Fernando Guimarães Pinheiro

Aplicação do método FrameWeb no desenvolvimento de um sistema de informação utilizando um framework PHP e um framework JavaScript

Vitória, ES

2017

Fernando Guimarães Pinheiro

Aplicação do método FrameWeb no desenvolvimento de um sistema de informação utilizando um framework PHP e um framework JavaScript

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

Vitória, ES

2017

Fernando Guimarães Pinheiro

Aplicação do método FrameWeb no desenvolvimento de um sistema de informação utilizando um framework PHP e um framework JavaScript

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, XX de dezembro de 2017:

Prof. Dr. Vítor E. Silva Souza
Orientador

Membro 1
Universidade Federal do Espírito Santo

Membro 2
Universidade Federal do Espírito Santo

Vitória, ES
2017

Agradecimentos

Nem momento de conclusão desse ciclo e após muito tempo investido nele eu me reservo o direto de ser breve e agradecer algumas das várias pessoas que me foram importantes. A minha mãe, Andressa, pela implacável disposição para me dar todo o aparato necessário para a conclusão do curso e para todas as demais coisas da minha vida. A minha namorada, Amanda, pelo incentivo e palavras de conforto nos momentos de maior fragilidade. Aos meus colegas de curso, amigos e sócios, André, Denis, Zanardo, Tomir e Murillo; pela paciência e compreensão durante esses vários anos de caminhada. A todos os sócios da Resultate por me exigirem o melhor. Ao meu orientador, Vítor, por todo auxílio para que esse projeto se tornasse realidade. A todos os professores da UFES, principalmente mas não exclusivamente os do NEMO, que me fizeram ver todo o lado bom do curso. E por fim os colegas de classes que estiveram juntos durante todo o curso.

"Você é aquilo que você faz continuamente.
Excelência não é uma eventualidade. É um hábito"

(Aristóteles)

Resumo

FrameWeb é um método para o projeto de Sistemas de Informação Web que utilizam em sua arquitetura alguns *frameworks* comuns no desenvolvimento Orientado a Objetos, como controlador frontal, mapeamento objeto/relacional e injeção de dependência. Criado em 2007 e recentemente revisado dentro da metodologia de Desenvolvimento Orientado a Modelos, o FrameWeb vem sendo usado em diferentes projetos e arquiteturas/plataformas para verificar a generalidade de suas propostas.

Um desses projetos é o Sistema de Controle de Afastamento de Professores (SCAP), sistema Web desenvolvido em dois outros Projetos de Graduação na plataforma Java, com *frameworks* controladores frontais diferentes. Partindo do levantamento de requisitos já realizado nestes projetos passados, este trabalho tem como objetivo reimplementar este sistema em outra plataforma de desenvolvimento, combinando as linguagens de programação PHP e JavaScript, de modo a avaliar se o método FrameWeb se adequa a este novo contexto.

Dado o objetivo de verificar o quão aderente o método FrameWeb se apresenta quando as tecnologias que foram utilizadas para o desenvolvimento da aplicação mudam, utilizou-se como metodologia a Engenharia Reversa, desenvolvendo em primeiro lugar a aplicação para, em momento posterior, documentá-la com os modelos de projeto prescritos pelo FrameWeb.

Palavras-chaves: Sistema Web, FrameWeb, PHP, JavaScript, Laravel, Vue.js, Quasar

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

AJAX	*Asynchronous JavaScript And XML*
API	Interface de Programação de Aplicação, do inglês *Application Programming Interface*
CSS	Folhas de estilo em cascata, do inglês *Cascading Style Sheets*
CT	Centro Tecnológico
DI	Departamento de Informática
HTML	Interface de Programação de Aplicação, do inglês *HyperText Markup Language*
HTTP	Protocolo de Transferência de Hipertexto, do inglês *Hypertext Transfer Protocol*
IDE	Ambiente de Desenvolvimento Integrado, do inglês *Integrated Development Environment*
JSON	Notação de Objetos JavaScript, do inglês *JavaScript Object Notation*
NPM	Gerenciador de pacotes do Node.js, do inglês *Node.js Package Manager*
ORM	Mapeamento de Objeto/Relacionamento, do inglês *Object-relational mapping*
PRPPG	Pró-reitoria de Pesquisa e Pós-Graduação
PWA	*Progressive Web App*
REST	Transferência de Estado Representacional, do inglês *Representational State Transfer*
RIA	Aplicações Ricas para Internet, do inglês *Rich Internet Applications*
SPA	Aplicativo de página única, do inglês *Single-Page Applications*
UML	Linguagem de Modelagem Unificada, do inglês *Unified Modeling Language*
URL	Localizador Uniforme de Recursos, do inglês *Uniform Resource Locator*

Sumário

1 Introdução

Com a crescente demanda do uso e facilidade de acesso à Internet, temos como consequência a popularização das aplicações de Sistemas de Informação Web (*Web Information Systems* ou WISs). Com grande flexibilidade esses sistemas podem resolver uma vasta série de problemas.

Como o desenvolvimento de uma aplicação do zero pode ser muito custosa, foram surgindo os *frameworks*, cujo o objetivo é oferecer um conjunto de conceitos, funcionalidades, bibliotecas, etc. para que o desenvolvedor reaproveite todo esse arcabouço e possa focar cada vez mais nas regras de negócio.

Uma das importantes etapas da construção de uma aplicação é a fase de projeto. É nela será definido e especificado o que será desenvolvido. Tendo isso em vista, o método FrameWeb (SOUZA, 2007; SOUZA; FALBO; GUIZZARDI, 2009) se destaca, principalmente para o desenvolvimento de WISs. O método define uma arquitetura padrão para facilitar a integração com *frameworks* muito utilizados, além de propor um conjunto de modelos que traz para o projeto arquitetural do sistema conceitos inerentes a estes *frameworks*.

Baseado neste cenário, este projeto propõe a construção de uma aplicação que usará dois *frameworks* distintos, com duas linguagens de programação distintas. Cada um desses *frameworks* constituem uma parte da aplicação. A primeira parte, referente ao lado do servidor, é a API (*Application Programming Interface*) e usa PHP. A segunda, referente ao lado do cliente, são interfaces de interação com o cliente e usa JavaScript.

O sistema em questão é o SCAP (Sistema de Controle de Afastamentos de Professores), valendo ressaltar que a Engenharia de Requisitos já foi feita e será reaproveitada (DUARTE, 2014; PRADO, 2015), fazendo com que o foco maior seja no projeto e na implementação.

Com a aplicação em mãos o próximo passo será a construção de modelos FrameWeb, em uma espécie de engenharia reversa, onde seguimos o fluxo contrário de projeto, implementando primeiro para depois passar para o projeto do sistema. Por fim, será feita a análise desses modelos, de modo a avaliar se o FrameWeb se adequa a uma plataforma e a um estilo de arquitetura para o qual, ao menos nos trabalhos feitos nesta universidade, nunca foi utilizado.

1.1 Objetivos

O objetivo de um Projeto de Graduação é primeiramente colocar em práticas os conhecimentos que foram obtidos ao longo da gradução do curso de Ciência da Computação. Dentre as várias matérias que podem ser citadas as mais relevantes são: Programação III (DEITEL; DEITEL, 2011), Engenharia de Software (PRESSMAN, 2011), Projeto de Sistemas (BUSCHMANN et al., 1996) e Banco de Dados (ELMASRI; NAVATHE, 2011.). Isto será evidenciado na aplicação a ser desenvolvida.

Em particular, o objetivo deste trabalho é contribuir com a pesquisa do método FrameWeb, construindo e analisando os modelos propostos pelo método com base em uma implementação do SCAP (DUARTE, 2014; PRADO, 2015) em uma nova plataforma de desenvolvimento, visto que até o momento ele foi implementado apenas na plataforma Java.

1.2 Método de Trabalho

Para atingir o objetivo geral deste trabalho, os seguintes passos são realizados:

- Revisão da literatura: etapa dedicada ao estudo dos conteúdos relevantes para a condução do trabalho;
- Revisão da Documentação de Requisitos: etapa dedicada ao estudo da documentação da aplicação que será desenvolvida, o SCAP;
- Desenvolvimento da aplicação – *API*: etapa dedicada a implementação da *API* da aplicação, levando em consideração a documentação de requisitos;
- Desenvolvimento da aplicação – Interfaces: etapa dedicada a implementação de todas as interfaces da aplicação, usando a *API* desenvolvida;
- Criação dos modelos FrameWeb: etapa dedicada a criação dos modelos de projeto de acordo com o método FrameWeb, baseado no código da aplicação já desenvolvida (i.e., Engenharia Reversa);
- Redação da monografia e apresentação dos resultados.

1.3 Organização do Texto

Esse trabalho é composto por quatro partes, além dessa introdução, que são descritas a seguir:

- **Capítulo 2** – Revisão bibliográfica: revisão e apresentação dos conteúdos que foram essenciais para a construção do projeto;
- **Capítulo 3** – SCAP: apresentação dos requisitos do trabalho a ser desenvolvido;
- **Capítulo 4** – Projeto: apresentação da arquitetura usada no desenvolvimento do projeto, subdividida em 2 subsistemas; apresentação dos subsistemas do lado do servidor; apresentação do subsistema do lado do cliente; apresentação dos resultados obtidos; e por fim a aplicação do método FrameWeb.
- **Capítulo 5** – Considerações Finais: análise feito sobre o trabalho desenvoldido onde são apresentadas as conclusões, limitações e opotrunidade de trabalhos futuros.

2 Engenharia Web, *frameworks* e FrameWeb

Neste capítulo são apresentados todos os importantes conceitos usados no projeto. Inicialmente falamos sobre a Engenharia Web e como as aplicações voltadas para a Internet são construídas. Após isso são apresentados os *frameworks* utilizados no projeto. Por fim, descrevemos o FrameWeb, um método voltado para o desenvolvimento de sistemas de informação Web baseados em *frameworks*.

2.1 Engenharia Web

A Internet se tornou uma tecnologia indispensável para os negócios, comércio, comunicação, educação, engenharia, entretenimento, medicina e várias outras áreas. Podemos ver sua importância e o seu impacto em nossas vidas: a usamos para comprar produtos, conhecer pessoas, consumir conteúdos, expressar nossa opinião e muito mais. O veículo usado para que tudo isso seja possível são as aplicações baseadas na Web (*Web-based Applications*), também conhecida como WebApps (PRESSMAN; LOWE, 2009).

Nos primórdios da Internet, por volta de 1990 a 1995, os *websites* não passavam de vários arquivos de hipertexto vinculado uns ao outros. Com o passar do tempo e com o surgimentos de novas tecnologias e ferramentas de desenvolvimentos houve um aumento da capacidade de desenvolvimento, tanto para o *client-side* (lado do cliente) quanto para o *server-side* (lado do servidor) desses *websites* (PRESSMAN; LOWE, 2009).

Quando falamos de WebApps alguns atributos estão presentes na maioria delas. Entre outros vários temos: intensidade de rede, concorrência, carga imprevisível, sensibilidade a performance, alta disponibilidade, orientação aos dados, imediatividade, segurança.

O *HyperText Transfer Protocol* (HTTP) é o ingrediente mais básico pelo qual a Web foi fundada. É um protocolo de aplicação cliente-servidor que define um formato padrão para que recursos sejam requisitados na Web. Foi inventado no começo da década de 1990 visando a criação de sistema distribuídos de hipermídia, usando o protocolo TCP/IP (CASTELEYN et al., 2009).

Para cada requisição HTTP feita é obtida uma resposta. Quando um usuário entra com uma URI na barra de endereço de um navegador, esse navegador se encarrega de fazer a requisição requerendo um determinado recurso de um determinado servidor e obtendo a resposta desse servidor.

Na requisição é necessário informar o seguinte:

- *Method*: o método a ser utilizado. Os mais utilizados e que são implementados nos

principais navegadores do mercado são o *GET* e o *POST*, porém temos outros como *PUT*, *PATCH*, *DELETE*, *OPTIONS*, etc;

- URI: identificador do recurso requerido;
- *Header*: (opicional) cabeçalho com instruções para o servidor;
- *Body*: (opicional) corpo da requisição, podem se usado como parâmetros.

Já na resposta da requisição são retornados:

- *Status Code*: código da resposta. É um número de três dígitos onde o primeiro define a classe do código. Podem ser 1xx para informação; 2xx para sucesso; 3xx para redirecionamento; 4xx para erros do cliente; e 5xx para erros do servidor;
- *Header*: cabeçalho com informações adicionais passada pelo servidor;
- *Body*: (opicional) corpo da resposta, o conteúdo em si.

2.1.1 Arquiteturas Multicamadas de Aplicações Web

Aplicações Web de grande escala, tais como portais e aplicações de comércio eletrônico, tipicamente estão expostas a um grande número de requisições concorrentes e devem exibir um alto nível de disponibilidade. Para tal, são necessárias arquiteturas de software modulares e multicamadas, nas quais os diferentes componentes possam ser facilmente replicados para aumentar o desempenho e evitar pontos de falha (CASTELEYN et al., 2009).

Arquiteturas de software de aplicações Web dessa natureza são normalmente organizadas em três camadas de software (CASTELEYN et al., 2009):

- Camada de Apresentação: responsável por processar as requisições vindas do cliente e construir as páginas HTML. É desenvolvida usando as extensões de servidores Web, as quais são capazes de construir dinamicamente as páginas HTML a serem enviadas como resposta para o cliente. Essas páginas são geradas tomando por base os dados produzidos pela execução de componentes de negócio, que estão na camada abaixo.
- Camada de Lógica de Negócio: responsável por executar componentes que realizam a lógica de negócio da aplicação. Para tal, comunica-se com a camada de gerência de recursos para acessar bases de dados persistentes, sistemas legados ou para invocar serviços externos.
- Camada de Gerência de Recursos: representa o conjunto de serviços oferecidos por

diferentes sistemas, tais como aqueles suportando o acesso a bases de dados, a outros sistemas ou, de maneira geral, a serviços Web externos.

A Figura 1 ilustra como é o ambiente de execução comumente utilizado na construção de WebApps.

