UNIVERSIDADE DO ESTADO DO RIO GRANDE DO NORTE
FACULDADE DE CIÊNCIAS EXATAS E NATURAIS
DEPARTAMENTO DE COMPUTAÇÃO
CURSO DE CIÊNCIA DA COMPUTAÇÃO

RAYSSA CATHARINA CUNHA DE MESQUITA

DESENVOLVIMENTO E INTEGRAÇÃO DO MÓDULO DE ATENDIMENTO PARA O SISTEMA DE CONTROLE DE VISITANTES, O *GUEST*

NATAL
2018

RAYSSA CATHARINA CUNHA DE MESQUITA

DESENVOLVIMENTO E INTEGRAÇÃO DO MÓDULO DE ATENDIMENTO PARA O SISTEMA DE CONTROLE DE VISITANTES, O *GUEST*

Monografia apresentada à Universidade do Estado do Rio Grande do Norte - UERN - como requisito obrigatório para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Dr. Alberto Signoretti

NATAL

2018

Catalogação da Publicação na Fonte.
Universidade do Estado do Rio Grande do Norte.

M582d Mesquita, Rayssa Catharina Cunha de
DESENVOLVIMENTO E INTEGRAÇÃO DO MÓDULO DE ATENDIMENTO PARA O SISTEMA DE CONTROLE DE VISITANTES, O GUEST. / Rayssa Catharina Cunha de Mesquita. - Universidade do Estado do Rio Grande do Norte, 2018.
54p.

Orientador(a): Prof. Dr. Alberto Signoretti.
Monografia (Graduação em Ciência de Computação). Universidade do Estado do Rio Grande do Norte.

1. Ciência de Computação. 2. Sistema WEB. 3. Desenvolvimento de software. 4. Java. 5. MVC. I. Signoretti, Alberto. II. Universidade do Estado do Rio Grande do Norte. III. Título.

O serviço de Geração Automática de Ficha Catalográfica para Trabalhos de Conclusão de Curso (TCC´s) foi desenvolvido pela Diretoria de Informatização (DINF), sob orientação dos bibliotecários do SIB-UERN, para ser adaptado às necessidades da comunidade acadêmica UERN.

RAYSSA CATHARINA CUNHA DE MESQUITA

DESENVOLVIMENTO E INTEGRAÇÃO DO MÓDULO DE ATENDIMENTO PARA O SISTEMA DE CONTROLE DE VISITANTES, O *GUEST.*

Monografia apresentada à Universidade do Estado do Rio Grande do Norte - UERN - como requisito obrigatório para obtenção do título de Bacharel em Ciência da Computação.

Aprovado em: ___/___/___.

Banca Examinadora:

Dr. Alberto Signoretti
UERN

Dr. Francisco Dantas de Medeiros Neto
UERN

Dr. Isaac de Lima Oliveira Filho
UERN

À Deus.

AGRADECIMENTOS

Primeiramente a Deus, eu jamais conseguiria chegar até aqui sem todo o cuidado que Ele tem comigo. Ao meu pai e minha mãe, vocês me deram a vida e todo o amor que precisei, vocês são os melhores. Obrigada pai, pela horas que gastou me ajudando. Obrigada mãe, por todos os investimentos e os incentivos. Obrigada Deus, por ter me dado pais tão maravilhosos.

Aos meus irmãos, pelo o apoio e os conselhos. Aos meus professores, sou grata por todo o conhecimento que compartilharam. A Justiça Federal do Rio Grande do Norte por liberarem o sistema para o desenvolvimento deste trabalho, principalmente a Daniel, Jackson, David e Diogenis. A Alan Martins pelo apoio e todas as dúvidas que você ajudou a entender Um obrigado ao meu noivo, por dizer “você consegue” em todos os momentos que eu duvidei de mim. E um agradecimento especial a Francisco Dantas, Isaac Lima e Alberto Signoretti pela oportunidade.

