação do sistema);
4. Examinar o resultado para ter precisão (testes e garantia da qualidade).

Dessa forma, os métodos da Engenharia de Software envolvem uma ampla gama de tarefas, que incluem: comunicação, análise de requisitos, modelagem de projeto, construção de programa, testes e suporte (PRESSMAN, 2011).

2.3.1 Especificação e Análise de Requisitos

Requisitos são descrições do que o sistema deve fazer, as funcionalidades que o mesmo oferece e as restrições às quais está submetido. Os requisitos expressam as necessidades dos clientes que irão utilizar o sistema desenvolvido. O processo de descobrir, analisar, documentar e verificar essas funcionalidades e restrições é chamado Engenharia de Requisitos (SOMMERVILLE, 2011).

A Engenharia de Requisitos têm um papel fundamental no desenvolvimento de software, pois um software é considerado bem sucedido quando atende os requisitos determinados para a sua construção. Requisitos estão presentes ao longo de todo o ciclo de vida de um software, servindo de base para a modelagem, projeto, implementação, testes e até mesmo para a manutenção do sistema (FALBO, 2017).

Segundo Sommerville (2011), os requisitos de software são divididos em dois tipos, os requisitos funcionais e os requisitos não-funcionais. Dessa forma, temos que:

- **Requisitos funcionais**: são especificações das funcionalidades que o sistema deve fornecer, de como o sistema deve reagir a entradas específicas e de como o sistema deve se comportar em determinadas situações. Em alguns casos, os requisitos funcionais também podem explicitar o que o sistema não deve fazer.
- **Requisitos não funcionais**: são restrições sobre as funções oferecidas pelo sistema, tais como restrições de tempo, de uso de recursos, entre outras. De maneira geral, requisitos não funcionais referem-se a atributos de qualidade que o sistema deve apresentar, tais como confiabilidade, usabilidade, eficiência, portabilidade, manutenibilidade e segurança.

O processo de Engenharia de Requisitos começa pelo levantamento de requisitos, atividade na qual devem ser coletadas informações sobre as necessidades dos usuários e clientes, informações do domínio, sistemas existentes, regulamentos, leis, etc. Uma vez

identificados os requisitos, é iniciada a etapa de análise, quando os requisitos levantados são usados como base para a modelagem do sistema. Tanto no levantamento quanto na análise de requisitos é importante documentar requisitos e modelos. Para documentar requisitos, dois documentos são normalmente utilizados: o **Documento de Definição de Requisitos** e o **Documento de Especificação de Requisitos**. No Documento de Definição de Requisitos se encontra a descrição do minimundo, na qual é apresentada, em forma de texto corrido, uma visão geral do domínio, do problema a ser resolvido e dos processos de negócio apoiados, além das principais ideias e sugestões do cliente sobre o sistema a ser desenvolvido. Nele também se encontra a lista dos requisitos de cliente identificados. Já no Documento de Especificação de Requisitos são registrados os requisitos de sistema e os vários diagramas resultantes do trabalho de análise. Os documentos produzidos são, então, verificados e validados (FALBO, 2017).

De acordo com Falbo (2017), a fase de análise de requisitos, em sua essência, é uma atividade de modelagem. A modelagem nesta fase se preocupa apenas com o domínio do problema e não com soluções técnicas para o mesmo, por isso é chamada de modelagem conceitual. Os modelos de análise são elaborados para se obter um entendimento maior acerca do sistema a ser desenvolvido. Diferentes modelos podem ser construídos para representar diferentes perspectivas, dentre eles podemos citar o diagrama de classes e o modelo de entidades e relacionamentos, que buscam modelar os conceitos, propriedades e relações do domínio que são relevantes para o sistema em desenvolvimento. Além disso, também são utilizados os diagramas de casos de uso, diagramas de atividades, diagramas de estados e diagramas de interação para modelar o comportamento geral do sistema, de suas funcionalidades ou de uma entidade específica ao longo do tempo.

2.3.2 Projeto e Implementação

O objetivo da fase de projeto é produzir uma solução para o problema identificado e modelado nas fases de levantamento e análise de requisitos, incorporando a tecnologia aos requisitos e projetando o que será construído na implementação. Sendo assim, é necessário conhecer a tecnologia disponível e os ambientes de hardware e software onde o sistema será desenvolvido e implantado. Durante o projeto, deve-se decidir como o problema será resolvido, começando em um alto nível de abstração, próximo da análise, e progredindo sucessivamente para níveis mais detalhados até se chegar a um nível de abstração próximo da implementação (FALBO, 2016). O projeto de software encontra-se no núcleo técnico do processo de desenvolvimento de software e é aplicado independentemente do modelo de ciclo de vida e paradigma adotados. É iniciado assim que os requisitos do software tiverem sido modelados e especificados pelo menos parcialmente e é a última atividade de modelagem. Por outro lado, corresponde à primeira atividade que leva em conta aspectos tecnológicos (PRESSMAN, 2011).

Partindo dos modelos conceituais do sistema, a fase de projeto inicia-se com o projeto da arquitetura do sistema, que visa definir os elementos estruturais do software e seus relacionamentos. Feito isso, o projeto passa a se concentrar no detalhamento de cada um desses elementos, o que envolve a decomposição de módulos em outros módulos menores. Uma vez especificado o projeto dos elementos da arquitetura, pode dar-se início à implementação, quando as unidades de software do projeto detalhado são implementadas e testadas individualmente (teste de unidade). Aos poucos, cada elemento da arquitetura vai sendo integrado e testado (teste de integração), até se obter o sistema completo. Por fim, deve ser feito um teste completo do sistema (teste de sistema) (FALBO, 2016).

De acordo com Pressman (2011), a importância do projeto de software pode ser definida em uma única palavra: qualidade. A fase de projeto é a etapa em que a qualidade é incorporada na Engenharia de Software. O projeto nos fornece representações de software que podem ser avaliadas em termos de qualidade. Na fase de projeto é onde podemos traduzir precisamente os requisitos dos interessados em um software finalizado.

2.4 Desenvolvimento Web

Nos primórdios da World Wide Web (por volta de 1990 a 1995), os sites eram formados por apenas conjuntos de arquivos de hipertexto que apresentavam informações usando texto e gráficos limitados. Com o surgimento de novas ferramentas de desenvolvimento, como o XML e o Java, tornou-se possível aos engenheiros da Internet oferecerem capacidade computacional juntamente com as informações. Dessa forma, nasceram os sistemas e aplicações baseados na Web (PRESSMAN, 2011).

Dois atributos-chave distinguem o desenvolvimento de softwares baseados na Web do desenvolvimento de softwares tradicionais: o crescimento rápido de seus requisitos e a velocidade alta com o qual o seu seu conteúdo muda. Com isso, aplicações Web precisam ser planejadas para serem escaláveis e de fácil manutenção, uma vez que tais recursos não podem ser adicionados posteriormente. O sucesso na construção, desenvolvimento, implementação e manutenção de um sistema Web depende do quão bem essas questões foram tratadas (GINIGE; MURUGESAN, 2001).

2.4.1 O Ambiente Web

De acordo com Mai (2017), o ambiente Web é sustentado por 3 componentes principais: o protocolo de comunicação HyperText Transfer Protocol (HTTP), o sistema de endereçamento Uniform Resource Locator (URL) e a linguagem HyperText Markup Language (HTML).

O protocolo HTTP é o ingrediente mais básico sobre o qual a Web está fundamentada. Ele é um protocolo de aplicações cliente-servidor que define um formato padrão

para especificar a requisição de recursos na Web. Através dele, o cliente (navegador ou dispositivo que fará a requisição) pode requisitar recursos disponíveis a um servidor remoto. Como esses recursos estão distribuídos ao longo de toda a Internet, é necessário um mecanismo de identificação que permita localizá-los e acessá-los. O identificador usado para referenciar esses recursos é uma URL (CASTELEYN et al., 2009).

Segundo Casteleyn et al. (2009), ao acessar uma URL, o cliente envia uma requisição HTTP, sendo que esta requisição possui um cabeçalho especificando um método HTTP. Os métodos HTTP mais utilizados são:

- **GET**: solicita um determinado recurso. Requisições GET devem ser usadas apenas para recuperar dados e não devem ter qualquer outro efeito;
- **POST**: envia dados para serem processados para o recurso especificado. Os dados são incluídos no corpo da requisição;
- **PUT**: envia os dados de forma semelhante ao POST, porém é utilizado para atualizar dados já existentes no servidor;
- **DELETE**: utilizado para excluir um recurso do servidor.

Além desses métodos, também existem os métodos HEAD, OPTIONS, TRACE e CONNECT.

Ao receber uma requisição, o servidor localiza o recurso solicitado e envia uma resposta para o cliente. Essa resposta inclui o seu status, um número que indica se a requisição feita pelo cliente foi bem sucedida ou não. De acordo com Fielding et al. (1999), o código de status é formado por três dígitos, sendo que o primeiro dígito representa a categoria a qual a resposta pertence. Uma resposta pode ser classificada em cinco categorias:

- **1xx:** *Informational* (Informação): utilizada para informar o cliente que sua requisição foi recebida e está sendo processada;
- **2xx:** *Success* (Sucesso): indica que a requisição do cliente foi bem sucedida;
- **3xx:** *Redirection* (Redirecionamento): informa a ação adicional que deve ser tomada para completar a requisição;
- **4xx:** *Client Error* (Erro no cliente): o cliente fez uma requisição que não pode ser atendida;
- **5xx:** *Server Error* (Erro no servidor): ocorreu um erro no servidor ao cumprir uma requisição válida.

Para expressar o conteúdo e a formatação visual das páginas Web é utilizada a linguagem HTML. O HTML é uma linguagem de marcação, ou seja, um documento HTML é composto por *tags* que delimitam um certo conteúdo e definem como o mesmo deve ser representando visualmente. O navegador recebe como entrada o conteúdo marcado, interpreta o significado das *tags* e transforma o conteúdo recebido em um documento renderizado (CASTELEYN et al., 2009).

Complementar ao HTML, o CSS (Folha de Estilo em Cascatas, do inglês *Cascading Style Sheets*) é utilizado para definir a aparência de uma página Web, separando o conteúdo HTML da sua representação visual. Uma folha de estilos é um conjunto de regras que definem como o navegador deve apresentar o documento. Um regra é composta de duas partes: um seletor, que define para qual *tag* HTML a regra deve ser aplicada, e uma declaração de estilo, que define as propriedades que devem ser adicionadas a tag HTML mencionada no seletor (CASTELEYN et al., 2009).

2.4.2 O padrão MVC de Arquitetura de Software

A arquitetura Modelo-Visão-Controlador (MVC) é um dos vários padrões de arquitetura sugeridos para aplicações Web (PRESSMAN, 2011). Esse padrão é uma das soluções mais avançadas propostas pela Engenharia de Software para a necessidade de modularização, já que foi desenvolvido com a escalabilidade e a separação de conceitos em mente (CASTELEYN et al., 2009).

As noções de separação e independência entre componentes são essenciais para o projeto de arquitetura, pois permitem que alterações sejam feitas de forma localizada, sem comprometer outros componentes. O padrão MVC separa os elementos de uma aplicação, permitindo modificá-los de forma independente (SOMMERVILLE, 2011).

Nesse padrão o sistema é separado em três camadas (FOWLER, 2002). Na camada *Modelo* se encontram todos os dados do sistema e as ações associadas a esses dados. Essa camada pode ser compartilhada entre diversas aplicações. O Modelo ignora como requisições de usuários são feitas e como os dados são apresentados para o usuário. A camada *Visão* contém todas as funções específicas à interface com o usuário e define como os dados do modelo serão exibidos. Por fim, a camada *Controlador* coordena o fluxo de dados entre o modelo e a visão. O Controlador recebe requisições do usuário, manipula o Modelo e atualiza a Visão de acordo com as modificações executadas.

Na Figura 2 é possível ver o comportamento do padrão MVC. Nele, os dados ou requisições de usuários são tratados pelo controlador, que também seleciona qual componente da Visão exibir baseado na requisição do usuário. Uma vez determinado o tipo da requisição recebida, o Controlador transmite uma solicitação de comportamento ao Modelo, que executa uma ação ou recupera os dados necessários para atender à requisição.

Para fazer isso, o Modelo pode acessar dados armazenados em um banco de dados externo. Os dados retornados pelo Modelo devem ser formatados e organizados pela Visão para serem exibidos de forma apropriada na máquina do usuário que fez a requisição.

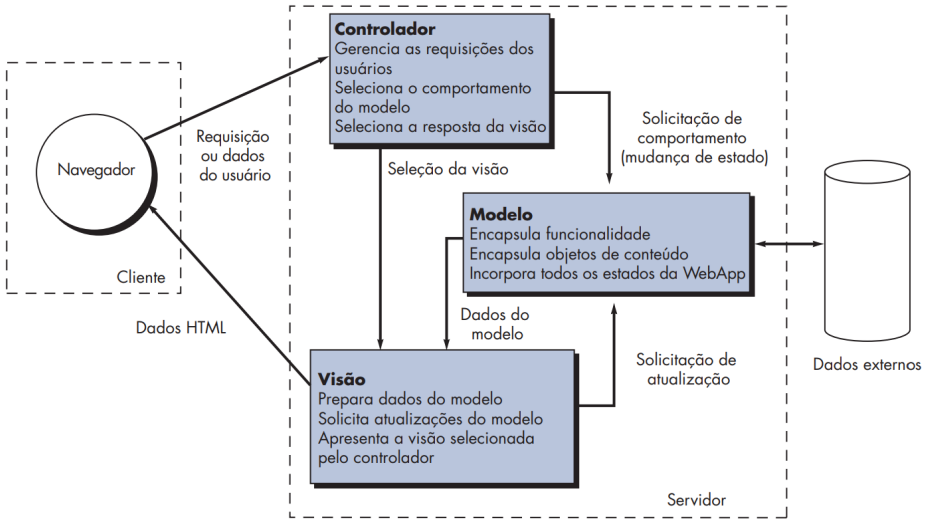

Figura 2 – A arquitetura MVC (PRESSMAN, 2011).

2.4.3 Node.js e o Framework Express.js

O Node.js é um ambiente de execução assíncrono orientado a eventos construído sobre o motor JavaScript do Google Chrome (Engine V8), ele foi criado com o objetivo de permitir a construção de aplicações de rede escaláveis. Node.js usa um modelo de E/S (Entrada e Saída) orientado a eventos e não bloqueante, o que o torna leve e eficiente, ideal para aplicações em tempo real com troca intensa de dados executadas em dispositivos distribuídos (NODEJS.ORG, 2019).

A principal característica de um servidor feito com Node.js é o fato de sua execução ser *single-threaded* com chamadas de E/S não-bloqueantes, enquanto outros servidores tradicionais possuem uma execução *multi-threaded* com chamadas bloqueantes. Na prática isso significa que em um servidor tradicional, para cada requisição recebida é criada uma nova *thread* para tratá-la. Já em um servidor Node.js, apenas uma *thread* é responsável por tratar as requisições. Essa *thread*, conhecida como *Event Loop* fica em execução aguardando a chegada de novos eventos. Quando um determinado código emite um evento, o mesmo é enviado para a fila de eventos para que o *Event Loop* o execute. Diferentemente dos servidores tradicionais, a *thread* não fica esperando que as operações que serão executadas sejam concluídas, pois essas operações serão executadas através de chamadas não-bloqueantes, e o mecanismo de notificação de eventos do Node.js ficará responsável por notificar o servidor quando a resposta dessas operações for obtida.

O Node.js, porém, não possui nativamente suporte a algumas tarefas comuns do desenvolvimento Web. O tratamento específico para métodos HTTP, especificação de rotas para requisições, uso de *templates* para criar páginas HTML dinamicamente, são algumas das funções que não existem nativamente no Node.js. A forma mais fácil de obtê-las é utilizando um *framework*.

O Express.js é o *framework* mais popular do Node.js. Ele é um *framework* minimalista e flexível que fornece um conjunto robusto de recursos para o desenvolvimento de aplicações Web. Complementar a esses recursos, desenvolvedores criaram uma série de pacotes de *middleware* que adereçam basicamente qualquer necessidade existente no desenvolvimento Web. Existem pacotes para se trabalhar com *cookies*, sessões, autenticação de usuários, segurança, *upload* de arquivos e muito mais (MOZILLA, 2019).

Dessa forma, utilizando o Express.js, é possível definir rotas para receber requisições de usuários, utilizar uma *template engine* para renderizar páginas HTML dinamicamente, conectar com um banco de dados para armazenar os dados da aplicação. Tais características possibilitam o desenvolvimento de um sistema Web robusto.

2.5 O Método FrameWeb

O FrameWeb (*Framework-based Design Method for Web Engineering*) é um método de projeto para construção de sistemas de informação Web utilizando *frameworks*. Ele é baseado em metodologias e linguagens de modelagens bastante utilizadas na área de Engenharia de Software (SOUZA, 2007).

O método, sugere um processo de software orientado a objetos contendo as fases de levantamento de requisitos, análise, projeto, codificação, testes e implantação, mas podendo ser estendido e adaptado pela equipe de desenvolvimento. A linguagem de modelagem UML (*Unified Modeling Language*) é utilizada durante todas as etapas do processo. Na fase de levantamento de requisitos, é sugerida a elaboração de modelos de casos de uso. Para a fase de análise, é sugerido o desenvolvimento de diagramas de classes com alto nível de abstração, construindo um modelo conceitual do domínio do problema (SOUZA, 2007).

O foco do método FrameWeb, porém, são as fases de projeto e implementação. É durante a fase de projeto que a plataforma onde o software será desenvolvido e implantado é levada em consideração. São feitas, portanto, as seguintes propostas: (i) definição de uma arquitetura de software que divida o sistema em camadas, de modo a se integrar bem com os *frameworks* utilizados; (ii) elaboração de um conjunto de modelos que utilizam conceitos utilizados por *frameworks*. Para isso é proposto um perfil UML que possibilite que os diagramas fiquem mais próximos da implementação do sistema (SOUZA, 2007).

Dessa forma, o FrameWeb propõe uma arquitetura lógica baseada no padrão

arquitetônico *Service Layer* definido por Fowler (2002) e que pode ser visto na Figura 3. A primeira camada dessa arquitetura possui os pacotes Visão e Controle, responsáveis por apresentar dados na interface e receber requisições vindas do usuário. A segunda camada é responsável pela lógica de negócio da aplicação e possui os pacotes Domínio e Aplicação, responsáveis por armazenar as classes que representam conceitos do domínio do problema e por implementar os casos de uso definidos na especificação de requisitos, respectivamente. Por fim, a terceira e última camada é responsável pela lógica de acesso a dados, contém apenas o pacote Persistência, que é responsável pela persistência dos dados.

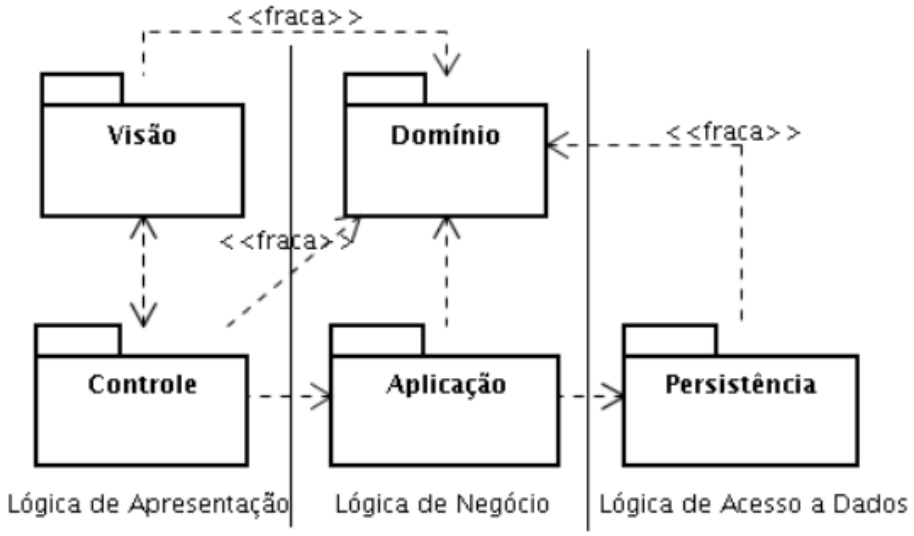

Figura 3 – Arquitetura proposta pelo FrameWeb (SOUZA, 2007).

A partir da arquitetura definida, é proposto uma extensão da linguagem UML (MARTINS; SOUZA, 2015) que possibilita a construção de quatro tipos de diagramas:

- **Modelo de Entidades** (antes Modelo de Domínio): um diagrama de classes da UML que representa os objetos de domínio do problema e seu mapeamento para a persistência em banco de dados relacional. Nele, o modelo de classes construído na fase de análise de requisitos é adequado para à plataforma de implementação escolhida. Para isso, os tipos de dados de cada atributo são indicados, a navegabilidade das associações é definida, entre outras ações. Além disso, são adicionados os mapeamentos de persistência, meta-dados das classes de domínio que permitem que os *frameworks* ORM (*Object/Relational Mapping*) transformem objetos que estão na memória em linhas de tabelas no banco de dados relacional e vice-versa. Estereótipos e restrições definidas pela linguagem FrameWeb guiam os desenvolvedores na configuração do *framework* ORM. Apesar de tais configurações serem relacionadas mais à persistência do que ao domínio, elas são representadas no Modelo de Entidades porque as classes que são mapeadas e seus atributos são exibidas neste

diagrama (SOUZA, 2007). A partir desse modelo são implementadas as classes da camada *Domínio* da arquitetura proposta pelo FrameWeb.

- **Modelo de Persistência**: um diagrama de classes da UML que representa as classes DAO existentes, que pertencem ao pacote *Persistência*, responsáveis pela persistência das instâncias das classes de domínio. Nesse modelo são apresentados, para cada classe de domínio que necessita de lógica de acesso a dados, uma interface e uma classe concreta DAO que a implementa. A interface define os métodos de persistência existentes para aquela classe, tais métodos são implementados por uma ou mais classes concretas;

- **Modelo de Navegação**: um diagrama de classe da UML que representa os diferentes componentes que formam a camada de Lógica de Apresentação, como páginas Web, formulários HTML e classes controladoras. Esse modelo é utilizado pelos desenvolvedores para guiar a codificação das classes e componentes dos pacotes *Visão* e *Controle*. Nele é possível identificar como os dados são submetidos pelo usuário e quais resultados são possíveis a partir de uma certa entrada de dados;

- **Modelo de Aplicação**: um diagrama de classes da UML que representa as classes de serviço, que são responsáveis pela codificação dos casos de uso, e suas dependências. Esse diagrama é utilizado para guiar a implementação das classes do pacote Aplicação e a configuração das dependências entre os pacotes *Controle*, *Aplicação* e *Persistência*, ou seja, quais classes controladoras dependem de quais classes de serviço e quais DAOs são necessários para que as classes de serviço alcancem seus objetivos.

3 Especificação de Requisitos

Este capítulo aborda alguns resultados da Engenharia de Requisitos da ferramenta Portal de Biodiversidade de Chalcidoidea. Na Seção 3.1, é apresentado o minimundo do projeto, na Seção 3.2, é apresentado o diagrama de pacotes, na Seção 3.3, são apresentados diagramas de casos de uso e na Seção 3.4, são apresentados os diagramas de classes. Os requisitos funcionais, requisitos não funcionais e regras de negócio podem ser encontrados no **Documento de Especificação de Requisitos** que está disponível no Apêndice ao final dessa monografia.

3.1 Descrição do Escopo

O Portal de Biodiversidade de Chalcidoidea é uma aplicação Web desenvolvida para o Laboratório de Biodiversidade de Insetos da Universidade Federal do Espírito Santo (LaBI/UFES). O portal tem como objetivo promover a divulgação científica para aumentar a visibilidade da pesquisa em *Chalcidoidea*, permitindo que pesquisadores divulguem informações sobre publicações científicas, projetos, laboratórios, pesquisadores e textos relacionadas a área.

A aplicação permite ainda que pesquisadores façam *upload* de ferramentas de identificação taxonômica, desenvolvidas por taxônomos usando plataformas como Lucid e Delta, por meio de arquivos em formato SDD. O arquivo SDD é interpretado e as suas informações são armazenadas em um banco de dados. A partir dessas informações a chave de identificação é disponibilizada no portal onde poderá ser utilizada pelo público em computador ou celular sem a necessidade do software originalmente utilizado para o preparo da mesma.

3.1.1 Recursos Disponíveis

O portal oferecerá uma série de recursos relacionados a Chalcidoidea, sendo eles: publicações científicas, projetos, laboratórios, pesquisadores e postagens. Cada um desses recursos poderá ser encontrado em sua própria seção dentro do portal.

A seção de publicações científicas do portal exibe todas as publicações disponíveis, organizadas pelo ano da publicação, da mais recente para a menos recente. De uma publicação deseja-se cadastrar/exibir o título, o veículo onde foi publicada, o ano em que foi publicada e um link para o seu arquivo PDF.

Da mesma forma, a seção de projetos exibe os projetos disponíveis organizados por data de postagem, com os mais recentes aparecendo primeiro. Nesta seção são exibidos

o título de cada projeto, sua data de criação e uma pequena descrição sobre o mesmo. Cada projeto deve possuir sua própria página onde o mesmo é detalhado, nela, além das informações anteriores, também são exibidos o código de cadastro do projeto, a lista de pesquisadores participantes, e o conteúdo textual completo do projeto.

O portal também possui seções para exibir os laboratórios e pesquisadores da área de taxonomia de Chalcidoidea. Para cada laboratório é exibido o seu nome, a cidade e estado onde o mesmo está localizado. Para cada pesquisador é exibido o seu nome, área de interesse, e-mail para contato e link para o seu currículo Lattes.

Por fim, recursos educacionais relacionados a Chalcidoidea podem ser encontrados na seção de postagens. Nesta seção, que deve ser organizada por data de publicação em ordem decrescente, são exibidos o título, a data de publicação e uma pequena descrição de cada postagem. Cada postagem deve possuir a sua própria página onde será possível ler o seu conteúdo completo.

3.1.2 Chaves de Identificação

Para interagir com as chaves de identificação adicionadas no portal, o usuário deve escolher qual chave deseja acessar na seção de chaves de identificação, onde são exibidos o nome, a data de publicação, o autor e uma pequena descrição de cada chave.

Ao acessar uma chave é possível ver a seção de caracteres e a seção de entidades da mesma. Caracteres podem estar organizados em agrupamentos. Da mesma forma, agrupamentos também podem estar contidos dentro de um agrupamento maior, definindo assim uma hierarquia de exibição dos caracteres. Dessa forma, a seção de caracteres exibe a lista de caracteres e agrupamentos disponíveis, organizados de acordo com a hierarquia definida.

Ao acessar um caractere o usuário pode visualizar a sua lista de estados de caractere, onde cada estado possui um nome, e opcionalmente uma imagem e uma pequena descrição sobre ele. O usuário pode selecionar ou desmarcar quantos estados quiser. Ao selecionar um estado o mesmo é adicionado a lista de estados selecionados pelo usuário. A lista de entidades é filtrada de acordo com os estados selecionados, e apenas as entidades que possuem os estados selecionados são exibidas na seção de entidades.

A seção de entidades exibe a lista de entidades da chave de identificação. Como mencionado, essa lista é filtrada de acordo com os estados selecionados na seção de caracteres, caso nenhum estado tenha sido selecionado todas as entidades são exibidas. Cada entidade possui um texto descritivo sobre a mesma, a lista de estados em que essa entidade pode se encontrar e uma galeria de imagens.

A lista de estados selecionados pelo usuário também fica disponível para visualização na página da chave utilizada, sendo possível remover cada estado dessa lista individualmente,

ou remover todos de uma vez.

Por fim, existe a opção de enviar o resultado obtido na chave. Ao fazer isso o usuário envia para o sistema a lista de estados que selecionou e quais entidades foram obtidas como resultado, podendo também adicionar um comentário referente ao resultado que obteve. Essa resultado ficará então disponível para que os administradores do portal o visualizem, servindo de *feedback* para que os mesmos possam refinar e corrigir a chave de identificação caso seja necessário.

3.1.3 Dashboard

Na *dashboard* ocorre o gerenciamento do conteúdo do portal. Para acessá-la é necessário ser autenticado primeiro. Feito isso a *dashboard* é exibida e o administrador pode escolher entre as seguintes opções: **Gerenciar Administradores**, **Gerenciar Publicações**, **Gerenciar Projetos**, **Gerenciar Pesquisadores**, **Gerenciar Laboratórios**, **Gerenciar Postagens**, **Gerenciar Chaves de Identificação**, **Lista de Resultados**.

A opção **Gerenciar Administradores** permite que o administrador cadastre novos administradores. Para isso ele deve preencher um formulário com o nome, nome de usuário, e-mail e senha do novo administrador. Também é exibida a lista de todos os administradores do portal, por meio a qual é possível alterar a senha de um administrador ou excluir a sua conta.

A Partir das opções **Gerenciar Publicações**, **Gerenciar Projetos**, **Gerenciar Pesquisadores**, **Gerenciar Laboratórios** e **Gerenciar Postagens**, o administrador pode visualizar e gerenciar os recursos existentes no portal, podendo editar ou excluir os mesmos. Também é possível adicionar um novo recurso.

A opção **Gerenciar Chaves de Identificação** permite que o administrador visualize e gerencie as chaves de identificação disponíveis no portal. O administrador tem a opção de editar ou excluir uma chave. Também é possível adicionar uma nova chave. Ao adicionar uma chave o administrador deve fazer *upload* de um arquivo SDD, arquivos de imagem e arquivos HTML. O arquivo SDD contém informações sobre a lista de caracteres, a lista de agrupamentos, a lista de estados e a lista de entidades da chave. Também possui informações sobre com quais entidades e estados os arquivos HTML e imagens estão associados. Por fim, o arquivo indica quais são as associações entre agrupamentos, caracteres, estados e entidades. O sistema deve ler e interpretar todas as informações no arquivo SDD e armazená-las no banco de dados, criando assim a chave de identificação.

A opção **Lista de Resultados** permite que o administrador visualize a lista com todos os resultados enviados por usuários. Na lista é exibido o nome da chave de identificação referente a cada resultado enviado. Ao acessar o resultado, o administrador poder visualizar a lista de estados selecionados pelo usuário e as entidades resultantes.

Caso exista, também é exibido o comentário feito pelo usuário. Também existe a opção de excluir um resultado da lista.

3.2 Diagrama de Pacotes

Dada a descrição do minimundo, foi possível estruturar o sistema em dois pacotes, como pode ser visto na Figura 4. A descrição de cada um dos subsistemas está contida na Tabela 1.