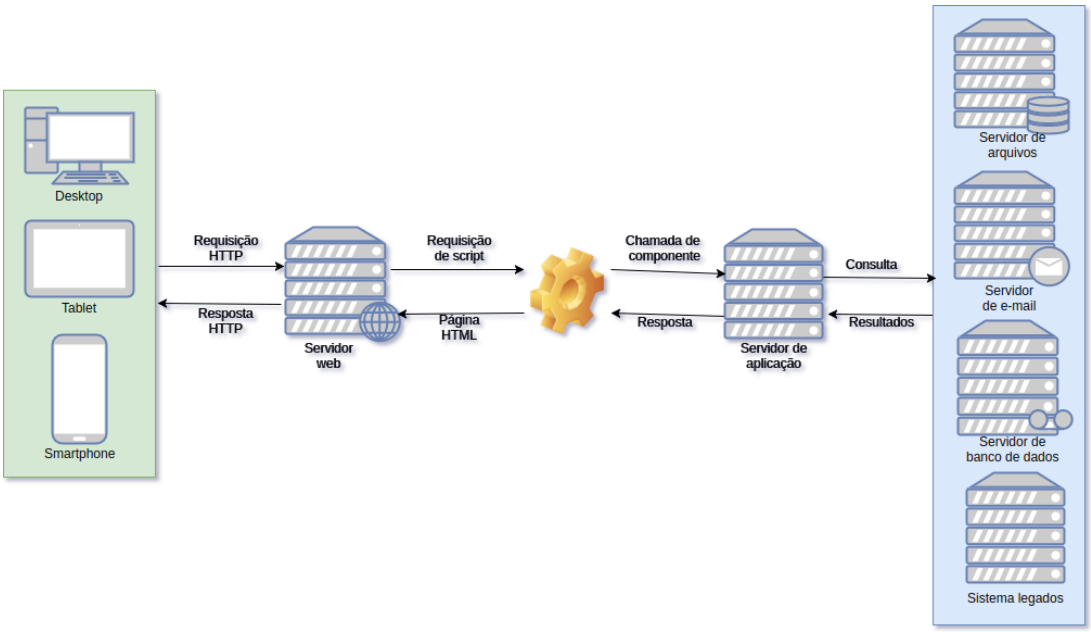

Figura 1 – Ambiente de Execução para Arquitetura Multicamadas (adaptado de (CASTELEYN et al., 2009)).

2.1.2 Uma abordagem diferente

Dois fatores são importantes de serem analisados ao perceber que o ambiente de execução apresentado na Figura 1 tem sofrido algumas alterações. O primeiro é o aumento da necessidade de comunicação entre sistema independentes.

Cada vez mais precisamos comunicar sistemas com outros sistemas terceiros, a construção e o consumo de APIs tem-se tornado uma prática comum no desenvolvimento de WebApps. Há inclusive quem defenda que a primeira coisa que deva ser desenvolvida na aplicação é a API, o movimento API-First[1] tem chamado a atenção de vários desenvolvedores.

O outro fator é o constante avanço das tecnologias voltadas para a construção de RIAs (*Rich Internet Applications*), caracterizadas por suportar comportamentos sofisticados na interface (e.g. *drag&drop*); mecanismos para minimizar a transferência de dados fazendo com que a interface faça o gerenciamento das interações e apresentações; armazenando dados tanto no lado do cliente quanto no lado do servidor; e execução também feita tanto no lado do cliente quanto no lado do servidor (CASTELEYN et al., 2009).

[1] <http://www.api-first.com/>

Hoje podemos ter interfaces construídas somente com o uso de JavaScript.[2] Além disso essa linguagem de programação está entre a mais popular no mundo segundo Weinberger (2017).

Com isso em vista podemos juntar esses dois pontos, ou seja, para a construção de uma WebApp podemos dividi-la em duas partes, uma responsável pela API e outra responsável pela RIA. A Figura 2 demonstra como seria o ambiente de execução dessa nova abordagem.

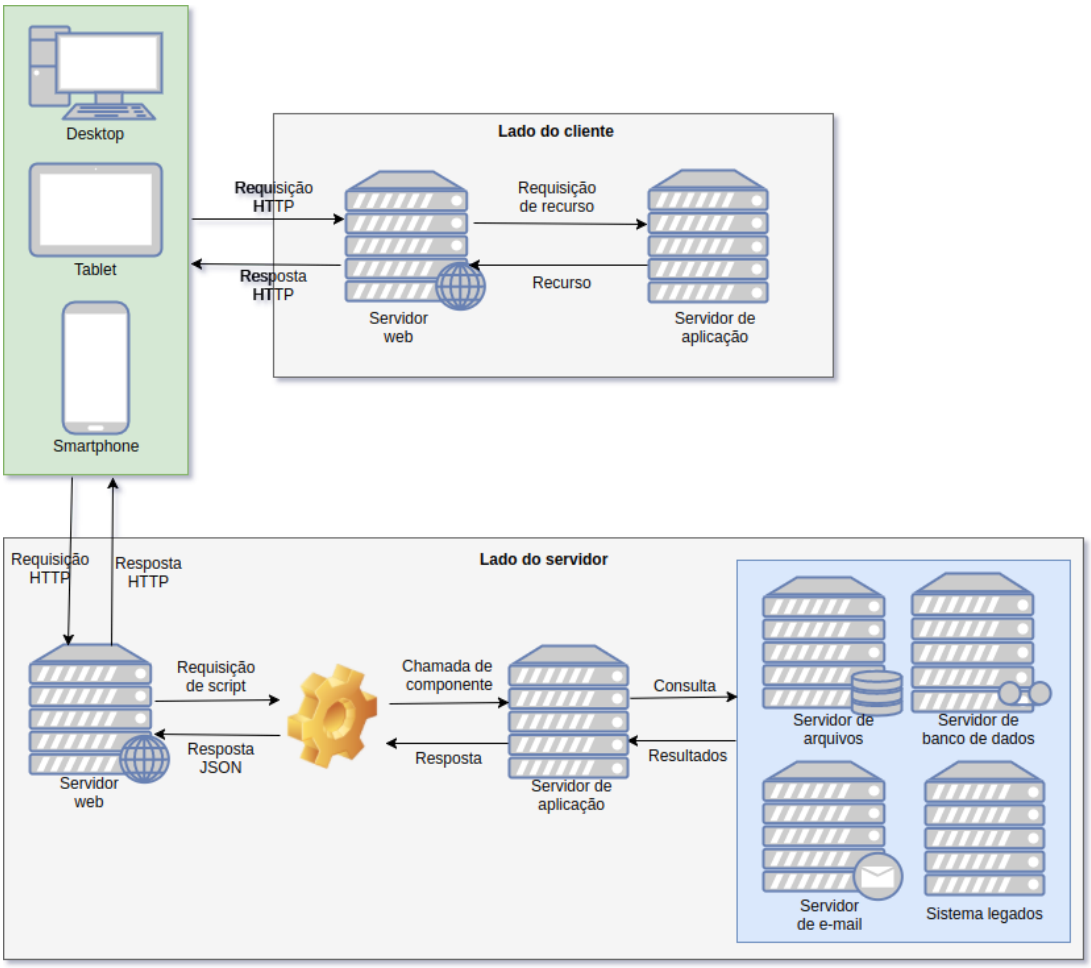

Figura 2 – Ambiente de execução multicamadas vista de uma nova abordagem.

A primeira requisição HTTP é feita no lado do cliente, que irá retornar os recursos necessários para executar a RIA no navegador do cliente, que pode ser um desktop, tablet, smartphone ou qualquer dispositivo que rode um navegador.

Após a RIA estar rodando no navegador do usuário, uma outra aplicação é requerida: aquela destinada para a lógica de negócio, chamada de "lado do servidor". Requisições HTTP são feitas para o servidor Web, que por sua vez faz uma requisição para o motor de script, que executa as rotinas existentes no servidor de aplicação. O servidor de aplicação por sua vez pode fazer várias consultas para outros servidores como de banco de dados,

[2] <https://www.javascript.com/>

e-mail e gerenciamento de arquivos. O retorno da requisição HTTP sempre será um JSON (*JavaScript Object Notation*).

JSON é uma formatação leve de troca de dados. Para seres humanos, é fácil de ler e escrever. Para máquinas, é fácil de interpretar e gerar. Está baseado em um subconjunto da linguagem de programação JavaScript mas mantém seu formato texto completamente independente de linguagem (JSON.ORG, 2017).

2.2 Frameworks

Para a construção de WebApps uma prática muito comum é a utilização de *framework*s. Um *framework* em desenvolvimento de software, é uma abstração que une códigos comuns entre vários projetos de software provendo uma funcionalidade genérica. O objetivo ao se usá-los é não perder tempo fazendo atividades repetitivas e que são comuns a vários sistemas.

2.2.1 MVC e Laravel

MVC, abreviação de *Model-View-Controller* (GAMMA et al., 1994), é uma arquitetura de software desenvolvida para o Smalltalk-80™. Tem uma aceitação alta e atualmente é uma das arquiteturas mais utilizadas para a construção de sistemas Web.

As três camadas são usadas para separar a aplicação. A camada de interação do usuário (*View*), a camada de manipulação dos dados (*Model*) e a camada de controle (*Controller*).

Toda a lógica de negócio da aplicação fica no *Model*. Já a *View* é a camada de interação com o usuário. Ela apenas faz a exibição dos dados, sendo ela por meio de um HTML, XML, JSON, entre outros. O *Controller* é usado para intermediar o *Model* e a *View*. A Figura 3 demonstra a comunicação entre esses componentes do padrão.

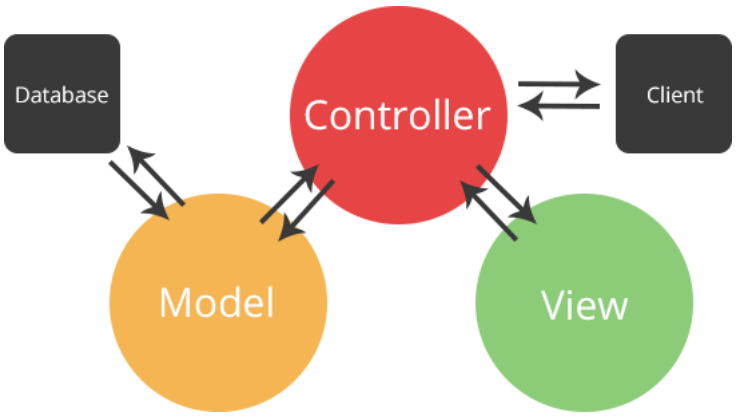

Figura 3 – Padrão MVC (PROGRAMMINGHELP, 2013).

O Lavarel[3] é um *framework* MVC feito em PHP bastante expressivo e tem uma sintaxe bem atraente, valorizando a simplicidade e legibilidade. Com uma documentação bem completa e fácil compreensão tem grande aderência de programadores iniciais e além disso a curva de aprendizado baixa.

O *framework* em si é a composição dezenas de pacotes, que não necessariamente são feitos pela mesma pessoa. A ideia por trás disso é reaproveitar o máximo possível de código e se concentrar no que é realmente relevante para o desenvolvedor. Esses pacotes são gerenciados pelo *Composer*, que é um gerenciador de pacotes PHP.

O Laravel é feito por programadores para programadores e há um vasto ecossistema em torno do *framework*. Várias bibliotecas (pacotes) são feitas sob medida para funcionarem com o Laravel, a comunidade é bem presente e ativa e tudo é *open source*. Por esses motivos não é de se estranhar que seja um dos mais populares *frameworks* de PHP.

Alguns de seus pacotes possuem destaque, como por exemplo o *Eloquent*,[4] que é o pacote ORM utilizado por padrão. Por ter o *Eloquent* como classe herdada, os modelos da aplicação já tem, sem nenhuma configuração, uma série de métodos disponíveis para interação com o banco de dados. Se usadas as convenções propostas pelo *framework* ganha-se muita agilidade nesse mapeamento, mas como é prática no Laravel, o programador tem a opção de fazer isso por conta própria.

Funcionando como se fosse um sistema de versão para o banco de dados, o Laravel tem uma funcionalidade interessante chamada de *Migrations*.[5] As *Migrations* são classes que possuem instruções relativas ao banco de dados, como a criação de tabelas, alterações de colunas, etc., e com um ordem de prioridade. Sendo assim é garantido que os todos os comandos referente ao esquema do banco de dados sejam executados na mesma ordem, independente de onde esse sistema esteja rodando.

Outros pacotes conhecidos do *framework* são o Homestead[6] e Valet[7] para criação de todo o ambiente de desenvolvimento (sistema operacional e todos os programas e serviços configurados); Cashier[8] para interfaces com *gateway* de pagamento; Horizon[9] para gerenciamento de filas; entre outros.

---

3 <https://laravel.com>
4 <https://laravel.com/docs/master/eloquent>
5 <https://laravel.com/docs/master/migrations>
6 <https://laravel.com/docs/master/homestead>
7 <https://laravel.com/docs/master/valet>
8 <https://laravel.com/docs/master/billing>
9 <https://laravel.com/docs/master/horizon>

2.2.2 MVVM, Vue.js e Vuex

O padrão MVVM (*Model*, *View*, *View-Model*) foi revelado pela primeira vez por John Gossman, arquiteto do WPF[10] e Silverlight[11] pela Microsoft[12] e é uma variação do padrão PM (*Presentation Model*) (FOWLER, 2004). O interessante desse padrão é que uma abstração da *view* é criada, onde compartamentos e estados são separados da apresentação (SMITH, 2009). É importante notar que ambos MVVM e PM são derivados do padrão MVC.

Nesse padrão o *Model* se refere ao nível do domínio, representando o estado ou ao acesso aos dados, similar ao domínio apresentado no padrão MVC. Em seguida temos a *View*, que assim como o padrão MVC, é responsável pela apresentação, tudo que aparece na tela do usuário. Por fim, o *ViewModel*, atua entre o *Model* e a *View* e é responsável pela lógica de negócio da *View*. Normalmente interege invocando métodos do *Model* e provendo dados para a *View*. A diferença do *ViewModel* para o *Controller* da arquitetura MVC reside no fato de que o *ViewModel* não é separado da apresentação e, além disso, suporta o fluxo de dados nos dois sentidos (*two-way data binding*) facilitando a propagação das alterações, entre *ViewModel* e *View*. A Figura 4 demonstra a comunicação entre esses compomentes do padrão.