RESUMO

Este trabalho trata da criação e inclusão de um novo módulo ao sistema de controle de visitas da Justiça Federal do RN, com propósito de informatizar todos atendimentos e encaminhamentos ocorridos nos prédios da Justiça Federal do Rio Grande do Norte. Para isso, foi feito um levantamento de requisitos por meio de reuniões com um grupo selecionado de servidores responsáveis pela realização desses atendimentos. Com essas informações, foi desenvolvida a documentação do sistema e diagramas UML que guiaram todo o processo de criação. Foi utilizado a linguagem java, com alguns *frameworks* e padrões específicos dela, para alcançar os objetivos definidos. Entretanto, esses não puderam ser analisados em sua totalidade, pois o sistema ainda encontra-se em processo de adaptação na JFRN.

Palavras-chave: Sistema WEB. Desenvolvimento de software.Java, MVC, Spring.

ABSTRACT

This development is about the creation and inclusion of a new module to the Federal Justice of RN visit control system, with the purpose of computerizing all customer services occurring in Federal Justice buildings in Rio Grande do Norte. For this, a requirements survey was made through meetings with a selected group of employees responsible for performing these services. With this information, we developed the system documentation and UML diagrams that guided the entire creation process. It was used the language java, with some frameworks and specific standards of it, to reach the defined objectives. However, these could not be analyzed in their entirety, because the system is still in the process of adaptation in FJRN.

Keywords: Web System. Software Development.Java, MVC, Spring.

Lista de Ilustrações

Lista de Abreviaturas e Siglas

NTI	Núcleo de Tecnologia da Informação
JFRN	Justiça Federal do Rio Grande do Norte
CA	Central de Atendimento
UA	Unidade de Atendimento
JSF	JavaServer Faces
JPA	Java Persistence API
MVC	Model-View-Control
DTO	Data Transfer Object
SGBD	Sistema de Gerenciamento de Banco de Dados
BD	Banco de Dados
DAO	*Data Access Object*
RG	Registro Geral
RGE	Registro Geral Estrangeiro
MF	Matrícula Funcional
OAB	Ordem dos Advogados do Brasil
UF	Unidade Federativa

SUMÁRIO

1. Introdução

Para as instituições públicas e/ou privadas que possuem um sistema que controle a circulação de pessoas em seus prédios e setores agregados, se faz necessário a utilização de um sistema eficiente, rápido e confiável. Para isso, o mesmo deve operar na gestão do acesso do público às suas repartições, gerenciar os atendimentos, organizar os achados e perdidos, e outras funcionalidades específicas de cada instituição.

A Superintendência Regional da Polícia Federal no Rio Grande do Norte, o Tribunal de Justiça do RN (TJRN), o fórum Miguel Seabra (extensão do TJRN), o Ministério Público estadual e federal, a Advocacia-Geral da União (AGU) e a Justiça Federal do Rio Grande do Norte (JFRN) são alguns dos prédios públicos localizados em Natal que possuem sistemas especializados em controle de acessos. Todos eles tem um fato em comum, para adentrar qualquer setor diferente da recepção, é preciso estar cadastrado no sistema e explicitar o(s) local(is) aos quais se deseja visitar, para que este acesso seja registrado no sistema.

No caso do JFRN, ao adentrar a recepção, situada no prédio principal, de imediato percebe-se os atendentes recepcionistas aptos a realizar a comunicação entre o visitante e o sistema que faz estes procedimentos de controle, o *Guest*.

Este sistema controla todo o processo de entrada e saída das pessoas que visitam a JFRN, com exceção à servidores e estagiários. Por meio deste, as informações pessoais de um visitante são armazenadas, bem como, será criado um registro de visitas, guardando detalhes referentes aos horários de entrada e saída, assim como os locais do prédio que serão frequentados.

O primeiro fator a ser identificado pelo recepcionista é se o visitante já possui cadastro no sistema, em caso negativo, será solicitado um documento com foto que traga suas informações oficiais e, logo em seguida, será capturada uma imagem do mesmo, por meio da *webcam* conectada ao computador do recepcionista. Findo o cadastro, será indagado qual(is) o(s) local(is) o visitante irá

se dirigir, importante frisar que essas informações deverão ser armazenadas ao iniciar a visita pelo sistema *Guest.*

Os locais possuem uma classificação específica, que definem o quão restrito é aquele ambiente, sendo esta classificação definida por três valores que são representados pela seguintes cores: Verde, amarelo e vermelho, respectivamente do menos ao mais limitado. O sistema analisa essa indicação e define qual será a classificação do crachá[1] da visita, auxiliando o pessoal da segurança a verificar se os visitantes estão nos lugares que lhes foram delimitados.