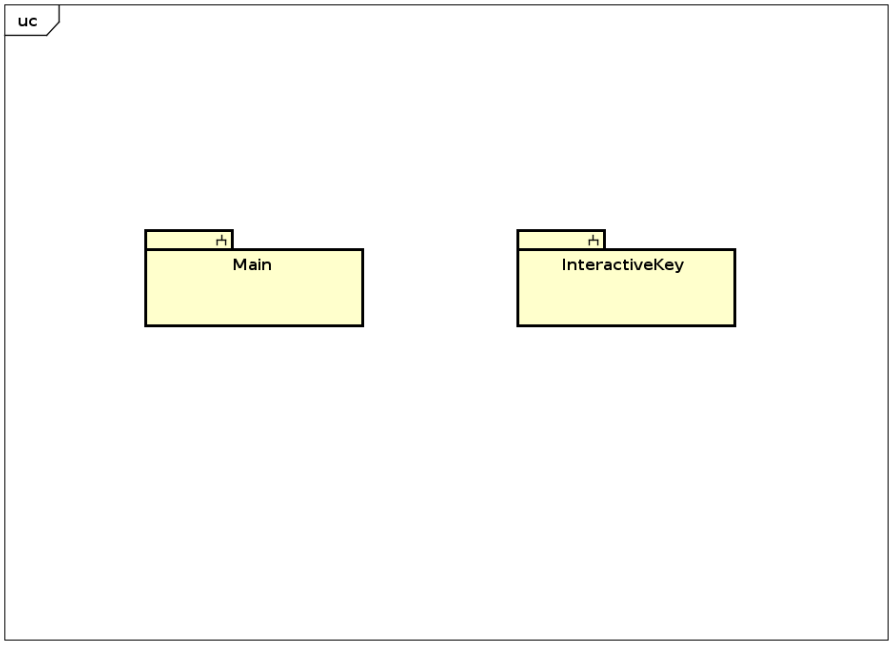

Figura 4 – Diagrama de pacotes para a ferramenta Portal de Biodiversidade de Chalcidoidea.

Tabela 1 – Descrição dos subsistemas.

Subsistema	**Descrição**
Main	Envolve todas as funcionalidades relacionadas ao cadastramento, alteração, visualização e remoção de dados do sistema.
InteractiveKey	Subsistema contendo todas as funcionalidades relacionadas diretamente a chaves de identificação.

3.3 Diagrama de Casos de Uso

O modelo de casos de uso visa capturar e descrever as funcionalidades que um sistema deve prover para os atores que interagem com o mesmo. No contexto deste projeto, existem dois tipos de atores: usuários não autenticados que visitam o portal (Visitante) e usuários autenticados que gerenciam o conteúdo do portal (Administrador). Maiores

informações e detalhes sobre os casos de uso poderão ser consultados no **Documento de Especificação de Requisitos** que está disponível no Apêndice ao final dessa monografia.

Como visto na Seção 3.2, este projeto foi divido em dois subsistemas. A subseção 3.3.1 apresenta os casos de uso do subsistema **Main** e a subseção 3.3.2 apresenta os casos de uso do subsistema **InteractiveKey**.

3.3.1 Subsistema Main

A Figura 5 mostra os casos de uso do subsistema **Main** que serão descritos a seguir. O subsistema **Main** tem como objetivo gerenciar as funcionalidades relacionadas a publicações, projetos, pesquisadores, laboratórios, postagens e administradores.

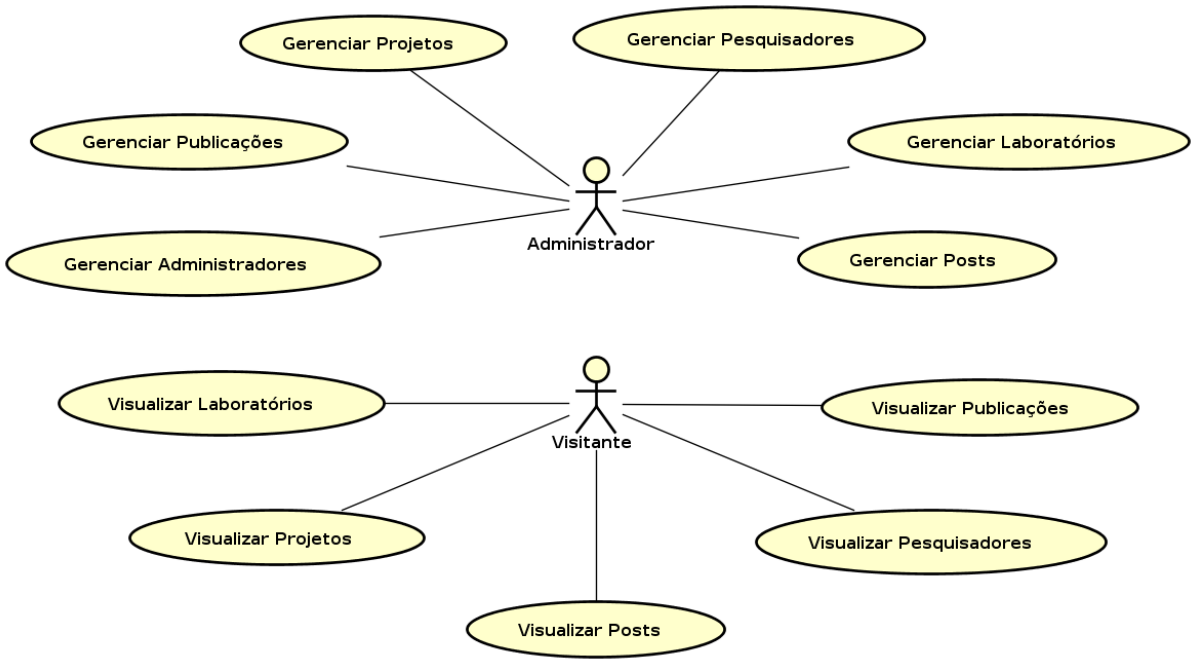

Figura 5 – Diagrama de Casos de Uso para o subsistema Main.

Os casos de uso **Gerenciar Administradores**, **Gerenciar Publicações**, **Gerenciar Projetos**, **Gerenciar Pesquisadores**, **Gerenciar Laboratórios** e **Gerenciar Posts** são do tipo cadastrais. Para cada caso de uso cadastral, espera-se que o administrador possa criar, alterar, consultar e excluir a classe em questão, passando as informações relativas a ela no momento de sua criação ou alteração.

O caso de uso **Visualizar Publicações** permite que uma publicação seja visualizada na página de publicações. Uma pulicação representa um artigo científico, e deve possuir um título, veículo onde foi publicado, ano da publicação e um link para o seu arquivo PDF.

O caso de uso **Visualizar Pesquisadores** permite que um pesquisador seja visualizado na página de pesquisadores. Um pesquisador deve possuir um nome, área de

interesse, link para o currículo Lattes e e-mail para contato.

O caso de uso **Visualizar Posts** permite que um post seja visualizado na página de posts. Um Post é uma postagem de cunho informativo, e deve possuir um título, data de postagem, uma pequena descrição e o seu conteúdo.

O caso de uso **Visualizar Projetos** permite que um projeto seja visualizado na página de projetos. Projetos são desenvolvidos por pesquisadores e devem possuir um título, uma data, número de cadastro, uma pequena descrição, conteúdo do projeto, e a lista de pesquisadores responsáveis pelo mesmo junto com links para o currículo Lattes de cada pesquisador.

O caso de uso **Visualizar Laboratórios** permite que um laboratório seja visualizado na página de laboratórios. Para cada laboratório, espera-se ver o seu nome, cidade e estado.

3.3.2 Subsistema InteractiveKey

A Figura 6 mostra os casos de uso do subsistema **InteractiveKey** que serão descritos a seguir. O subsistema **InteractiveKey** tem como objetivo gerenciar as funcionalidades relacionadas a chaves de identificação, desde o seu cadastro até a interação do visitante com a chave.

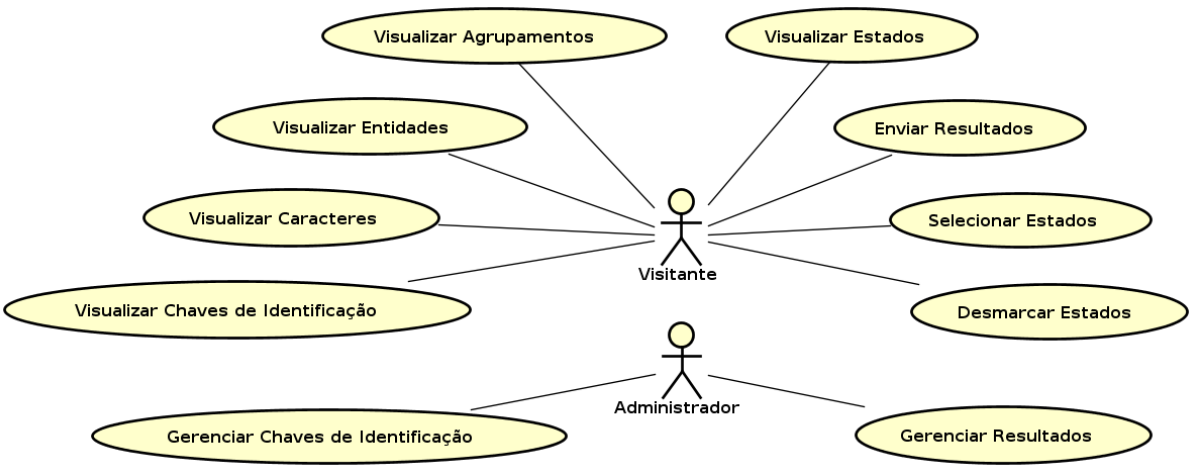

Figura 6 – Diagrama de Casos de Uso para o subsistema InteractiveKey.

O caso de uso **Gerenciar Chaves de Identificação** permite ao administrador criar, alterar, consultar e excluir uma chave de identificação. A criação de uma chave de identificação é um processo complexo que envolve muitos passos. Primeiro, o administrador informa o nome do autor, data de criação e descrição da chave de identificação. Em seguida, o administrador informa o nome da pasta que será criada no sistema para armazenar os arquivos referentes à chave de identificação. Por fim, o administrador faz *upload* de um arquivo SDD, arquivos de imagem e arquivos HTML. O arquivo SDD contém informações

sobre uma chave de identificação, sendo elas: o nome da chave de identificação, nomes dos agrupamentos, nomes dos caracteres, nomes dos estados e suas respectivas descrições, nomes das entidades, informações sobre os arquivos HTML e imagens e, por fim, as associações entre todas estas classes. O sistema deve ler e interpretar todas as informações no arquivo SDD e armazená-las no banco de dados, criando assim a chave de identificação.

O caso de uso **Visualizar Chaves de Identificação** permite que o visitante visualize uma chave de identificação. Uma chave de identificação pode ser visualizada na página de chaves de identificação. Para cada chave, espera-se ver o seu nome, data de criação, autor, descrição uma lista de entidades e outra de caracteres.

O caso de uso **Visualizar Entidades** permite que o visitante visualize a lista de entidades de uma chave de identificação. Cada entidade possui uma lista de caracteres e, opcionalmente, uma galeria de imagens e um arquivo HTML contendo um texto descritivo sobre a entidade.

O caso de uso **Visualizar Caracteres** permite que o visitante visualize a lista de caracteres de uma chave de identificação. Cada caractere possui um nome e uma lista de estados com os quais o visitante pode interagir. Caracteres também podem estar organizados em agrupamentos, que por sua vez também podem estar dentro de outro agrupamento. O caso de uso **Visualizar Agrupamentos** permite que o visitante visualize um agrupamento.

O caso de uso **Visualizar Estados** permite que o visitante visualize a lista de estados de um caractere. Para cada estado, espera-se ver o seu nome e, opcionalmente, uma imagem e uma descrição sobre o mesmo.

Os casos de uso **Selecionar Estados** e **Desmarcar Estados** permitem que o visitante interaja com os estados de um caractere em uma chave de identificação. Para selecionar um estado, o visitante acessa um caractere dentro de uma chave de identificação e indica qual estado da lista de estados desse caractere deseja selecionar. Feito isso, o estado é adicionado à lista de estados selecionados e o sistema filtra a lista de entidades deixando apenas as entidades que possuem todos os estados da lista de estados selecionados. Para desmarcar um estado o visitante indica um estado que tenha sido previamente selecionado, esse estado é então removido da lista de estados selecionados e a lista de entidades é novamente atualizada. Também existe a opção de desmarcar todos os estados selecionados, deixando, assim, a lista de estados selecionados vazia. Neste caso, o sistema retorna a lista de entidades para o seu estado original, em que todas as entidades fazem parte da mesma. Em todos os cenários a seção de entidades da chave de identificação é sempre atualizada com a lista de entidades atual.

O caso de uso **Enviar Resultados** permite que o visitante envie um resultado obtido em uma chave de identificação. Um resultado possui uma lista de caracteres e seus

estados selecionados pelo visitante, a lista de entidades resultantes e, opcionalmente, um comentário feito pelo visitante. Após isso, o resultado fica disponível na *dashboard* onde administradores podem consultar um resultado ou exclui-lo, o caso de uso **Gerenciar Resultados** permite esse controle.

3.4 Diagrama de Classes

Assim como os diagramas de casos de uso da Seção 3.3, os diagramas de classes estão divididos de acordo com a divisão dos subsistemas, na Subseção 3.4.1 estão as classes pertecentes ao subsistema **Main** e na Subseção 3.4.2 estão as classes pertencentes ao subsistema **InteractiveKey**.

3.4.1 Subsistema Main

A Figura 7 exibe o diagrama de classes do subsistema **Main**. A classe central deste subsistema é a classe **Admin**, que representa o administrador, um usuário autenticado no sistema. A classe **Publication** armazena dados referentes a uma publicação científica. A classe **Researcher** armazena informações de um pesquisador. A classe **Post** representa uma postagem. A classe **Lab** armazena dados referentes a um laboratório. A classe **Project** representa um projeto e a classe **ProjectResearcher** representa um pesquisador envolvido em um projeto.

O administrador é responsável por adicionar todo o conteúdo do portal, dessa forma, todas as outras classes estão associados com um administrador.

3.4.2 Subsistema InteractiveKey

A Figura 8 exibe o diagrama de classes do subsistema **InteractiveKey**. A classe central deste subsistema é a classe **IdentificationKey**, que representa uma chave de identificação. Uma chave de identificação possui uma lista de entidades (**Entity**) e caracteres (**Feature**) e também pode possuir agrupamentos **Node**. Um agrupamento é um conjunto de caracteres ou agrupamentos, podendo os dois estarem presentes dentro de um único agrupamento.

Caracteres possuem uma lista de possíveis estados (**State**) que o mesmo pode se encontrar. Cada estado pode estar associado com dois arquivos (**File**) que auxiliam na sua descrição, sendo que um desses arquivos deve ser uma imagem e o outro deve ser um arquivo HTML contendo um texto descritivo sobre o estado.

Entidades também estão associadas com caracteres, sendo que cada caractere possui um ou mais estados em que a entidade se encontra. Uma entidade pode estar associada com vários arquivos, sendo que no máximo apenas um desses arquivos pode ser um texto

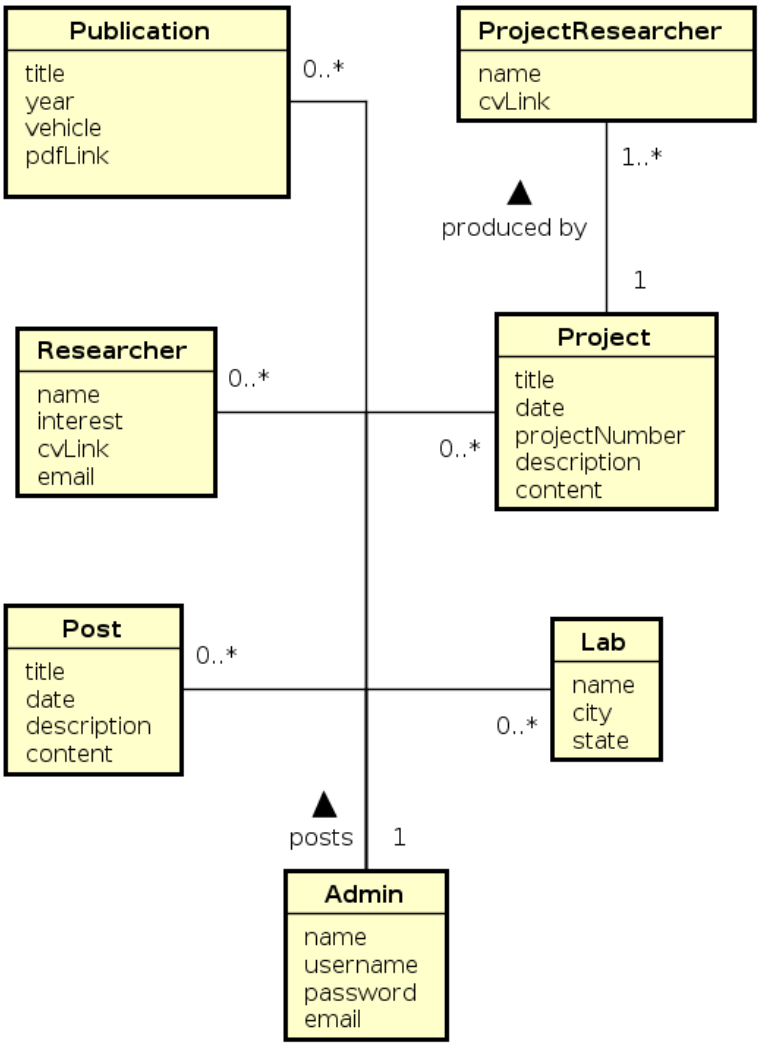

Figura 7 – Diagrama de Classes para o subsistema Main.

descritivo sobre a entidade e os outros devem ser imagens.

Arquivos devem sempre ser do tipo **Image** ou do tipo **Text**. Um arquivo do tipo **Image** representa uma imagem, enquanto um arquivo do tipo **Text** representa um arquivo HTML contendo texto. Arquivos devem sempre estar associados a uma entidade ou a um estado.

Por fim, a classe **Result** representa um resultado obtido por um visitante em uma chave de identificação. Dessa forma, um resultado está associado à chave de identificação da qual o mesmo foi obtido e possui uma lista de caracteres com os seus respectivos estados que o usuário selecionou e uma lista de entidades resultantes, obtida a partir das seleções feitas pelo visitante, podendo ambas as listas estarem vazias.

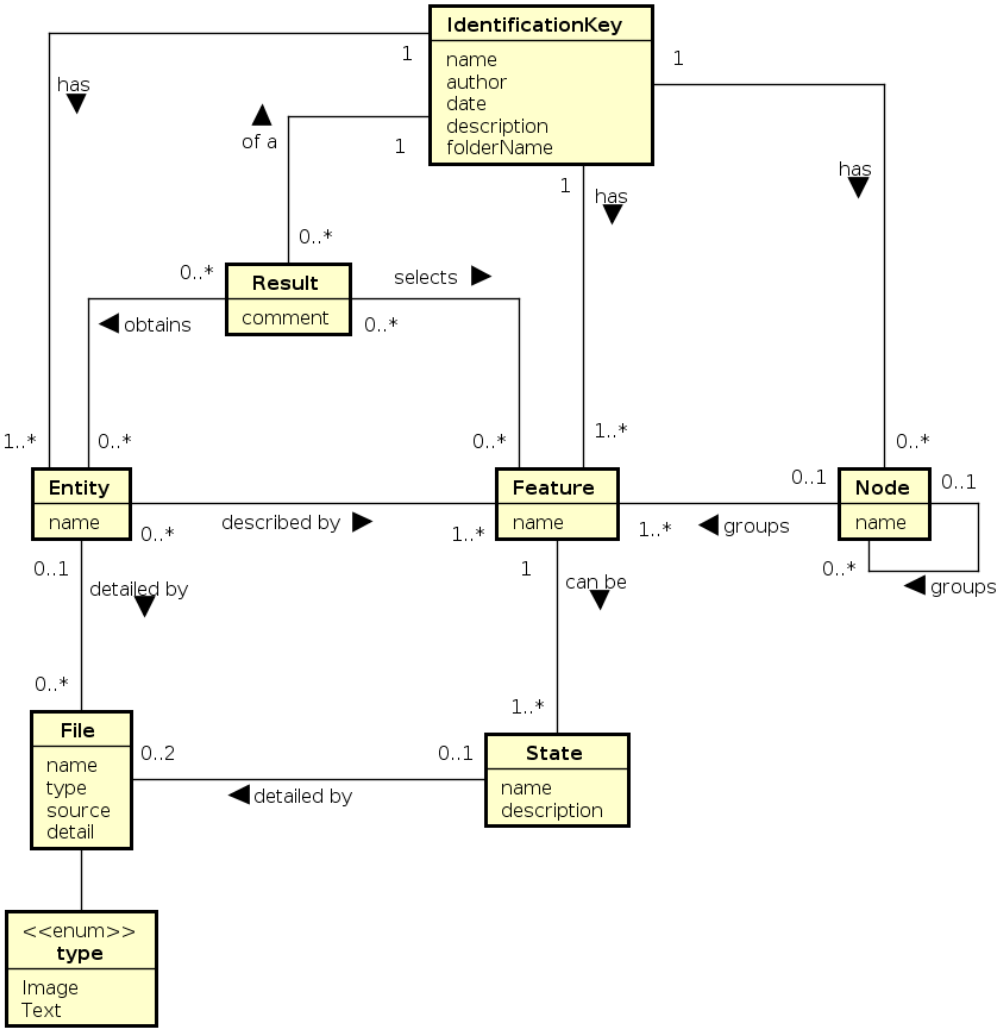

Figura 8 – Diagrama de Classes para o subsistema InteractiveKey.

4 Projeto Arquitetural

Como proposto na Seção 2.3, após as fases de especificação e análise de requisitos, ocorre a fase de projeto do sistema. O objetivo dessa fase é produzir uma solução para o problema identificado e modelado nas fases de levantamento e análise de requisitos, incorporando a tecnologia aos requisitos e projetando o que será construído na implementação (FALBO, 2016).

Este capítulo exibe os resultados obtidos na fase de projeto do sistema. A Seção 4.1 apresenta o projeto da arquitetura de software, enquanto a Seção 4.2 discute as tecnologias utilizadas para o desenvolvimento do projeto e, por fim, a Seção 4.3 apresenta os modelos FrameWeb. A lista de táticas utilizadas para tratar os requisitos não funcionais (atributos de qualidade) pode ser encontrada no **Documento de Projeto de Sistema** que está disponível no Apêndice ao final dessa monografia.

4.1 Arquitetura do Sistema

A arquitetura de software do Portal de Biodiversidade de Chalcidoidea baseia-se no padrão MVC (*Model*, *View*, *Controller*). Esse padrão tem como objetivo dividir a aplicação em 3 partes interconectadas, isso é feito para separar como a informação é representada internamente da forma como ela é apresentada para o usuário. A Figura 9 apresenta a visão geral das camadas juntamente com os relacionamentos entre elas. Dessa forma, temos que:

- **Model:** camada responsável por gerenciar os dados da aplicação. Neste projeto foi utilizada a biblioteca *Mongoose* para modelagem dos dados e para a interação com o banco de dados;
- **View:** camada responsável pela interação com o usuário. Define como os dados da aplicação serão apresentados, através de páginas HTML, *templates*, formúlários, entre outros. Também é utilizada para receber dados de entrada do usuário;
- **Controller:** camada responsável por atualizar o modelo e a visão em resposta a requisições do usuário. O *Controller* interpreta as requisições vindas da *View*, atualiza e buscas as informações necessárias na camada *Model* e a partir disso envia uma resposta de volta para a *View*.

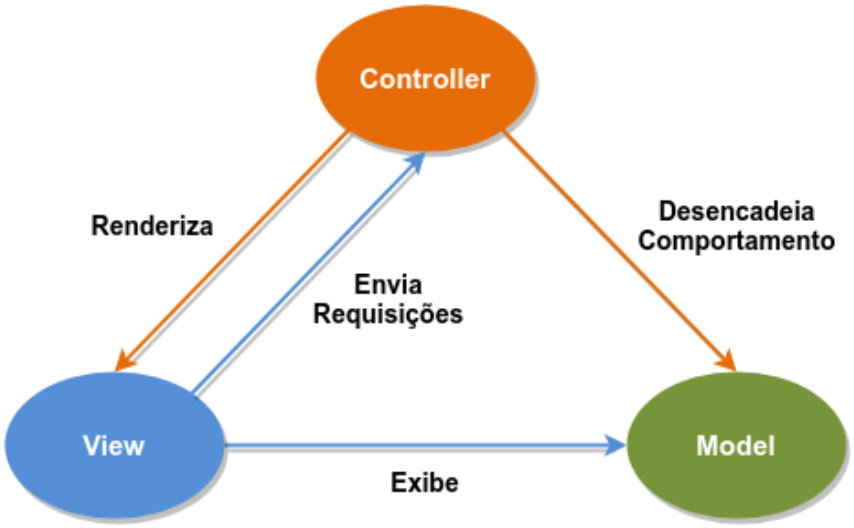

Figura 9 – Representação do padrão de design MVC.

4.2 Tecnologias Utilizadas

Foram utilizadas diversas tecnologias para implementar o Portal de Biodiversidade de Chalcidoidea. Sendo a principal delas o *framework* Express.js, um *framework* Web minimalista e flexível do Node.js para a construção de aplicações Web e APIs. Ele é o responsável por integrar todas as tecnologias que serão apresentadas a seguir, seguindo o padrão de arquitetura MVC.

Para implementar a camada *Controller* foi utilizado a classe **express.Router** do próprio Express. A partir dessa classe são definidas as rotas da aplicação. Essas rotas definem como a aplicação responde a uma requisição do cliente por um *endpoint* específico. Sendo que cada rota possui uma função que é executada quando o *endpoint* é correspondido, ou seja, quando a requisição feita pelo usuário possui o mesmo método HTTP e a mesma URL da rota definida. Dentro dessas funções ocorre a atualização do modelo e da visão.

Para armazenar os dados da aplicação foi utilizado o MongoDB, um banco de dados NoSQL que utiliza um modelo de dados orientado a documentos. Em conjunto com o MongoDB também foi utilizado o *framework* ORM (*Object/Relational Mapping*) Mongoose. O Mongoose oferece uma solução baseada em esquemas para modelar os dados da aplicação e age como o *front end* do MongoDB, representando os dados do banco de dados como objetos JavaScript. Essas duas tecnologias são as responsáveis por implementar a camada *Model*.

Por fim, para implementar a camada *View* foi utilizado a *template engine* Pug. Uma *template engine* é uma ferramenta que permite criar conteúdo HTML de forma dinâmica, oferecendo mecanismos para renderizar dados fornecidos pela aplicação em páginas HTML. Com o objetivo de auxiliar o desenvolvimento dessa camada, optou-se também por utilizar o **jQuery**, uma biblioteca de funções JavaScript que simplifica a criação de *scripts* para

o navegador do cliente, e o **Bootstrap**, um *framework* web que oferece uma série de componentes responsivos pré-definidos, agilizando assim o desenvolvimento das páginas da aplicação.

Além disso, foram utilizados diversos *middlewares* que fornecem funcionalidades essenciais para uma aplicação Web. Tais funcionalidades não estão presentes no Express.js originalmente devido ao fato do mesmo ser um *framework* minimalista. Dentre esses *middlewares* podemos citar: o **Helmet**, uma coleção de 13 pequenas funções que configuram cabeçalhos HTTP relacionados a segurança; o **Multer**, que fornece suporte para *upload* de arquivos no sistema; o **Passport** é um *middleware* de autenticação para Node.js e fornece o controle de acesso a certas áreas do sistema através da autenticação de usuários; o **Xml2js** é um conversor de XML para JSON e auxilia a interpretação de arquivos SDD; o **Compression**, *middleware* utilizado para a compressão de respostas HTTP; o **Csurf**, utilizado para proteger o sistema contra falsificações de solicitação *cross-site*.

O Controle de tráfego para a aplicação e feito por meio do **Nginx**, um servidor HTTP e proxy reverso. E o controle de sessões do usuário é feito por meio do **Redis**, que permite armazenamento de estrutura de dados de chave-valor na memória, com o auxílio do *middleware* **connect-redis**, responsável por realizar a conexão entre o Express.js e o Redis.

4.3 Modelos FrameWeb

Nesta seção, são exibidos os modelos FrameWeb, citados anteriormente na Seção 2.5. Esses modelos também estão divididos nas camadas da arquitetura do sistema, conforme citado na Seção 4.1.

4.3.1 Modelo de Domínio

O Modelo de Domínio é um diagrama de classes da UML que representa os objetos de domínio do problema e seu mapeamento para a persistência em banco de dados relacional. Nele, o modelo de classes construído na fase de análise de requisitos é adequado para à plataforma de implementação escolhida.

Para este projeto o Modelo de Domínio foi adaptado para o MongoDB, um banco de dados NoSQL que utiliza um modelo de dados orientado a documentos. No MongoDB uma coleção é equivalente a uma tabela em um banco de dados relacional, da mesma forma, documentos equivalem a linhas da tabela.

Todas as classes de domínio são modeladas a partir de *Schemas* do Mongoose, uma biblioteca do Node.js utilizada para modelar os dados da aplicação, sendo que cada *Schema* é mapeado para uma coleção no MongoDB. Os seguintes tipos de variáveis são

aceitos em um *Schema*:

- String;
- Number;
- Date;
- Buffer;
- Boolean;
- Mixed;
- ObjectId;
- Array;
- Decimal128;
- Map.

Informações adicionais sobre *Schemas* estão disponíveis na documentação oficial do Mongoose.[1]

A Figura 10 mostra o modelo de entidades para o subsistema **Main**, nela é possível ver que todas as classes possuem um atributo do tipo **ObjectId** com o esteriótipo **«id»**, indicando que esse atributo é a chave primária da classe.

A Figura 11 representa o modelo de entidades para o subsistema **InteractiveKey**. Nela é possível ver a presença da restrição *null* em atributos das classes **Result**, **File** e **State**, indicando que esses campos podem ser nulos no banco de dados.

Diferentemente da abordagem original proposta em 2007, todos os atributos que não podem ser nulos tiveram a *tag not null* omitida e aqueles que podem tiveram a *tag null* adicionada de forma a reduzir o impacto visual nos diversos diagramas. Além disso, foi considerada que a estratégia padrão de recuperação de uma associação é do tipo *lazy*, e não *eager* como proposto pelo FrameWeb, devido a forma como o MongoDB opera.

4.3.2 Modelo de Navegação

O Modelo de Navegação é um diagrama de classe da UML que representa os diferentes componentes que formam a camada de Apresentação, como páginas Web, formulários HTML e classes de ação. Esse modelo é utilizado para guiar a codificação das camadas *View* e *Controller*. A classe de ação é o principal componente deste modelo e

[1] <https://mongoosejs.com/docs/guide.html>.