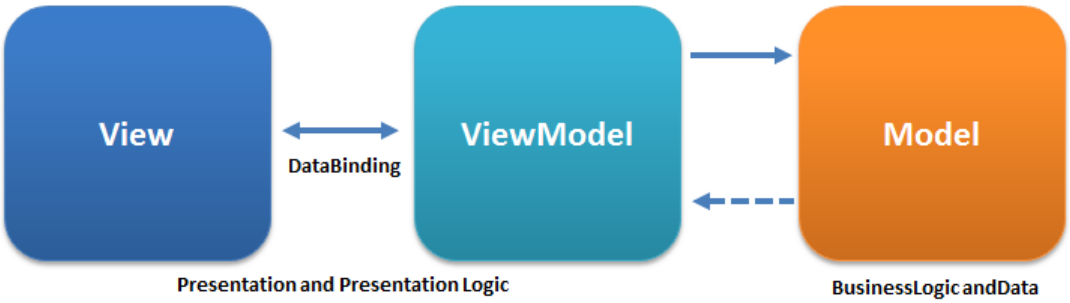

Figura 4 – Padrão MVVM.

Após a criação desse padrão alguns *frameworks* foram criados o levando em consideração. Um desses é o Vue.js,[13] um *framework* feito em JavaScript que tem com ideia separar uma coisa grande em várias pequenas, que são chamadas de componentes. A partir de um componente principal são incluídos vários outros componentes, que por sua vez incluem outros componentes, formando uma árvore de componentes. Cada componente desses possui seus próprias variáveis, métodos, eventos e funções de controle relacionados ao clico de vida (iniciação, criação, montagem, atualização, destruição, etc.).

Para que um componente possa acessar os dados dos outros componentes, prática

10 <https://msdn.microsoft.com/pt-br/library/cc564903.aspx>
11 <https://www.microsoft.com/silverlight>
12 <https://www.microsoft.com>
13 <https://vuejs.org/>

que é muito normal dentro da abordagem, foi utilizado o Vuex,[14] que é uma biblioteca de gerenciamento de estado feito para o Vue.js. O *State* (estado) nada mais é do que as variáveis existentes na aplicação com os seus respectivos valores. Tendo em vista que eventos de mouse e teclado são disparados o tempo todo e que esses eventos podem invocar métodos do componente em questão, convenciona-se que o *State* só pode ser alterado por meio de *Actions* (ações), que são despachados e executados como se estivessem em uma fila, assim sabemos exatamente qual é o *State* da aplicação antes e depois de cada evento disparado. O Vuex é inspirado no Flux[15] e no Redux.[16]

Uma das várias possibilidades na utilização desse padrão é na criação de SPAs (*Single Page Applications*). Uma SPA é uma aplicação Web que interege com o usuário reescrevendo dinamicamente a página atual sem a necessidade do carregamento inteiro de novas páginas. Todo o código necessário para a execução dessa aplicação é carregado uma única vez, na primeira requisição. Outras requisições — assíncronas — podem ser feitas de acordo com a interação do usuário, são chamadas de requisições AJAX (*Asynchronous JavaScript and XML*).

Outra possibilidade são as PWAs (*Progressive Web Apps*). PWAs são comumente SPAs e são construídas de uma maneira que se assemelham muito com os aplicativos para dispositivos móveis. Os principais motivadores por trás dos PWAs são confiabilidade, pois mesmo em situações adversas de conectividade após o carregamento inicial não há perda de usabilidade; rapidez, com animações fluidas e respostas instantâneas às interações; e engajamento, dado que maneira do usuário usar a PWA vai ser parecida com a que ele já usa em todos os demais aplicativos no seu dispositivo.

2.3 FrameWeb

O FrameWeb é um método de projeto para construção de sistemas de informação Web (Web Information Systems – WISs) baseado em *frameworks* (SOUZA, 2007). O método propõe uma arquitetura básica que muito se aproxima da implementação do sistema e é usada durante a construção de uma aplicação.

Como é um método utilizado na fase de projeto, o FrameWeb não descreve um processo de software completo. Sugere-se que as fases apresentadas na Figura 5 para o processo de desenvolvimento.

FrameWeb define uma arquitetura lógica padrão para WISs baseada no padrão arquitetônico Service Layer (Camada de Serviço), proposto por Randy Stafford em (FOWLER, 2002). Como mostra a Figura 4.6, o sistema é dividido em três grandes camadas:

[14] <https://vuex.vuejs.org>
[15] <https://facebook.github.io/flux>
[16] <https://redux.js.org>

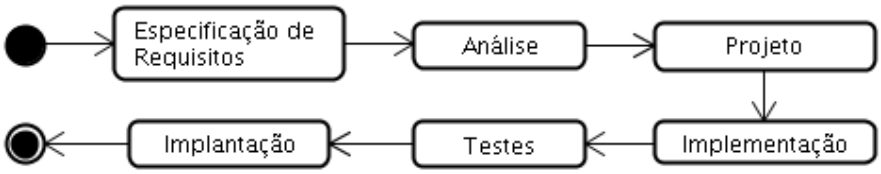

Figura 5 – Um processo de desenvolvimento de software simples sugerido por FrameWeb (SOUZA, 2007).

lógica de apresentação, lógica de negócio e lógica de acesso a dados.

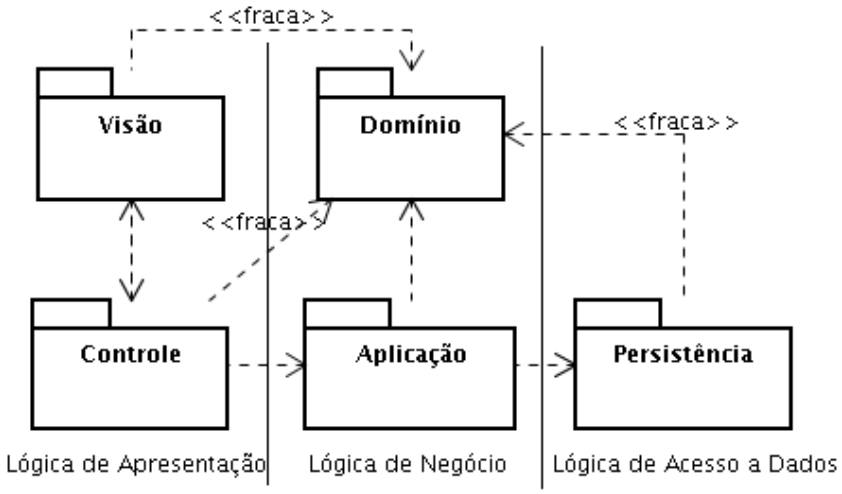

Figura 6 – Arquitetura padrão para WIS baseada no padrão arquitetônico *Service Layer* (FOWLER, 2002).

A camada de apresentação é responsável pelas interfaces gráficas e estão presentes nela os pacotes de Visão e Controle. Implementada na segunda camada está a lógica de negócio, onde estão presentes dois pacotes, o de Domínio e o de Aplicação. Já na terceira camada temos a pacote de Persistência.

O FrameWeb define uma linguagem de modelagem baseada na UML. São quatro modelos distintos que possuem como função auxiliar os programadores na fase de desenvolvimento. Os modelos são:

- Modelo de Entidades: representa os objetos de domínio do problema e seu mapeamento para a persistência em banco de dados relacional. A partir dele são implementadas as classes do pacote de Domínio na atividade de implementação (SOUZA, 2007).
- Modelo de Persistência: representa as classes DAO (ALUR; CRUPI; MALKS, 2003) existentes, responsáveis pela persistência das instâncias das classes de domínio. Esse diagrama guia a construção das classes DAO, que pertencem ao pacote Persistência (SOUZA, 2007).

- Modelo de Navegação: representa os diferentes componentes que formam a camada de Lógica de Apresentação, como páginas Web, formulários HTML e classes de ação. Esse modelo é utilizado pelos desenvolvedores para guiar a codificação das classes e componentes dos pacotes Visão e Controle (SOUZA, 2007).

- Modelo de Aplicação: 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 (SOUZA, 2007).

O método FrameWeb foi aplicado no desenvolvimento deste trabalho. Todos os modelos criados serão apresentados na Seção 4.4.

3 SCAP

Neste capítulo apresentamos resumidamente o SCAP (Sistema de Controle de Afastamentos de Professores). As documentações técnicas de requisitos e análise foram realizadas por Duarte (2014) e Prado (2015) e, por isso, serão reaproveitadas nesse trabalho. Tendo isso em vista, apresentamos apenas a descrição do escopo e os modelos de casos de uso.

3.1 Descrição do escopo

No Departamento de Informática (DI) da UFES os pedidos de afastamento são solicitados para eventos no Brasil ou no exterior. Tais solicitações são avaliadas por professores do DI e, em alguns casos, também pela diretoria do Centro Tecnológico (CT) juntamente com a Pró-reitoria de Pesquisa e Pós-Graduação (PRPPG). Caso aprovado em todas as instâncias, o afastamento é aceito (PRADO, 2015). O objetivo do sistema é apoiar os professores e o secretário do departamento, visando o aumento da agilidade e organização nesse contexto de solicitações de afastamento.

O afastamento tem um fluxo diferente dependendo do seu tipo, que pode ser nacional ou internacional. Para os afastamentos nacionais o fluxo é mais curto e pode ser resolvido no próprio departamento em que o professor estiver lotado. Já para os afastamentos internacionais temos um fluxo que não depende apenas do departamento mas também do centro de ensino (CT, no caso da Informática) e da PRPPG.

Os professores que solicitarem o afastamento para um evento que acontecerá no Brasil precisam da aprovação da Câmara Departamental, que é composta pelos docentes e funcionários do departamento. Dez dias corridos após o pedido feito pelo professor e se ninguém se manifestar contra o pedido em questão, o pedido será aprovado.

Já para as solicitações de afastamento para eventos que forem internacionais, será necessário definir um relator, que nada mais é do que um professor a ser escolhido pelo Chefe do Departamento para emitir um parecer inicial sobre o pedido. Se o parecer for positivo, a solicitação ainda precisará dos pareceres do CT e da PRPPG, nesta ordem.

Como o SCAP é um sistema interno do DI, as interações com o CT e com a PRPPG não são feitas pelas pessoas que de fato estão cuidado do caso, e sim pelo secretário do departamento que irá passar o que foi decidido por essas pessoas para dentro do sistema.

3.2 Modelo de casos de uso

Para a apresentação dos requisitos do SCAP serão utilizados diagramas de casos de uso. Como o próprio nome sugere, um caso de uso é uma maneira de usar o sistema. Usuários interagem com o sistema por meio de casos de uso. Tomados em conjunto, os casos de uso de um sistema definem a sua funcionalidade. Casos de uso são, portanto, os "itens" que o desenvolvedor negocia com seus clientes (FALBO, 2017a).

O SCAP foi dividido em 2 subsistemas: Núcleo e Secretaria. O primeiro contempla os casos de usos dos professores e do chefe de departamento, enquanto o segundo envolve os casos de uso dos secretários. As figuras 7 e 8 mostram os diagramas de casos de uso destes subsistemas. Nos parágrafos que se seguem, apresentamos uma descrição sucinta dos casos de uso levantados para o SCAP (PRADO, 2015). Uma versão mais detalhada dessa descrição pode ser vista nos apêndices do trabalho do Prado (2015).

Antes da apresentação dos modelos é importante descrever os atores do sistema. Dá-se nome de ator a um papel desempenhado por entidades físicas (pessoas ou outros sistemas) que interagem com o sistema em questão da mesma maneira, procurando atingir os mesmos objetivos. Uma mesma entidade física pode desempenhar diferentes papéis no mesmo sistema, bem como um dado papel pode ser desempenhado por diferentes entidades (OLIVé, 2007). Os atores do SCAP são listados na Tabela 1.

Tabela 1 – Atores do SCAP (PRADO, 2015).

Ator	**Descrição**
Professor	Professor efetivo do DI/UFES.
Chefe de departamento	Professores do DI/UFES que estão realizando a função administrativa de chefe e sub-chefe do departamento.
Secretário	Secretário do DI/UFES.

Os secretários são responsáveis pela administração do sistema. São eles que fazem os cadastros dos professores, assim como a indicação de parentescos e mandatos dos chefes de departamento. Além disso, são eles que fazem os registros dos pareceres do CT e da PRPPG para as solicitações de afastamento internacionais. Por fim, ainda podem consultar solicitações e arquivar as que estiverem concluídas.

Os professores podem criar solicitações de afastamento e cancelar as suas próprias se assim desejar. Nas solicitações internacionais, quando designado relator, é papel dele dar o parecer em relação a essa solicitação. Já nas nacionais ele pode se manifestar contra, caso ainda esteja no prazo. Para isso ele pode consultar as solicitações de afastamentos já cadastradas.

O chefe do departamento é a pessoa que, além de professor, exerce a função administrativa de chefe do DI por um mandato. Ele é quem escolhe quem serão os relatores

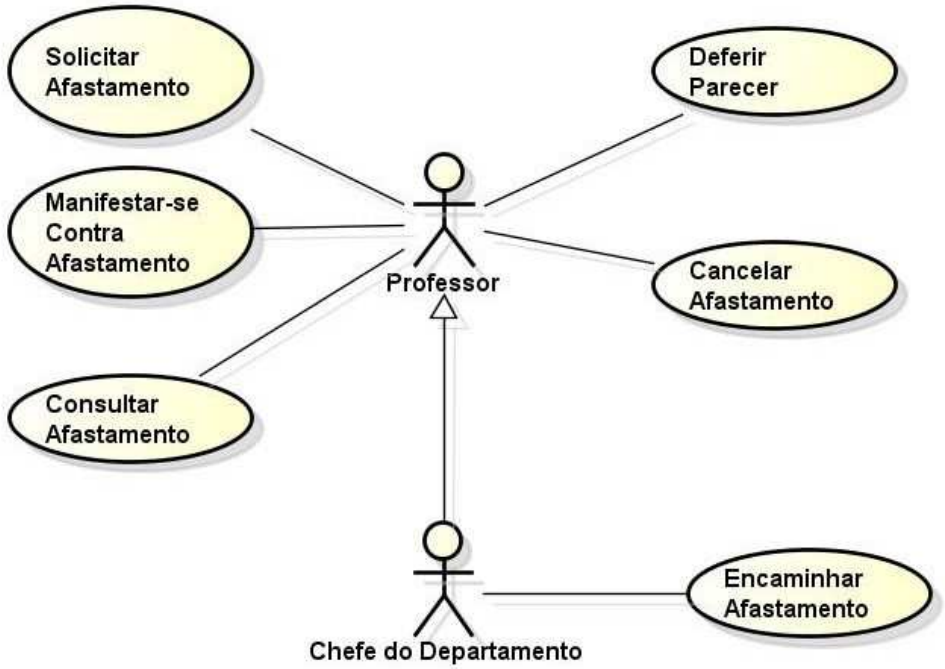

Figura 7 – Diagrama de Casos de Uso do subsistema Núcleo (PRADO, 2015).