O prédio principal e o anexo da JFRN são compostos por cerca de trinta e dois setores, divididos entre os quatro andares do prédio principal e os quatro andares do anexo. Dentre estes setores, estão inclusos as dezesseis Unidades de Atendimento (UA) e a Central de Atendimento (CA).

A UA se trata de qualquer setor responsável pelos serviços burocráticos referentes a um processo, respeitando a sua especificação, que realizam atendimento ao público referente a jurisdição. Para a JFRN se tratam de quinze varas (nomeadas de 1ª a 15ª Vara) e a Turma Recursal. Delas, oito[2] varas e a Turma Recursal estão situadas em Natal, estando todas localizadas no prédio principal, com exceção a 14ª Vara e a Turma Recursal que estão no prédio anexo.

Por outro lado, a CA é responsável por realizar os demais atendimentos, no geral, não referentes a processos, a não ser casos específicos da 3ª Vara, 7ª Vara e a Turma Recursal.

Por isso, caso o visitante procure algum tipo de atendimento, será enviado para a Central de Atendimento (CA) ou uma das Unidades de Atendimento (UA) disponíveis, dependendo da especificidade do atendimento que o mesmo procura. Sendo encaminhado para CA, sua situação será analisada e, resolvida ou encaminhada para a UA mais indicada.

A JFRN tem como propósito futuro que, o primeiro atendimento dos visitantes, com exceção dos Advogados, seja realizado apenas pela Central de Atendimento.. No entanto, esta regra já está em vigência para a 3ª Vara, 7ª Vara e

[1] Os crachás deveriam ser entregues aos visitantes antes de iniciar as visitas, porém, a JFRN está, atualmente, sem crachás.

[2] 1ª Vara, 2ª Vara, 3ª Vara, 4ª Vara, 5ª Vara, 6ª Vara, 7ª Vara e a 14ª Vara.

a Turma Recursal.

Além disso, a JFRN possui um número alto de visitas, utilizando o mês de Abril de 2018 como referência, eis que é o mês pós festividades carnavalescas, de modo que os dados têm maior integridade do que o resto do ano, ou seja, são mais fidedignos. Fora constatado que, em um dia ordinário, o prédio principal e seu anexo, em Natal, recebeu 824 visitas. Neste mesmo período, em âmbito estadual, este número cresce para 1.083. Em suma, durante todo este mês, verificou-se que este total chegou a 14.945, no lapso temporal de 30 (trinta) dias.

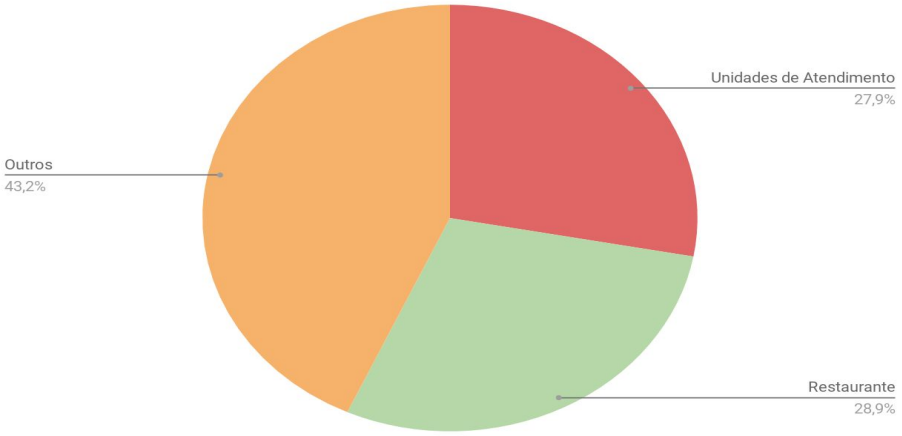


Figura 1 - Gráfico de locais visitados

Ao analisar a Figura 1, percebe-se que cerca de 27,9% das visitas estão direcionadas para as UAs. Calculando tal percentual, trazendo para a proporção do mês de observação, tem-se que, em Natal, das 824 visitas, cerca de 230 (27,9%) eram para UAs, aumentando para cerca de 302 no RN e 4.170 em todo mês.