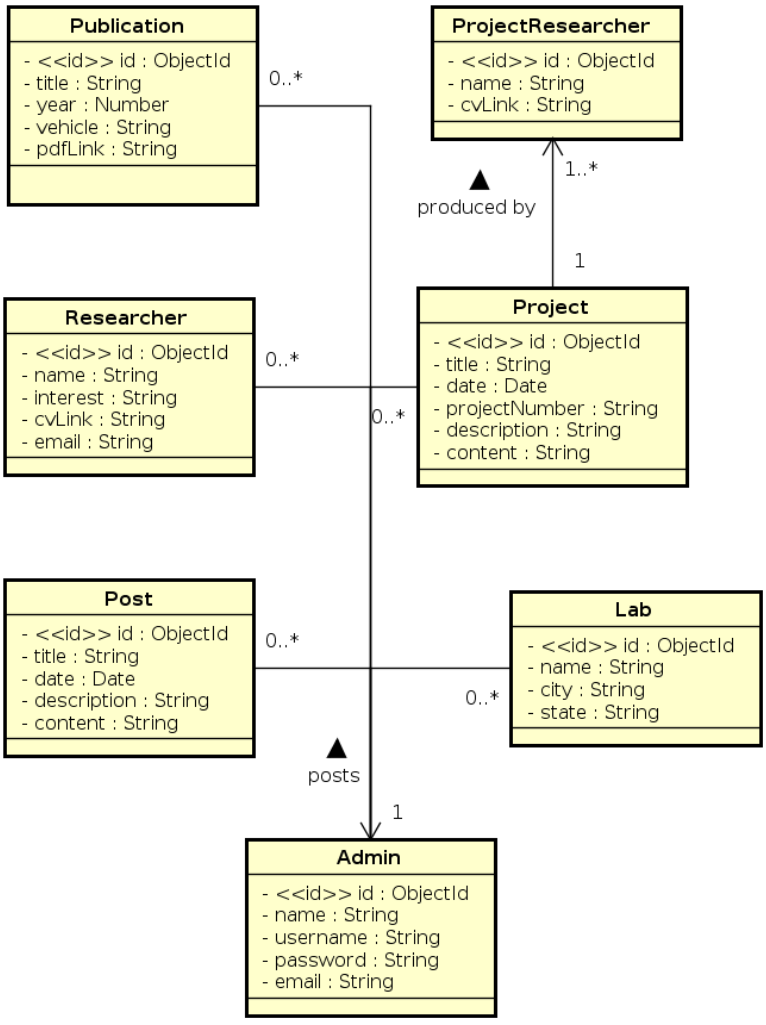

Figura 10 – Modelo de Entidades do subsistema Main.

suas associações de dependência ditam o controle de fluxo quando uma requisição é feita pelo usuário.

As funcionalidades criar, visualizar, editar e excluir (abreviadas de CRUD, do inglês *create, read, update and delete*), seguem um mesmo fluxo de execução e de interação com o usuário. Devido ao fato dessas funcionalidades serem similares para todos os casos de uso cadastrais, optou-se por representar esse fluxo de execução na Figura 12 por meio de um Modelo de Navegação genérico. Para isso foi utilizado o caso de uso cadastral de **Publicação** como exemplo.

Este modelo genérico pode ser aplicado para todas as entidades que possuem funcionalidades do tipo CRUD. Dessa forma, cada entidade possui um *controller* que receberá as requisições dos usuários. Para exibir as informações na camada de visão, o sistema utiliza um *template* para a listagem de entidades (*/publications*), um para detalhamento (*/publications/:id*) e um para gerenciamento (*/dashboard/publications*), sendo que na página de gerenciamento da entidade existe um formulário para a criação

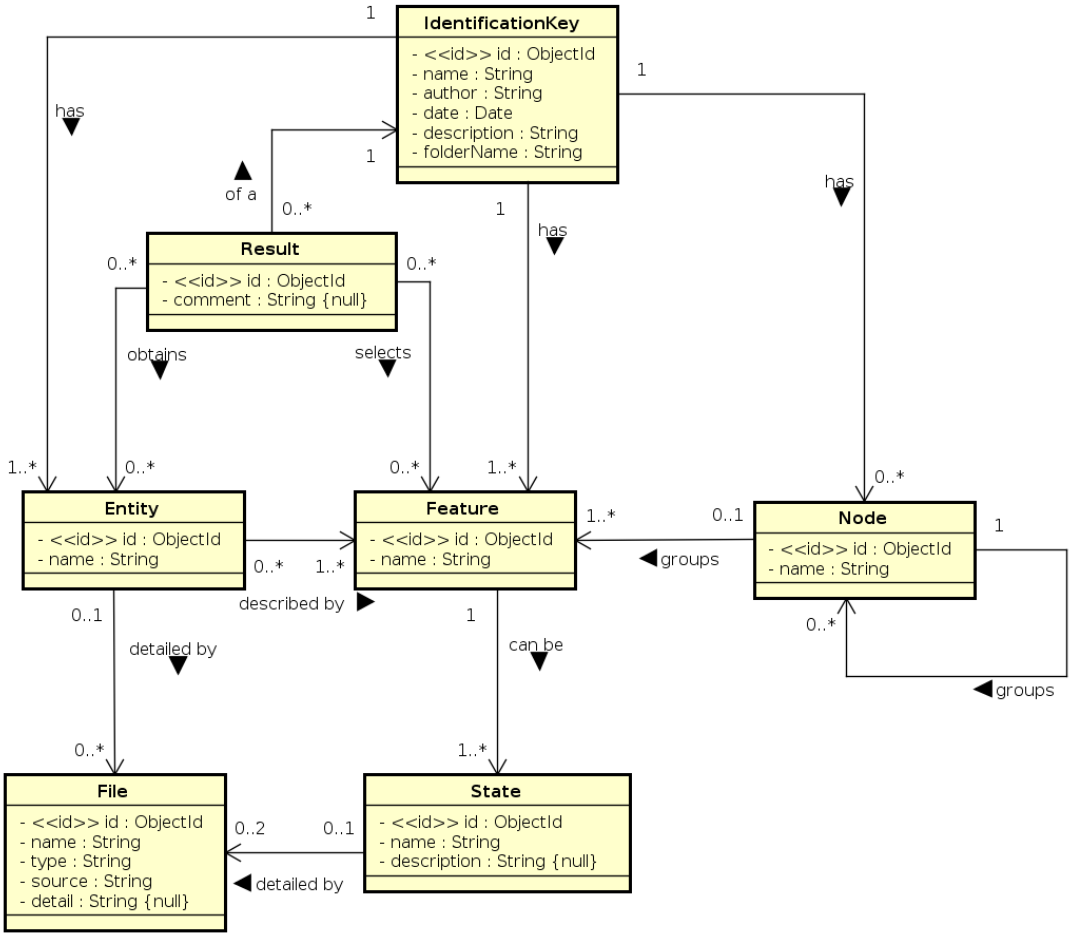

Figura 11 – Modelo de Entidades do subsistema InteractiveKey.

de uma nova entidade (*/dashboard/publications/create*) e formulários para a edição de entidades existentes (*/dashboard/publications/:id/edit*). É importante notar a presença da restrição **ajax**, indicando que os métodos **getPublication** e **delete** são feitos através de uma requisição AJAX (*Asynchronous JavaScript and XML*) por meio da qual o usuário indica na interface a entidade que deseja visualizar ou deletar, respectivamente, e uma requisição é enviada para o controlador contendo o ID da entidade escolhida.

Para os casos de uso que apresentam funções diferentes do que apenas as funcionalidades do tipo CRUD, o modelo de navegação mostrado anteriormente não pode ser aplicado. A Figura 13 apresenta o modelo de navegação para os casos de uso relacionados à autenticação no sistema, incluindo as funcionalidades para registrar um administrador, efetuar *login*, efetuar *logout*, alterar senha e deletar um administrador. Nesse diagrama, é possível observar a presença da restrição **result** indicando o retorno obtido na requisição. Caso ocorra algum erro durante a autenticação de um usuário (*result = error*), ele é redirecionado para a página de login novamente para que possa corrigir seu erro, caso tudo ocorra normalmente (*result = success*) ele é redirecionado para a *dashboard*.

A Figura 14 apresenta o modelo de navegação para o fluxo de uma chave de

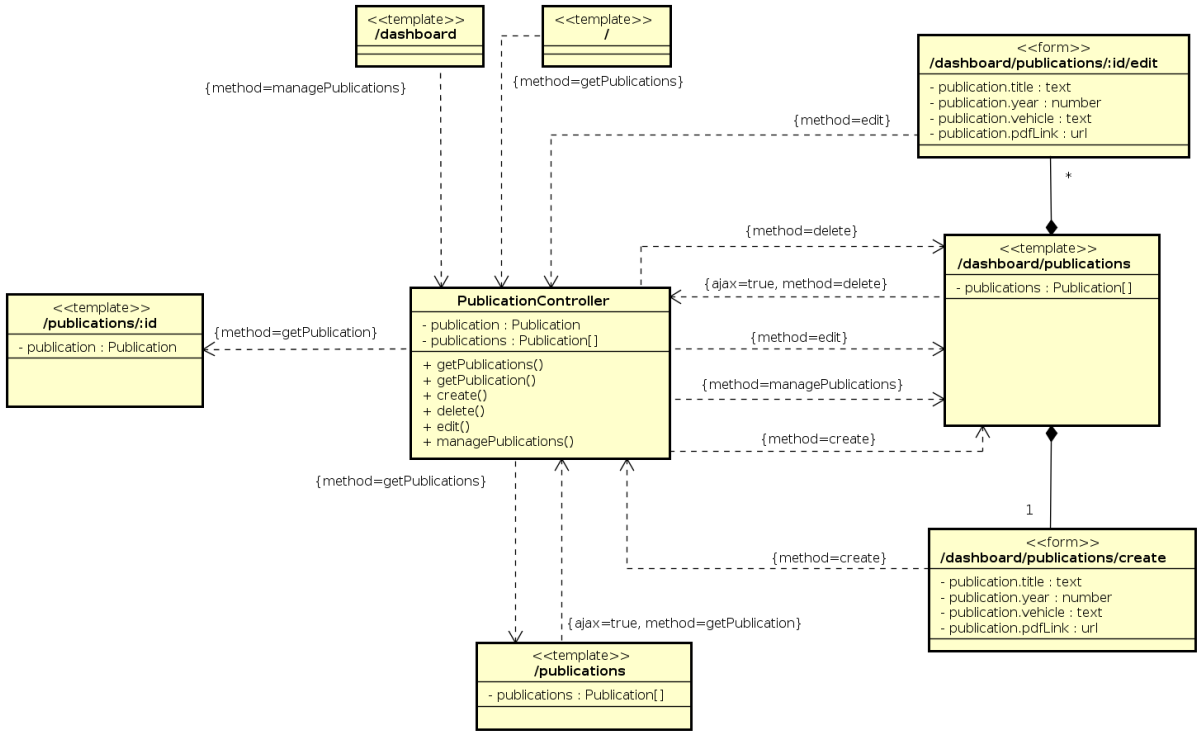

Figura 12 – Modelo de navegação de um CRUD usando como base os casos de uso cadastrais para publicações.

identificação, incluindo os casos de uso **Selecionar Estados**, **Desmarcar Estados**, **Desmarcar Todos os Estados** e **Visualizar Entidades**. A partir da página principal de uma chave de identificação (*/key/:id*) o usuário pode acessar os seus caracteres e selecionar e desmarcar os estados do caractere acessado. Essa informação é enviada para o controlador através de requisições AJAX, o controlador por sua vez atualiza a lista de estados selecionados (exibida no template */key/:id/selectedStates*) e a lista de resultados obtidos (*/key/:id/results*). Também existe a opção de desmarcar todos os estados selecionados, deixando assim a lista de estados selecionados vazia, nesse caso, o sistema retorna a lista de resultados para o seu estado original, onde todas as entidades fazem parte da mesma.

Nesse diagrama, é possível notar a presença da restrição **resultType** indicando o tipo de retorno da requisição, os métodos **addState**, **removeState** e **clearStates** retornam um conteúdo em formato JSON (*JavaScript Object Notation*) como resposta da requisição.

4.3.3 Modelos de Persistência e Aplicação

O Modelo de Persistência é um diagrama de classes da UML que representa as classes DAO existentes, responsáveis pela persistência das instâncias das classes de domínio. Esse diagrama guia a construção das classes DAO, que pertencem ao pacote de

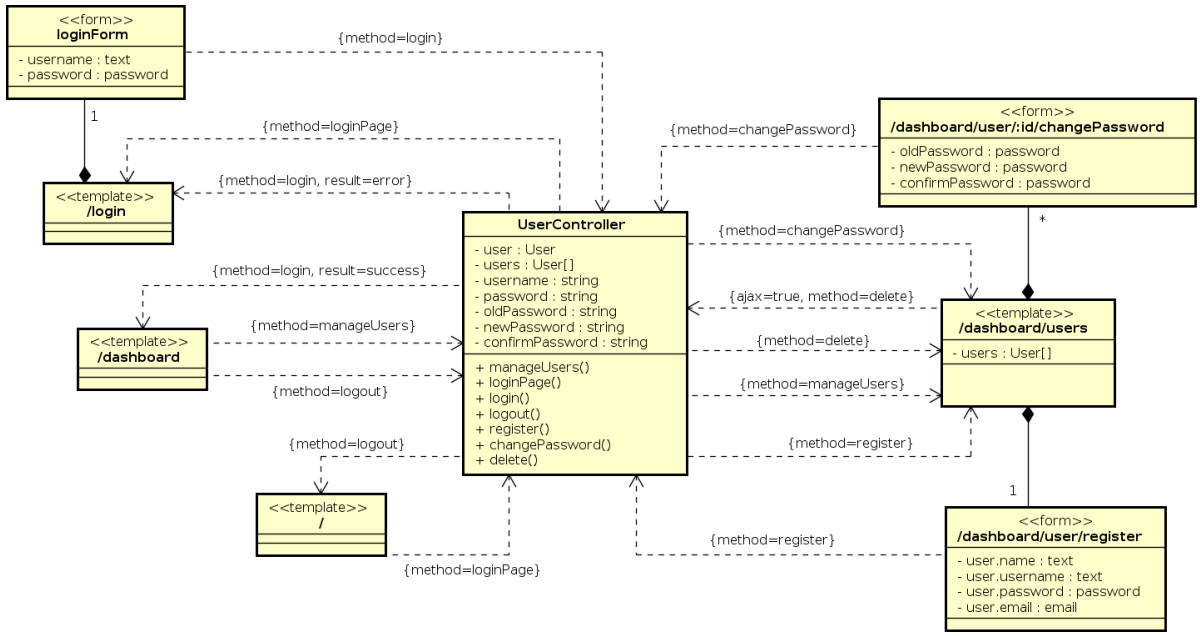

Figura 13 – Modelo de navegação para o fluxo de autenticação no sistema.

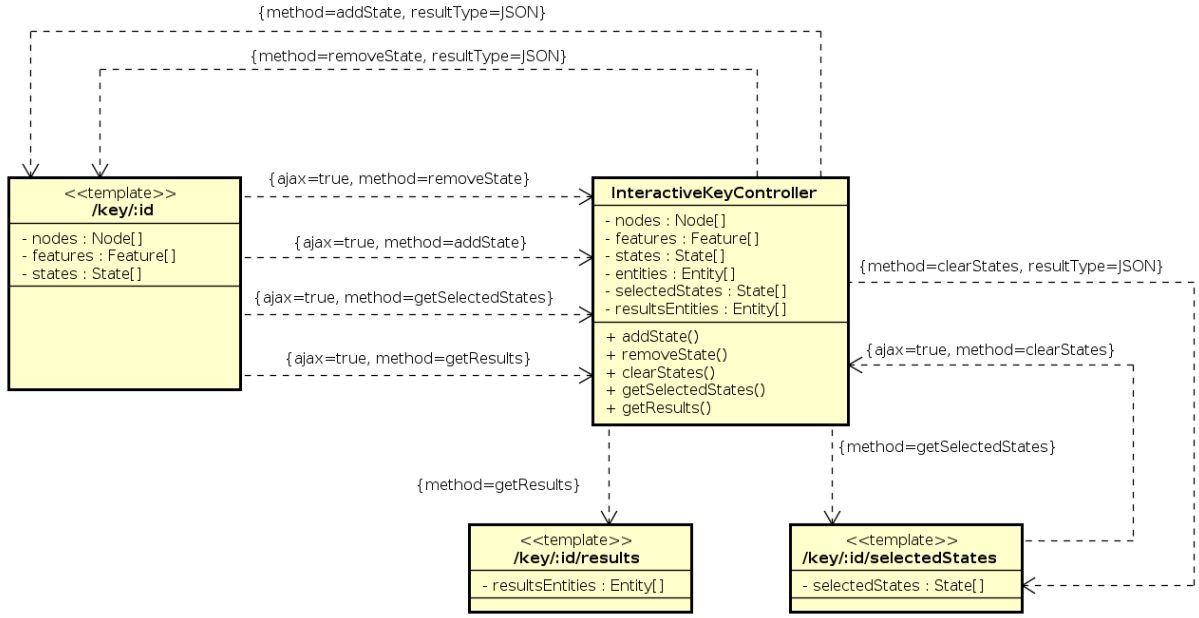

Figura 14 – Modelo de navegação para as funcionalidades relacionadas a uma chave de identificação.

persistência (SOUZA, 2007).

O Modelo de Aplicação, por sua vez, representa as classes de serviço, que são responsáveis pela codificação dos casos de uso, e suas dependências. Esse diagrama é utilizado para guiar a implementação das classes do pacote Aplicação e a configuração das dependências entre os pacotes Controle, Aplicação e Persistência, ou seja, quais classes de ação dependem de quais classes de serviço e quais DAOs são necessários para que as classes de serviço alcancem seus objetivos (SOUZA, 2007).

Ambos os modelos não foram utilizados na elaboração dessa monografia, sendo que a principal razão para isso foi o uso da biblioteca *Mongoose* do Node.js, que fornece um mapeamento de objetos do MongoDB similar ao ORM (*Object Relational Mapping*). O *Mongoose* oferece métodos para fazer consultas ao banco, adicionar, editar e deletar dados. Além disso, todos os dados do banco de dados são traduzidos para objetos JavaScript, tornando a manipulação desses dados algo bem simples e trivial de ser feito. Com isso, não foram utilizadas classes responsáveis pela persistência de dados, uma vez que os métodos oferecidos pelo *Mongoose* foram suficientes para cumprir esse papel sem a necessidade de uma camada adicional. Por fim, todas as funcionalidades propostas para o sistema possuem uma complexidade baixa, sendo que em sua grande maioria são apenas CRUDs. Juntando isso à facilidade de manipular o banco de dados oferecida pelo *Mongoose*, foi descartada a necessidade de implementar classes de serviço. Dessa forma, todas as funcionalidades do sistema foram implementadas diretamente nos controladores utilizando o *Mongoose* para acessar o banco de dados.

5 Implementação e Apresentação

Neste capítulo serão apresentados aspectos da implementação do sistema, bem como seu resultado final por meio de diversas capturas de tela. A Seção 5.1 apresenta detalhes da implementação do sistema com o framework *Express.js*, a Seção 5.2 apresenta o resultado final através de capturas de tela das principais funcionalidades do sistema.

5.1 Implementação do Sistema

Conforme mencionado na Seção 4.1, o Portal de Biodiversidade de Chalcidoidea foi desenvolvido utilizando o padrão MVC e o framework *Express.js*. Dessa forma, o sistema foi organizado em diversos diretórios, conforme a Figura 15. Sendo que dentre todas as pastas e arquivos, destacam-se os seguintes (pastas são precedidas por "/"e arquivos não):

- **/models** – Contém todos os modelos do sistema que são armazenados no banco de dados, referente à camada *Model* do padrão MVC;
- **/routes** – Contém todas as rotas responsáveis por receber requisições do usuário e atualizar o modelo e a visão, referente à camada *Controller* do padrão MVC;
- **/views** – Contém todas as páginas Web que serão exibidas ao usuário, referente à camada *View* do padrão MVC;
- **/public** – Contém todos os arquivos estáticos que são enviados para o cliente:
  - **/javascripts** – Contém os arquivos JavaScript enviados para o cliente;
  - **/stylesheets** – Contém os arquivos CSS enviados para o cliente;
  - **/uploads** – Contém os arquivos referentes às chaves de identificação do portal.
- **/node_modules** – Contém todos os pacotes instalados utilizados pela aplicação;
- **app.js** – Arquivo responsável pela inicialização de toda aplicação, nele ocorre a inclusão dos demais arquivos e pacotes utilizados pelo sistema;
- **readfile.js** – Arquivo responsável pela leitura e interpretação de arquivos SDD e pelo armazenamento de suas informações no banco de dados;
- **package.json** – Arquivo responsável por informar as dependências do projeto. Também pode conter outras informações importantes, como o nome do projeto, versão, autor, entre outros detalhes;

- **/bin** – Contém arquivos utilizados para inicializar a aplicação:
  - **www** – Arquivo utilizado para inicializar o servidor e para ouvir e reportar informações vindas do mesmo. Também é responsável por chamar **app.js**.

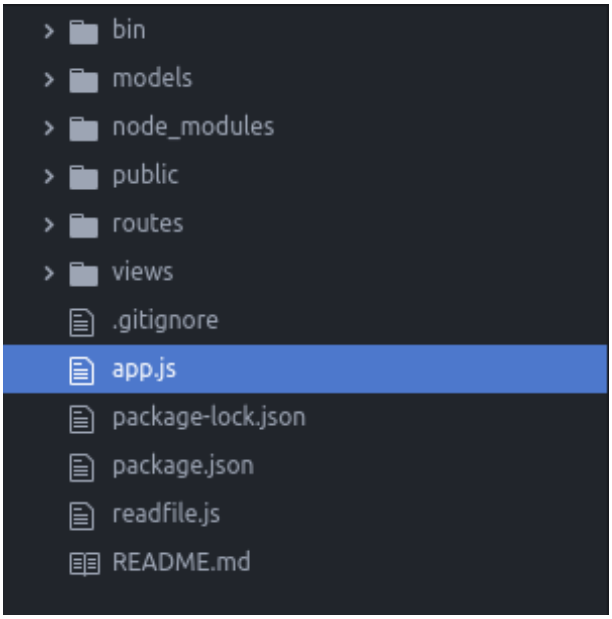

Figura 15 – Estrutura de diretórios utilizada no sistema.

5.1.1 package.json

Esse arquivo fica localizado na raiz do projeto, e é responsável por dar informações ao npm (o gerenciador de pacotes do node.js) para que este possa lidar com a lista de dependências da aplicação. Também pode conter outras informações importantes, como o nome do projeto, versão, autor, entre outros detalhes. A Listagem 5.1 mostra o conteúdo deste arquivo.

Listagem 5.1 – package.json

```
1  {
2    "name": "chalcidoidea",
3    "version": "1.0.0",
4    "private": true,
5    "scripts": {
6      "start": "node ./bin/www",
7      "devstart": "nodemon ./bin/www",
8      "teste": "mocha"
9    },
10   "dependencies": {
11     "async": "^2.6.1",
12     "body-parser": "^1.18.3",
13     "compression": "^1.7.3",
14     "connect-ensure-login": "^0.1.1",
15     "connect-flash": "^0.1.1",
```

```
16      "connect-mongo": "^2.0.1",
17      "cookie-session": "^2.0.0-beta.3",
18      "csurf": "^1.9.0",
19      "debug": "^3.1.0",
20      "express": "~4.16.0",
21      "express-session": "^1.15.6",
22      "express-validator": "^5.3.0",
23      "fs-extra": "^7.0.0",
24      "helmet": "^3.13.0",
25      "http-errors": "^1.7.0",
26      "lodash": "^4.17.10",
27      "marko": "^4.13.4-1",
28      "memorystore": "^1.6.0",
29      "mkdirp": "^0.5.1",
30      "moment": "^2.22.2",
31      "mongoose": "^5.2.10",
32      "morgan": "^1.9.0",
33      "multer": "^1.3.1",
34      "passport": "^0.4.0",
35      "passport-local": "^1.0.0",
36      "passport-local-mongoose": "^5.0.1",
37      "promise": "^8.0.1",
38      "pug": "^2.0.3",
39      "sanitize-html": "^1.18.4",
40      "session-file-store": "^1.2.0",
41      "xml2js": "^0.4.19"
42    },
43    "devDependencies": {
44      "mocha": "^5.2.0",
45      "nodemon": "^1.18.3"
46    }
47  }
```

Além das dependências de produção (**"dependencies"**), necessárias para o funcionamento da aplicação, também existem as dependências de desenvolvimento (**"devDependencies"**), utilizadas apenas no ambiente de desenvolvimento. Por fim, na seção **"scripts"** são definidos scripts para inicializar aplicação, rodar testes, entre outras possibilidades.

Maiores informações e detalhes sobre os pacotes utilizados no projeto podem ser consultados no **Documento de Projeto de Sistema** que está disponível no Apêndice ao final dessa monografia.

5.1.2 app.js

Esse arquivo cria uma instância do express, determina uma série de configurações e *middlewares* a serem utilizados no projeto, e por fim, exporta a aplicação para ser utilizada no script de inicialização. A Listagem 5.2 mostra apenas as partes utilizadas para criar o exportar a aplicação.

Listagem 5.2 – Criando e exportando uma istância do express.

```
1 const express = require('express');
```

```
2 const app = express();
3 ...
4 module.exports = app;
```

A Listagem 5.3 demonstra a adição de *middlewares* utilizados pela aplicação através do comando **app.use**, *middlewares* são funções que podem modificar a requisição vinda do cliente, e a resposta a ser enviada. O **helmet**, utilizado para proteger a aplicação contra vulnerabilidades da Web, e o **compression**, utilizado para comprimir as repostas HTTP são exemplos de *middlewares* utilizados nesse projeto. Também é possível ver a definição do valor **"view engine"**, responsável por definir a *template engine* utilizada no projeto, e a definição da pasta onde se encontram os arquivos estáticos, por meio da função **express.static**.

Listagem 5.3 – Adição dos middlewares utilizados pela aplicação.

```
1 app.use(helmet());
2 app.use(compression());
3 app.use(express.static(path.join(__dirname, 'public')));
4
5 // view engine setup
6 app.set('views', path.join(__dirname, 'views'));
7 app.set('view engine', 'pug');
8
9 app.use(logger('dev'));
10 app.use(express.json());
11 app.use(express.urlencoded({extended: true}));
```

Por fim, são definidos os controladores (previamente importados) responsáveis por cada rota da aplicação. Na Listagem 5.4 abaixo é possível ver que todas as requisições para a rota **/users** serão gerenciadas pelo controlador **usersRouter**, por exemplo.

Listagem 5.4 – Adição das rotas da aplicação.

```
1 app.use('/', indexRouter);
2 app.use('/users', usersRouter);
3 app.use('/entities', entitiesRouter);
4 app.use('/nodes', nodesRouter);
5 app.use('/keys', keysRouter);
6 app.use('/results', resultsRouter);
7 app.use('/dashboard', dashboardRouter);
8 app.use('/publications', publicationsRouter);
9 app.use('/projects', projectsRouter);
10 app.use('/researchers', researchersRouter);
11 app.use('/labs', labsRouter);
12 app.use('/posts', postsRouter);
13 app.use('/saves', savesRouter);
14 app.use('/about', aboutRouter);
```

5.1.3 Models

Para armazenar os dados da aplicação foi utilizado o **MongoDB**, um banco de dados NoSQL que utiliza um modelo de dados orientado a documentos. No MongoDB uma coleção é equivalente a uma tabela em um banco de dados relacional, da mesma forma, documentos equivalem a linhas da tabela. Em conjunto com o MongoDB também foi utilizado o ORM (Object/Relational Mapping) *Mongoose*. O *Mongoose* oferece uma solução baseada em esquemas para modelar os dados da aplicação e age como o *front end* do MongoDB, representando os dados do banco de dados como objetos JavaScript.

Modelos são definidos a partir de *Schemas*. Cada *Schema* é mapeado para uma coleção no banco de dados, e define os campos que cada documento deve ter. *Schemas* são convertidos em modelos usando o método **mongoose.model()**. A partir de um modelo é possível utilizar métodos fornecidos pelo *Mongoose* para criar, procurar, atualizar e excluir objetos no banco de dados.

Na Listagem 5.5 é exibido o modelo para uma Publicação. Uma publicação possui os campos **title**, **vehicle**, **year** e **pdfLink**, sendo que cada campo é representado por um objeto contendo as características desse campo, como o seu tipo (**type**), e a sua obrigatoriedade (**required**). O *Mongoose* também cria um campo **id** automaticamente, caso um não seja definido explicitamente.

Também é possível observar a criação de uma propriedade virtual de nome **url**, em conjunto com um método **get** para a mesma, a partir do comando **virtual('url')**. Essa propriedade define a URL de uma publicação a partir do seu **id**.

Por fim, o modelo da publicação é criado a partir do seu *schema* com o comando **mongoose.model('Publication', PublicationSchema)** e exportado para posteriormente ser utilizado pelo controlador **publicationsRouter** para fazer as operações com o banco de dados.

Listagem 5.5 – Criando um Modelo para Publicação. .

```
1  const mongoose = require('mongoose');
2
3  const Schema = mongoose.Schema;
4
5  const PublicationSchema = new Schema(
6    {
7      title: {type: String, required: true},
8      vehicle: {type: String, required: true},
9      year: {type: Number, required: true},
10     pdfLink: {type: String},
11   }
12 );
13
14 PublicationSchema
15 .virtual('url')
16 .get(function () {
```

```
17     return '/publications/' + this._id;
18 });
19
20 //Export model
21 module.exports = mongoose.model('Publication', PublicationSchema);
```

5.1.4 Routes

As rotas definem como a aplicação responde a uma requisição do cliente por um *endpoint* específico. Um *endpoint* nada mais é que uma URL (ou caminho) e um método de solicitação HTTP específico (GET, POST, PUT, entre outros). Cada rota possui uma função que é executada quando o *endpoint* é correspondido, ou seja, quando a requisição feita pelo usuário possui o mesmo método HTTP e a mesma URL da rota definida. No *Express.js* uma rota é definida como na Listagem 5.6, onde:

- **app** é uma instância do express;
- **METHOD** é um método de solicitação HTTP;
- **PATH** é a URL que deve ser correspondida;
- **HANDLER** é a função executada quando a rota é correspondida.

Listagem 5.6 – Exemplo de uma definição de rota.

```
1 app.METHOD(PATH, HANDLER)
```

Para modularizar o gerenciamento das diferentes partes do portal, foram criados controladores diferentes para cada seção, como visto na Listagem 5.4. Dessa forma, quando a URL de uma seção específica é acessada, o seu controlador fica responsável por gerenciar as requisições recebidas.

A Listagem 5.7 exibe uma das rotas definidas pelo controlador de Laboratórios (**labsRouter**), essa rota possui o método HTTP GET e a URL **/labs/** (A URL é a junção da rota especificada quando o controlador foi importado em app.js, '**/labs**', com a URL definida dentro do controlador, '/'), e é responsável por retornar todos os laboratórios disponíveis. É possível observar a utilização do método **Lab.find()** fornecido pelo *Mongoose* para consultar o banco de dados. Por fim, o método **res.render** renderiza a página **'labs'** como resposta a requisição.

Listagem 5.7 – Rota do Controlador de Laboratórios responsável por retornar todos os laboratórios.

```
1 router.get('/', (req, res, next) => {
2   Lab.find()
3   .sort({year: -1})
```

```
4   .exec().then((labs) => {
5     if(labs == null){
6       var err = new Error('Labs not found');
7       err.status = 404;
8       next(err);
9     }
10    res.render('labs', {labs: labs, user: req.user});
11    }).catch((err) => {
12      next(err);
13    });
14  });
```

A Listagem 5.8 exibe a rota responsável por criar um novo laboratório. Essa rota possui o método HTTP POST e a URL **/labs/create**. É possível observar a utilização do método **lab.save()** fornecido pelo *Mongoose* para salvar no banco de dados um novo laboratório. Também é importante notar a presença do método **ensureLoggedIn**, responsável por permitir que apenas usuários autenticados tenham acesso a essa rota. Por fim, o método **res.redirect** redireciona o usuário para a URL **/dashboard** como resposta a requisição.