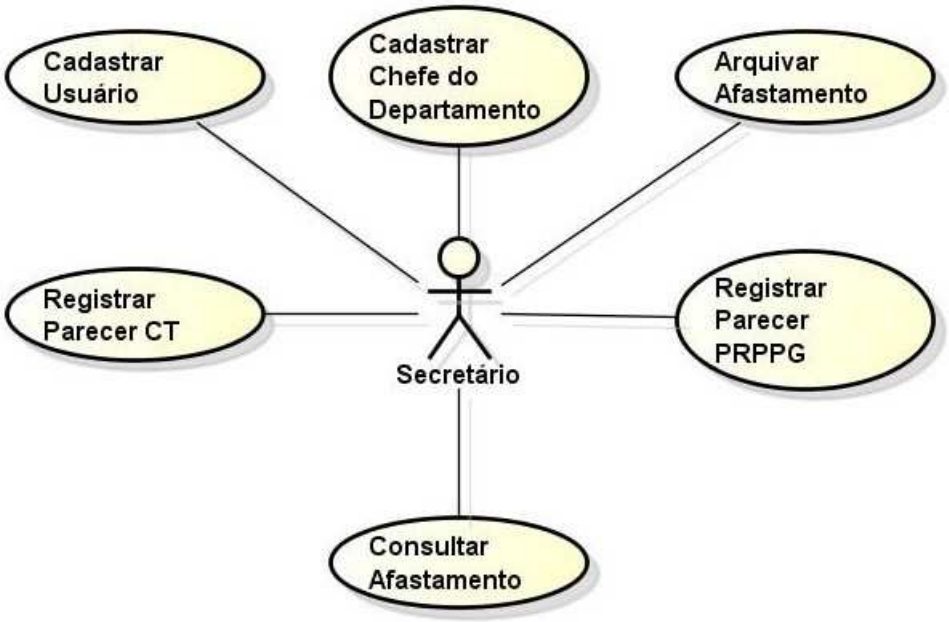

Figura 8 – Diagrama de Casos de Uso do subsistema Secretário (PRADO, 2015).

das solicitações internacionais.

4 Projeto e implementação

Nesta fase de projeto o objetivo é apresentar o que foi desenvolvido, a solução proposta para o problema identificado e modelado no levantamento de requisitos. Para isso se faz necessário ter, além do conhecimento do domínio, também ter o conhecimento das tecnologias que serão utilizadas e a arquitetura a ser utilizada (FALBO, 2017b).

É iniciada após a fase de levantamentos de requisitos, que pressupõe que a tecnologia é perfeita, quando na realidade não é. Por isso é necessário levar em conta que os requisitos de caráter não funcional (PRESSMAN, 2011).

Nas seções seguintes será descrito como o SCAP foi construído. Na Seção 4.1 será apresentada a arquitetura do sistema, bem como todas as tecnologias envolvidas. Já na Seção 4.2 será explicado como foi organizado o sistema. Adiante, na Seção 4.4 serão apresentados os modelos do FrameWeb criados.

4.1 Arquitetura do sistema

O SCAP é composto pela junção de dois subsistemas, um destinado para o lado do servidor e outro para o lado do cliente. No subsistema referente ao lado do servidor, que foi desenvolvido para ser uma API, é usado o Laravel, que por sua vez é feito em cima do PHP. O *framework* ORM (*Object/Relational Mapping*) usado foi o Eloquent. Já do lado do cliente é usado como base o *framework* Quasar, que utiliza o Vue.js, que é feito em cima do JavaScript.

Para esse projeto foi utilizado o sistema de gerenciamento de banco de dados MySQL. A interação com ele é inteiramente responsabilidade do subsistema do lado do servidor, uma ferramenta chamada *artisan* faz criações, alterações e exclusões na estrutura do banco de dados, por meio de uma série de classes que possuem comandos de manipulação dessa estrutura. Já para a manipulação de dados o Eloquent é usado.

Vale ainda citar outras tecnologias que também foram utilizadas, ainda que em menor quantidade:

- HTML: linguagem de marcação de hipertexto utilizada pelos navegadores;
- CSS: mecanismo para adicionar estilo (cores, fontes, espaçamento, etc.) a um documento Web;
- Stylus: preprocessador de CSS, open source;
- Node.js: Javascript rodando no servidor, open source;

- Git: sistema de controle de versão distribuído e gerenciamento de código fonte;
- Visual Studio Code: IDE open source;
- Docker: fornece contêineres, uma camada adicional de abstração e automação de virtualização do sistema operacional;
- Composer: gerenciador de pacotes PHP;
- NPM: gerenciador de pacotes Node.js;
- entre outros.

4.2 Organização do Projeto

Tendo em vista que foram desenvolvidos dois subsistema com propostas bem distintas e tecnologias diferentes, cabe a explicação dos *frameworks* utilizados em cada um deles separadamente.

4.2.1 Lado do servidor

Para o construção desse subsistema foi utilizado o Laravel na versão 5.4 e o PHP na versão 7.1, ambas as mais recentes até a conclusão do desenvolvimento. Toda a lógica de negócio do sistema como um todo deve estar presente nesse subsistema, o objetivo dele é prover uma interface de comunicação para demais sistemas, por isso ele foi construído para ser uma API. Todas as interações com ele retornaram uma *string* em formato JSON em vez de uma página HTML com folhas de estilo, arquivos binários etc. Essa é a única diferença entre a arquitetura *Service Layer* proposta pelo (FOWLER, 2002), apresentada na Figura 6, na Seção 2.3.

O subsistema é organizado conforme a Figura 9, onde é mostrada a organização de pastas. A pasta *app* é onde fica toda a lógica de negócio e é, consequentemente, onde o maior esforço foi despendido. A estrutura dela pode ser vista na Figura 10. Os arquivos que estão logo na raiz são os modelos, assim como proposto no padrão arquitetural MVC. Já na pasta *Enums* são representados os tipos enumerados de dados, que são classes auxiliares que tem como objetivo facilitar a interação com os valores desse tipo de dados. Além de fazerem a tradução do nome de máquina usado no banco de dados para o nome a ser exibido na interface do usuário, também possuem várias operações de recuperação.

Visto que o Laravel não precisa necessariamente ser utilizado nos navegadores, a existência da pasta *Http* se faz necessária para acomodar as classes que somente serão utilizadas estivermos construindo uma aplicação que será utilizada em navegadores. A estrutura desta pasta pode ser vista na Figura 11.

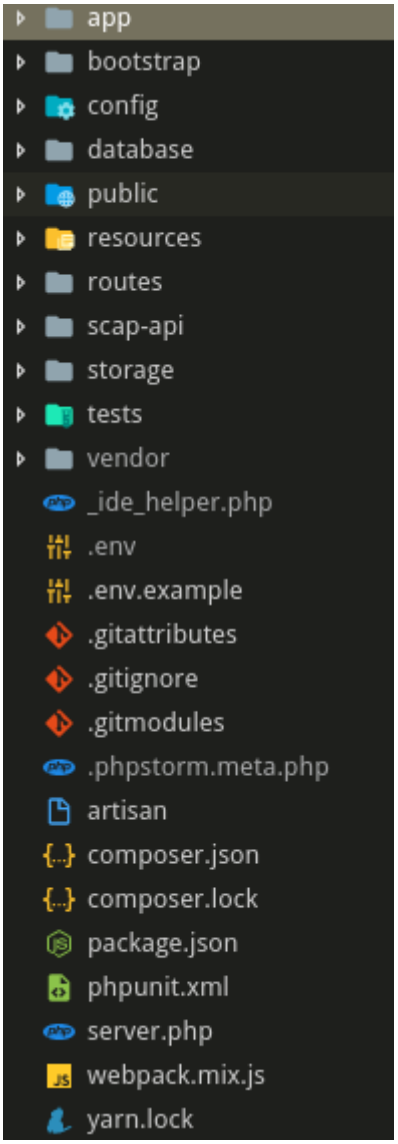

Figura 9 – Estrutura de pastas do subsistema do lado do servidor.

Nesta pasta *Http* teremos os *Controllers*, que são os controladores propostos no MVC; *Middlewares*, que são classes chamadas antes do código principal ser executado, normalmente para autorização; e *Requests* que são as classes responsáveis pela validação das requisições HTTP. Já uma aplicação que não utiliza o navegador poderia usar linhas de comando para executar o que for necessário, tendo assim como sua interface o terminal.

A pasta *Jobs* possui as classes da camada de serviço e comumente são invocadas pelos controladores. Cada ação é representada numa classe distinta, isso porque essas ações podem ser enfileiradas e serem executadas paralelamente. Porém, para este projeto não utilizamos essa funcionalidade. Como pode ser visto na Figura 12, temos um agrupamento de classes similares, a fim de organização.

Mais adiante temos a pasta *Repositories*, onde ficam todas as classes intituladas repositórios, responsáveis pela camada de persistência. Todas as interações com o banco de

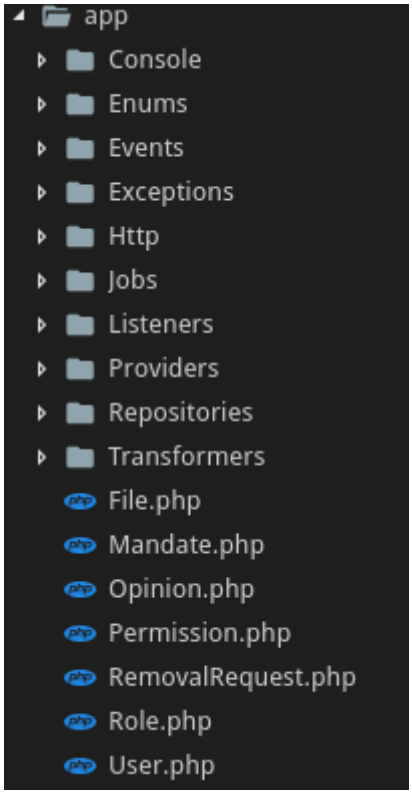

Figura 10 – Estrutura da pasta *app* do subsistema do lado do servidor.

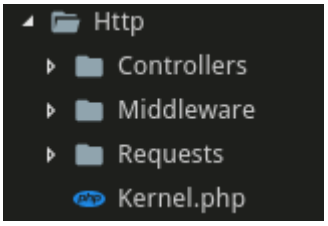

Figura 11 – Estrutura da pasta *app/Http* do subsistema do lado do servidor.

dados são feitas através dos métodos dessas classes. Para esse trabalho, nessa camada de persistência, foi utilizado o *framework* ORM *Eloquent*. Na Figura 13 é possível ver como isso foi organizado.

Outra pasta encontrada é *Transformers*, onde ficam todas as classes intituladas transformadoras, responsáveis pela apresentação da resposta para o usuário. Dada uma entidade do domínio a classes transformadora irá convertê-la para o formato JSON. Apenas é apresentado o que faz sentido para o cliente, como por exemplo um campo booleano que é representado no banco de dados como um *TINYINT* será convertido para `true` ou `false`. Datas, tipos enumerados, campos números e quaisquer outros campos podem ser traduzidos.

Por fim, temos algumas pastas em que não tivemos alterações em relação ao projeto base do Laravel. Na pasta *Console* podemos criar classes que representam comandos, que pode ser chamados diretamente do terminal ou por um agendador de tarefas.

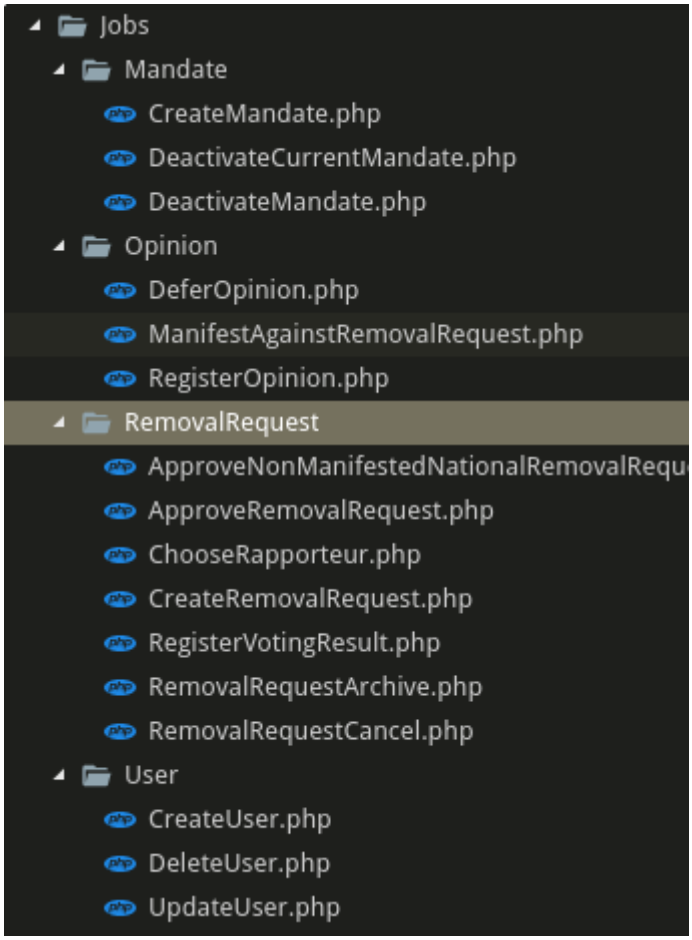

Figura 12 – Estrutura da pasta *app/Jobs* do subsistema do lado do servidor.

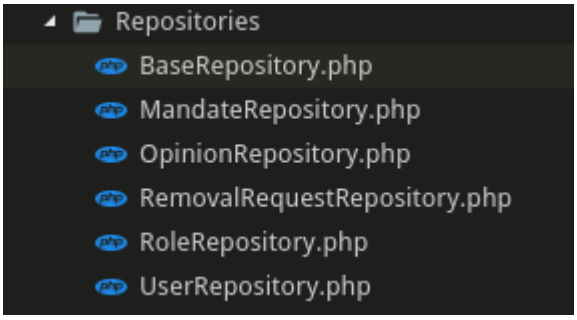

Figura 13 – Estrutura da pasta *app/Repositories* do subsistema do lado do servidor.