1.1. Problemática

Ao analisar este cenário, de imediato, surge um questionamento, como controlar essas movimentações referentes a UAs? Como não há nenhum tipo de controle sobre encaminhamentos, o fluxo determinado pela JFRN acaba, muitas vezes, não sendo seguido, e, consequentemente, o *Guest* é impedido de realizar

aquilo ao que se propõe, o monitoramento das visitas.

O principal problema de atendimento enfrentado pela 3° e 7° Vara, bem como da turma recursal, é o grande número de atendimentos que prejudicam a sua eficiência e eficácia nos demais serviços prestados, logo que, muitas vezes realizam atendimentos que, a priori, haveriam de ser sob a responsabilidade da Central de Atendimento, pois o visitante, com o propósito de acelerar o êxito na informação perseguida, informa que já havia passado pelo setor que lhe fora indicado, sem de fato haver sido. Neste sentido, percebe-se que o próprio sistema *Guest* não resguarda a possibilidade de criar obstáculos para que tal problema ocorra, de modo que o visitante obtém sucesso em atravessar, diretamente, as UAs vindo a adentrar nas respectivas secretarias da 3ª e 7ª Vara e Turma Recursal.

Outra situação que afeta diretamente a segurança do prédio, é quando os visitantes afirmam que vão ao banheiro ou ao restaurante, no entanto acabam por ir à alguma vara criminal (Que são de maior classificação de segurança que os demais locais) para realizarem um atendimento sem terem sido credenciados conforme, o que é mais acentuado quando a JFRN está sem crachás, como ocorre atualmente.

Existe outra regra que todo objeto perdido deve ser entregue à sala de segurança, porém, muitas vezes o tempo de entrega de um objeto perdido no prédio anexo é muito extenso, e quando o visitante volta a justiça para procurá-lo, o mesmo ainda não está cadastro no sistema.

Dessa forma, a JFRN requisitou ao Núcleo de Tecnologia de Informação (NTI) um sistema que fosse capaz de resolver essa situação, tornando o processo mais rápido e mais fácil para os seus usuários.

Assim, surgiu o novo módulo para o *Guest* que, além de controlar os acesso dos visitantes, será capaz de administrar o atendimento e os encaminhamentos, mantendo um registro interno de tudo, criando um histórico acessível a todos os atendentes, para que eles possam ter acesso a mais informações e estar previamente preparados para os atendimentos que virão em seguida, sempre visando a melhoria da usabilidade e, também, a manutenção do sistema.

Outro ponto importante, é o reaproveitamento dos dados pertencentes ao

Guest, sem perder o desempenho que esse trabalho iria gerar ou repetir, em outro banco de dados, as informações que o ele possui. Sincretizando teoria e prática.

Outra modificação favorável é a possibilidade de abrir o módulo de achados e perdidos, e ao ser apresentado um objeto na lista de achados e perdidos, será informado o local onde esse objeto se encontra atualmente, assim, criou-se o fluxo exposto na Figura 2.