Listagem 5.8 – Rota do Controlador de Laboratórios responsável por criar um laboratório.

```
1 router.post('/create', require('connect-ensure-login').ensureLoggedIn('/users/
    login'), [
2   body('name').isLength({ min: 1 }).trim().withMessage('Name must be specified.')
      ,
3   body('city').isLength({ min: 1 }).trim().withMessage('City name must be
      specified.'),
4   body('state').isLength({ min: 1 }).trim().withMessage('State name must be
      specified.'),
5   ], (req, res, next) => {
6
7   const lab = new Lab({
8     name: req.body.name,
9     city: req.body.city,
10    state: req.body.state
11  });
12  lab.save(function (err) {
13    if (err) { return next(err); }
14      res.redirect('/dashboard');
15    });
16 });
```

A Listagem 5.9 exibe a rota responsável por retornar um laboratório específico. Essa rota possui o método HTTP GET e a URL **/labs/:id**, onde **:id** é o ID referente ao laboratório. É possível observar a utilização do método **Lab.findById()** fornecido pelo *Mongoose* para consultar o banco de dados por um laboratório com um **id** específico. Assim como na Listagem 5.8, essa rota também é protegida pelo método **ensureLoggedIn**. Por fim, o método **res.render** renderiza a página **'lab_detail'** como resposta a requisição.

Listagem 5.9 – Rota do Controlador de Laboratórios responsável por retornar um laboratório específico.

```
router.get('/:id', (req, res, next) => {
  Lab.findById(req.params.id)
  .exec().then((lab) => {
    if (lab==null) {
      const err = new Error('Lab not found');
      err.status = 404;
      return next(err);
    }
    res.render('lab_detail', { title: lab.title});
  });
});
```

A Listagem 5.10 exibe a rota responsável por atualizar os dados de um laboratório específico. Essa rota possui o método HTTP PUT e a URL **/labs/:id/update**, onde **:id** é o ID referente ao laboratório. É possível observar a utilização do método **Lab.findByIdAndUpdate()** fornecido pelo *Mongoose* para consultar o banco de dados por um laboratório com um ID específico e atualizá-lo com novos dados. Assim como na Listagem 5.8, essa rota também é protegida pelo método **ensureLoggedIn**. Por fim, o método **res.redirect** redireciona o usuário para a URL **/dashboard** como resposta a requisição.

Listagem 5.10 – Rota do Controlador de Laboratórios responsável por atualizar dados de um laboratório.

```
router.put('/:id/update', require('connect-ensure-login').ensureLoggedIn('/users/
    login'), [
  body('name').isLength({ min: 1 }).trim().withMessage('Name must be specified.')
      ,
  body('city').isLength({ min: 1 }).trim().withMessage('City name must be
      specified.'),
  body('state').isLength({ min: 1 }).trim().withMessage('State name must be
      specified.'),
  ], (req, res, next) => {
    const id = mongoose.Types.ObjectId(req.params.id);
    const lab = new Lab({
      _id: id,
      name: req.body.name,
      city: req.body.city,
      state: req.body.state
    });
    Lab.findByIdAndUpdate(req.params.id, lab, {}, function (err, lab) {
      if (err) {
        return next(err);
      }
      res.redirect('/dashboard');
    });
});
```

Por fim, a Listagem 5.11 exibe a rota responsável por deletar um laboratório

específico. Essa rota possui o método HTTP DELETE e a URL **/labs/:id/delete**, onde **:id** é o ID referente ao laboratório. É possível observar a utilização do método **Lab.findByIdAndRemove()** fornecido pelo *Mongoose* para consultar o banco de dados por um laboratório com um ID específico e removê-lo do banco de dados. Assim como na Listagem 5.8, essa rota também é protegida pelo método **ensureLoggedIn**. Por fim, o método **res.send** envia um JSON com a string **"success"** como resposta à requisição.

Listagem 5.11 – Rota do Controlador de Laboratórios responsável por deletar um laboratório.

```
router.delete('/:id/delete', require('connect-ensure-login').ensureLoggedIn('/
    users/login'), (req, res, next) => {
  const id = mongoose.Types.ObjectId(req.params.id);
  Lab.findByIdAndRemove(id,function(err, lab) {
    if(err) {
      next(err);
    } else {
      res.send('success');
    }
  });
});
```

5.1.5 Views

As *views* (*templates* utilizados para criar conteúdo HTML) ficam armazenadas na pasta **/views**. Existem diversas *template engines* que podem ser utilizadas com o *framework Express.js*, para esse projeto foi utilizado a *template engine* **Pug**.

Uma *template engine* é uma ferramenta que permite criar conteúdo HTML de forma dinâmica, oferecendo mecanismos para renderizar dados fornecidos pela aplicação em páginas HTML. **Pug** utiliza uma sintaxe similar à linguagem de programação Python, onde a indentação é utilizada para definir elementos aninhados dentro de outros.

A Listagem 5.12 exibe como a rota **/publications/** do controlador **publicationsRouter** utiliza o método **res.render** para renderizar uma página HTML a partir do template **'publications'**, passando as variáveis **publications** e **user** como parâmetros.

Listagem 5.12 – Renderizando um template a partir do Controlador.

```
router.get('/', function(req, res, next) {
  Publication.find()
  .sort({year: -1})
  .exec().then((publications) => {
    if(publications == null){
      var err = new Error('Publications not found');
      err.status = 404;
      next(err);
    }
    res.render('publications', {publications: publications, user: req.user});
  }).catch((err) => {
```

```
12      next(err);
13    });
14  });
```

O *template* correspondente à rota acima (**publications.pug**) é mostrado na Listagem 5.13. Nele é possível observar mecanismos utilizados pelo Pug para exibir o *array* de publicações passado como parâmetro pela rota. Para iterar pelo *array* é usado o comando **each-in**, caso o *array* esteja vazio, o comando **else** é invocado. Variáveis são representadas dentro de **#{}**. Por fim, também é possível estender outros *templates* a partir do comando **extends**.

Listagem 5.13 – Template para exibir a lista de publicações utilizando a template engine Pug.

```
1  extends layout
2
3  block content
4    div(class="page-header")
5      h1 Portal de biodiversidade de Chalcidoidea
6      h2 Publicações
7    div(class="publications-list feed")
8      each publication in publications
9        article(class="publication article")
10         header(class='publication-header')
11           div.publication-year-box
12             span(class="publication-year") #{publication.year}
13           h2 #{publication.title}
14         div(class='publication-content')
15           div(class="publication-description")
16             p #{publication.vehicle}
17             div(class="link")
18               a(href=publication.pdfLink, target="_blank")
19                 span Ver PDF
20         hr
21     else
22       p(class='else-p') Nenhuma publicação disponível.
```

5.2 Apresentação do Sistema

Esta seção apresenta o Portal de Biodiversidade de Chalcidoidea por meio de uma série de capturas de tela. A Figura 16 mostra a página inicial do portal, onde é feita uma breve introdução sobre o seu conteúdo. A partir dessa tela todas as outras seções podem ser acessadas através de links no *header* ou através do menu lateral esquerdo.

O portal possui um *layout* responsivo, dessa forma, todas as páginas se adequarão ao tamanho da tela do aparelho sendo utilizado para acessar o portal. A Figura 17 exibe a página inicial do portal sendo visualizada a partir de um celular.

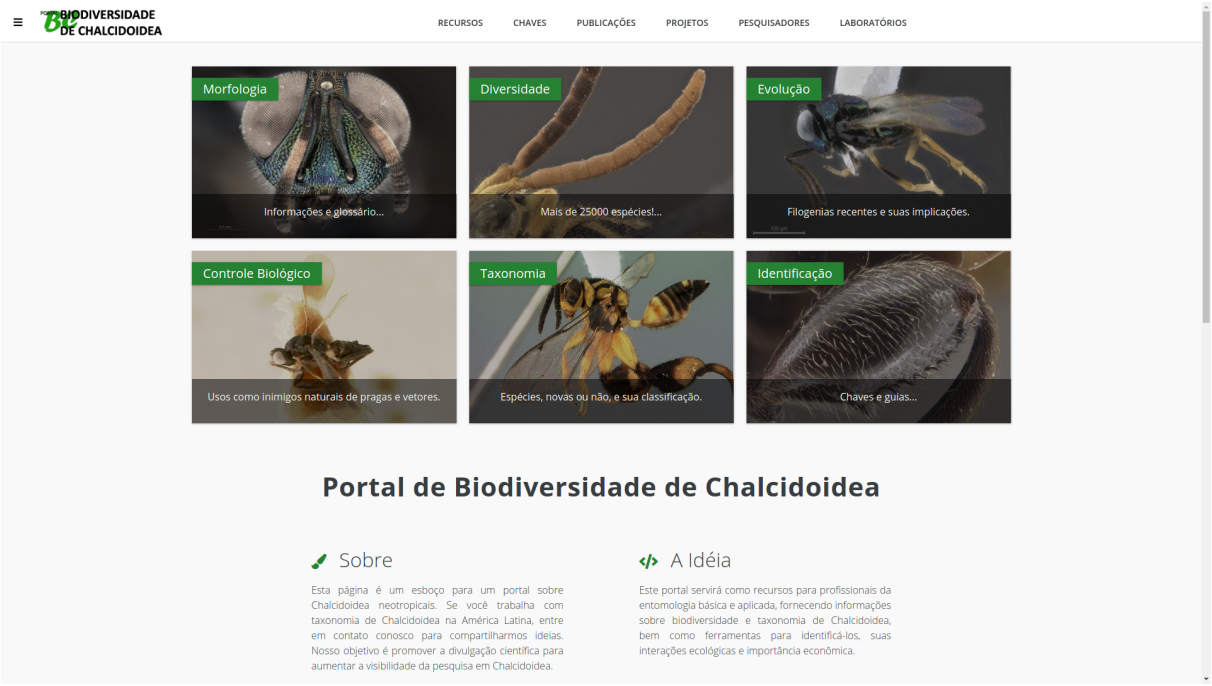

Figura 16 – Página inicial do portal.

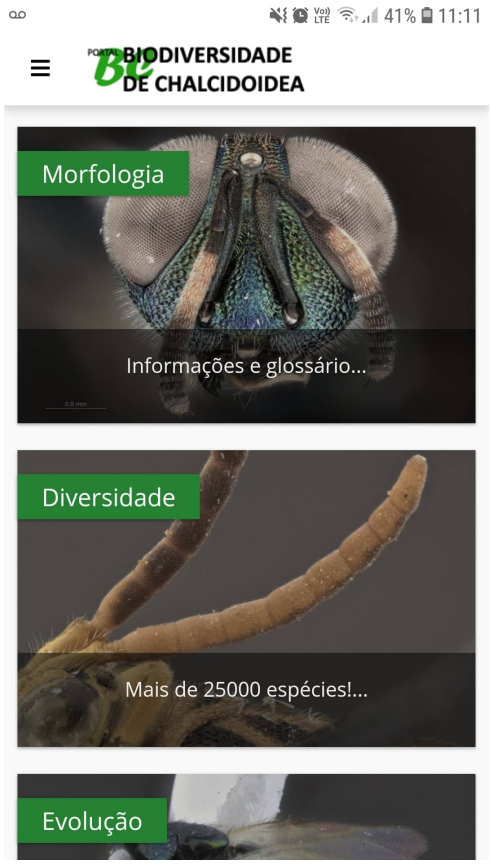

Figura 17 – Página inicial do portal visualizada em um dispositivo móvel.

5.2.1 Recursos

O portal possui telas específicas para cada tipo de recurso existente, porém, para evitar repetição, serão mostradas apenas as telas de listagem de publicações, listagem de

projetos e listagem de pesquisadores, uma vez que, as telas de laboratórios e postagens são bem similares às telas mostradas. É importante mencionar também, que todas as telas de listagem utilizam *scroll* infinito para carregar os recursos dinamicamente, sem a necessidade de recarregar a página.

A seção de publicações (Figura 18) exibe a lista de todas as publicações científicas do portal, ordenadas por data de publicação. Para cada publicação o usuário pode ver o seu nome, o veículo onde foi publicada, a data de publicação e um botão que leva ao seu PDF.

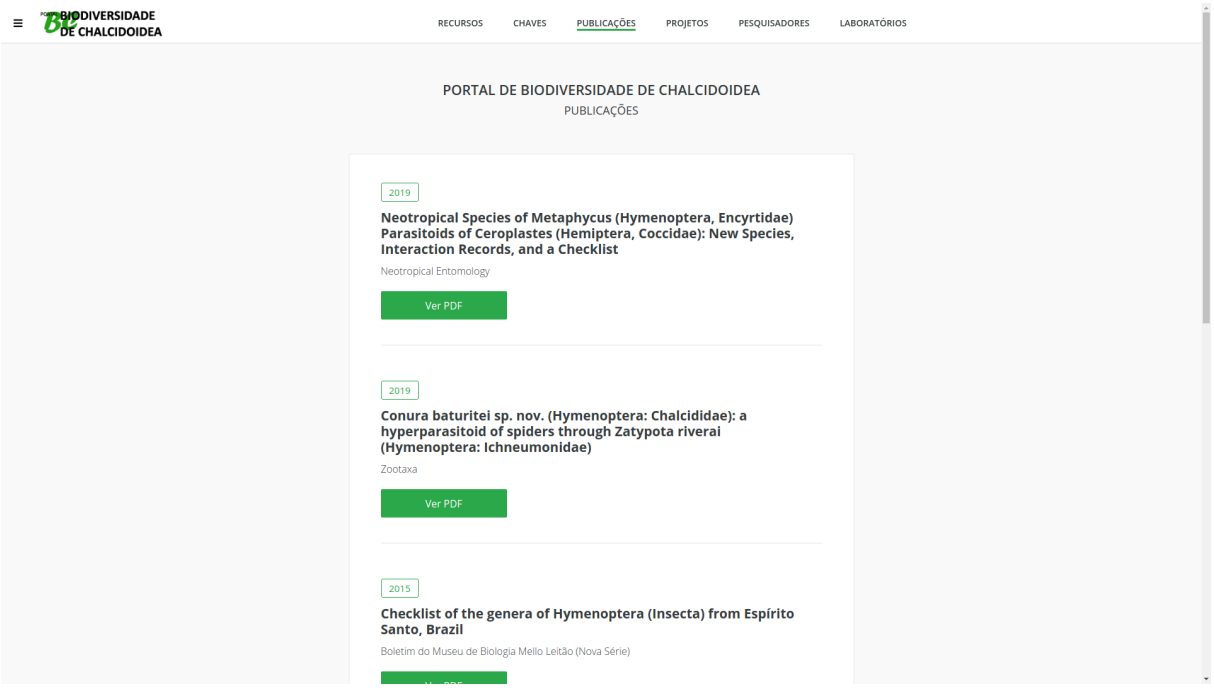

Figura 18 – Seção de publicações científicas.

A seção de projetos (Figura 19) exibe a lista de projetos. Para cada projeto são exibidos o seu título, a data de publicação, uma pequena descrição sobre o mesmo e um botão que leva para a sua página.

A seção de pesquisadores (Figura 20) exibe a lista de pesquisadores. Para cada pesquisador são exibidos o seu nome, a sua área de interesse, um e-mail para contato e um botão que leva ao seu currículo Lattes.

5.2.2 Chaves de Identificação

Na seção de chaves de identificação o usuário terá acesso a lista de chaves de identificação disponíveis. Nesta página ele poderá ver o nome, a data de publicação, o autor e a descrição de cada chave, conforme a Figura 21.

Ao escolher uma chave de identificação o usuário é levado à sua página. Essa tela é dividida em duas abas, a aba de caracteres, onde são exibidos os caracteres e agrupamentos

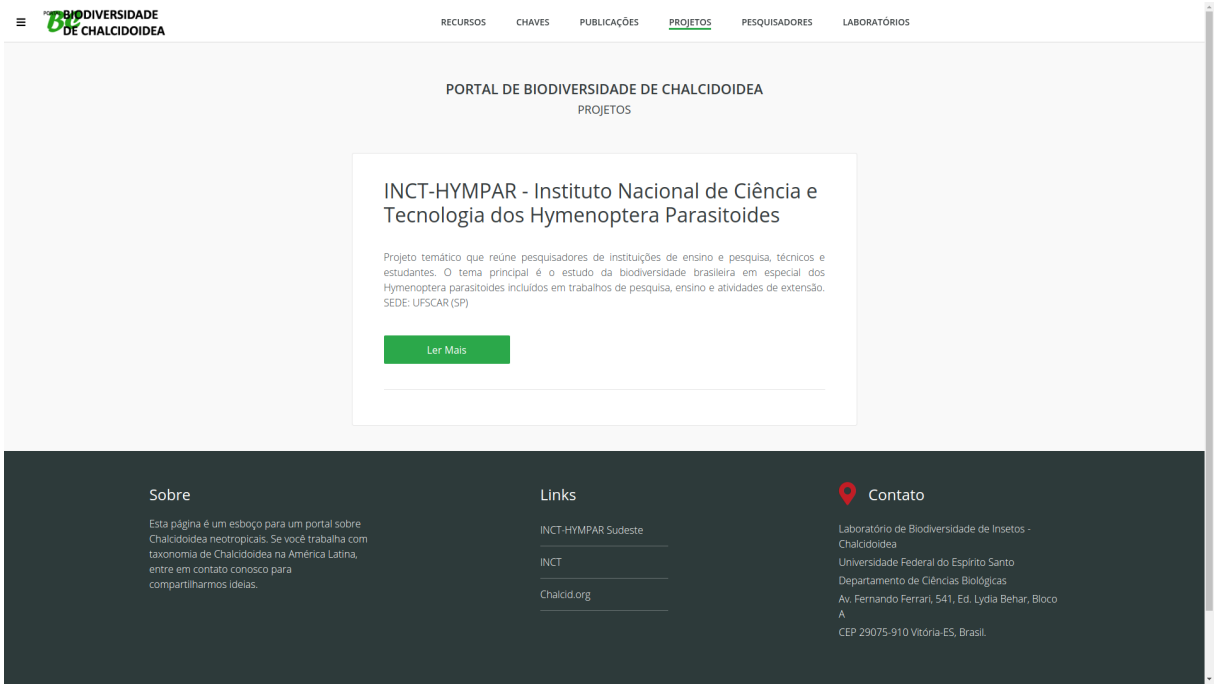

Figura 19 – Seção de projetos.

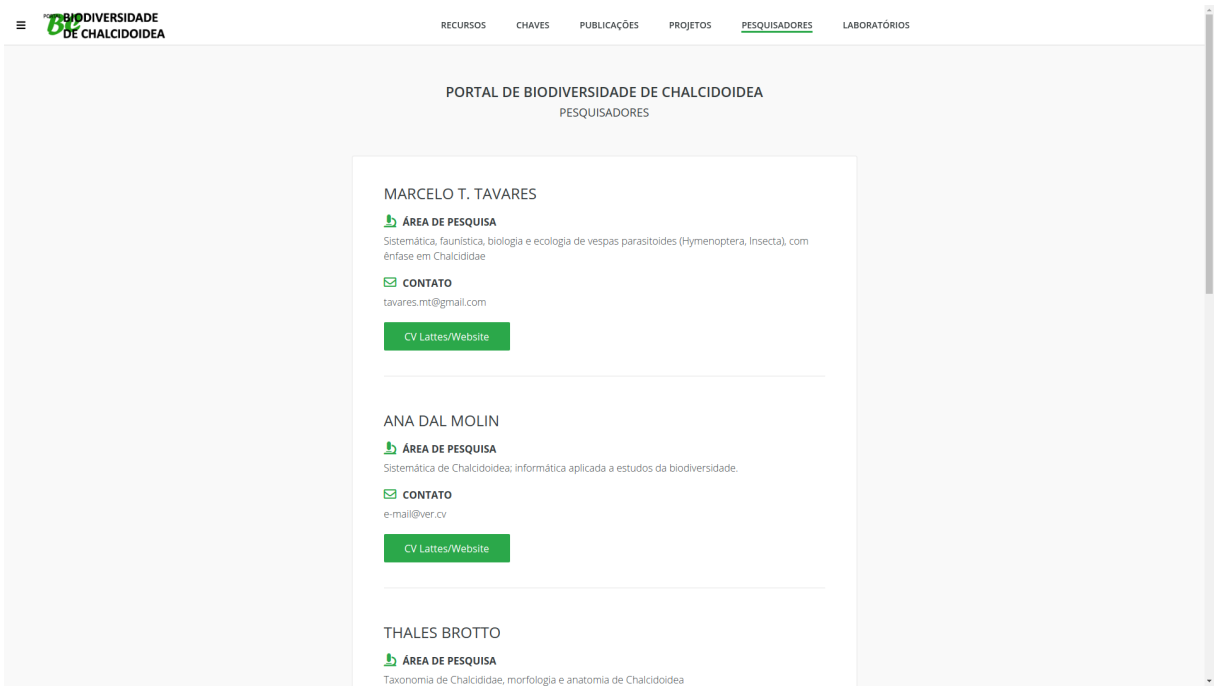

Figura 20 – Seção de pesquisadores.

da chave, e a aba de entidades, onde é exibida a lista de entidades da chave, filtrada de acordo com os estados selecionados. Na Figura 22 é possível ver a aba de caracteres da chave **Teste/Demo - Caminalculos 0.5**.

Ao acessar um agrupamento o usuário pode visualizar todos os caracteres ou agrupamentos que estão contidos dentro do agrupamento acessado. Na Figura 23 é possível ver o agrupamento **Arms** com a sua lista de caracteres.

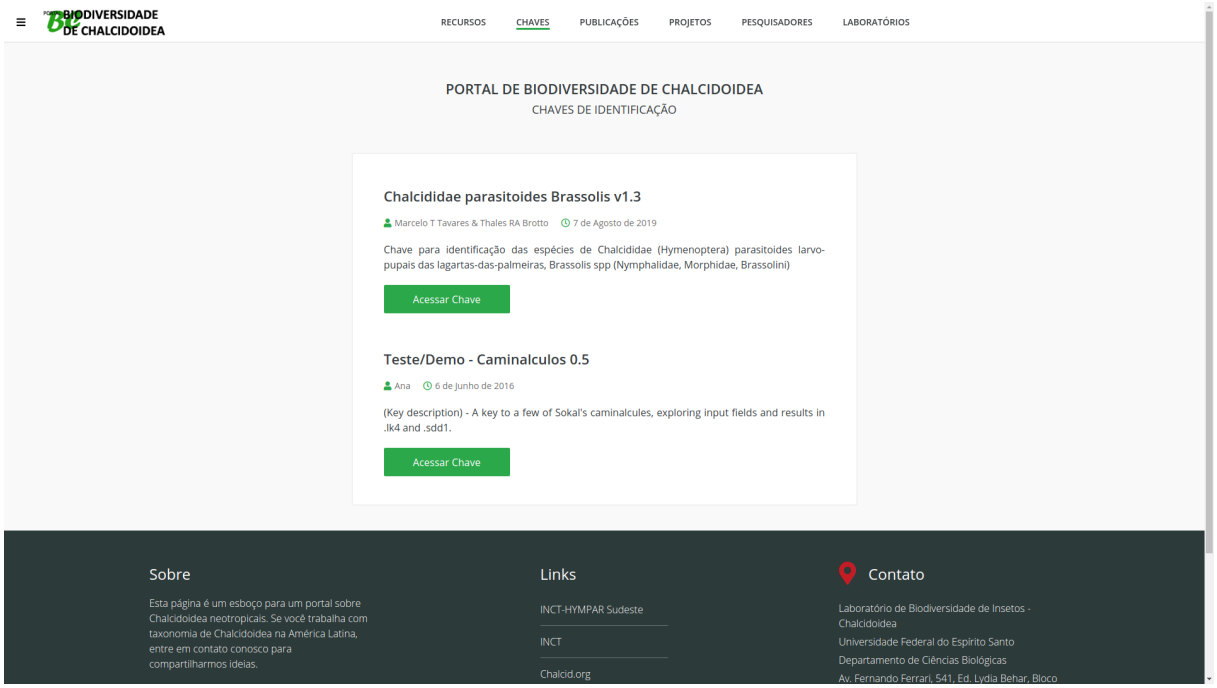

Figura 21 – Seção de Chaves de Identificação.

Figura 22 – Página de uma chave de identificação.

Acessando um caractere o usuário é levado para a sua página (Figura 24), onde pode ver a lista de estados na qual o caractere pode se encontrar. Para cada estado são exibidos o seu nome, uma imagem, e um modal com informações adicionais sobre o mesmo. Também existe um botão ao lado do nome do caractere onde o usuário pode adicionar ou remover o estado da lista de estados selecionados.

A Figura 25 exibe a lista de estados selecionados, que pode sempre ser acessada

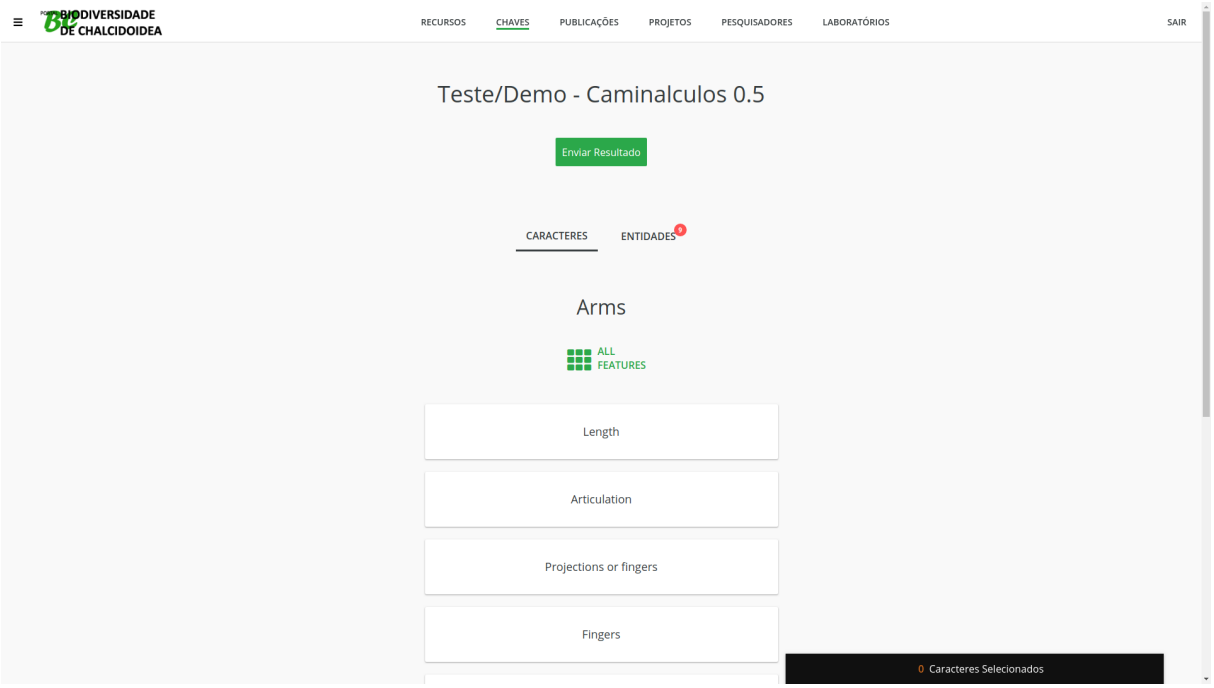

Figura 23 – Agrupamento de uma chave de identificação.

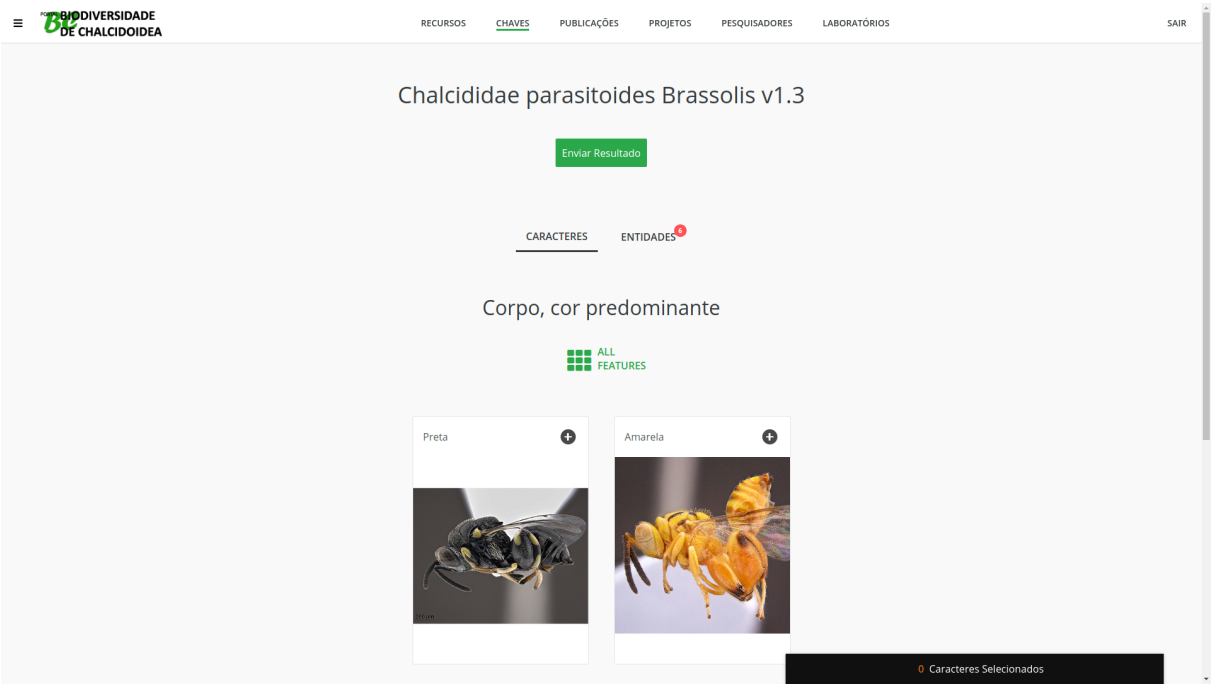

Figura 24 – Estados de um caractere.

no canto inferior direito da página de uma chave de identificação. Nela o usuário pode visualizar todos os estados que selecionou, também é possível remover cada estado da lista individualmente ou remover todos de uma vez através do botão **Remover tudo**.