Já as pastas *Events* (eventos) e *Listeners* (ouvintes) são destinadas a implementação de simples observadores. A partir do momento que uma classe de evento é registrada e com ela suas respectivas classes de ouvintes, pode-se disparar esse evento de qualquer lugar da aplicação, garantindo que todos os ouvintes sejam executados. É muito interessante quando temos uma sequência de execuções que precisam ser feitas em série.

Seguindo, temos a pasta *Exceptions*, local onde podem ser criadas exceções personalizadas, não presentes de antemão no *framework*. Para concluir temos a pasta *Provider*, que contém classes que são executadas uma única vez na inicialização do *framework*.

Voltando à raiz do projeto, além da pasta *app* temos: a pasta *bootstrap*, que é utilizada pelo *framework*; a pasta *config*, que tem as configurações dos pacotes que são reaproveitados; a pasta *database*, que contém principalmente as *migrations*, uma maneira de se versionar os comandos usados no banco de dados como a criação e alteração do esquema; a pasta *public*, que é onde servidor Web tem acesso e por isso contem tudo que, no caso de uma aplicação que será utilada no navegador, ele pode requerer; a pasta *resources*, que contém os arquivos responsáveis pela visão, que não foram utilizados nesse projeto, e demais arquivos destinados a interface com o usuário; a pasta *routes*, que contem o mapeamento de todas as rotas disponíveis no subsistema e será explicada melhor mais adiante; a pasta *scap-api*, que contem tudo relacionado ao Docker; a pasta *storage*, que contem arquivos utilizados pelo *framework* como *cache* e *log*; a pasta *tests*, que contem todos os testes automatizados que foram feitos para o subsistema; e, por fim, a pasta *vendor*, que contem todos pacotes, gerenciados pelo *Composer*, que são requeridos para o funcionamento do subsistema.

As rotas nada mais são do que o mapeamento entre as requisições HTTP que podem ser feitas no subsistema e o método do controlador que será responsável para dar a resposta a essa requisição. Para esse projeto, foi usado o estilo arquitetural REST (FIELDING, 2000) e, sendo assim, precisamos informar o método do HTTP para cada uma das rotas criadas, podendo ser *POST* para criar um recurso; *GET* para ler um recurso; *PUT* para atualizar um recurso; *PATCH* para atualizar parte de um recurso; e *DELETE* para excluir um recurso.

4.2.2 Lado do cliente

O motivo da criação de duas aplicações separadas é a disponibilização de uma aplicação rica para o usuário final, com todas as características de uma RIA (*Rich Internet Applications*) como descrito em (CASTELEYN et al., 2009). Para isso essa aplicação foi desenvolvida para ser um SPA (*Single-Page Applications*) e PWA (*Progressive Web App*).

Para a criação desse subsistema foi utilizado como linguagem de programação principal o JavaScript. Além disso também foi utilizado o Vue.js e o Quasar. Para criar a identidade visual foi usado o Stylus, que ao ser compilado resulta em CSS.

Apesar do subsistema ter sido escrito em JavaScript, o código fonte final não é disponibilizado para o cliente. Com o apoio do Node.js ele é compilado e como resultado temos apenas um arquivo, que é disponibilizado para o navegador. Nesse arquivo contem toda a aplicação, que a partir do momento que é carregada no navegador, não precisa mais fazer nenhuma requisição, caracterizando um SPA.

Além disso, usando o Quasar, temos uma aplicação adaptada para aplicativos móveis. Ela é um PWA, na mesma aplicação atinge usuário que usam computadores de

mesa, notebooks e smartphones. Se ainda for necessário, podemos envelopá-lo em um aplicativo, fazendo com que o mesmo código fonte possa ser usado em aplicativos Android e iOS, através de uma *WebView*[1,2].

O subsistema é organizado conforme a Figura 14, onde é mostrada a organização de pastas. Na pasta *src*, abreviação de *source*, fica o código fonte desenvolvido para esse subsistema, como o nome já sugere. Os arquivos *main.js* e *App.vue* são responsável pela inicialização de toda aplicação, neles que fazemos as inclusões dos demais arquivos. Já o arquivo *router.js* é responsável pelo sistema de rotas, em que, para cada caminho passado, faremos o carregamento de um componente diferente. O componentes que são carregados pelas rotas, e que são destinado a estrutura da página são chamados de *views*.

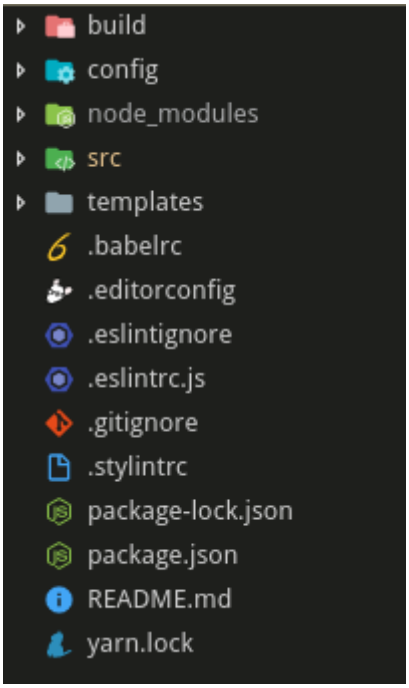

Figura 14 – Organização de pastas do subsistema do lado do cliente.

Na Figura 15 pode ser vista a estrutura dessa pasta e mais adiante a explicação de cada uma delas.

Já na Figura 16 é apresentado como foram organizadas as *views*. No primeiro nível temos as pastas *admin* e *auth*, separadas assim para diferenciar a área administrativa (usuário já autenticado) da área de autenticação (usuário ainda não autenticado). Dentro de cada uma delas, na raiz, há os arquivos que serão utilizados para dar estrutura básica da página, evitando repetição de código. Já as pastas agrupam os arquivos destinados para cada conjunto de funcionalidade, como por exemplo a pasta *src/views/admin/removal-request*, que terá somente as *views* das solicitações de afastamento da área administrativa.

[1] <https://developer.android.com/reference/android/webkit/WebView.html>
[2] <https://developer.apple.com/documentation/uikit/uiwebview>

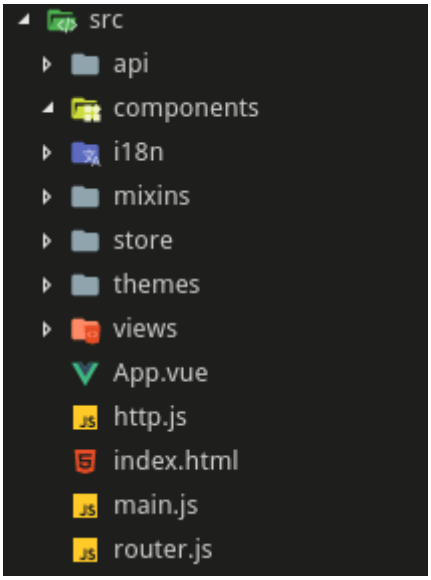

Figura 15 – Estrutura da pasta *src* do subsistema do lado do cliente.

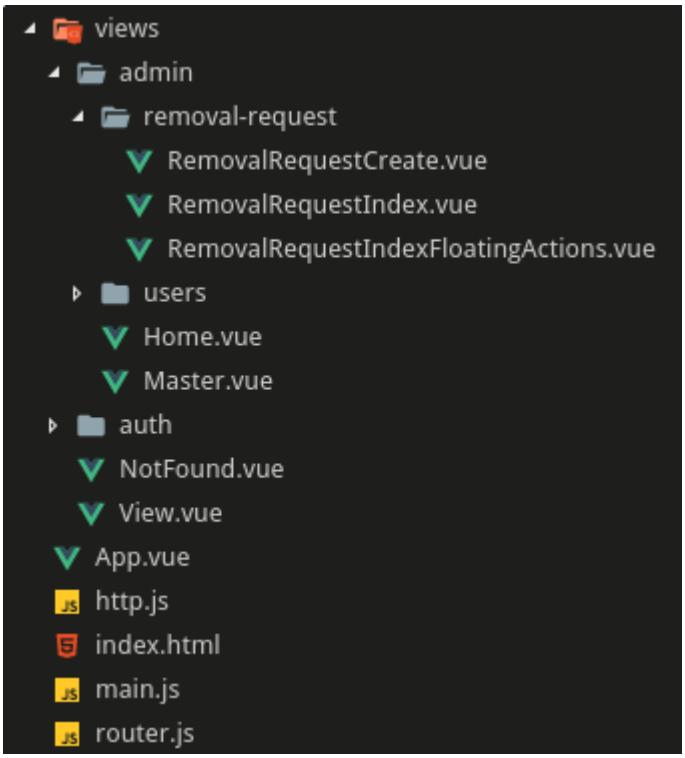

Figura 16 – Estrutura da pasta *src/views* do subsistema do lado do cliente.

Como toda a comunicação com o subsistema do lado do servidor é feita por meio de requisições HTTP, foi criada uma pasta para fazer a abstração de todos os recursos que podem ser requisitados. É na pasta *src/api* que é feito o mapeamento, com uma função específica para cada recurso desejado. A estrutura dessa pasta pode ser vista na Figura 17.

É na pasta *src/store*, como pode ser visto também na Figura 17, que é feito o gerenciamento do estado da aplicação. Já na raiz podemos ver os arquivos *state*, *mutations* e *actions*. A mesma ideia é seguida nos módulos, representados pelas diferentes pastas.

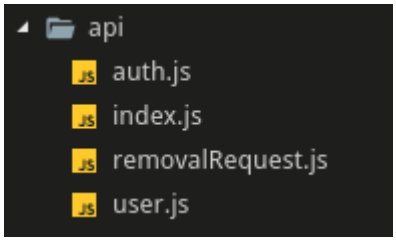

Figura 17 – Estrutura da pasta *src/api* do subsistema do lado do cliente.

As *actions* são responsável por boa parte da lógica de negócio e controle de fluxo dessa aplicação.

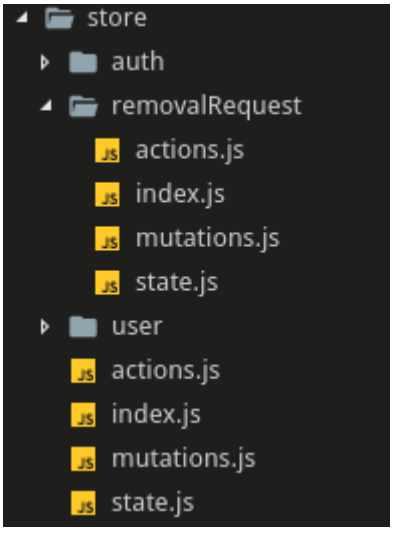

Figura 18 – Estrutura da pasta *src/store* do subsistema do lado do cliente.

Por fim, na raiz do subsistema, temos a pasta *build*, contendo arquivos responsáveis pela compilação; a pasta *config*, contendo arquivos de configuração dos ambientes de desenvolvimento e produção; a pasta *node_modules* que é onde fica todos os pacotes necessários para rodar a aplicação; e a pasta *templates*, que contém alguns exemplos de como as páginas podem ser construídas. Todas elas não foram alteradas em relação à instalação base do Quasar.

4.3 Apresentação dos resultados

Nessa seção apresentamos capturas de tela do protótipo desenvolvido. Como a mesma implementação gera resultados diferente de acordo com o dispositivo que o usuário estiver usando, serão levados em consideração dois tipos de clientes (com tamanhos diferentes de tela): dispositivos móveis (*smartphones*) e computadores de mesa (*desktop*).

A Figura 19 apresenta a tela de autenticação, primeira interface apresentada ao usuário.

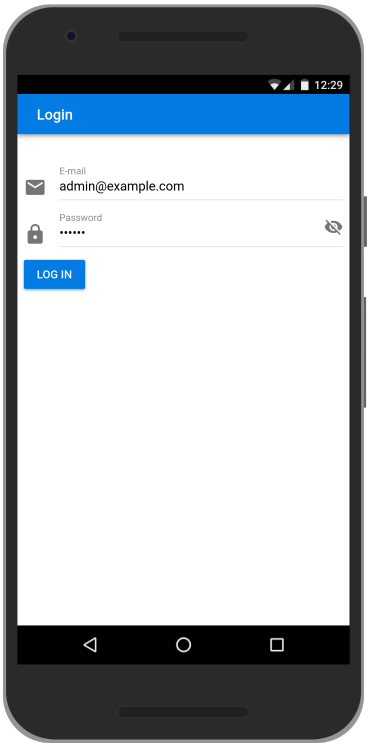

Figura 19 – Tela de login - Dispositivos móveis.

Após o login ao clicar no botão no canto superior esquerdo vemos as opções disponíveis no menu. O resultado é apresentado na Figura 20.

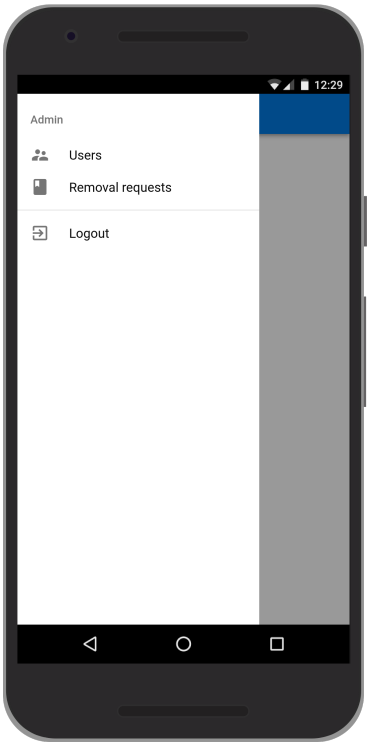

Figura 20 – Tela do menu - Dispositivos móveis.

Através do menu é possível navegar até a tela que lista os afastamentos e a tela

que lista os usuários, que são apresentadas nas figuras 21 e 22, respectivamente. É possível ver que essas telas possuem um botão com o sinal de mais (+) no canto inferior direito. Ao clicar nesse botão o usuário irá para a tela de criação da entidade em questão.