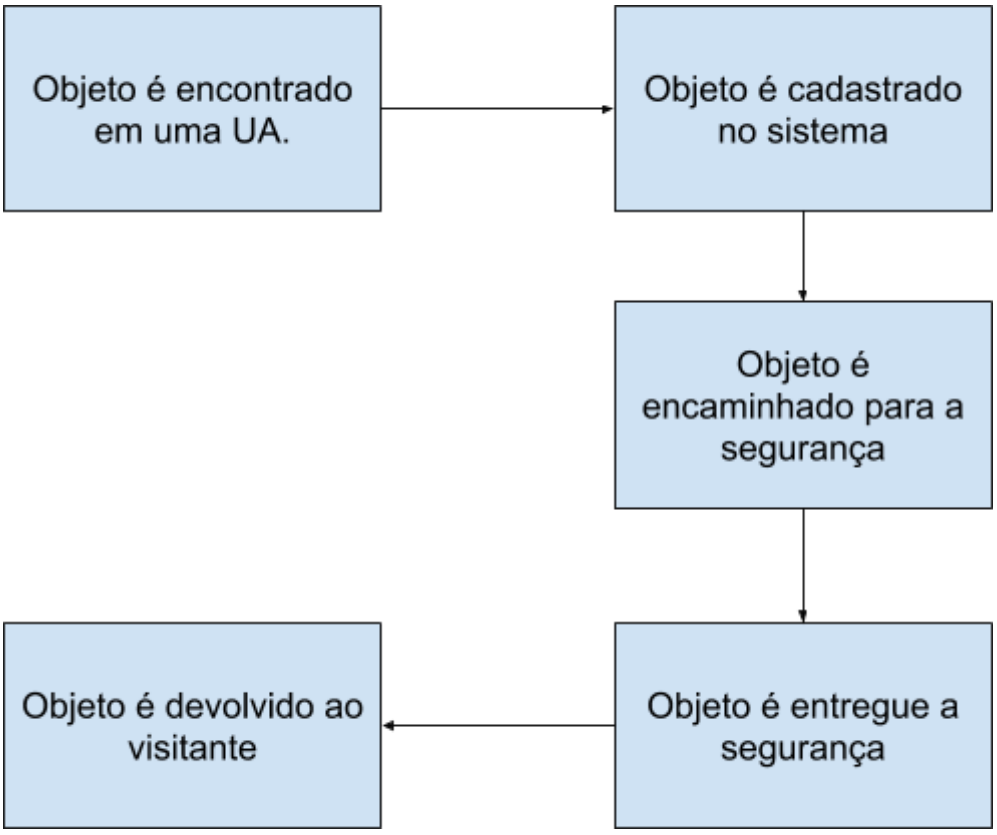


Figura 2 - Proposta de Fluxo de Achados e Perdidos

Os servidores cadastram esses objetos, encaminham para a segurança, e a equipe de seguranças confirmam recebimento. Todos esses estados serão monitorados, quando o visitante aparecer a procurar o seu objeto perdido, saberá exatamente onde está os seus pertences, podendo ser devolvido por qualquer um dos funcionários que tenham acesso ao sistema.

1.2. Objetivos e metodologia

1.2.1. Objetivo geral

Desenvolver para o sistema Guest da JFRN, um novo módulo que gerencie os atendimentos e evoluir o módulo de achados e perdidos.

1.2.2. Objetivos específicos

- Coletar os requisitos
- Aprender as tecnologias necessárias para desenvolver o produto.
- Prototipar e documentar o sistema
- Desenvolver o módulo
- Gerar uma versão estável do módulo

1.2.3. Metodologia

O tipo de pesquisa, para os requisitos, escolhida para o desenvolvimento do trabalho foi a pesquisa exploratória, por haver uma necessidade de se familiarizar com o ambiente de trabalho, para gerar algo que seja mais adequado para aqueles que irão utilizá-los em seu dia-a-dia.

Será utilizado o *framework* estrutural *Scrum* (Otávio, 2015), por ser iterativo e incremental e utilizar do conhecimento adquirido para melhorar o controle dos riscos e das tomadas de decisões futuras, sendo assim, as *sprints* duram, em média, duas semanas. Outra razão de sua utilização, é por identificar o projeto com a necessidade de seguir a essência do manifesto ágil.

A principal fonte de pesquisa foi uma amostra dos usuários do sistema, contemplando os antigos e os futuros, sendo ela um representante de cada unidade de atendimento, da central de atendimento e da segurança. Observando aquilo que é mais adequado para cada um deles, conversando com os mesmos para verificar se aquilo que está sendo proposto é o suficiente para resolver os problemas que eles enfrentam, se a interface é acessível e agradável, se não, o que será necessário para que se torne.

Assim como a utilização de prototipação de telas que foram aprovados antes de serem oficialmente implementadas. Também foram utilizadas outras

fontes de pesquisa secundária, como livros, artigos e manuais, para fundamentar as tecnologias utilizada, preferindo o material disponibilizado pela JFRN. Também foi feita uma análise de outros sistemas voltados a atender um público alvo que possuíssem algum tipo de semelhança com a ideia do módulo de atendimento do *Guest,* os principais critérios, utilizados para definir quais sistemas deveriam ser analisados, foram a visibilidade do código (sistemas de código-aberto ou sistemas criados para trabalhos acadêmicos), o quão recente é o sistema ou o se está sendo utilizado atualmente, e a sua documentação.