A aba de entidades (Figura 26) exibe a lista de entidades da chave de identificação, filtradas de acordo com os estados selecionados pelo usuário. Caso nenhum estado tenha sido selecionado é exibido a lista completa de entidades existentes. Para cada entidade

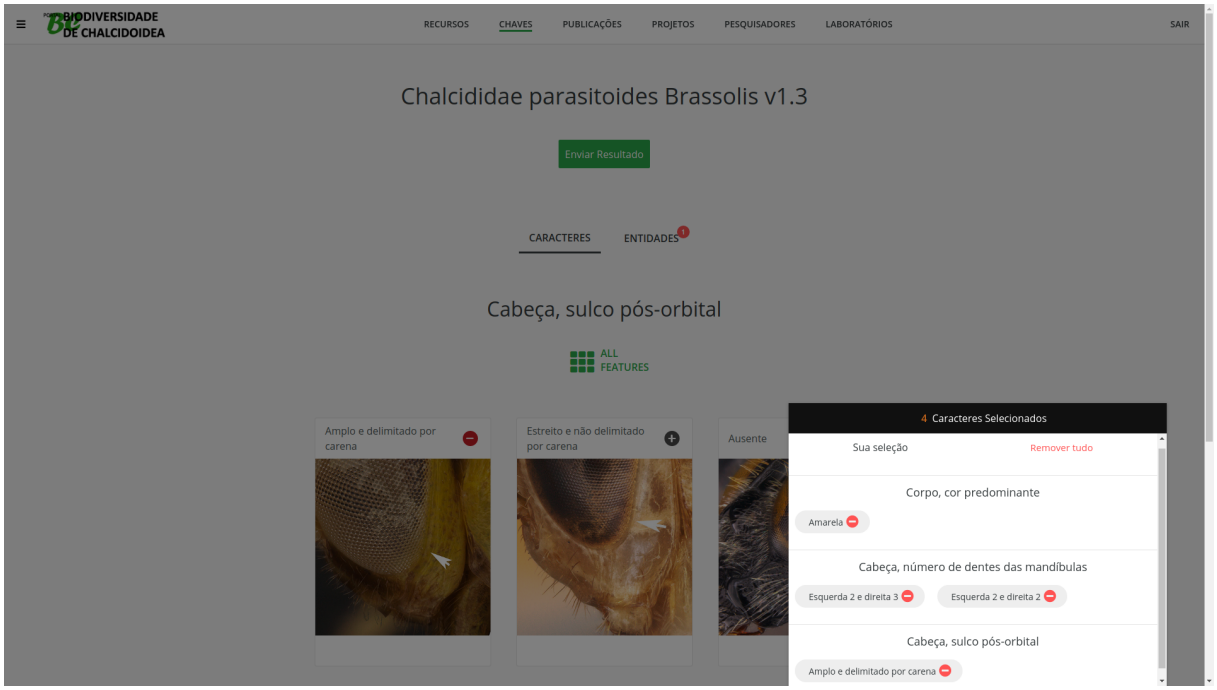

Figura 25 – Lista de estados selecionados.

são exibidos o seu nome, uma imagem, e um modal (Figura 27) onde é possível ler um texto com informações sobre a entidade e visualizar a lista de estados no qual a entidade se encontra para cada caractere.

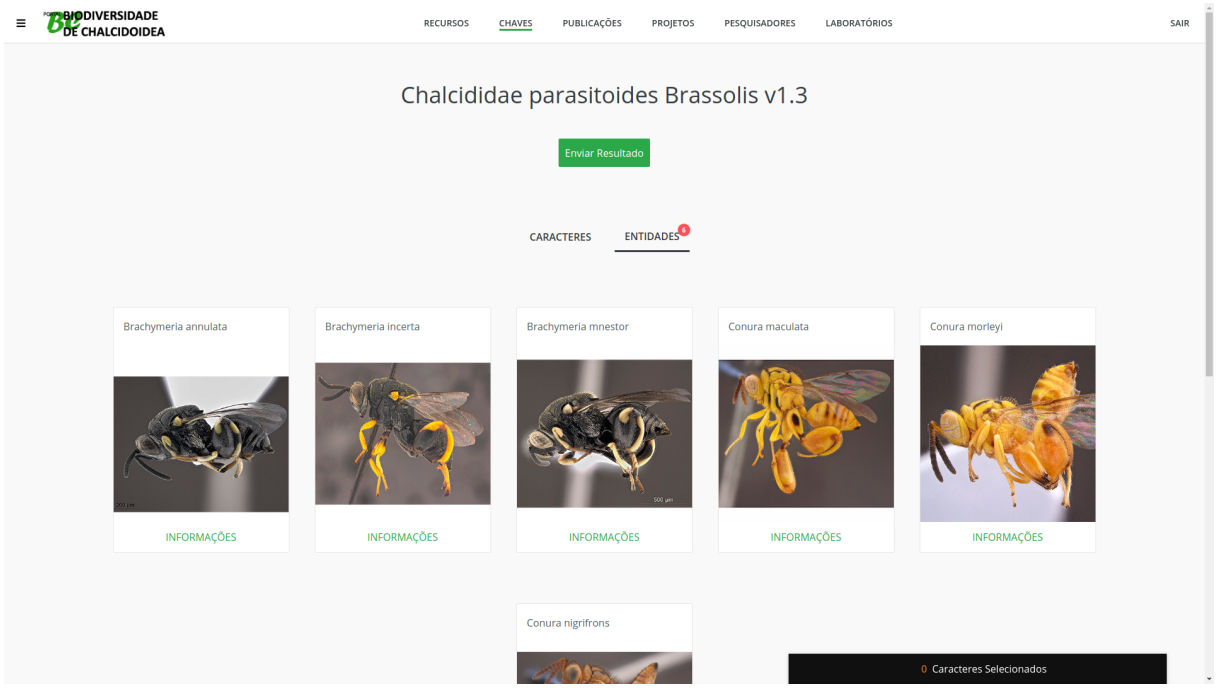

Figura 26 – Aba de entidades da chave de identificação.

Por fim, clicando no botão **Enviar Resultado**, que se encontra abaixo do nome da chave de identificação, é exibido um modal (Figura 28) onde o usuário pode enviar para o sistema o resultado obtido na chave de identificação, contendo a lista de estados que

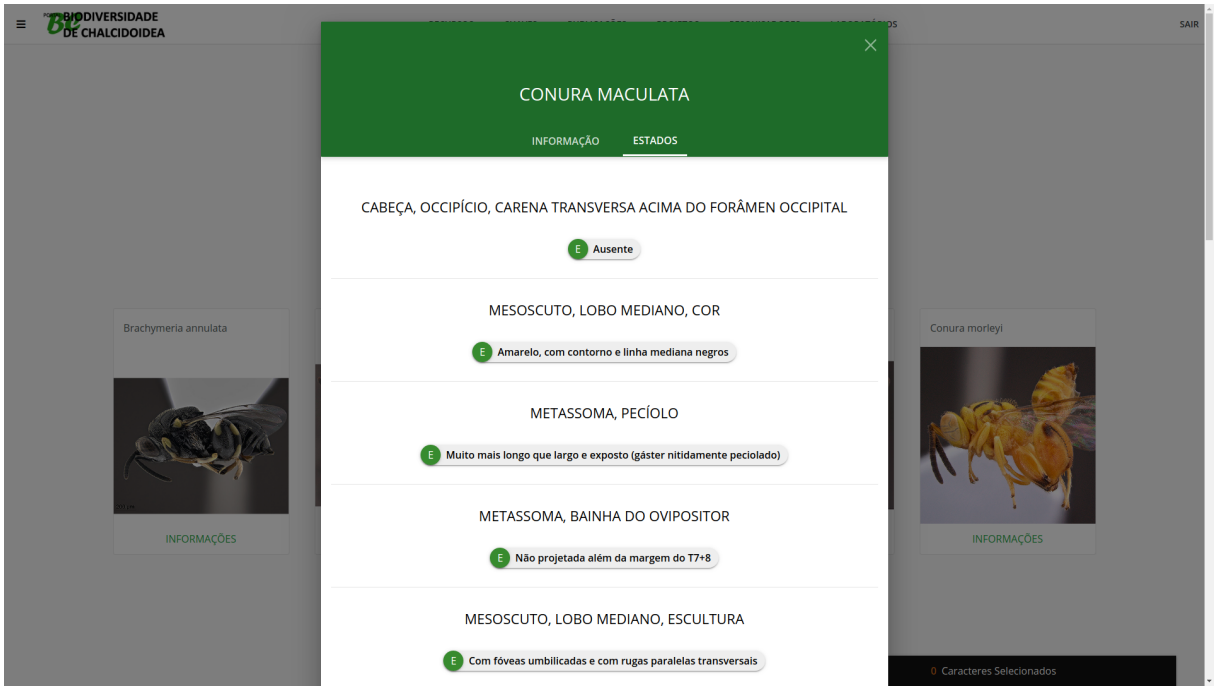

Figura 27 – Modal com detalhes sobre a entidade.

selecionou, a lista de entidades resultantes e, opcionalmente, um comentário. Feito isso o resultado ficará disponível na *dashboard* para que os administradores possam avaliá-lo.

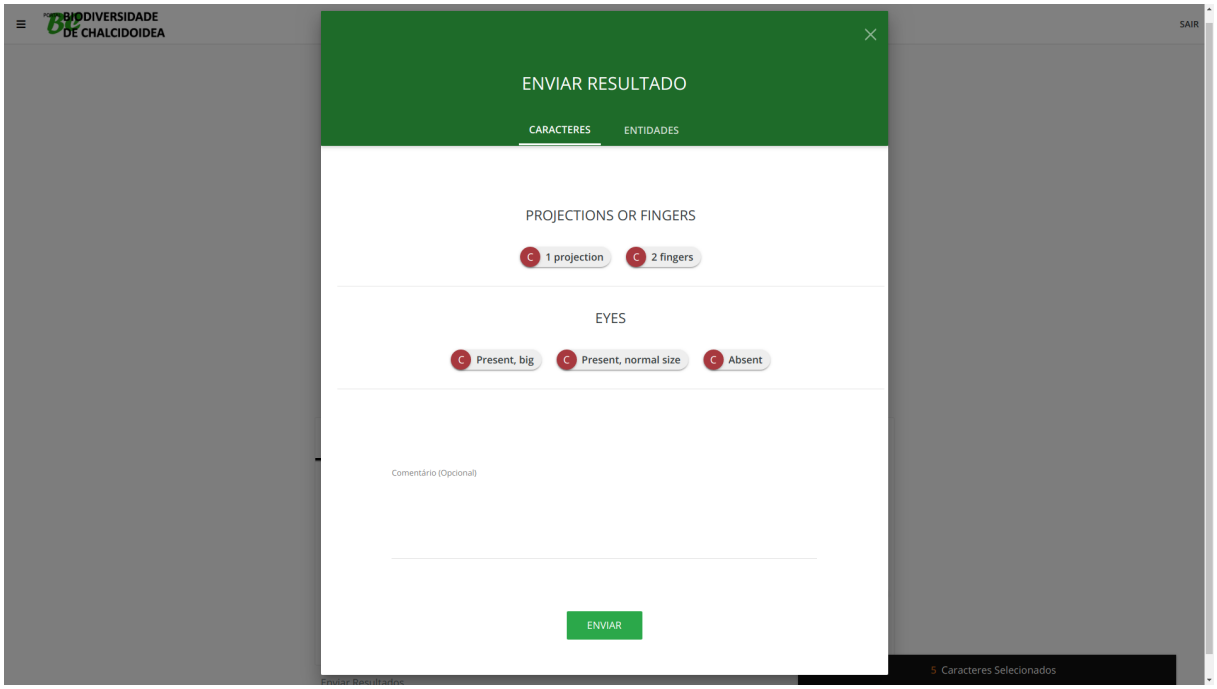

Figura 28 – Enviar resultado da chave de identificação.

5.2.3 Dashboard

Para ter acesso a *dashboard* o usuário deve fazer login no sistema, para isso deve informar o seu nome de usuário e a sua senha, conforme a Figura 29.

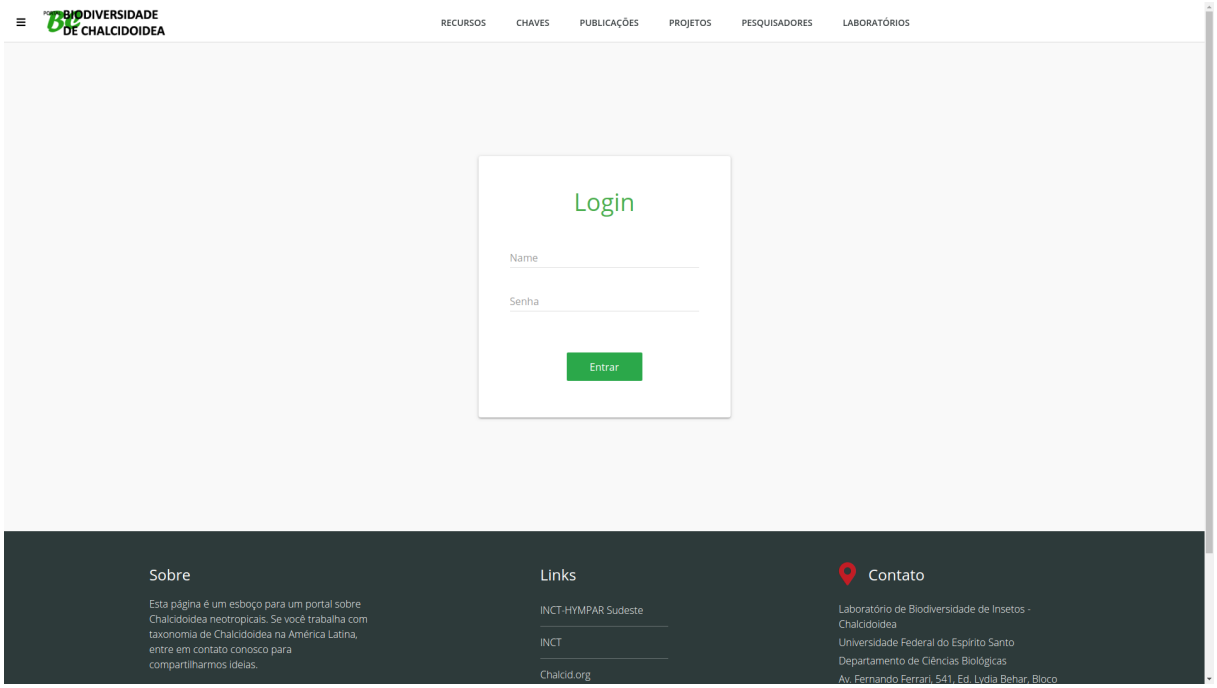

Figura 29 – Tela de login.

Após o login, o administrador terá acesso à página inicial da *dashboard* (Figura 30), onde é exibida a lista de todos os recursos existentes no portal e as suas respectivas quantidades. A partir dessa tela o usuário escolhe qual recurso deseja gerenciar.

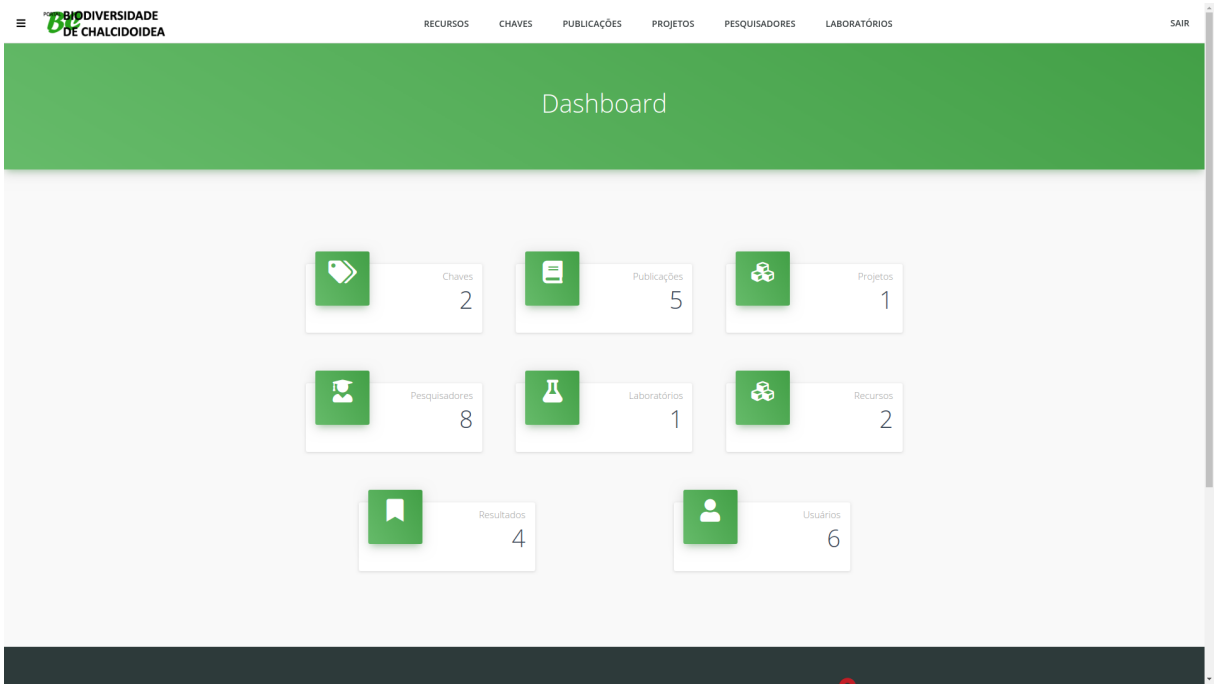

Figura 30 – Página inicial da Dashboard.

As páginas de gerenciamento de cada recurso possuem telas que seguem exatamente o mesmo padrão e fluxo. Para evitar repetição, serão mostradas apenas as telas do gerenciamento de publicações. A Figura 31 exibe a tela que lista todas as publicações

cadastradas no sistema. A partir dela é possível editar uma publicação (clicando no lápis), excluir uma publicação (clicando na lata de lixo vermelha), ou adicionar uma publicação (clicando no símbolo de soma — "+").

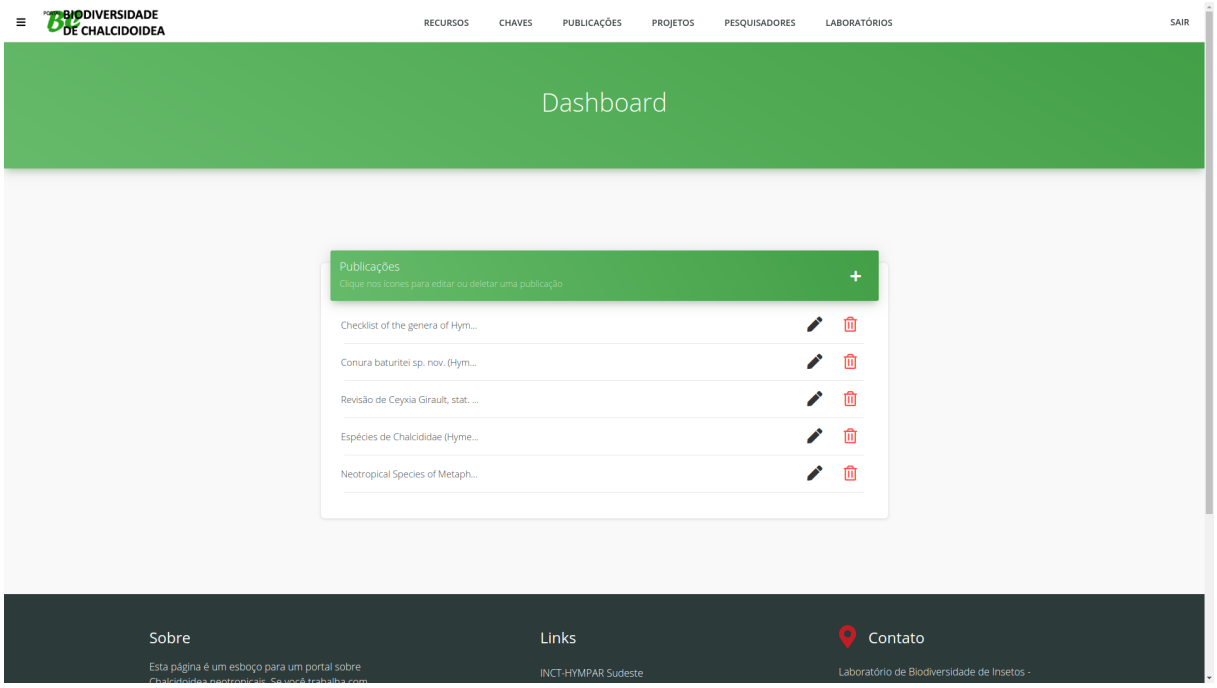

Figura 31 – Gerenciar Publicações.

Ao selecionar a opção de adicionar uma publicação, é exibido o formulário com os campos que devem ser preenchidos pelo administrador, conforme a Figura 32. Da mesma forma, ao escolher editar uma publicação, o administrador deve editar um formulário similar, modificando os campos que deseja alterar.

Os formulários para adicionar uma chave de identificação ou editar uma chave existente possuem campos especiais que permitem o *upload* de arquivos. Na Figura 33 é possível ver os campos para adicionar o arquivo SDD, as imagens e os arquivos HTML de uma chave de identificação. O usuário tem a opção de arrastar os arquivos até a área azul de *upload* onde eles serão identificados automaticamente.

Por fim, o administrador tem acesso a lista de resultados enviados por usuários (Figura 34). Nessa lista, resultados ainda não visualizados pelo administrador ficam destacados com um ponto vermelho. Para cada resultado é exibido o nome da chave de identificação à qual ele pertence. O administrador tem a opção de excluir o resultado enviado clicando na lata de lixo vermelha. Para visualizar o resultado basta clicar no nome da chave e um modal (Figura 35) será exibido. Nele, o administrador pode ver a lista de estados selecionados pelo usuário, a lista de entidades resultantes, e caso exista, o comentário feito pelo usuário.

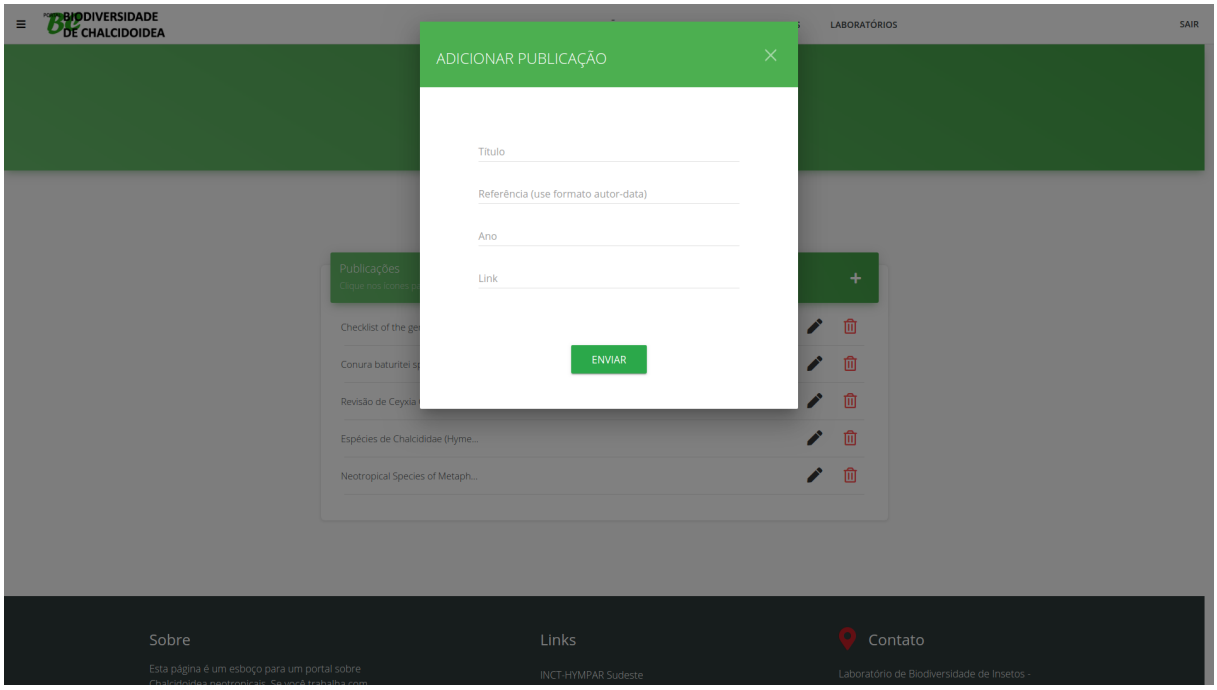

Figura 32 – Adicionar Publicação.

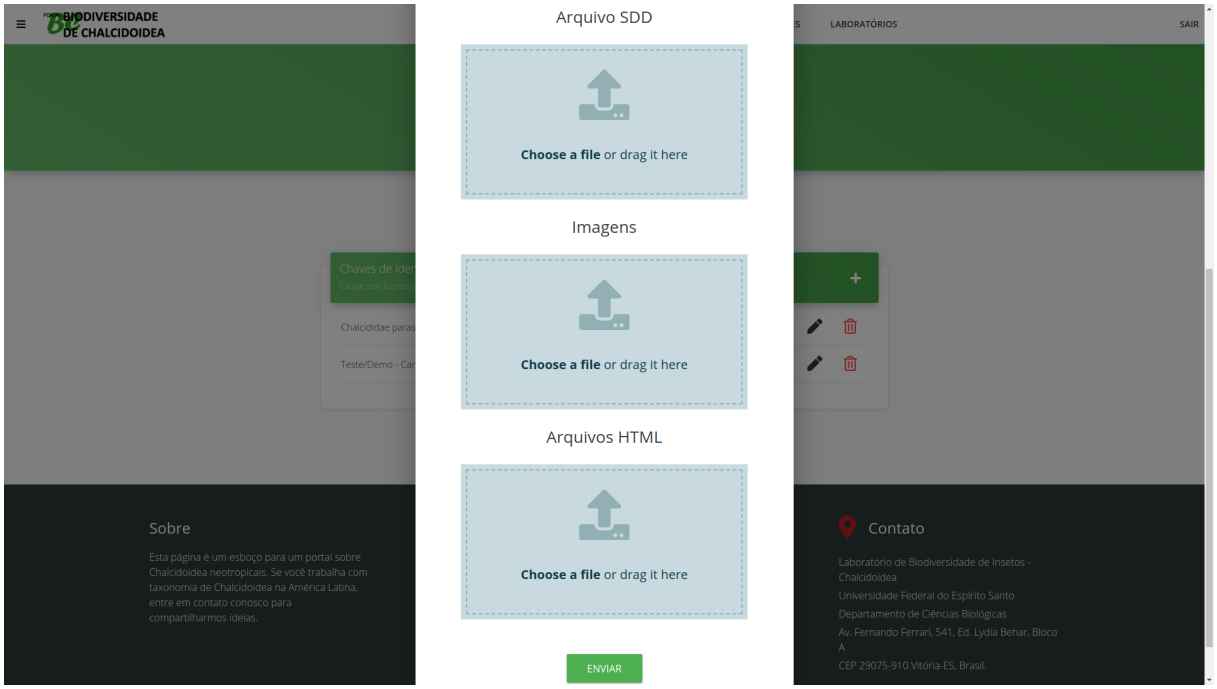

Figura 33 – Upload de arquivos.

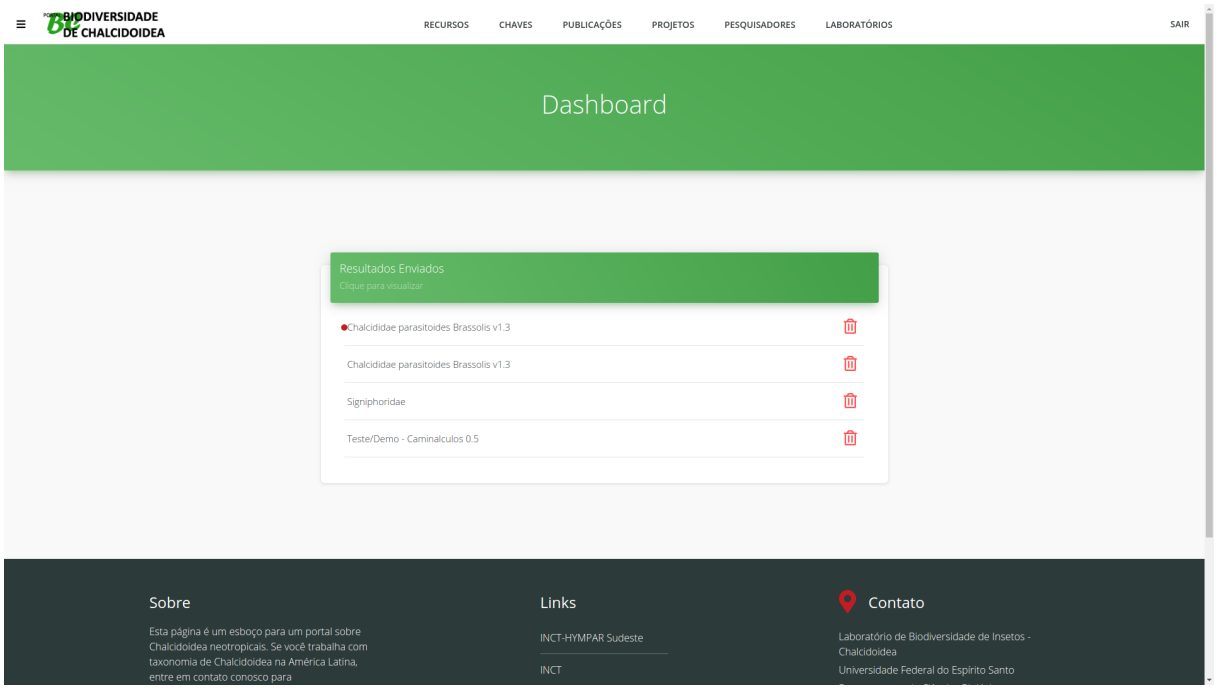

Figura 34 – Lista de resultados enviados.

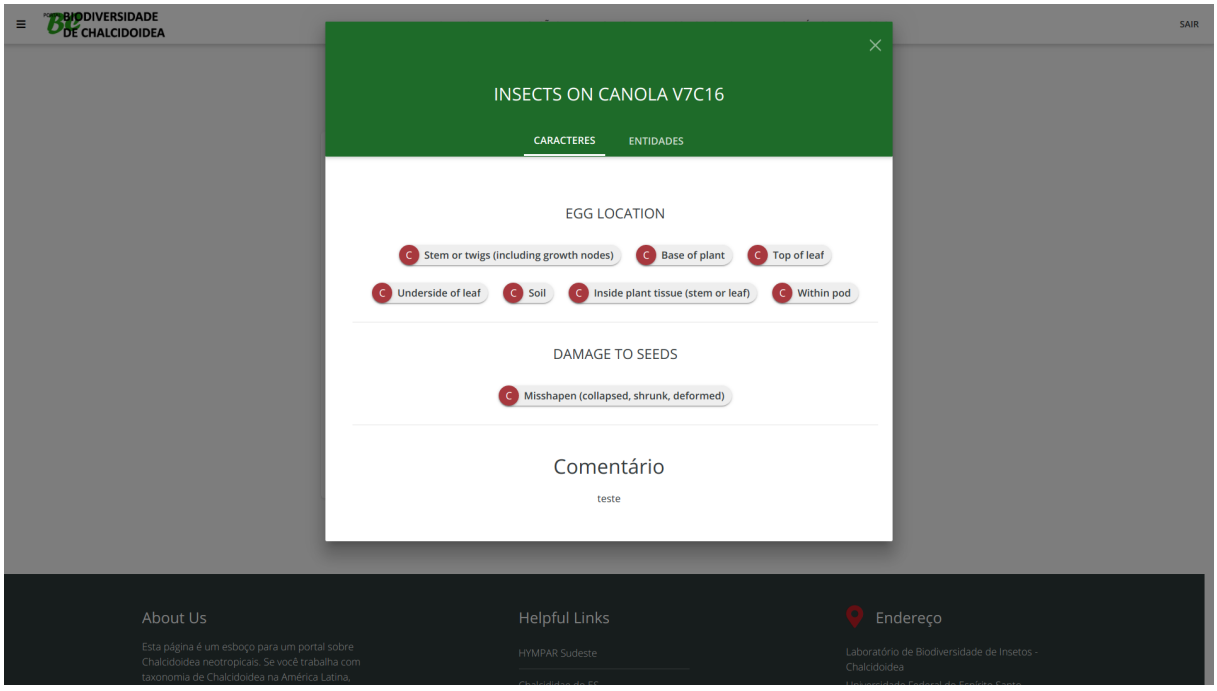

Figura 35 – Modal com detalhes do resultado.