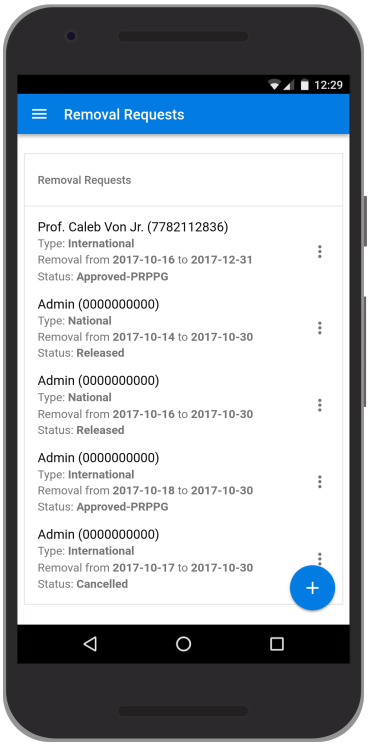

Figura 21 – Tela que lista todos afastamentos - Dispositivos móveis.

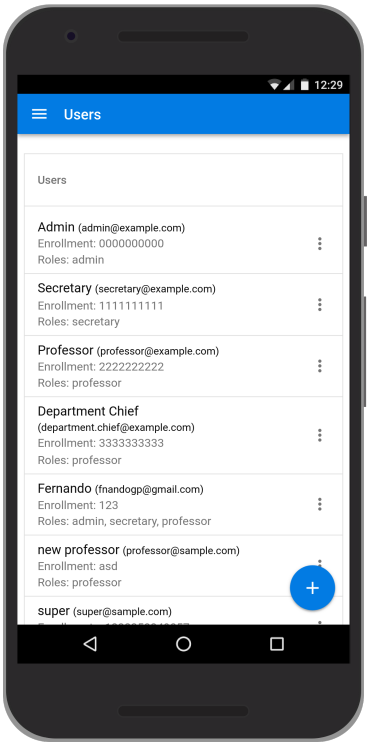

Figura 22 – Tela que lista todos usuários - Dispositivos móveis.

As telas que possuem os formulários de criação dessas entidades podem ser vistos nas figuras 23 e 24.

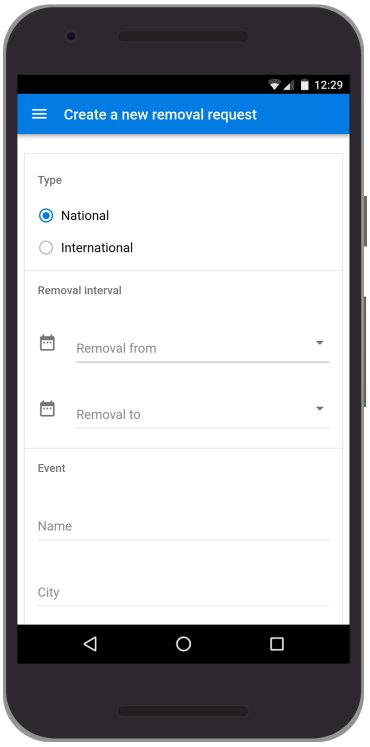

Figura 23 – Tela que cria um novo afastamento - Dispositivos móveis.

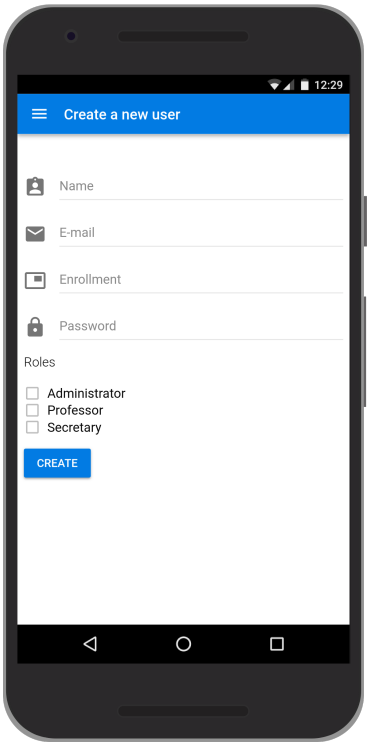

Figura 24 – Tela que cria um novo usuário - Dispositivos móveis.

Por fim temos as ações que podem ser feitas em cada uma das solicitações de afastamento. Para cada uma delas temos apresentadas as ações disponíveis na tela que

lista todos os afastamentos, em um botão que são três pontos alinhados verticalmente. Na Figura 25 podemos ver a lista de ações e na Figura 26 vemos o formulário que abre para a conclusão dessa ação.

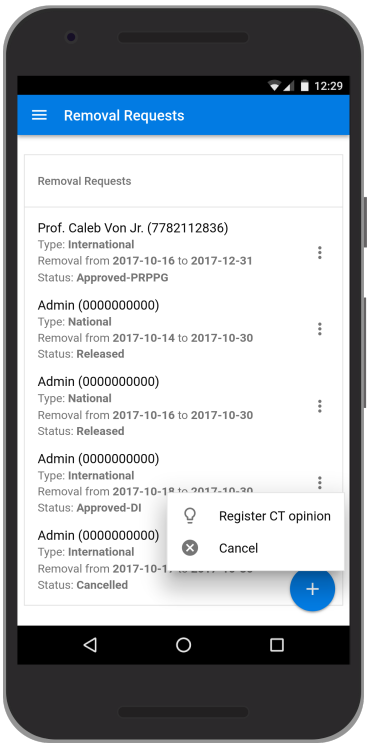

Figura 25 – Tela que lista ações disponíveis para uma solicitação de afastamento - Dispositivos móveis.

Nas telas para computadores de mesa temos os resultados parecidos com os que foram apresentados para os dispositivos móveis. A diferença mais evidente é o menu que não é retrátil, ele está sempre sendo exibido. Podemos ver isso na Figura 27. Já a Figura 28 vemos a diferença em relação à Figura 26.

4.4 Criação dos modelos FrameWeb

Tendo em mãos toda a implementação do projeto, o passo seguinte foi construir os modelos de domínio, persistência, navegação e aplicação. Como o sistema é dividido em dois subsistemas em cada um dos modelos, que serão apresentados a seguir, será indicada qual é a relação com cada um dos subsistemas.

4.4.1 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. A partir dele são implementadas as classes da camada de Domínio na atividade de implementação (SOUZA, 2007).

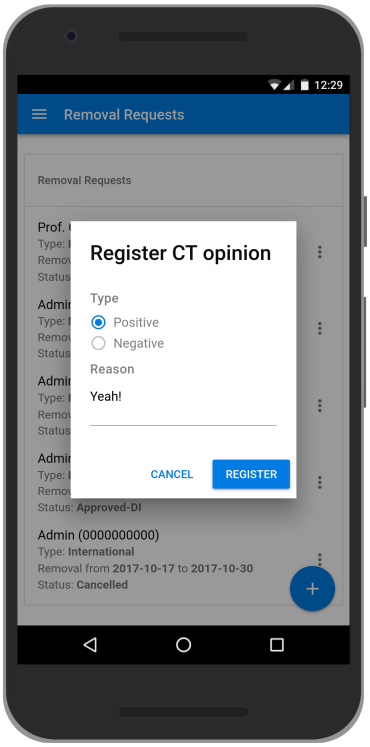

Figura 26 – Tela que exibe o formulário referente a ação escolhida para uma solicitação de afastamento - Dispositivos móveis.

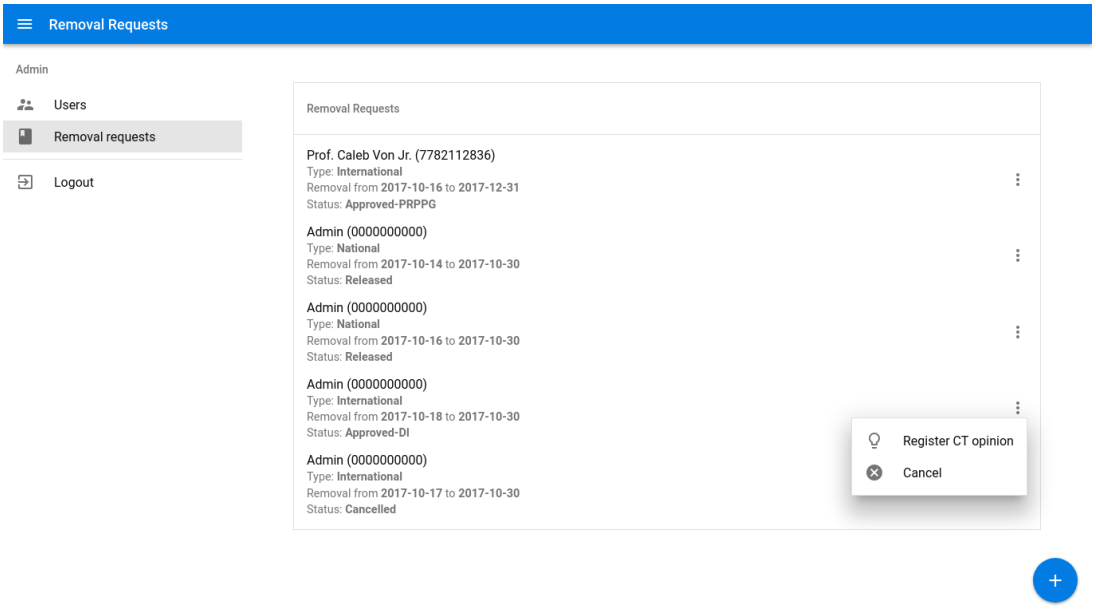

Figura 27 – Tela que lista ações disponíveis para uma solicitação de afastamento - Computadores de mesa.

A Figura 29 representa o modelo de domínio. Já na Figura 30 são apresentados os tipos enumerados já utilizados no modelo de domínio.

É necessário olhar apenas para o subsistema do lado do servidor para criar esse diagrama e dois lugares merecem atenção especial. O primeiro deles é a pasta de *database/migrations*, que contém todas as instruções usadas para a criação do banco de

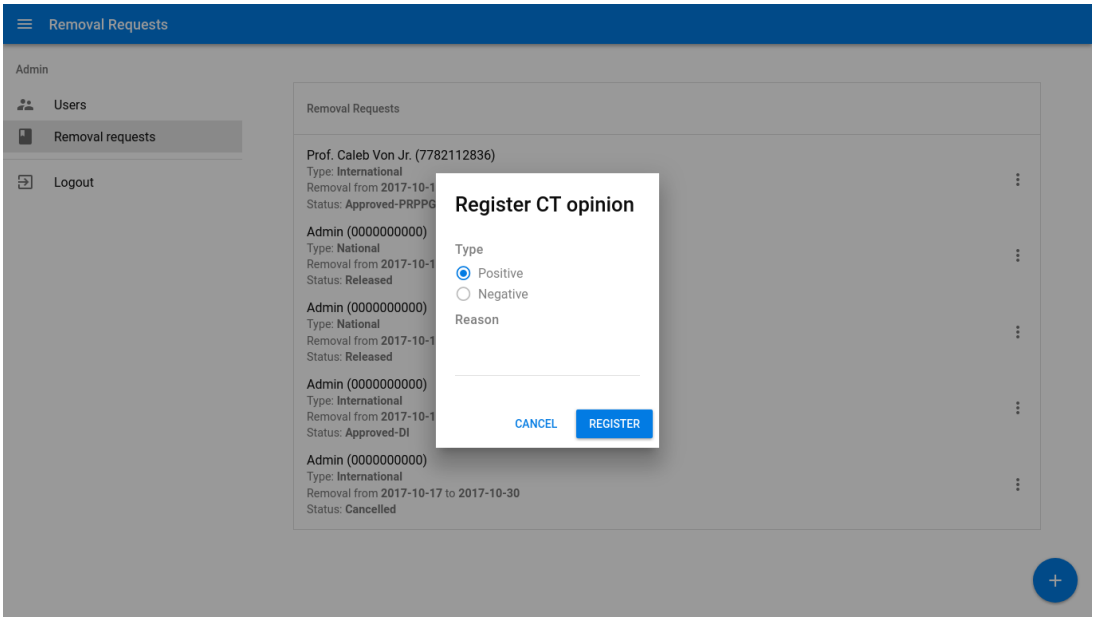

Figura 28 – Tela que lista ações disponíveis para uma solicitação de afastamento - Computadores de mesa.

dados.

O segundo é a pasta *app*, que contém as classes responsáveis pelo modelo logo na raiz. No modelo, caso não seja seguida a convenção do *framework*, informamos qual é a tabela que ele irá usar e, com isso, sabemos exatamente todas as variáveis disponíveis, que possuem o mesmo nome das colunas da tabela. Também é no modelo que informamos e temos acesso os relacionamentos com os demais modelos.

4.4.2 Modelo de Persistência

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 Persistência (SOUZA, 2007). A Figura 31 representa o modelo de persistência do SCAP.

É no subsistema do lado do servidor podemos identificar as classes responsáveis para a construção desse modelo. Na pasta *app/Respositories* temos acesso a todos os repositórios e em cada uma das classes podemos ver os métodos usados na implementação do subsistema.

Tendo isso em vista, para criar o modelo começamos pela a criação das interfaces dos DAOs e, logo após, a criação da implementação concreta dos DAOs com cada um dos métodos presentes na classe.

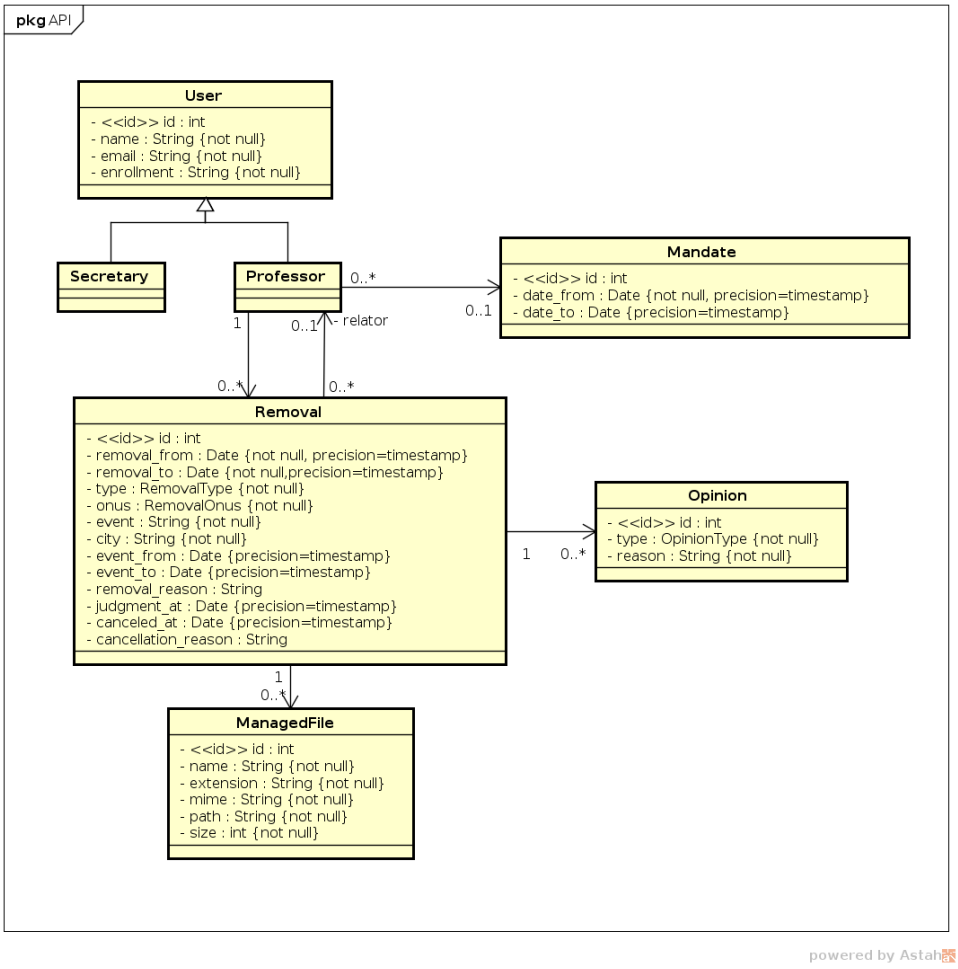

Figura 29 – Modelo de domínio do SCAP.