Para auxiliar no desenvolvimento, toda a codificação foi incluída em uma ferramenta própria da JFRN de versionamento baseada em *Git*. Para isso, ao início de cada *sprint* era feita uma revisão do ciclo anterior, observado os principais desafios, e feito o planejamento da *sprint* consecutiva, definindo as tarefas e o esforço para a realização de cada uma delas, por fim, é definido o objetivo da *sprint,* e as tarefas são incluídas na ferramenta.

Para cada tarefa é criada uma subversão do sistema, onde irão ocorrer todas as modificações de código referentes, ao ser finalizada, é solicitada a união da subversão com o projeto original, neste momento o supervisor auxiliar dos estagiários verificava se existia alguma possível melhoria, seja a nível de organização de código, como de performance e de layout gráfico.

Caso houvesse, eram abertas discussões, pela ferramenta, para averiguar se essas modificações explicitadas eram viáveis. Em caso positivo, na própria subversão eram corrigidos e, novamente, analisados se foram feitos em conformidade com o requisitado, e, ao fim, a subversão era fundida a versão original.

Após a finalização do módulo, ele foi devidamente testado pelos supervisores e estagiários do setor, conforme foi estipulado pela JFRN, onde foram encontradas falhas e problemas à que foram corrigidos antes da entrega aos clientes, que, por sua vez, testaram o sistema, que foi colocado em um servidor de homologação, realizando todos os procedimentos que lhes foram informados que o sistema é capaz, respeitando as permissões de cada usuário, e reportaram algumas melhorias para os módulos. E assim, chegou-se a um aval final de aprovação. Estando agora em fase de adaptação para começar a ser efetivamente

utilizado para todos os atendimentos.

1.3. Estrutura

Os capítulos que estão por vir apresentam toda a pesquisa realizada, as principais funcionalidades e modificações feitas no sistema, assim como a conclusão do trabalho, as referências e os apêndices.

O capítulo de fundamentação teórica é dividido em duas principais seções, a de sistemas de atendimentos, que explicita as semelhanças do módulo de atendimento com outros sistemas de atendimento, e a seção de tecnologias utilizadas, que informa e conceitua as principais tecnologias que participam da criação do módulo.

O desenvolvimento foi dividido em quatro seções secundárias. A seção 3.1 explica como o sistema se comporta, dando apenas uma visão geral. A seção 3.2 fundamenta toda a arquitetura utilizada, explicando como é feita a comunicação entre as camadas. A seção 3.3 informa todas as principais funcionalidades implementadas no sistema. E por fim, a seção 3.4, dá uma breve explanação dos artefatos de *software* em apêndices.

A conclusão define as contribuições alcançadas e se os objetivos foram atingidos conforme o esperado, apresentando algumas pequenas funcionalidades que podem vir a ser incluídas no sistema.

As referências tratam de todo o material de pesquisa utilizado para desenvolver este trabalho de conclusão de curso. E os apêndices, imagens e documentos que complementam este trabalho.

2. Fundamentação teórica

Este capítulo apresenta a base teórica, que resultou da pesquisa em fontes secundárias, bem como, os principais conceitos e elementos utilizados para subsidiar esse sistema, analisando as melhores tecnologias para implementar o módulo de atendimento, respeitando as limitações impostas pelo sistema previamente desenvolvido. Além disso, outros sistemas, com propósitos e funções semelhantes ao requisitado, foram analisados e utilizados como base de algumas funcionalidades.

2.1. Sistemas de Atendimento

Dois sistemas foram escolhidos para compor essa seção, o sistema *Clínica Fácil* (Meneses, 2017), por ter sido um trabalho de conclusão de curso com grande parte do seu código exposto no mesmo e pela qualidade do mesmo, e o *Novo SGA,* por ser de código aberto e bastante robusto. Ambos são constituídos de algumas funcionalidades bastante equivalentes, o que ajudou a constituir melhor as regras de negócio a serem aplicadas para essas funções.