6 Considerações Finais

Este capítulo apresenta as conclusões do trabalho realizado, mostrando suas contribuições. Por fim, são apresentadas suas limitações e perspectivas de trabalhos futuros.

6.1 Conclusões

Sabendo da importância do desenvolvimento de ferramentas de identificação taxonômica para a pesquisa científica em Biologia, surgiu a necessidade da criação de um sistema que oferecesse uma forma simples para taxônomos compartilharem essas ferramentas. As soluções existentes atualmente, em geral, são programas que devem ser instalados no computador do usuário, possuem código fechado e em muitos casos são pagos. Programas como o Lucid só permitem a exportação de chaves no formato Java, o que dificulta seu uso, por exemplo, em celulares. Além disso, a conversão de chaves desenvolvidas em Lucid para o formato Web e como aplicativo de celular envolve alto custo financeiro. Em contrapartida o Portal de Biodiversidade de Chalcidoidea, oferece uma página Web com interface amigável e responsiva, podendo ser acessada de qualquer dispositivo com acesso à Internet. O portal permite o envio de chaves de identificação por meio de arquivos SDD, um formato de arquivo que, além de possuir código aberto, é independente de qualquer ferramenta e pode ser utilizado de forma gratuita. Por fim, embora o portal tenha sido concebido com foco na área de taxonomia de Chalcidoidea, o seu código fonte pode ser utilizado para exibir chaves de identificação de qualquer área da Biologia.

Todos os objetivos propostos no Capítulo 1 foram alcançados. Começando pelo levantamento e análise de requisitos, onde foram produzidos o **Documento de Requisitos**, documento que apresenta uma descrição do sistema, de seu minimundo, definição dos requisitos funcionais e não funcionais, além das regras de negócio, e o **Documento de Especificação de Requisitos**, que apresenta a identificação dos subsistemas, modelos de casos de uso, modelo estrutural, modelo dinâmico e glossário do projeto. Com a etapa de análise finalizada, foi elaborado o Documento de Projeto de Sistema, onde foi definida a plataforma de desenvolvimento do sistema, seus atributos de qualidade e táticas, a arquitetura do software e projeto detalhado de cada um dos seus componentes, seguindo a abordagem FrameWeb, descrita na Seção 2.5. Para esses objetivos, toda a documentação proposta foi produzida de acordo com as boas práticas propostas pela Engenharia de Software.

Durante a etapa de Projeto de Sistema, foi utilizado o método FrameWeb para a elaboração dos modelos necessários para esta fase. Porém, como o *framework* Express.js segue algumas convenções diferentes das esperadas pelo método, foi necessário realizar

algumas pequenas adaptações. O Modelo de Entidades foi adaptado para o MongoDB, um banco de dados NoSQL que utiliza um modelo de dados orientado a documentos. Já no modelo de navegação, embora não foram feitas adaptações ao método, foi necessário abstrair alguns conceitos do *framework* utilizado, como o tipo da requisição HTTP de cada método, uma vez que o modelo não oferece tais informações. No entanto, estas adaptações se mostraram simples de serem realizadas e os modelos, mesmo com abstrações, foram capazes de prover as informações esperadas. Dessa forma, foi possível observar a flexibilidade do método com os diversos *frameworks* existentes atualmente.

Ao longo de todo o processo de desenvolvimento do sistema, indo desde a fase de documentação até a implementação do mesmo, diversos desafios foram encontrados e tiveram que ser superados. Sendo que, o domínio da linguagem de programação JavaScript, em conjunto com o Node.js e o estudo do *framework* Express.js, provaram ser o maior desafio. O mesmo foi superado por meio da leitura de livros e das documentações oficiais dessas tecnologias disponibilizadas na Internet. Além disso, devido ao fato, do Express.js ser um *framework* minimalista, muitas funcionalidades necessárias para o desenvolvimento do sistema não existem nativamente nele. Dessa forma. foi preciso escolher separadamente cada um dos *middlewares* necessários para a implementação das funcionalidades do sistema, sendo que a escolha dos *middlewares* relacionados à autenticação de usuários e segurança do sistema exigiram uma atenção e análise bem cuidadosa. Por fim, a utilização do método FrameWeb também se provou um desafio no início, uma vez que o mesmo apresentou modelos e conceitos diferente dos aprendidos durante a graduação. Mas com alguma leitura e prática, foi possível elaborar os modelos corretamente e em tempo hábil.

É fundamental destacar também o grande desafio que foi integrar as diferentes disciplinas realizadas durante o curso de Ciência da Computação. Na etapa de análise e levantamento de requisitos foram utilizados conceitos das disciplinas de Engenharia de Software e, em particular, Engenharia de Requisitos. A etapa de projeto de sistema relaciona-se com as disciplinas de Projeto de Sistemas e Desenvolvimento Web e Web Semântica (optativa). Por fim, durante a implementação do sistema exercitou-se conceitos das disciplinas de Programação, Linguagens de Programação, Banco de Dados, Desenvolvimento Web e Interface Humano-Computador.

A experiência adquirida com o desenvolvimento deste trabalho foi muito proveitosa, pois foi possível colocar em prática conceitos aprendidos em sala de aula, em conjunto com o aprendizado de novas tecnologias e ferramentas, possibilitando assim o desenvolvimento de uma ferramenta de grande utilidade.

6.2 Limitações e Perspectivas Futuras

O desenvolvimento de software é considerado um processo que nunca termina, pois sempre podem ocorrer mudanças ao longo de todo o ciclo de vida do software. Novas funcionalidades podem ser adicionadas, enquanto outras podem ser removidas, o código fonte pode ser substituído por outro mais eficiente ou moderno, correções de possíveis falhas que venham a ser encontradas podem ser feitas. Dessa forma, a partir dos resultados obtidos, foi possível verificar que existe a possibilidade da adição de novas funcionalidades ao sistema, além de uma modularização maior do mesmo. O que abre as portas para a realização de trabalhos futuros. Essas limitações são apresentadas abaixo.

- Adicionar uma ferramenta para produzir etiquetas de espécimes, onde o usuário informa os dados sobre o espécime coletado, e as etiquetas são geradas já formatadas, prontas para impressão;
- Definir novos atores para o portal além do visitante e do administrador. Esse atores teriam permissões limitadas de gerenciamento de dados, podendo adicionar conteúdo ao portal como chaves de identificação e publicações, mas não podendo excluir dados, ou gerenciar contas de outros administradores;
- Utilizar um *framework* JavaScript específico para a implementação da interface do sistema, permitindo assim o desenvolvimento de uma interface mais complexa e uma separação maior das camadas do sistema;
- Adicionar formas de representar o tipo da requisição HTTP de cada método no modelo de navegação do FrameWeb.

Referências

CASTELEYN, S. et al. *Engineering Web Applications*. 1st. ed. [S.l.]: Springer Publishing Company, Incorporated, 2009. ISBN 3540922008, 9783540922001. Citado 2 vezes nas páginas 22 e 23.

DALLWITZ, M.; PAINE, T.; ZURCHER, E. *Principles of interactive keys*. 2018. Disponível em: <https://www.delta-intkey.com/www/interactivekeys.htm>. Citado na página 18.

FALBO, R. d. A. *Engenharia de Software*. [s.n.], 2014. 144 p. Disponível em: <https://inf.ufes.br/~falbo/files/ES/Notas_Aula_Engenharia_Software.pdf>. Citado 2 vezes nas páginas 14 e 18.

FALBO, R. d. A. *Projeto de Sistemas*. [s.n.], 2016. 138 p. Disponível em: <https://inf.ufes.br/~falbo/files/PSS/Notas_Aula_Projeto_Sistemas_2016.pdf>. Citado 4 vezes nas páginas 14, 20, 21 e 38.

FALBO, R. d. A. *Engenharia de Requisitos*. [s.n.], 2017. 178 p. Disponível em: <https://inf.ufes.br/~falbo/files/ER/Notas_Aula_Engenharia_Requisitos.pdf>. Citado 3 vezes nas páginas 14, 19 e 20.

FIELDING, R. et al. *RFC 2616, Hypertext Transfer Protocol – HTTP/1.1*. 1999. Disponível em: <http://www.rfc.net/rfc2616.html>. Citado na página 22.

FOWLER, M. *Patterns of Enterprise Application Architecture*. [S.l.: s.n.], 2002. Citado 3 vezes nas páginas 14, 23 e 26.

GIBSON, G. et al. *Annotated Keys to the Genera of Nearctic Chalcidoidea (Hymenoptera)*. NRC Research Press, 1997. (Monograph Publishing Program). ISBN 9780660166698. Disponível em: <https://books.google.com.br/books?id=50tXxazrCvoC>. Citado na página 16.

GINIGE, A.; MURUGESAN, S. Web engineering-: An introduction. *Multimedia, IEEE*, v. 8, 2001. Citado na página 21.

HERATY, J. M. et al. A phylogenetic analysis of the megadiverse chalcidoidea (hymenoptera). *Cladistics*, v. 29, n. 5, p. 466–542, 2013. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1111/cla.12006>. Citado 3 vezes nas páginas 6, 16 e 17.

MAI, L. F. F. *Khoeus: uma plataforma de aprendizado focada no ensino da programação*. Vitória, ES, Brasil, 2017. Citado na página 21.

MARTINS, B. F.; SOUZA, V. E. S. A Model-Driven Approach for the Design of Web Information Systems based on Frameworks. In: *Proc. of the 21st Brazilian Symposium on Multimedia and the Web*. Manaus, AM, Brazil: [s.n.], 2015. p. 41–48. Disponível em: <http://dl.acm.org/citation.cfm?id=2820439>. Citado na página 26.

MOZILLA, W. D. *Express/Node introduction*. 2019. Disponível em: <https:

//developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction>. Citado na página 25.

NODEJS.ORG. *About Node.js*. 2019. Disponível em: <https://nodejs.org/en/about/>. Citado na página 24.

POLYA, G. *How to Solve It*. [S.l.]: Princeton University Press, 1945. Citado na página 18.

PRESSMAN, R. S. *Engenharia de Software - Uma abordagem profissional*. 7. ed. [S.l.]: McGrawHill, 2011. Citado 7 vezes nas páginas 6, 18, 19, 20, 21, 23 e 24.

SOMMERVILLE, I. *Engenharia de Software*. 9. ed. [S.l.]: Pearson Prentice Hall, 2011. Citado 2 vezes nas páginas 19 e 23.

SOUZA, V. E. S. *FrameWeb: um Método baseado em Frameworks para o Projeto de Sistemas de Informação Web*. [S.l.], 2007. Disponível em: <http://portais.ufes.br/PRPPG/ext/mono.php?progpess=2032&curso=9&prog=30001013007P0>. Citado 7 vezes nas páginas 6, 13, 14, 25, 26, 27 e 45.

WIKIPEDIA. *Identification key*. 2018. Disponível em: <"https://en.wikipedia.org/w/index.php?title=Identification_key&oldid=826023226">. Citado na página 17.

WINSTON, J. *Describing Species: Practical Taxonomic Procedure for Biologists*. [S.l.]: Columbia University Press, 1999. Citado na página 17.

Apêndices

Documento de Requisitos

Projeto: Portal de Biodiversidade de Chalcidoidea

Registro de Alterações:

Versão	Responsáveis	Data	Alterações
1.0	Gustavo Epichin Monjardim	02/09/2018	Versão inicial da documentação.
1.1	Vítor E. Silva Souza	28/09/2018	Primeira revisão.
1.2	Gustavo Epichin Monjardim	01/10/2018	Acertos da primeira revisão.
1.3	Vítor E. Silva Souza	17/10/2018	Segunda revisão
1.4	Gustavo Epichin Monjardim	20/10/2018	Acertos da segunda revisão.
1.5	Vítor E. Silva Souza	07/11/2018	Versão final.

1. Introdução

Este documento apresenta os requisitos de usuário da ferramenta Portal de Biodiversidade de Chalcidoidea e está organizado da seguinte forma: a seção 2 contém uma descrição do propósito do sistema; a seção 3 apresenta uma descrição do minimundo apresentando o problema; e a seção 4 apresenta a lista de requisitos de usuário levantados junto ao cliente.

2. Descrição do Propósito do Sistema

O Portal de Biodiversidade de Chalcidoidea é uma aplicação web desenvolvida para o Laboratório de Biodiversidade de Insetos da Universidade Federal do Espírito Santo (LaBI/UFES). O portal tem como objetivo promover a divulgação científica para aumentar a visibilidade da pesquisa em Chalcidoidea, permitindo que pesquisadores divulguem informações sobre publicações científicas, projetos, laboratórios, pesquisadores, e recursos relacionados a área. Além disso, a aplicação permite que pesquisadores façam *upload* de ferramentas de identificação taxonômica desenvolvidas por taxônomos usando plataformas como Lucid e Delta, através de arquivos em formato matricial ou XML. O arquivo será então interpretado e a chave será disponibilizada no portal onde poderá ser utilizada pelo público em computador ou celular sem a necessidade do software originalmente utilizado para o preparo da chave.

3. Descrição do Minimundo

O Portal de Biodiversidade de Chalcidoidea servirá como recurso para profissionais da entomologia básica e aplicada, fornecendo informações sobre biodiversidade e taxonomia de Chalcidoidea. Chacidoidea é uma das várias superfamílias de vespas conhecidas como "parasitoides", ou seja, cujo ciclo de vida inclui uma fase larval que se desenvolve dentro de ou sobre outro organismo, normalmente outro inseto, que é consumido para a emergência da vespa adulta. Ao contrário das vespas mais popularmente conhecidas, elas não constroem ninhos nem possuem ferrão. São extremamente diversas morfologicamente, a maioria com 0,5 a 10mm de comprimento, sendo que este grupo inclui os menores insetos conhecidos (0,15mm). Estima-se que existam cerca de 500.000 espécies, a maioria delas com importância ecológica ou econômica justamente devido ao seu ciclo de vida, que as caracteriza como inimigos naturais de outros organismos que podem ser pragas agrícolas ou florestais. No portal será possível encontrar ferramentas para identificá-los. Essa ferramentas são conhecidas como chaves de identificação taxonômicas. O desenvolvimento dessas chaves é uma maneira do especialista no grupo de organismos possibilitar ao não especialista que realize ao menos parte do trabalho de identificação. Tais ferramentas frequentemente são apresentadas no formato dicotômico e desenvolvidas manualmente. No entanto, o formato dicotômico não é ideal para o usuário leigo, mesmo que seja ricamente ilustrada, uma vez que, se um passo da chave se refere a características difíceis ou impossíveis de observar (por exemplo, uma característica de machos quando o usuário só possui fêmeas) a continuação do uso da ferramenta sem risco de identificação errônea fica impossibilitada. Esse problema não é encontrado nas chaves multi-entrada, que são baseadas em matrizes, uma vez que o usuário pode responder às questões em qualquer ordem a fim de refinar cada vez mais o conjunto de resultados possíveis.

Para o desenvolvimento de chaves multi-entrada, uma série de softwares já se encontram disponíveis. Um dos softwares mais utilizados para este fim é o Lucid (www.lucidcentral.com), outra ferramenta que também é utilizada desde os anos 80 é a plataforma Delta (www.delta-intkey.com). A dificuldade atualmente encontrada com Lucid é que este é um programa de código fechado, que só permite a exportação de chaves no formato Java, o que dificulta seu uso, por exemplo, em celulares. Além disso, a conversão de chaves desenvolvidas em Lucid para o formato web e como aplicativo de celular envolve alto custo financeiro. No entanto, com o avanço da pressão para que se produzam ferramentas científicas em formatos abertos, as chaves podem ser exportadas em um formato XML simples, SDD (*Structured Descriptive Data*). As convenções associadas ao formato SDD e outros baseados em XML são mantidas por uma equipe internacional envolvendo profissionais da biologia e da informática, a *Taxonomic Database Working Group* (TDWG). No entanto, não há uma ferramenta disponível que faça a conversão automática dos formatos exportados para uma página de Internet de fácil navegação que possa ser usada por um usuário leigo.

Desta maneira, propõe-se desenvolver uma ferramenta web, que permita ao taxônomo fazer *upload* do arquivo contendo a chave em formato matricial ou XML e os arquivos de imagens e textos associados, sem maiores necessidades de configurações. A chave disponibilizada dessa maneira poderá então ser utilizada pelo público em computador ou celular sem a necessidade do software originalmente utilizado para o preparo da chave.

Complementar à disponibilização das chaves de identificação, o portal também possuirá uma série de recursos sobre a Chalcidoidea, sendo eles: publicações, projetos,

pesquisadores, laboratórios e postagens de um blog com fins educacionais. Cada um desses recursos é uma seção dentro do portal, em conjunto com a seção de chaves de identificação e a *dashboard*, onde os administradores gerenciam as informações. Todas as seções são detalhadas a seguir.

3.1. Publicações

Nesta seção é possível visualizar todas as publicações científicas disponíveis, organizadas pelo ano da publicação, da mais recente para a menos recente. Um publicação contém um título, o veículo onde foi publicada, o ano em que foi publicada, o(s) autor(es) e um link para o pdf.

3.2. Projetos

Na seção de projetos é possível ver todos os projetos adicionados, organizados por data, os mais recentes aparecendo no topo. Nessa página é possível ver a data do projeto, o seu título e uma pequena descrição sobre o mesmo. Ao clicar em um projeto a página desse projeto é exibida, nela, além das informações anteriores, também é exibido o código de cadastro do projeto, a lista de pesquisadores participantes, e o conteúdo completo do projeto.

3.3. Pesquisadores

Na seção de pesquisadores é possível ver a lista de todos os pesquisadores adicionados. Um pesquisador possui um nome, área de interesse, e-mail para contato e link para o seu currículo Lattes.

3.4. Laboratórios

Na seção de laboratórios é possível ver a lista de todos os laboratórios adicionados. Um laboratório possui um nome, cidade e estado.

3.5. Postagens

Na seção de postagens encontram-se recursos educacionais relacionados a Chalcidoidea. A lista de postagens é organizada por data de publicação, exibindo as postagens mais recentes no topo. Ao clicar em uma postagem a sua página é acessada. Uma postagem possui um título, uma data e um conteúdo em texto.

3.6. Chaves de Identificação

Nessa seção são exibidas todas as chaves de identificação disponíveis. É possível ver o nome da chave, a data de publicação, o autor da chave e uma pequena descrição sobre a mesma. Ao clicar em uma chave é possível interagir com a mesma, a interação é explicada a seguir.

3.6.1. Interagindo com uma chave de identificação

A página de uma chave de identificação possui uma seção onde é exibida a lista de caracteres (características) disponíveis e outra seção onde são exibidas as entidades, o usuário pode alternar entre as duas seções a qualquer momento.

3.6.1.1. Caracteres

Na seção de caracteres o usuário pode acessar um caractere e ver os estados referentes ao mesmo, um estado possui um nome, e opcionalmente uma imagem e uma pequena descrição sobre ele. O usuário pode selecionar ou desmarcar quantos estados quiser. Ao selecionar um estado a lista de entidades é filtrada de acordo com o estado selecionado. Caracteres também podem estar agrupados em um agrupamento com um nome específico, agrupamentos também podem estar agrupados em um agrupamento.

3.6.1.2. Entidades

Na seção de entidades é exibida a lista de entidades, essa lista é filtrada de acordo com os estados selecionados na aba de caracteres, caso nenhum estado tenha sido selecionado toda as entidades são exibidas. Ao clicar em uma entidade é exibido um texto descritivo sobre a mesma, a sua lista de estados e uma galeria de imagens.

3.6.1.3. Opções Adicionais

Além disso, existe uma área. mostrando todos os estados que o usuário selecionou, nela é possível remover cada um desses estados individualmente, ou remover todos de uma vez.
Por fim existe a opção de enviar o resultado obtido na chave. Ao fazer isso o usuário envia a lista de estados que selecionou e quais entidades foram obtidas como resultado, também há a opção de adicionar um comentário referente ao resultado.

3.7. Dashboard

Na *dashboard* é onde ocorre o gerenciamento do conteúdo do portal. Para acessá-la é necessário ser autenticado primeiro, para isso uma tela de login é exibida onde deve ser fornecido o *username* e a senha. Caso estejam corretos a *dashboard* é exibida. Na *dashboard* o administrador pode escolher entre as seguintes opções: gerenciar publicações, gerenciar projetos, gerenciar pesquisadores, gerenciar laboratórios, gerenciar postagens, gerenciar chaves, gerenciar administradores, lista de resultados. Cada uma dessas opções é apresentada a seguir.

3.7.1. Gerenciar publicações

Ao clicar nessa opção é exibida a lista de todas as publicações disponíveis, o administrador tem a opção de editar ou excluir uma publicação. Também é possível adicionar uma nova publicação.

3.7.2. Gerenciar projetos

Ao clicar nessa opção é exibida a lista de todos os projetos disponíveis, o administrador tem a opção de editar ou deletar um projeto. Também é possível adicionar

um novo projeto. Ao editar ou adicionar um projeto, o administrador tem a opção de fornecer um texto normal ou um texto com tags HTML para o campo “conteúdo do projeto”.

3.7.3. Gerenciar pesquisadores

Ao clicar nessa opção é exibida a lista de todos os pesquisadores disponíveis, o administrador tem a opção de editar ou excluir um pesquisador. Também é possível adicionar um novo pesquisador.

3.7.4. Gerenciar Laboratórios

Ao clicar nessa opção é exibida a lista de todos os laboratórios disponíveis, o administrador tem a opção de editar ou excluir um laboratório. Também é possível adicionar um novo laboratório.

3.7.5. Gerenciar Postagens

Ao clicar nessa opção é exibida a lista de todas as postagens disponíveis, o administrador tem a opção de editar ou deletar uma postagem . Também é possível adicionar uma nova postagem. Assim como em um projeto, o administrador também pode fornecer texto em HTML para o campo “conteúdo da postagem”.

3.7.6. Gerenciar Chaves de identificação

Ao clicar nessa opção é exibida a lista de todas as chaves de identificação disponíveis, o administrador tem a opção de editar ou excluir uma chave. Também é possível adicionar uma nova chave. Ao adicionar uma chave o administrador deve fazer *upload* de um arquivo SDD, esse arquivo será lido e interpretado e a partir dele serão identificados caracteres e entidades. A chave de identificação será então exibida na página de chaves e o usuário poderá interagir com a mesma. Também existe a opção de fazer *upload* de imagens e arquivos HTML na hora de adicionar ou editar chaves, estes arquivos serão associados com caracteres e entidades.

3.7.7. Gerenciar Administradores

Ao escolher essa opção o administrador tem a opção de cadastrar um novo administrador, para isso ele deve preencher um formulário com o nome, *username*, e-mail e senha do novo administrador. Também é exibida a lista de todos os administradores, onde é possível editar informações de um administrador ou excluir a sua conta.

3.7.8. Lista de Resultados

Ao escolher essa opção é exibida uma lista com todos os resultados enviados por usuários, na lista é exibido o nome da chave de identificação referente ao resultado enviado, ao clicar nesse nome é exibido a lista de caracteres selecionados pelo usuário e as entidades resultantes. Caso exista, também é exibido o comentário feito pelo usuário. Além disso, existe a opção de excluir um resultado da lista.

4. Requisitos de Usuário

Tomando por base o contexto do sistema, foram identificados os seguintes requisitos de usuário:

Requisitos Funcionais

Identificador	**Descrição**	**Prioridade**	**Depende de**
RF01	O sistema deve permitir o gerenciamento de administradores. Administradores possuem nome, username, email e senha.	Alta	
RF02	O sistema deve permitir que administradores gerenciem publicações. Publicações possuem título, veículo onde foi publicada, ano de publicação e um link para o pdf.	Alta	
RF03	O sistema deve permitir que administradores gerenciem projetos. Projetos possuem título, data, número de cadastro, lista de pesquisadores com links para o currículo Lattes de cada um, uma pequena descrição e o conteúdo do projeto, podendo este ser um texto normal ou em HTML.	Alta	
RF04	O sistema deve permitir que administradores gerenciem pesquisadores. Pesquisadores possuem um nome, área de interesse, email para contato e um link para o seu currículo Lattes.	Alta	
RF05	O sistema deve permitir que administradores gerenciem laboratórios. Laboratórios possuem nome, cidade e estado.	Alta	
RF06	O sistema deve permitir que administradores gerenciem postagens. Postagens possuem título, uma pequena descrição e o conteúdo da postagem, podendo este ser um texto normal ou HTML.	Alta	
RF07	O sistema deve permitir que administradores gerenciem informações de chaves de identificação. Chaves de identificação possuem as seguintes informações: nome, autor, data e descrição.	Alta	
RF08	O sistema deve permitir que administradores façam o *upload* em uma pasta criada no sistema de um arquivo de formato SDD e arquivos de imagens e texto associados a uma chave de identificação.	Alta	RF07
RF09	O sistema deve ser capaz de ler e interpretar o conteúdo de um arquivo SDD referente a uma chave de identificação, armazenando em um banco de dados o nome da chave de identificação, nomes dos	Alta	RF07, RF08

	agrupamentos, nomes dos caracteres, nomes dos estados e suas respectivas descrições, nomes das entidades, informações sobre os arquivos e, por fim, as associações entre todas as classes.		
RF10	O sistema deve permitir que administradores substituam imagens ou arquivos HTML referentes a uma chave de identificação por arquivos novos.	Alta	RF07, RF08
RF11	O sistema deve permitir que usuários visualizem todas as publicações na página de publicações.	Alta	RF02
RF12	O sistema deve permitir que usuários visualizem todos os projetos na página de projetos.	Alta	RF03
RF13	O sistema deve permitir que usuários visualizem todos os pesquisadores na página de pesquisadores.	Alta	RF04
RF14	O sistema deve permitir que usuários visualizem todos os laboratórios na página de laboratórios.	Alta	RF05
RF15	O sistema deve permitir que usuários visualizem todas as postagens na página de postagens.	Alta	RF06
RF16	O sistema deve permitir que usuários visualizem e interajam com todas as chaves de identificação na página de chaves de identificação.	Alta	RF07, RF08, RF09
RF17	O sistema deve permitir que usuários visualizem agrupamentos em uma chave de identificação. Agrupamentos possuem um nome e um conjunto de caracteres ou conjunto de agrupamentos.	Alta	RF07, RF08, RF09, RF16
RF18	O sistema deve permitir que usuários visualizem todos os caracteres e seus estados em uma chave de identificação. Estados possuem um nome, e opcionalmente, uma imagem e uma descrição.	Alta	RF07, RF08, RF09, RF16
RF19	O sistema deve permitir que usuários selecionem e desmarquem estados em uma chave de identificação, modificando assim a lista de entidades	Alta	RF07, RF08, RF09, RF16, RF17, RF19
RF20	O sistema deve permitir que usuários visualizem entidades em uma chave de identificação. Entidades possuem um nome, uma lista de estados, e opcionalmente, uma ou mais imagens e um texto descritivo.	Alta	RF07, RF08, RF09, RF16
RF21	O sistema deve permitir que usuários enviem resultados obtidos em uma chave de identificação. O resultado contém os caracteres selecionados pelo usuário, as entidades resultantes, e pode possuir um comentário feito pelo usuário.	Alta	RF07, RF08, RF09, RF16, RF18, RF19
RF22	O sistema deve permitir que administradores visualizem resultados	Alta	RF16, RF18, R19, RF20

	enviados pelos usuários.		

Requisitos Não Funcionais

Identificador	**Descrição**	**Categoria**	**Escopo**	**Prioridade**	**Depende de**
RNF01	A ferramenta deve estar disponível como uma aplicação Web, acessível a partir dos principais navegadores disponíveis no mercado.	Portabilidade	Sistema	Alta	
RNF02	A ferramenta deve ser de aprendizado fácil, não sendo necessário nenhum treinamento especial para seu uso.	Facilidade de Aprendizado	Sistema	Alta	
RNF03	A ferramenta deve ser de fácil operação, não sendo necessário uso contínuo para uma boa operação do sistema.	Facilidade de Operação	Sistema	Alta	
RNF04	O sistema deve controlar o acesso as funcionalidades de gerenciamento por meio de autenticação e autorização.	Segurança	Sistema	Alta	
RNF05	O sistema deve garantir a consistência de dados, obrigando o administrador a preencher todos os campos essenciais na hora de inserir dados.	Confiabilidade	Sistema	Alta	
RNF06	O sistema deve estar protegido contra os tipos de ataques mais comuns na internet.	Segurança	Sistema	Alta	RNF07
RNF07	O sistema deve usar ferramentas para sanitizar todos os dados fornecidos por usuários.	Segurança	Sistema	Alta	
RNF08	As chaves de identificação devem produzir os mesmo resultados gerados no software onde a mesma foi desenvolvida originalmente.	Confiabilidade	Sistema	Alta	
RNF09	O sistema pode ser acessado a qualquer momento a partir de um endereço web.	Disponibilidade	Sistema	Média	
RNF10	Todas as páginas devem ser carregadas com um tempo inferior a 2 segundos.	Eficiência em relação ao tempo	Sistema	Média	
RNF11	O sistema deve possuir mecanismos para se recuperar de falhas e continuar operacional.	Recuperabilidade	Sistema	Alta	
RNF12	O sistema deve utilizar uma linguagem de design moderna como o Material design da Google ou o Flat Design.	Estética da interface	Sistema	Média	
RNF13	O desenvolvimento do sistema deve facilitar manutenções futuras, utilizando padrões de design de software.	Redigibilidade	Sistema	Alta	
RNF14	O sistema deve ser fácil de manter, de modo a acomodar novas funcionalidades ou até mesmo adaptação para organizações específicas.	Manutenibilidade	Sistema	Alta	

RNF15	O sistema deve ser capaz de informar o administrador responsável pela adição de um novo recurso no sistema.	Responsabilização	Sistema	Média	

Documento de Especificação de Requisitos

Projeto: Portal de Biodiversidade de Chalcidoidea

Registro de Alterações:

Versão	Responsáveis	Data	Alterações
1.0	Gustavo Epichin Monjardim	18/09/2018	Versão inicial da documentação.
1.1	Vítor E. Silva Souza	18/10/2018	Primeira revisão.
1.2	Gustavo Epichin Monjardim	20/10/2018	Acertos da primeira revisão.
1.3	Vítor E. Silva Souza	07/11/2018	Versão final.

1. Introdução

Este documento apresenta a especificação dos requisitos da ferramenta Portal de Biodiversidade de Chalcidoidea. A atividade de análise de requisitos foi conduzida aplicando-se técnicas de modelagem de casos de uso e modelagem de classes. Os modelos apresentados foram elaborados usando a UML. Este documento está organizado da seguinte forma: a seção 2 apresenta os subsistemas identificados, mostrando suas dependências na forma de um diagrama de pacotes; a seção 3 apresenta o modelo de casos de uso, incluindo descrições de atores, os diagramas de casos de uso e descrições de casos de uso; a seção 4 apresenta o modelo conceitual estrutural do sistema, na forma de diagramas de classes; finalmente, a seção 5 apresenta o glossário do projeto, contendo as definições das classes identificadas.

2. Identificação de Subsistemas

A Figura 1 mostra os subsistemas identificados no contexto do presente projeto, os quais são descritos na tabela abaixo.