4.4.3 Modelo de Navegação

O Modelo de Navegação é um diagrama de classes da UML que representa os diferentes componentes que formam a camada de Lógica de Apresentação, como páginas Web, formulários HTML e classes de ação do *framework Front Controller*. Esse modelo é utilizado pelos desenvolvedores para guiar a codificação das classes e componentes dos pacotes Visão e Controle (SOUZA, 2007).

Sabendo que para a construção desse modelo foi necessário consultar o subsistema do lado do cliente, e que a única maneira que ele tem de comunicar com o subsistema do lado do servidor é através de requisições HTTP, foi necessário fazer uma adaptação. Como cada requisição HTTP é representada por uma rota no subsistema do lado do servidor, em vez de representar essa rota no Modelo de Navegação, utilizamos o método que é chamando quando essa rota é acionada. Para que isso fique mais claro basta vermos abaixo como uma rota é registrada.

```
1 Route::post('/removal-requests', '
    RemovalRequestController@store')
```

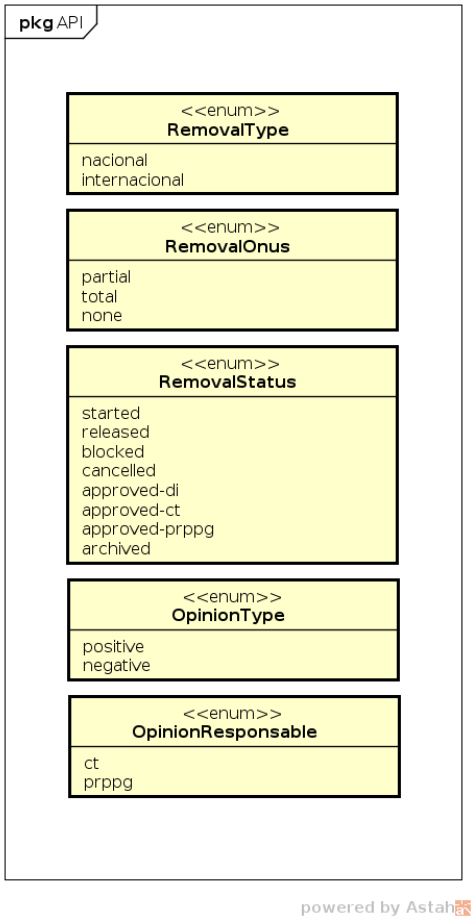

Figura 30 – Tipos enumerados do SCAP.

```
2        ->name('removal-request.create');
```

Podemos ler esse código da seguinte maneira: ao receber uma requisição HTTP com o método *POST* no endereço `/removal-requests`, o método *store* do controlador *RemovalRequestController* é chamado. A resposta dessa requisição HTTP é o retorno do método do controlador. Para fins de organização as rotas são identificada com nomes e no caso acima o nome dado foi *removal-request.create*.

Como exemplos, nas figuras 32 e 33 são exibidos os modelo de navegação referentes aos casos de uso "Solicitar Afastamento" e "Encaminhar Afastamento", respectivamente.

4.4.4 Modelo de Aplicação

O Modelo de Aplicação é 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 controladores dependem de quais classes de serviço e quais DAOs são necessários para que as classes de serviço alcancem seus objetivos (SOUZA, 2007). A Figura 34 apresenta o

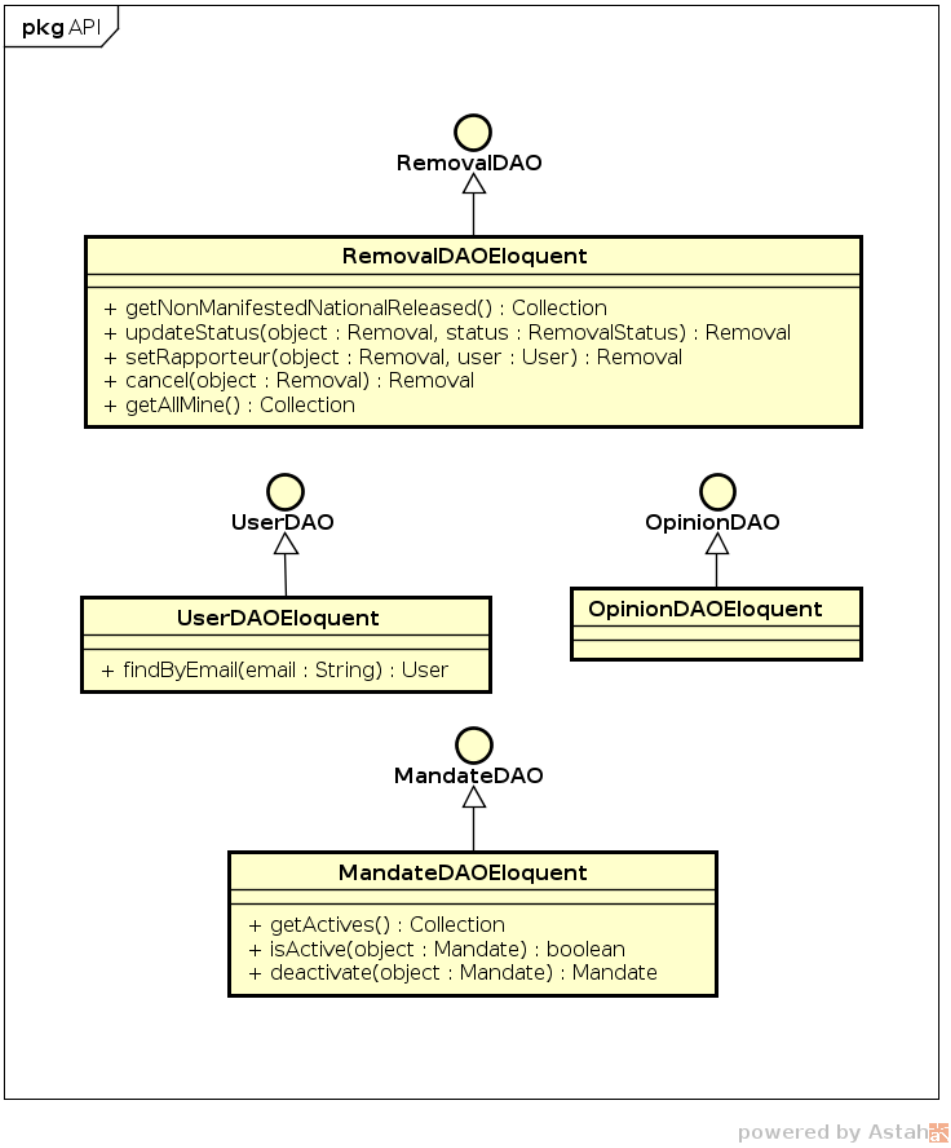

Figura 31 – Modelo de persistência do SCAP

modelo de aplicação.

Para a construção desse modelo os seguintes passos foram seguidos:

- Criação da interfaces das classes que implementam a lógica de negócio do sistema;
- Importação dos controladores;
- Identificação de cada um dos métodos utilizados na implementação do projeto e adição na classes que implementa a interface de aplicação.

Assim como os modelos de domínio e persistência, para a construção desse modelo foi necessário consultar o subsistema do lado do servidor.

Nesse caso, por conta de uma decisão de projeto, como explicado sobre os *Jobs* na Seção 4.2.1, foi representada uma única classe com todos os métodos. A outra opção

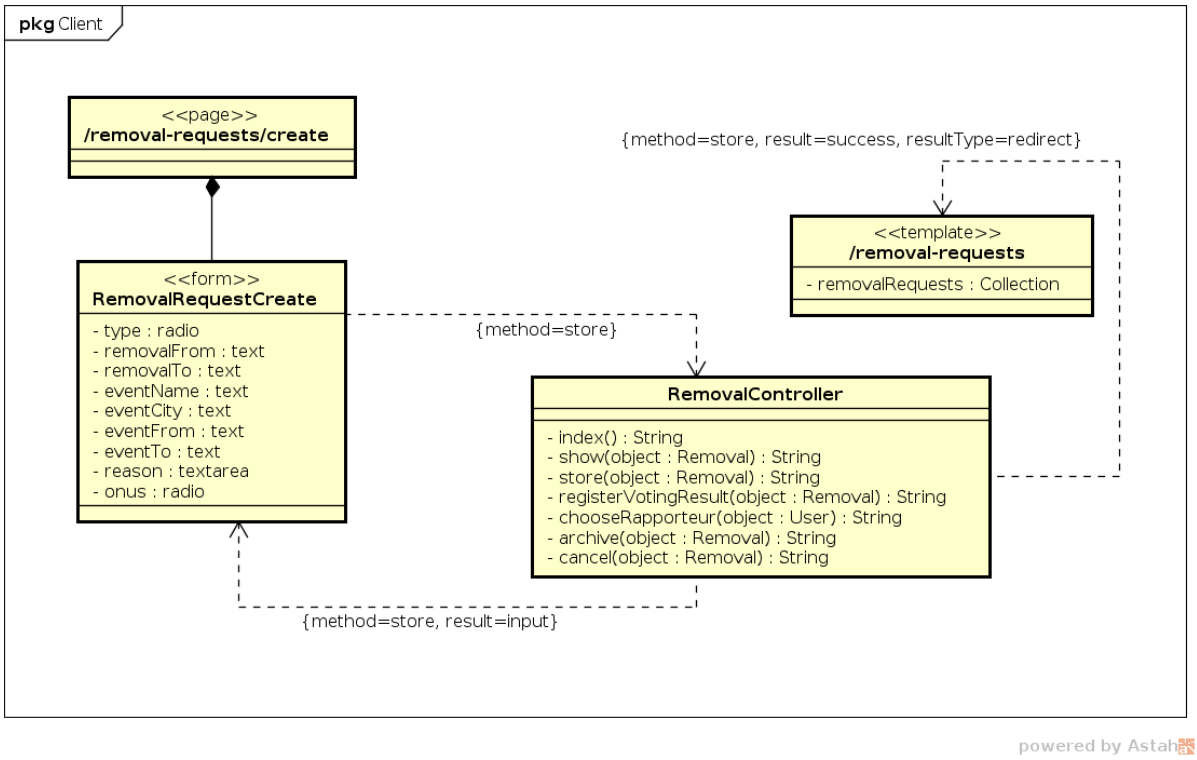

Figura 32 – Modelo de navegação do SCAP - Criação de uma solicitação de afastamento.

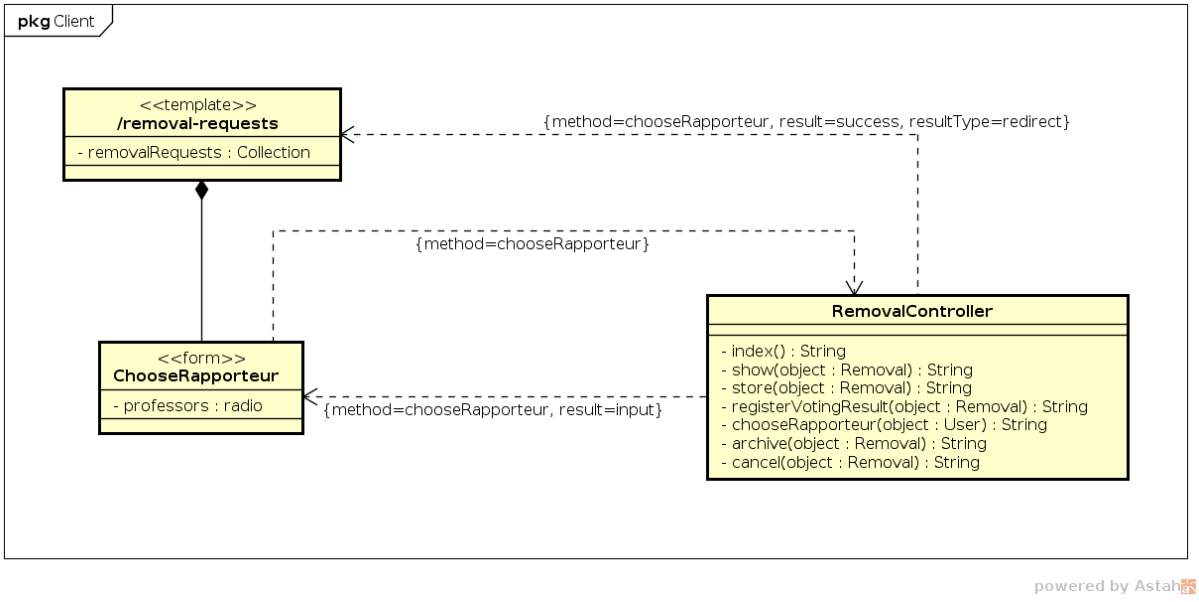

Figura 33 – Modelo de navegação do SCAP - Escolha de um relator para uma solicitação de afastamento.

seria representar várias classes com apenas um método, porém isso poluiria o modelo e dificultaria a leitura. Para a construção do diagrama isso foi abstraído, o que gerou somente uma classe de aplicação com vários métodos.