2.1.1. Cliníca Fácil

Foi criado como um sistema para auxiliar a recepção e o atendimento de pacientes na Clínica Médica Otoban. O funcionamento é bem simples: Um paciente entrará em contato com a Clínica Médica para marcar uma consulta. Caso ele não possua cadastro, o(a) recepcionista irá cadastrá-lo e em seguida marcar a consulta para o médico requerido, na data e no horário livre mais adequado para o paciente. Caso o paciente já possua cadastro, apenas será marcada a consulta.

Então, no dia da consulta, o paciente irá chegar e ser atendido conforme havia sido marcado e o médico deverá registrar as informações referentes a aquilo que ocorreu dentro do consultório e liberar o paciente, se necessário, o paciente poderá agendar uma nova consulta. Tudo isso será registrado em um histórico, ao

qual os usuários terão acesso para auxiliar em suas tomadas de decisões, principalmente para o médico, por possuir o histórico de medicações e doenças do paciente. (Meneses, 2017)

Os sistemas, *Clínica Fácil* e o *Guest,* possuem muita similaridade, apesar dos âmbitos totalmente diferentes. A necessidade de um cadastro é a primeira delas, onde o paciente e o visitante tem o mesmo papel. Outra forte semelhança é a marcação de consultas para médicos específicos, que faz a ponte com o encaminhamento para unidades de atendimento específicas, principalmente pelo fato de que, tanto o paciente, quanto o visitante, só poderá ser atendido, uma vez que foi esteja com uma consulta marcada, ou encaminhado para a UA, respectivamente.

Para finalizar, ambos possuem o histórico com propósito de auxiliar no momento do atendimento, o que, no novo módulo, pode dar celeridade ao serviço, logo que o atendente poderá saber, com antecedência, quais os tipos de atendimento mais comuns para aquele visitante, ou qual a necessidade do mesmo, podendo ajudar mais pontualmente a resolvê-los com agilidade.

2.1.2. Novo SGA

O Novo Sistema de Gerenciamento de Atendimento (SGA) tem o seu código aberto, disponibilizado no *GitHub.* Tem como propósito gerenciar filas - emitindo senhas e fazendo chamadas via painel, controlar o fluxo dos atendimentos, ajudar a gerenciar e administrar as unidades de atendimento. (Lino, 2013)

Funciona da seguinte forma: Ao chegar um visitante, sua situação será analisada e será definido se ele é, ou não, um visitante prioritário, e com qual serviço ele será atendido. Após isso, o mesmo receberá uma senha, e deverá aguardar até que a senha seja exibida no painel. Quando a senha é chamada, o visitante será atendido conforme o serviço requisitado. (Anderson, 2013)

A principal semelhança entre o Novo SGA e o módulo de atendimento do Guest, se trata da triagem inicial. O Novo SGA escolhe um serviço, enquanto a CA tem como propósito definir quais atendimentos podem ser resolvidos lá, e quais

necessitam de um encaminhamentos para Unidade de Atendimento a qual o processo do visitante se encontra.

2.2. Tecnologias utilizadas

Por já possuir um grupo robusto de tecnologias, só serão apresentas as mais essenciais para a construção deste trabalho, apresentando seus conceitos básicos e a razão do sistema usufruir delas.

2.2.1. Java EE

Java Platform, Enterprise Edition é um padrão de plataforma para grandes aplicações Java ou aplicações WEB. (FARIA, 2013) Além desse fator já ser o suficiente para utilizá-lo no *Guest*, pois é uma aplicação em java de grande porte e voltada para internet, ela ainda possui as tecnologias JavaServer Faces (JSF), Java Persistence API (JPA) e Servlet. (FARIA, 2013). Além, é claro, de suportar a linguagem de programação a qual foi constituído o sistema e seu novo módulo.

Os *servlets* são utilizados para capturar as imagens que completam o cadastro de pessoas e objetos perdidos. O JPA está presente em toda a camada de persistência e o JSF tem como principal função facilitar a criação de telas, foi utilizado de alguns de seus componentes com funcionalidades mais expressivas.

2.2.2. Padrão de Arquitetura Model-View-Control (MVC)

A arquitetura *Model-View-Control* foi escolhida por possuir uma dinâmica simples, separando “as camadas de apresentação, de lógica de negócio e de gerenciamento do fluxo da aplicação” (LUCIANO E ALVES, 2011, p. 103) com isso, facilita a manutenção do projeto, o que é extremamente necessário para a JFRN, como há um grande fluxo de estagiários que trabalham nos mais diversos sistemas.