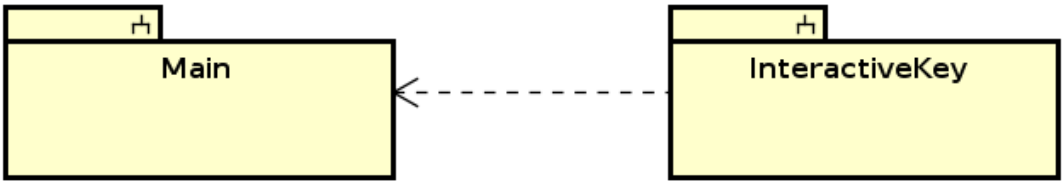

Figura 1 – Diagrama de Pacotes e os Subsistemas Identificados.

Tabela 1 – Subsistemas

Subsistema	**Descrição**
Main	Envolve todas as funcionalidades relacionadas ao cadastramento, alteração, visualização e remoção de dados do sistema.
InteractiveKey	Subsistema contendo todas as funcionalidades relacionadas diretamente a chaves de identificação.

3. Modelo de Casos de Uso

O modelo de casos de uso visa capturar e descrever as funcionalidades que um sistema deve prover para os atores que interagem com o mesmo. Os atores identificados no contexto deste projeto estão descritos na tabela abaixo.

Tabela 2 – Atores.

Ator	**Descrição**
Visitante	Qualquer pessoa que acesse o sistema e não tenha realizado o login.
Administrador	Usuário que realizou o login e pode adicionar, remover e alterar dados do sistema.

A seguir, são apresentados os diagramas de casos de uso e descrições associadas, organizados por subsistema.

3.1 - Subsistema Main

A Figura 2 apresenta o diagrama de casos de uso do subsistema Main.

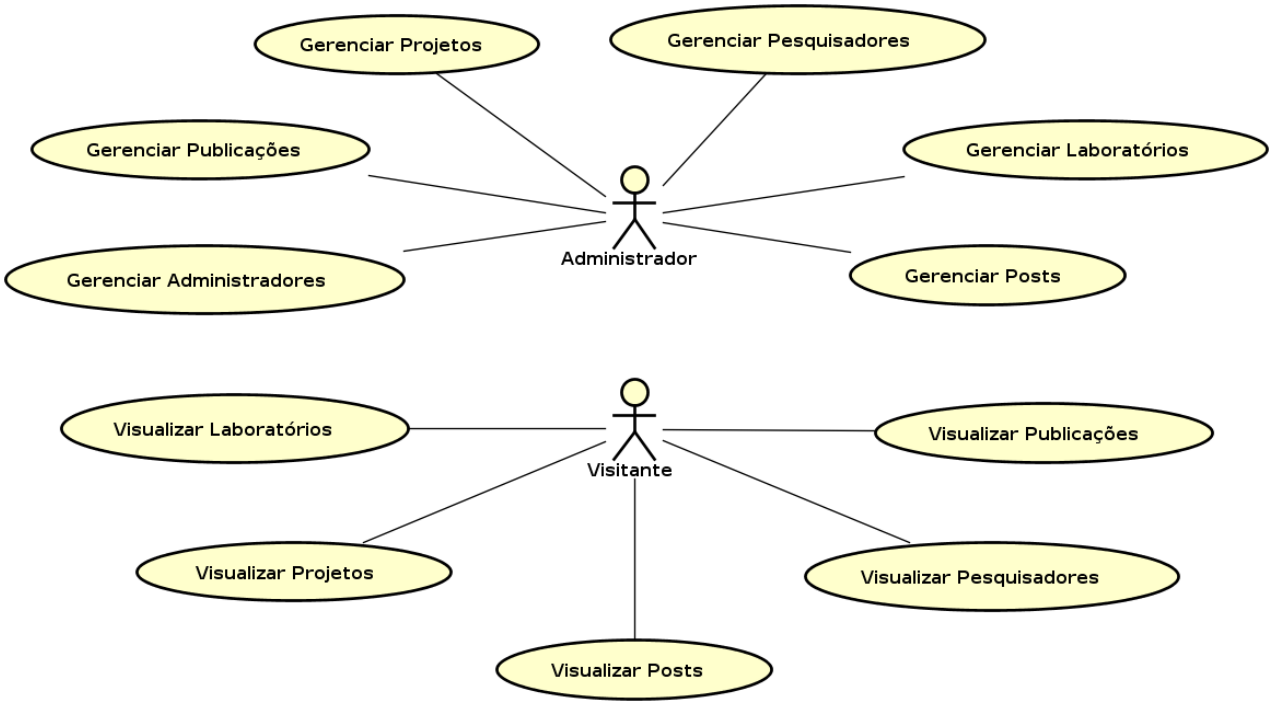

Figura 2 – Diagrama de Casos de Uso do Subsistema Main.

A seguir, são apresentadas as descrições de cada um dos casos de uso identificados. Os casos de uso cadastrais de baixa complexidade, envolvendo inclusão, alteração, consulta e exclusão, são descritos na tabela abaixo, segundo o padrão da organização.

Tabela 3 – Casos de Uso Cadastrais

Subsistema	**Main**				
Identificador	**Caso de Uso**	**Ações Possíveis**	**Observações**	**Requisitos**	**Classes**
UC01	**Gerenciar Administradores**	I, A, C, E	[I]: Informar: nome, nome de usuário, email, senha.	RF01	Admin
UC02	**Gerenciar Publicações**	I, A, C, E	[I]: Informar: título, veículo, ano de publicação, link para o pdf.	RF02	Publication
UC03	**Gerenciar Projetos**	I, A, C, E	[I]: Informar: título, data, número de cadastro, lista de pesquisadores com links para o currículo Lattes de cada um, descrição, conteúdo.	RF03	Project
UC04	**Gerenciar Pesquisadores**	I, A, C, E	[I]: Informar: nome, área de interesse, link para o currículo Lattes, e-mail para contato.	RF04	Researcher
UC05	**Gerenciar Laboratórios**	I, A, C, E	[I]: Informar: nome, cidade, estado.	RF05	Lab
UC06	**Gerenciar Posts**	I, A, C, E	[I]: Informar: título, data, descrição, conteúdo.	RF06	Post

Os casos de uso de consulta mais abrangente que as consultas a um único objeto (já tratadas como parte dos casos de uso cadastrais), mas ainda de baixa complexidade, tais como consultas que

combinam informações de vários objetos envolvendo filtros, estão descritos na tabela abaixo, segundo o padrão da organização.

Tabela 4 – Casos de Uso de Consulta

Subsistema	**Main**			
Identificador	**Caso de Uso**	**Observações**	**Requisitos**	**Classes**
UC07	**Visualizar Publicações**	Uma publicação pode ser visualizada na página de publicações. Para cada publicação, espera-se ver o seu título, ano de publicação, veículo e link para o pdf.	RF11	Publication
UC08	**Visualizar Projetos**	Um projeto pode ser visualizado na página de projetos. Para cada projeto, espera-se ver o seu título, data, número de cadastro, descrição, conteúdo e a lista de pesquisadores responsáveis pelo projeto junto com links para o currículo Lattes de cada pesquisador.	RF12	Project
UC09	**Visualizar Pesquisadores**	Um pesquisador pode ser visualizado na página de pesquisadores. Para cada pesquisador, espera-se ver o seu nome, área de interesse, link para o currículo Lattes e e-mail para contato.	RF13	Researcher
UC10	**Visualizar Laboratórios**	Um laboratório pode ser visualizado na página de laboratórios. Para cada laboratório, espera-se ver o seu nome, cidade e estado.	RF14	Lab
UC11	**Visualizar Posts**	Um Post pode ser visualizado na página de posts. Para cada Post, espera-se ver o seu título, data, descrição e conteúdo.	RF15	Post

3.2 - Subsistema InteractiveKey

A Figura 3 apresenta o diagrama de casos de uso do subsistema InteractiveKey.

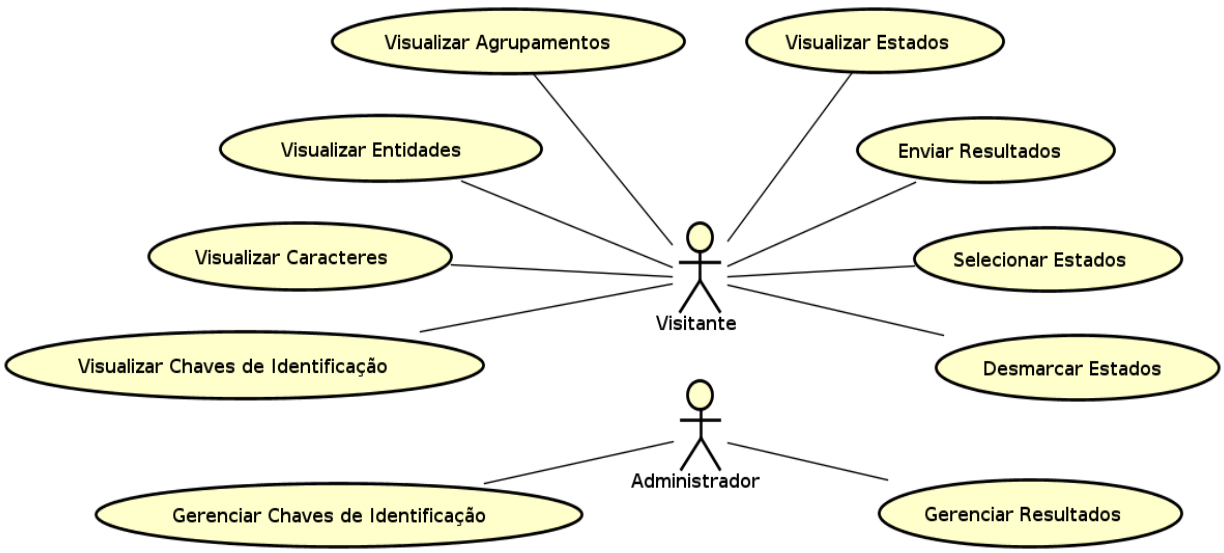

Figura 3 – Diagrama de Casos de Uso do Subsistema InteractiveKey.

A seguir, são apresentadas as descrições de cada um dos casos de uso identificados. Os casos de uso cadastrais de baixa complexidade, envolvendo inclusão, alteração, consulta e exclusão, são descritos na tabela abaixo, segundo o padrão da organização.

Tabela 5 – Casos de Uso Cadastrais

Subsistema	**InteractiveKey**				
Identificador	**Caso de Uso**	**Ações Possíveis**	**Observações**	**Requisitos**	**Classes**
UC12	**Gerenciar Resultados**	C, E		RF21	Result

Os casos de uso de consulta mais abrangente que as consultas a um único objeto (já tratadas como parte dos casos de uso cadastrais), mas ainda de baixa complexidade, tais como consultas que combinam informações de vários objetos envolvendo filtros, estão descritos na tabela abaixo, segundo o padrão da organização.

Tabela 6 – Casos de Uso de Consulta

Subsistema	**InteractiveKey**			
Identificador	**Caso de Uso**	**Observações**	**Requisitos**	**Classes**
UC13	**Visualizar Chaves de Identificação**	Uma chave de identificação pode ser visualizada na página de chaves de identificação. Para cada chave de identificação, espera-se ver o seu nome, data, autor, descrição e a lista de entidades e caracteres.	RF16	Identification Key, Entity, Feature, State
UC14	**Visualizar Agrupamentos**	Um agrupamento pode ser visualizado na seção de caracteres de uma chave de identificação. Um agrupamento possui um nome e um conjunto de caracteres, ou um conjunto de agrupamentos.	RF17	Identification Key, Feature, Node
UC15	**Visualizar Caracteres**	Um caractere pode ser visualizado na seção de caracteres de uma chave de identificação. Cada caractere possui uma lista de estados com os quais o visitante pode interagir.	RF18	Identification Key, Feature, State
UC16	**Visualizar Estados**	Um estado pode ser visualizado na página de um caractere. Para cada estado, espera-se ver o seu nome e, opcionalmente, uma imagem e uma descrição sobre o mesmo.	RF18	Identification Key, Feature, State
UC17	**Visualizar Entidades**	Uma entidade pode ser visualizada na aba de entidades de uma chave de identificação. Cada entidade possui uma lista de caracteres e, opcionalmente, galeria de imagens e um arquivo HTML contendo um texto descritivo.	RF20	Identification Key, Feature, State, Entity

A seguir, são apresentados os casos de uso de maior complexidade que não puderam ser descritos segundo os formatos tabulares simplificados. Esses casos de uso são descritos segundo o padrão de descrição completa de casos de uso definido.

Descrição de Caso de Uso

Projeto: Portal de Biodiversidade de Chalcidoidea
Subsistema: InteractiveKey
Identificador do Caso de Uso: UC18
Caso de Uso: Gerenciar Chaves de Identificação
Descrição Sucinta: Este caso de uso permite a criação, consulta, exclusão e alteração de chaves de identificação.

Fluxos de Eventos Normais

Nome do Fluxo de Eventos Normal	Precondição	Descrição
Incluir chave de identificação		1. O administrador informa o autor, data e descrição da chave de identificação. 2. O administrador informa o nome da pasta que será criada no sistema para armazenar os arquivos referentes a chave de identificação. 3. O sistema cria uma pasta com o nome informado. 4. O administrador faz upload de um arquivo de formato SDD para a pasta criada. 5. O administrador faz upload de imagens e arquivos HTML referentes a caracteres e entidades da chave da identificação para a pasta criada. 6. O sistema interpreta as informações contidas no arquivo SDD, sendo elas, o nome da chave de identificação, nomes dos agrupamentos, nomes dos caracteres, nomes dos estados e suas respectivas descrições, nomes das entidades, informações sobre os arquivos e, por fim, as associações entre todas estas classes. 7. O sistema armazena os dados coletados do arquivo SDD e os dados preenchidos pelo administrador no banco de dados. 8. A chave de identificação é exibida na página de chaves de identificação.
Alterar chave de identificação		1. O administrador escolhe a chave de identificação que deseja alterar. 2. O administrador preenche os campos que deseja alterar. 3. As alterações são registradas.
Excluir chave de identificação		1. O administrador escolhe a chave de identificação que deseja excluir. 2. A chave de identificação é excluída.

Fluxos de Eventos Variantes

Nome do Fluxo de Eventos Normal Relacionado	Variante	Descrição
Incluir chave de identificação	O sistema não consegue criar a pasta.	1.a Uma mensagem é exibida solicitando que o administrador informe um novo nome para a pasta.

Requisitos Relacionados: RF08, RF09, RF10

Classes Relacionadas: Identification Key, Entity, Feature, State, Node

Descrição de Caso de Uso

Projeto: Portal de Biodiversidade de Chalcidoidea
Subsistema: InteractiveKey
Identificador do Caso de Uso: UC19
Caso de Uso: Selecionar Estados

Descrição Sucinta: Este caso de uso permite que o visitante selecione um estado de um caractere em uma chave de identificação.

Fluxos de Eventos Normais

Nome do Fluxo de Eventos Normal	Precondição	Descrição
Selecionar Estado		1. O visitante acessa uma chave de identificação. 2. O visitante acessa um caractere da chave de identificação. 3. O visitante indica o estado que deseja selecionar. 4. O estado indicado é adicionado a lista de estados selecionados. 5. O sistema filtra a lista de entidades, deixando apenas as entidades que possuem todos os estados na lista de estados selecionados. 6. A lista de entidades na seção de entidades é atualizada.

Requisitos Relacionados: RF19

Classes Relacionadas: Identification Key, Entity, Feature, State

Descrição de Caso de Uso

Projeto: Portal de Biodiversidade de Chalcidoidea
Subsistema: InteractiveKey
Identificador do Caso de Uso: UC20
Caso de Uso: Desmarcar Estados

Descrição Sucinta: Este caso de uso permite que o visitante desmarque um estado previamente selecionado de um caractere em uma chave de identificação.

Fluxos de Eventos Normais

Nome do Fluxo de Eventos Normal	Precondição	Descrição
Desmarcar Estado	O visitante deve ter acessado uma chave de identificação e selecionado um de seus estados antes.	1. O visitante indica o estado que deseja desmarcar. 2. O estado indicado é removido da lista de estados selecionados. 3. O sistema filtra a lista de entidades, deixando apenas as entidades que possuem todos os estados na lista de estados selecionados. 4. A lista de entidades na seção de entidades é atualizada.
Desmarcar todos os estados	O visitante deve ter acessado uma chave de identificação e selecionado um ou mais estados antes.	1. O visitante indica que deseja desmarcar todos os estados selecionados. 2. O sistema remove todos os estados da lista de estados selecionados 3. O sistema retorna a lista de entidades para o seu estado original, onde todas as entidades fazem parte da mesma. 4. A lista de entidades na seção de entidades é atualizada.

Requisitos Relacionados: RF19

Classes Relacionadas: Identification Key, Entity, Feature, State

Descrição de Caso de Uso

Projeto: Portal de Biodiversidade de Chalcidoidea
Subsistema: InteractiveKey
Identificador do Caso de Uso: UC21
Caso de Uso: Enviar Resultados

Descrição Sucinta: Este caso de uso permite que o visitante envie um resultado obtido em uma chave de identificação. Um resultado possui uma lista de caracteres e seus estados selecionados, uma lista de entidades e, opcionalmente, um comentário feito pelo visitante

Fluxos de Eventos Normais

Nome do Fluxo de Eventos Normal	Precondição	Descrição
Enviar Resultado	O visitante deve ter acessado uma chave de identificação e selecionado um ou mais estados antes.	1. O visitante informa ao sistema que deseja enviar o resultado obtido na chave de identificação acessada. 2. O sistema exibe uma seção com os estados selecionados e outra com as entidades obtidas como resultado. 3. O visitante envia o resultado para o sistema. 4. O resultado fica disponível para administradores consultarem na *dashboard*.

Fluxos de Eventos Variantes

Nome do Fluxo de Eventos Normal Relacionado	Variante	Descrição
Enviar Resultado	Adicionar comentário	3.a O visitante adiciona um comentário referente ao resultado que obteve e envia o resultado.

Requisitos Relacionados: RF21

Classes Relacionadas: Identification Key, Entity, Feature, State, Result

4. Modelo Estrutural

O modelo conceitual estrutural visa capturar e descrever as informações (classes, associações e atributos) que o sistema deve representar para prover as funcionalidades descritas na seção anterior. A seguir, são apresentados os diagramas de classes de cada um dos subsistemas identificados no contexto deste projeto. Na seção 5 – Glossário de Projeto – são apresentadas as descrições das classes presentes nos diagramas apresentados nesta seção.

4.1 - Subsistema Main

A Figura 4 apresenta o diagrama de classes do subsistema Main.

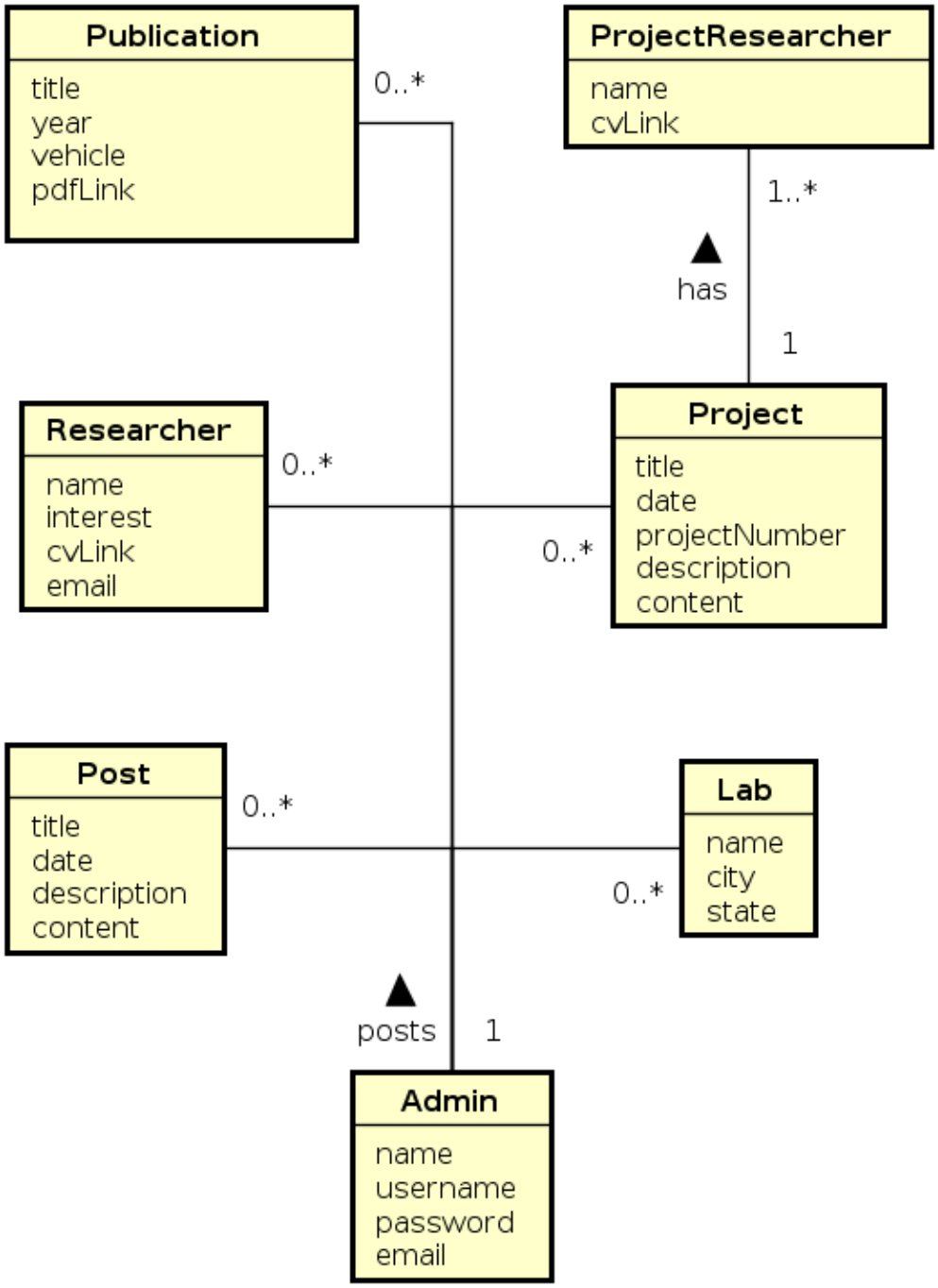

Figura 4 – Diagrama de Classes do Subsistema Main.

4.2 - Subsistema InteractiveKey

A Figura 5 apresenta o diagrama de classes do subsistema InteractiveKey.

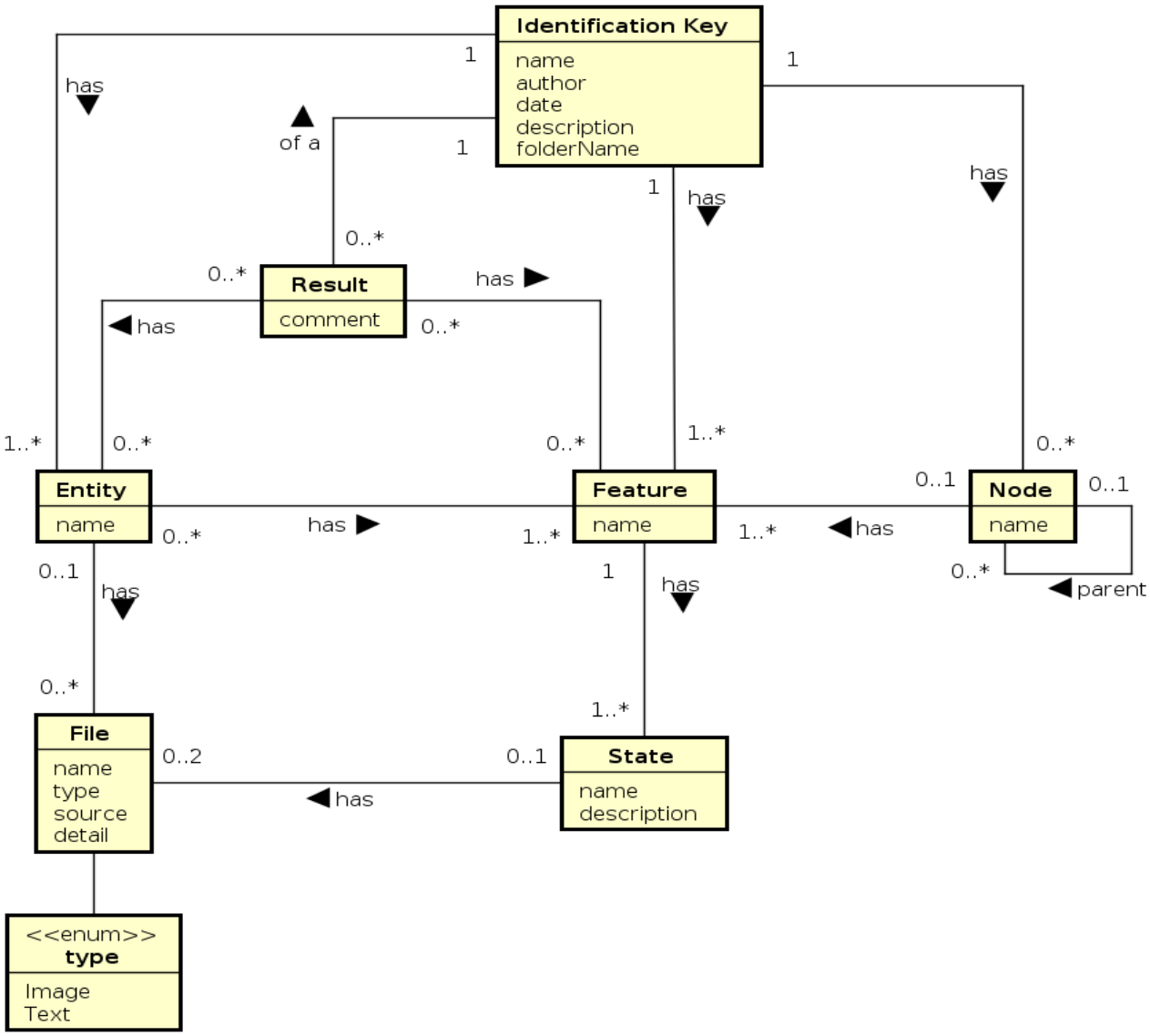

Figura 5 – Diagrama de Classes do Subsistema InteractiveKey.

As seguintes restrições de integridade devem ser observadas:

- RI-1: Um File deve estar sempre associado a uma Entity ou a um State.
- RI-2: Um State deve possuir no máximo apenas um File com type “Text”.
- RI-3: Um State deve possuir no máximo apenas um File com type “Image”.
- RI-4: Uma Entity deve possuir no máximo apenas um File com type “Text”.

5. Glossário de Projeto

Esta seção apresenta as definições dos principais conceitos envolvidos no projeto. Essas definições estão organizadas por subsistema.

5.1 - Subsistema Main

- **Admin:** armazena informações de um administrador, permitindo que o mesmo gerencie o conteúdo do portal.
  - name: nome do administrador.
  - username: nome de usuário do administrador, utilizado para fazer login no sistema.
  - email: e-mail do administrador.
  - password: senha do administrador
- **Publication:** armazena informações sobre publicações científicas.
  - title: título da publicação.
  - year: ano em que a publicação foi publicada.
  - vehicle: veículo onde a publicação foi publicada.
  - pdfLink: link para um pdf da publicação.
- **Project:** armazena informações sobre projetos desenvolvidos por pesquisadores.
  - title: título do projeto.
  - date: data de postagem do projeto.
  - projectNumber: número de cadastro do projeto.
  - description: pequena descrição sobre o projeto.
  - content: texto sobre o projeto.
- **ProjectResearcher:** armazena informações sobre os pesquisadores que estão envolvidos em um projeto.
  - name: nome do pesquisador associado ao projeto.
  - cvLink: link para o currículo Lattes do pesquisador.
- **Researcher:** armazena informações sobre pesquisadores.
  - name: nome do pesquisador.
  - interest: área de interesse do pesquisador.
  - cvLink: link para o currículo Lattes do pesquisador.
  - email: e-mail do pesquisador.
- **Lab:** armazena informações sobre laboratórios.
  - name: nome do laboratório.
  - city: cidade do laboratório.
  - state: estado do laboratório.
- **Post:** armazena informações sobre postagens.

- title: título da postagem.
- date: data de publicação da postagem.
- description: pequena descrição do conteúdo da postagem.
- content: conteúdo da postagem.

5.2 - Subsistema InteractiveKey

- **Identification Key**: armazena informações sobre chaves de identificação.
  - name: nome da chave de identificação.
  - author: nome do autor da chave de identificação.
  - date: data de criação da chave de identificação.
  - description: pequeno texto descrevendo a chave de identificação.
  - folderName: nome da pasta criada no sistema para o armazenamento dos arquivos da chave de identificação.
- **Node:** um Node é um agrupamento de caracteres ou um agrupamento de agrupamentos de uma chave de identificação.
  - name: nome do agrupamento de caracteres.
- **Feature:** armazena informações de um caractere de uma chave de identificação.
  - name: nome de um caractere.
- **State:** armazena informações de um estado de um caractere em uma chave de identificação.
  - name: nome de um estado.
  - description: descrição de um estado.
- **Entity:** armazena informações de uma entidade de uma chave de identificação.
  - name: nome de uma entidade.
- **File:** Armazena informações sobre um arquivo de texto ou de imagem armazenado no sistema, esse arquivo é associado a uma entidade ou a um estado.
  - name: nome de um arquivo.
  - type: tipo do arquivo, pode assumir dois valores: “Text”, “Image”.
  - source: contém o diretório onde o arquivo está armazenado no sistema.
  - detail: descrição de um arquivo.
- **Result:** armazena um resultado obtido por um visitante em uma chave de identificação.
  - comment: comentário feito pelo visitante sobre o resultado obtido.

Documento de Projeto de Sistema

Projeto: Portal de Biodiversidade de Chalcidoidea

Registro de Alterações:

Versão	Responsáveis	Data	Alterações
1.0	Gustavo Epichin Monjardim	26/10/2018	Versão inicial da documentação.
1.1	Vítor E. Silva Souza	07/01/2019	Primeira revisão.
1.2	Gustavo Epichin Monjardim	11/01/2019	Acertos da primeira revisão.
1.3	Vítor E. Silva Souza	13/02/2019	Segunda revisão.
1.4	Gustavo Epichin Monjardim	17/02/2019	Versão final.

1. Introdução

Este documento apresenta o documento de projeto (*design*) da ferramenta Portal de Biodiversidade de Chalcidoidea. Este documento está organizado da seguinte forma: a seção 2 apresenta a plataforma de software utilizada na implementação da ferramenta; a seção 3 trata de táticas utilizadas para tratar requisitos não funcionais (atributos de qualidade); a seção 4 apresenta o projeto da arquitetura de software; por fim, a seção 5 apresenta o projeto dos componentes da arquitetura.

2. Plataforma de Desenvolvimento

Na Tabela 1 são listadas as tecnologias utilizadas no desenvolvimento da ferramenta, bem como o propósito de sua utilização.

Tabela 1 – Plataforma de Desenvolvimento e Tecnologias Utilizadas.

Tecnologia	Versão	Descrição	Propósito
Node.js	8.11.4	Plataforma para desenvolvimento de aplicações *server-side* utilizando JavaScript.	Permitir a criação de aplicações de alta escalabilidade utilizando JavaScript.
JavaScript	8	Linguagem de programação interpretada utilizada em páginas web.	Adicionar interatividade para páginas web.
npm	6.4.1	Gerenciador de pacotes para a linguagem JavaScript.	Permitir a instalação de pacotes necessários para o sistema.