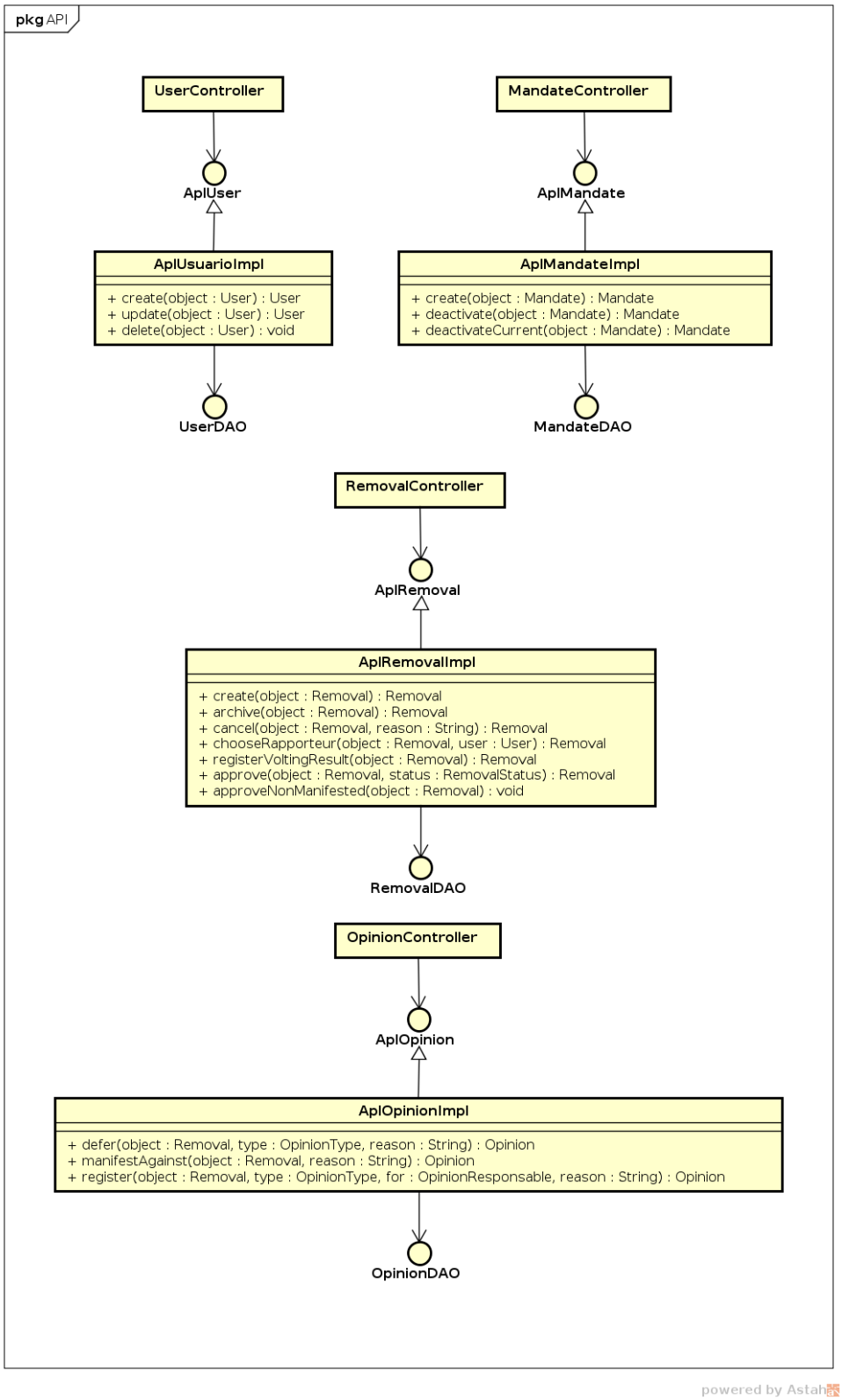

Figura 34 – Modelo de aplicação do SCAP.

4.5 Conclusões do capítulo

Nesse capítulo, é apresentada a arquitetura do projeto SCAP e como ele foi organizado. Como o projeto de implementação foi divido em dois subsistemas distintos,

um para o lado do servidor contendo a lógica de negócio e outro para o lado do cliente contendo a apresentação para o usuário, foram apresentados cada um dos *frameworks* utilizados e explicados os principais conceitos usados por cada um deles. A estrutura de pastas foi exibida em figuras para melhor visualização e entendimento.

A implementação do projeto foi feita com base em duas documentações de requisitos escritas anteriormente (DUARTE, 2014; PRADO, 2015), seguida da criação dos modelos FrameWeb. Para esse projeto o fluxo foi o inverso do que é proposto pelo FrameWeb, que prescreve primeiro construir os modelos para depois fazer a implementação, usando os modelos construídos como base.

Com exceção de um, a construção dos modelos foi feita de forma bem direta, consultando o código e traduzindo para o modelo. O trabalho maior foi simplesmente de montagem, pouco precisou ser pensado e arquitetado.

O modelo que não se encaixou muito bem foi o de Navegação, responsável pela lógica de apresentação. As páginas e formulários precisam interagir com uma classe de ação e como no subsistema do lado do cliente todas as interações são feitas através de requisições HTTP isso não foi possível de ser representado. Sendo assim eu o construi inferindo quais métodos da classe de ação eram chamados em cada uma das requisições que de fato eram feitas.

Nesse trabalho os dois subsistemas foram feitos pela mesma pessoa e o conhecimento de como cada coisa foi implementada podia ser obtido de maneira fácil. Hoje é comum vermos sistemas que se comunicam com outros sistemas, muitas vezes utilizando uma API fornecida por um deles, por isso é possível ver que não é um problema que aconteceu somente nesse caso específico. A comunicação entre sistemas, feitas através de requisições HTTP, poderia ser representável pelo modelo FrameWeb.

O próximo capítulo apresenta as conclusões deste trabalho.

5 Considerações Finais

Nesse capítulo serão apresentadas todas as considerações finais relacionadas ao trabalho feito, incluindo as conclusões, limitações e trabalhos futuros.

5.1 Conclusão

Sabendo que a ideia do trabalho partir da utilização de uma nova linguaguem de programação e consequentemente novos *frameworks* no contexto do uso de método FrameWeb, foi planejado que a ordem que em as coisas fossem feitas seria contrária ao que é proposto no método. Sendo assim a implementação foi feita antes da criação dos modelos do FrameWeb, sem deixar de lado que o arcabouço do método era conhecido.

Por ter sido subdivido em duas partes, o sistema pode ser visto sobre duas óticas diferentes, dois subsistemas. O primeiro, o lado do servidor, foi desenvolvido sem muitos desafios, visto que a linguagem de programação PHP e o *framework* Laravel já era de meu domínio. O mesmo vale para a organização do projeto e os outros tipos de *frameworks* (ORM, MVC, Decoradores, Injeção de dependência) que foram considerados desde o começo. A maior parte do esforço foi despendido no entendimento do que precisava ser construído.

Para o segundo subsistema, o do lado do servidor, que foi desenvolvida com a linguagem de programação JavaScript e os *frameworks* Vue.js, Vuex e Quasar eu tive um esforço maior. Nesse caso, além da implementação tive que me atualizar sobre como funcionava o Vue.js, dado que eu só havia trabalhado com ele na versão 1 e a versão utilizada foi a 2. Foi a primeira vez que trabalhei de fato com o Quasar e o Vuex e por isso tive o custo de aprender como eles funcionam e quais são as boas práticas sugeridas. Foi nesse subsistema que passei a maior parte trabalhando.

Com o desenvolvimento concluído o passo seguinte era a construção dos modelos FrameWeb. Com exceção de um, a construção dos modelos foi feita de forma bem direta, consultando o código e traduzindo para o modelo. O trabalho maior foi simplesmente de montagem, pouco precisou ser pensado e arquitetado. Isso porque o sistema já foi construído para satisfazer essas necessidades.

O método FrameWeb me pareceu muito sólido em relação a como a organização do projeto é dada, são decisões muito importantes de serem tomadas antes da implementação. Os modelos vêm para complementar essa organização e deixar tudo mais visual e de fácil consulta. É uma construção robusta e com um abordagem conceitual bem aberta, tornando fácil a mudança de componentes do sistema. Um exemplo disso pode ser a troca do sistema

de gerenciamento de banco de dados do projeto, basta implementar uma nova interface para isso e o restante do sistema continua funcionando.

5.2 Limitações e Trabalhos Futuros

Como havia uma liberdade muito grande na implementação do sistema, mesmo sabendo previamente da utilização de alguns tipos de *frameworks* específicos, não houve problema algum na etapa de construção da aplicação como um todo.

A primeira dificuldade encontrada foi na etapa de construção dos modelos FrameWeb. O modelo que não se encaixou muito bem foi o de Navegação, responsável pela lógica de apresentação. As páginas e formulários precisam interagir com uma classe de ação e como no subsistema do lado do cliente todas as interações são feitas através de requisições HTTP isso não foi possível de ser representado. Sendo assim eu o construi inferindo quais métodos da classe de ação eram chamados em cada uma das requisições que de fato eram feitas.

Nesse trabalho os dois subsistemas foram feitos pela mesma pessoa e o conhecimento de como cada coisa foi implementada podia ser obtido de maneira fácil. Hoje é comum vermos sistemas que se comunicam com outros sistemas, muitas vezes utilizando uma API fornecida por um deles, por isso é possível ver que não é um problema que aconteceu somente nesse caso específico. A comunicação entre sistemas, feitas através de requisições HTTP, poderia ser representável pelo modelo FrameWeb.

Referências

ALUR, D.; CRUPI, J.; MALKS, D. *Core J2EE Patterns: Best Practices and Design Strategies*. 2^{nd}. ed. [S.l.]: Prentice Hall / Sun Microsystems Press, 2003. Citado na página 23.

BUSCHMANN, F. et al. *Pattern-Oriented Software Architecture - Volume 1: A System of Patterns*. [S.l.]: Wiley Publishing, 1996. ISBN 0471958697, 9780471958697. Citado na página 13.

CASTELEYN, S. et al. *Engineering Web Applications*. [S.l.]: Springer, 2009. Citado 5 vezes nas páginas 6, 15, 16, 17 e 34.

DEITEL, P.; DEITEL, H. *Java How to Program*. 9th. ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2011. ISBN 0132575663, 9780132575669. Citado na página 13.

DUARTE, B. B. *Aplicação do Método FrameWeb no Desenvolvimento de um Sistema de Informação na Plataforma Java EE 7*. Monografia (Projeto de Graduação) — Departamento de Informática, Universidade Federal do Espírito Santo, Vitória, ES, Brasil, 2014. Citado 4 vezes nas páginas 12, 13, 25 e 49.

ELMASRI, R.; NAVATHE, S. B. *Sistemas de banco de dados /*. 6. ed.. ed. São Paulo :: Pearson Education do Brasil„ 2011. Contém índice. Citado na página 13.

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 na página 26.

FALBO, R. d. A. *Projeto de Sistemas*. [s.n.], 2017. 135 p. Disponível em: <https://inf.ufes.br/~falbo/files/PSS/Notas_Aula_Projeto_Sistemas_2017.pdf>. Citado na página 29.

FIELDING, R. T. *Architectural Styles and the Design of Network-based Software Architectures*. Tese (Doutorado) — University of California, Irvine, 2000. Citado na página 34.

FOWLER, M. *Patterns of Enterprise Application Architecture*. [S.l.]: Addison-Wesley Professional, 2002. ISBN 0321127420. Citado 4 vezes nas páginas 6, 22, 23 e 30.

FOWLER, M. *Presentation Model*. 2004. Disponível em: <https://martinfowler.com/eaaDev/PresentationModel.html>. Citado na página 21.

GAMMA, E. et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994. ISBN 0201633612. Disponível em: <https://www.amazon.com/Design-Patterns-Elements-Reusable-Object-Oriented/dp/0201633612?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0201633612>. Citado na página 19.

JSON.ORG. *Introducing JSON*. 2017. Disponível em: <https://www.json.org>. Citado na página 19.

OLIVé, A. *Conceptual Modeling of Information Systems*. [S.l.]: Springer, 2007. ISBN 9783540393900. Citado na página 26.

PRADO, R. C. do. *Aplicação do método FrameWeb no desenvolvimento de um sistema de informação utilizando o framework VRaptor 4*. Monografia (Projeto de Graduação) — Departamento de Informática, Universidade Federal do Espírito Santo, Vitória, ES, Brasil, 2015. Citado 9 vezes nas páginas 6, 8, 12, 13, 25, 26, 27, 28 e 49.

PRESSMAN, R. S. *Engenharia de software*. [S.l.]: Makron books Sao Paulo, 2011. v. 7. Citado 2 vezes nas páginas 13 e 29.

PRESSMAN, R. S.; LOWE, D. *Web engineering: a practitioner's approach*. [S.l.]: McGraw-Hill Education, 2009. v. 1. Citado na página 15.

PROGRAMMINGHELP. *Fundamentals of an MVC Framework*. 2013. Disponível em: <http://www.programminghelp.com/mvc/fundamentals-mvc-framework>. Citado 2 vezes nas páginas 6 e 19.

SMITH, J. *Patterns - WPF Apps With The Model-View-ViewModel Design Pattern*. 2009. Disponível em: <https://msdn.microsoft.com/en-us/magazine/dd419663.aspx>. Citado na página 21.

SOUZA, V. E. S. *FrameWeb: um Método baseado em Frameworks para o Projeto de Sistemas de Informação Web*. Dissertação (Mestrado) — Programa de Pós-Graduação em Informática — Universidade Federal do Espírito Santo, 2007. Citado 9 vezes nas páginas 6, 12, 22, 23, 24, 41, 43, 44 e 45.

SOUZA, V. E. S.; FALBO, R. A.; GUIZZARDI, G. Designing Web Information Systems for a Framework-based Construction. In: HALPIN, T.; PROPER, E.; KROGSTIE, J. (Ed.). *Innovations in Information Systems Modeling: Methods and Best Practices*. 1. ed. [S.l.]: IGI Global, 2009. cap. 11, p. 203–237. Citado na página 12.

WEINBERGER, M. *The 15 most popular programming languages, according to the 'Facebook for programmers'*. 2017. Disponível em: <http://www.businessinsider.com/the-9-most-popular-programming-languages-according-to-the-facebook-for-programmers-2017-10>. Citado na página 18.