MongoDB	4.0	Software de banco de dados orientado a documentos.	Persistência dos dados manipulados pela ferramenta.
Express.js	4.16.0	Framework web minimalista e flexível do Node.js para a construção de aplicações web e APIs.	Fornecer um conjunto robusto de ferramentas para o desenvolvimento de aplicações web.
Redis	4.0.11	Armazenamento de estrutura de dados de chave-valor na memória.	Armazenar os dados da sessão no servidor.
Pug	2.0.3	*Template engine* para Node.js	Permite criar e manipular conteúdo HTML de forma dinâmica.
HTML	5	Linguagem de marcação utilizada para a elaboração de páginas web.	Criação de páginas web para a exibição do conteúdo do sistema.
CSS	3	Linguagem utilizada para definir o estilo de um documento HTML.	Elaborar interfaces com design amigável para o usuário.
JQuery	3.3.1	Biblioteca de funções JavaScript que interage com o HTML.	Simplificar a criação de *scripts* para o navegador do cliente.
Bootstrap	4.0	Framework web para desenvolvimento de componentes de interface para sites e aplicações web usando HTML, CSS e JavaScript.	Melhorar a produtividade, utilizando componentes pré-definidos para o desenvolvimento de páginas web responsivas.
Mongoose	5.2.10	Biblioteca do Node.js que proporciona uma solução baseada em esquemas para modelar os dados da aplicação.	Reduzir a complexidade das operações com o banco de dados.
connect-redis	3.4.0	*Middleware* do Node.js utilizado para conectar com o Redis.	Permitir o armazenamento dos dados da sessão no Redis.
Helmet	3.13.0	Coleção de 13 pequenas funções de *middlewares* que configuram cabeçalhos HTTP relacionados a segurança.	Proteger a aplicação contra vulnerabilidades da web bastante conhecidas configurando os cabeçalhos HTTP adequadamente.
Multer	1.3.1	*Middleware* do Node.js utilizado para lidar com *upload* de arquivos.	Fornecer suporte para *upload* de arquivos no sistema.
Passport	0.4.0	*Middleware* de autenticação para Node.js	Fornecer o controle de acesso a certas áreas do sistema através da autenticação de usuários.
Xml2js	0.4.19	Conversor de XML para JSON (JavaScript Object Notation).	Facilitar a interpretação de arquivos SDD.
Compression	1.7.3	*Middleware* utilizado para a compressão de respostas HTTP.	Reduzir o tempo de resposta do servidor comprimindo o tamanho dos arquivos enviados.
Csurf	1.9.0	*Middleware* do Node.js para	Proteger o sistema contra falsificações

		proteção contra ataques CSRF (*Cross-site request forgery*).	de solicitação *cross-site*.
NGINX		Servidor HTTP e proxy reverso.	Controlar o tráfego para a aplicação e servir arquivos estáticos de forma eficiente.
Github	2.19	Sistema de Controle de Versão.	Controlar as várias versões do código-fonte do sistema desenvolvido.
Atom	1.26.1	Editor de texto de código aberto.	Permitir a escrita do código fonte do sistema.
Astah* UML	7.2.0	Ferramenta para modelagem em UML.	Modelar os diagramas necessários para o sistema implementado.
LibreOffice	6.0.6.2	*Suite* gratuita de aplicativos para escritório.	Apoiar a elaboração da documentação.

3. Atributos de Qualidade e Táticas

Na Tabela 2 são listados os atributos de qualidade considerados neste projeto, com uma indicação se os mesmos são condutores da arquitetura ou não e as táticas a serem utilizadas para tratá-los .

Tabela 2 – Atributos de Qualidade e Táticas Utilizadas.

Categoria	Requisitos Não Funcionais Considerados	Condutor da Arquitetura	Tática
Facilidade de Aprendizado, Facilidade de Operação	RNF02, RNF03, RNF12, RNF13, RNF14	Sim	• Adotar padrões de interface modernos e minimalistas como o Material Design e o Flat Design que facilitem a navegação pelo sistema. • Exibir mensagens de erro e avisos significativos para ações tomados pelo usuário.
Segurança de Acesso	RNF04, RNF06, RNF15	Sim	• Identificar usuários usando *login* e autenticá-los por meio de senha. • Utilizar ferramentas para impedir a injeção de conteúdo malicioso no sistema. • O sistema deve armazenar o administrador responsável pela inclusão de novo conteúdo no site.
Manutenibilidade, Portabilidade	RNF01	Sim	• Utilizar somente bibliotecas e *frameworks* livres e gratuitos, capazes de rodar nos principais navegadores do mercado. • O sistema deve ser organizado segundo o padrão MVC (*Model, View, Controller*).
Confiabilidade	RNF05, RNF07,	Não	• Validar o preenchimento de formulários, garantindo que todos os campos sejam preenchidos corretamente.
Aptidão Funcional	RNF08	Sim	• O sistema possuirá um algoritmo capaz de

			apresentar resultados corretos para chaves de identificação.
Desempenho	RFN10	Não	• O sistema deve utilizar mecanismos para reduzir o tamanho das solicitações HTTP enviadas e recebidas pelo servidor.
Disponibilidade	RNF09, RNF11	Não	• O sistema deverá utilizar ferramentas que permitam que o mesmo se recupere de falhas e volte a operar normalmente.

4. Arquitetura de Software

A arquitetura de software do sistema Portal de Biodiversidade de Chalcidoidea baseia-se no padrão MVC (*Model, View, Controller*). Esse padrão tem como objetivo dividir a aplicação em 3 partes interconectadas, isso é feito para separar como a informação é representada internamente da forma como ela é apresentada para o usuário. A Figura 1 apresenta a visão geral das camadas juntamente com os relacionamentos entre elas. Dessa forma, temos que:

- ***Model*:** Camada responsável por gerenciar os dados e as regras de negócio da aplicação.
- ***View*:** Camada que define como os dados da aplicação serão apresentados para o usuário.
- ***Controller*:** Camada responsável por atualizar o modelo e a visão em resposta a requisições do usuário.

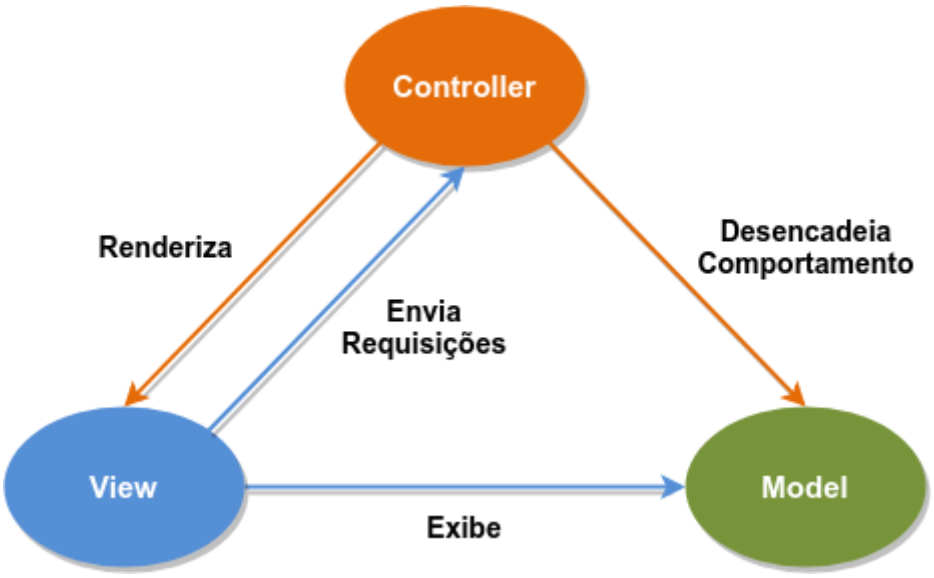

Figura 1 – Arquitetura de Software Completa.

Nas próximas seções, serão apresentados diagramas no padrão proposto pelo FrameWeb (SOUZA, 2007) de forma adaptada ao *framework* Express.js.

4.1. Model

Nessa seção são apresentados os modelos de entidade seguindo o método FrameWeb, porém adaptadas para um banco de dados orientado a documentos. Diferentemente da abordagem original proposta em 2007, todos os atributos que não podem ser nulos tiveram a *tag not null* omitida e aqueles que podem tiveram a *tag null* adicionada de forma a reduzir o impacto visual nos diversos diagramas. Além disso, foi considerada que a estratégia padrão de recuperação de uma associação é do tipo *lazy*, e não *eager* como proposto pelo FrameWeb, devido a forma como o MongoDB opera.

Todas as classes de domínio são modeladas a partir de *Schemas* do Mongoose, uma biblioteca do Node.js utilizada para modelar os dados da aplicação, sendo que cada *Schema* é mapeado para uma coleção no MongoDB. Os seguintes tipos de variáveis são aceitos em um *Schema:*

- String
- Number
- Date
- Buffer
- Boolean
- Mixed
- ObjectId
- Array
- Decimal128
- Map

Informações adicionais sobre Schemas estão disponíveis em https://mongoosejs.com/docs/guide.html.

A seguir, a Figura 2 representa o modelo de entidades para o subsistema **Main** e a Figura 3 representa o modelo de entidades para o subsistema **InteractiveKey**.

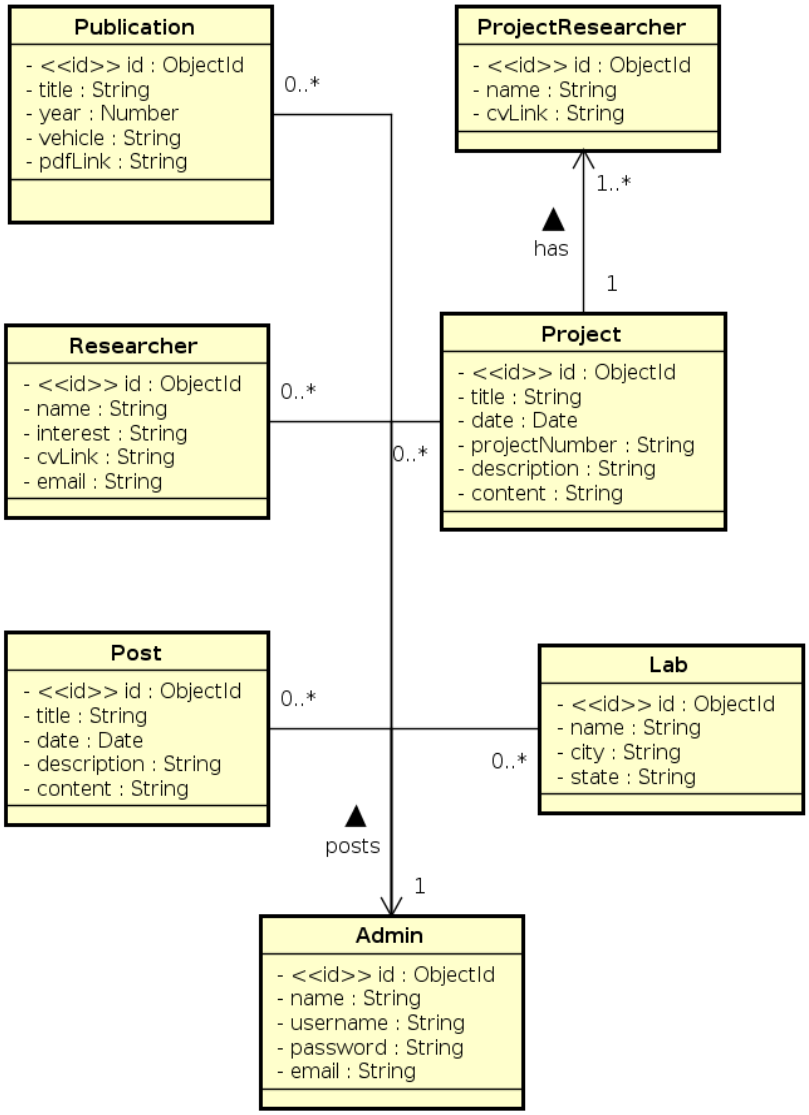

Figura 2 – Modelo de entidades para o subsistema Main.

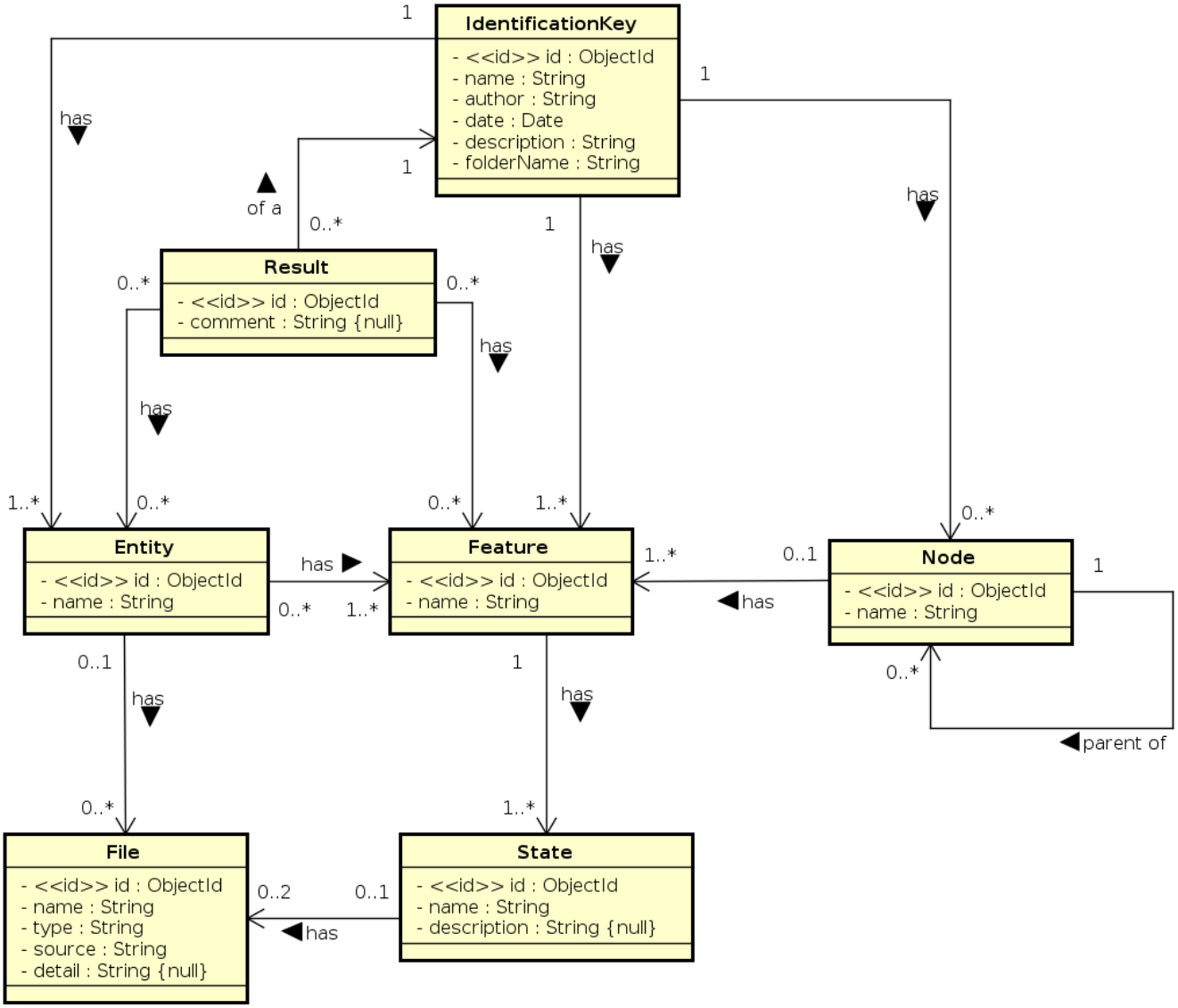

Figura 3 – Modelo de entidades para o subsistema InteractiveKey.

4.2. View e Controller

Essa seção apresenta os modelos de navegação referentes às camadas *View* e *Controller,* seguindo o método FrameWeb, porém adaptadas para o *framework* Express.js. As funcionalidades criar, visualizar, editar e excluir (abreviadas de CRUD, do inglês create, read, update and delete), seguem um mesmo fluxo de execução e de interação com o usuário. Devido ao fato dessas funcionalidades serem similares para todos os casos de uso cadastrais, optou-se por representar esse fluxo de execução na Figura 4 por meio de um Modelo de Navegação genérico. Para isso foi utilizado o caso de uso cadastral de **Publicação** como exemplo**.**

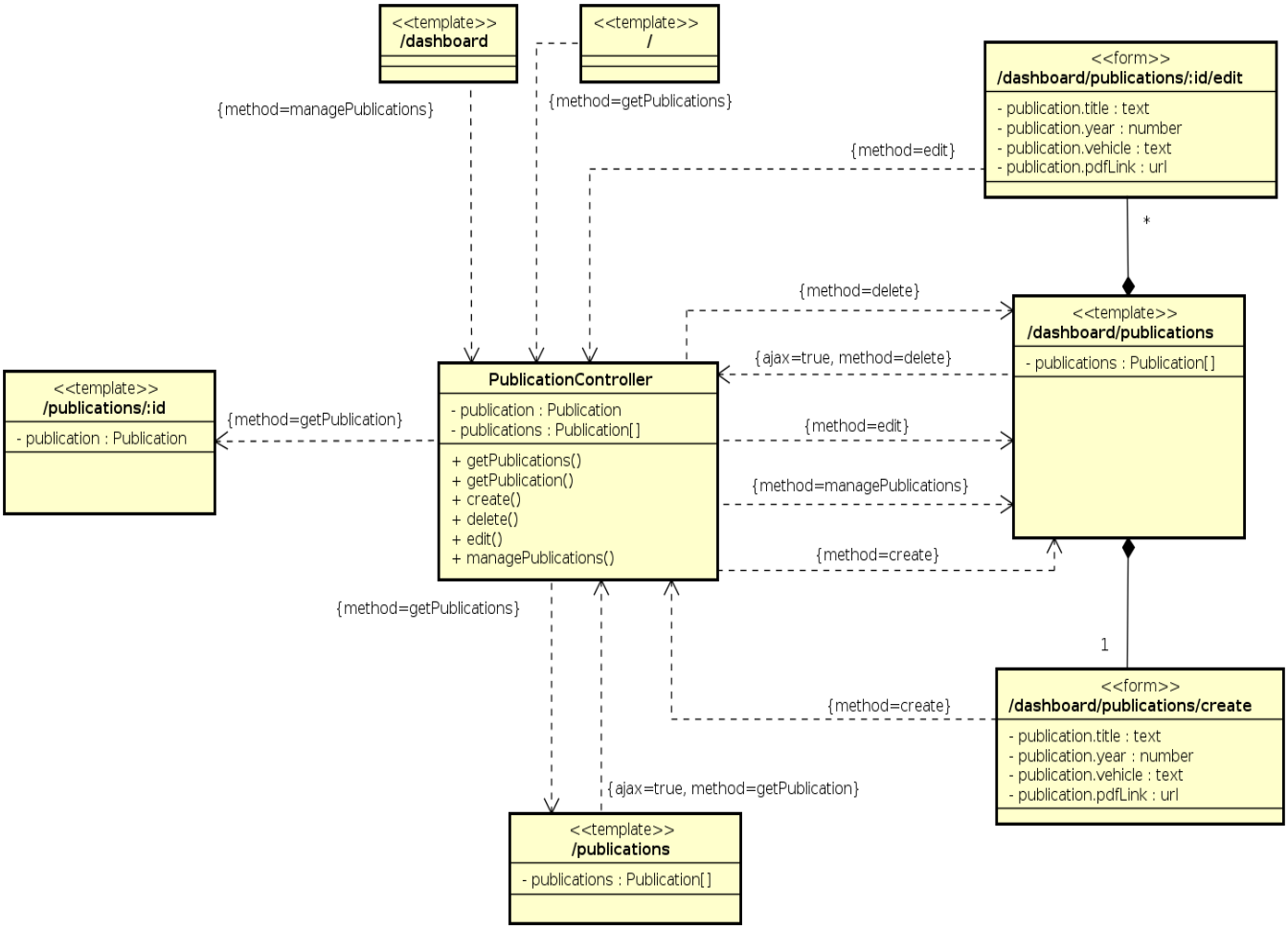

Figura 4 – Modelo de navegação de um CRUD usando como base os casos de uso cadastrais para publicações.

Este modelo genérico pode ser aplicado para todas as entidades que possuem funcionalidades do tipo CRUD. Dessa forma, cada entidade possui um *controller* que receberá as requisições dos usuários. Para exibir as informações na camada de visão, o sistema utiliza um *template* para a listagem de entidades (/publications), um para detalhamento (/publications/:id) e um para gerenciamento (/dashboard/publications), sendo que na página de gerenciamento da entidade existe um formulário para a criação de uma nova entidade (/dashboard/publications/create) e formulários para a edição de entidades existentes (/dashboard/publications/:id/edit).

Além disso, é importante notar a presença da restrição *ajax,* indicando que os métodos *getPublication* e *delete* são feitos através de uma requisição AJAX (Asynchronous JavaScript and XML) por meio da qual o usuário indica na interface a entidade que deseja visualizar ou deletar respectivamente, e uma requisição é enviada para o controlador contendo o id da entidade escolhida.

Para os casos de uso que apresentam funções diferentes do que apenas as funcionalidades do tipo CRUD, o modelo de navegação mostrado anteriormente não pode ser aplicado. A Figura 5 apresenta o modelo de navegação para os casos de uso relacionados à autenticação no sistema, incluindo as funcionalidades para registrar administrador, efetuar login, efetuar logout, alterar senha e deletar administrador.

Nesse diagrama, é possível observar a presença da restrição *result* indicando o retorno obtido na requisição. Caso ocorra algum erro durante a autenticação de um usuário (result = error), ele é redirecionado para a página de *login* novamente para que possa corrigir seu erro, caso tudo ocorra normalmente (result = success) ele é redirecionado para a *dashboard*.

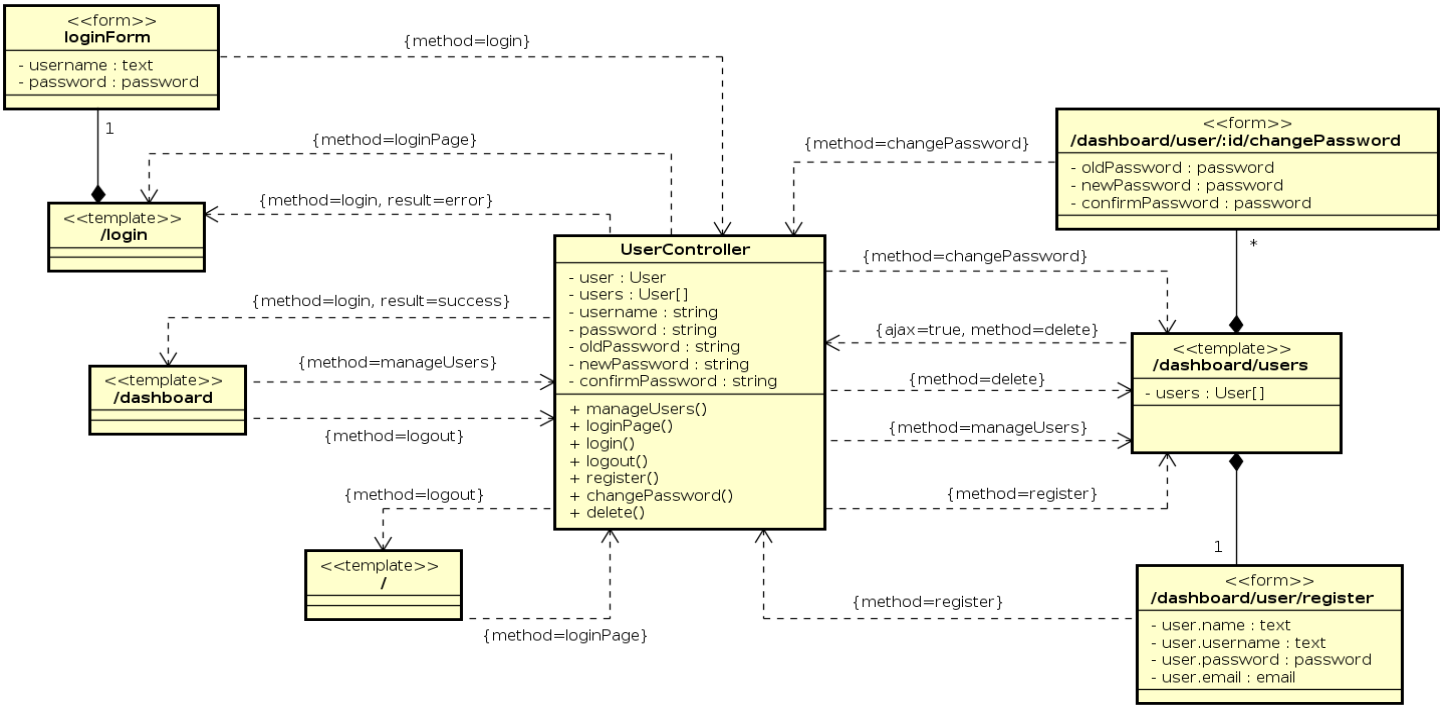

Figura 5 – Modelo de navegação para o fluxo de autenticação no sistema.

A Figura 6 apresenta o modelo de navegação para o fluxo de uma chave de identificação, incluindo os casos de uso **Selecionar Estados**, **Desmarcar Estados**, **Desmarcar Todos os Estados** e **Visualizar Entidades**. A partir da página principal de uma chave de identificação (/key/:id) o usuário pode selecionar e desmarcar os estados referentes a essa chave de identificação exibidos na interface. Essa informação é enviada para o controlador através de requisições AJAX, o controlador por sua vez atualiza a lista de estados selecionados (exibida no template /key/:id/selectedStates) e a lista de resultados obtidos (/key/:id/results).

Nesse diagrama, é possível notar a presença da restrição *resultType* indicando o tipo de retorno da requisição, os métodos **addState**, **removeState** e **clearStates** retornam um JSON (JavaScript Object Notation) como resposta da requisição.

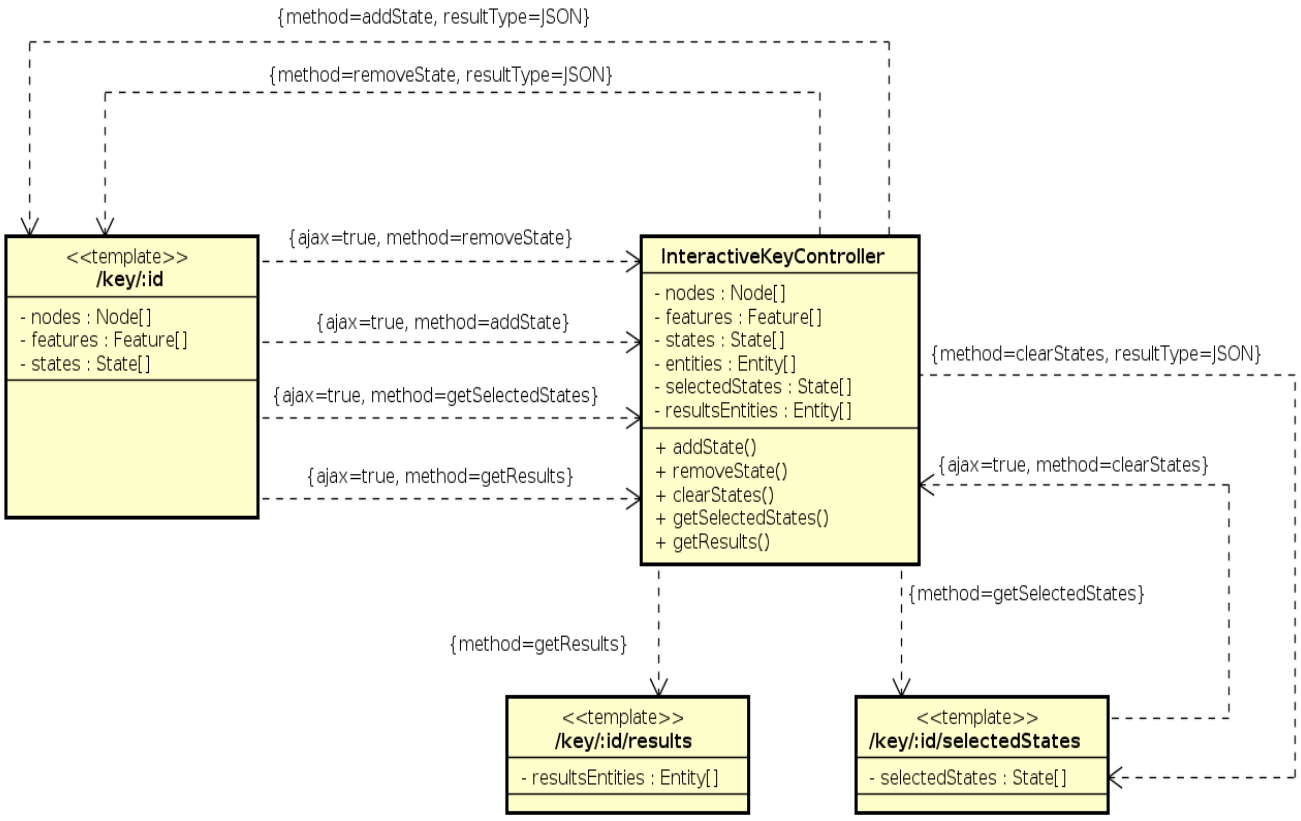

Figura 6 – Modelo de navegação para as funcionalidades relacionadas a uma chave de identificação.

4.3. Modelos de Persistência e Aplicação

O Modelo de Persistência é um diagrama de classes da UML que representa as classes DAO existentes, responsáveis pela persistência das instâncias das classes de domínio. Esse diagrama guia a construção das classes DAO, que pertencem ao pacote de persistência (SOUZA, 2007).
O Modelo de Aplicação, por sua vez, representa as classes de serviço, que são responsáveis pela codificação dos casos de uso, e suas dependências. Esse diagrama é utilizado para guiar a implementação das classes do pacote Aplicação e a configuração das dependências entre os pacotes Controle, Aplicação e Persistência, ou seja, quais classes de ação dependem de quais classes de serviço e quais DAOs são necessários para que as classes de serviço alcancem seus objetivos (SOUZA, 2007).

Ambos os modelos não foram utilizados na elaboração desse documento devido a natureza da arquitetura MVC adotada para o projeto. Na arquitetura MVC o controlador, além de ser responsável por executar a lógica de negócio, também faz toda a interação com o banco de dados, não sendo necessária nenhuma camada adicional para mediar essa interação. Com o objetivo de reduzir a complexidade das operações com o banco de dados, foi utilizado a biblioteca *Mongoose* do *Node.js*, que fornece um mapeamento de objetos do MongoDB similar ao ORM (Object Relational Mapping). Dessa forma, o *Mongoose* traduz os dados do banco de dados para objetos *JavaScript*, permitindo assim uma forma simples para manipular esses dados diretamente do controlador.

5. Referências

SOUZA, V. E. S. FrameWeb – A Framework-based Design Method for Web Engineering. Dissertação (Mestrado) — Universidade Federal do Espírito Santo, 2007.