Nesta arquitetura todas as a camada de controle (*control*), monitora o fluxo de dados entre a camada de modelo (*model*), onde são processados os dados, e, as telas (*view*), a camada responsável por se comunicar com o cliente ativamente. (LUCIANO E ALVES, 2011)

2.2.3. Hibernate

É um *framework* criado para mapear objeto relacionais para linguagem Java. Junto com o padrão JPA, a recuperação de dados das associações fica mais eficiente, e é possível gerenciar as transações com o banco de dados. Além disso, há uma diminuição de códigos de persistência, e com isso, o aumento de gradativo de performance das requisições. (BALDUÍNO, 2014)

Todas as classes que são armazenadas no banco de dados (BD) utilizam a anotação do hibernate *@Entity* para que o mapeamento seja feito automaticamente pelo sistema, também foram utilizadas o *@OneToMany, @ManyToMany* e o *@ManyToOne* para gerar os relacionamentos entre as entidades, definindo os relacionamento de 1...N (Um para muitos), N...N (Muitos para muitos) e 1...1 (Um para um), respectivamente. A *@Transient* para as classes de objeto de transferência de dados (*Data Transfer Objects* - DTO) que representa um conjunto de dados reunidos de forma coesa e serve para transportar este conjunto de dados entre as camadas de uma aplicação, sem a necessidade de efetuar uma chamada para cada atributo do objeto. Isto economiza tráfego de rede e são muito importante em sistemas. (FOWLER, RICE, FOEMMEL, HIEATT, MEE, STAFFORD, 2003, pag. 91-92)

2.2.4. Springframework e Spring Security

Se trata de outro *framework*, dessa vez de código aberto, que tem como propósito principal simplificar a programação do sistema. Existem tipos diferentes deste *framework* (BAWISKAR, SAWANT, KANKATE E MESHRAM, 2012), com diversas funcionalidades, porém, uma das mais importantes para o *Guest* é o *Spring Security*.

O *Spring Security* utiliza de métodos, ferramentas e atividades que para facilitar a parte de autorização e autenticação de um sistema, trabalhando com *roles* (permissões do sistema) que definem as permissões do sistema, baseado em padrões de segurança desenvolvido as melhores práticas de soluções para os

problemas de segurança mais comuns.(DIKANSKI, STEINEGGER e ABECK, 2012)

2.2.5. Primefaces

Além de ter uma documentação muito bem estruturada, o *Primefaces* é "uma bibliotecas de componentes ricos para aplicações criadas com JavaServer Faces" (ANDRADE, 2016). Sendo um complemento e não uma especificação do JSF, possui mais de 100 componentes "bonitos e cheios de recursos" (ANDRADE, 2016).

Outro ponto importante, é fato de que esses componentes já possuírem a funcionalidade de Ajax[3] integrado, ajudando na estruturação da arquitetura do sistema, além de muito simples de configurar em projetos Maven (BALDUÍNO e RUFINO, 2014), como é o caso do *Guest*.

2.2.6. Jasper Reports

É um *framework* de código aberto totalmente escrito em Java, com função de gerar e exportar relatórios dinâmicos em diferentes formatos, aceitando diversas formas de entradas de dados, permitindo a utilização de "diagramas, gráficos, e até código de barras" (MARTINS, 2010).

Além de tudo, é simples de gerar o relatório, e mais simples ainda de definir o *design* do relatório, pode-se montar uma imagem ou pdf arrastando os componentes para a tela, e ainda é possível testar o resultado final com a opção de *preview* (MARTINS, 2010).

[3] *Asynchronous Javascript and XML*, em português, Javascript e XML Assíncronos. Se trata de um conjunto de técnicas para programação e desenvolvimento web utilizando tecnologias como Javascript e XML para carregar informações de forma assíncrona

3. Desenvolvimento

O novo módulo tem como principal função auxiliar os servidores na hora de realizar atendimentos, sendo assim, foi necessária a criação de três entidades, quatro *data models*, cinco *data access objects*, três telas para manipulação de classes, quatro telas de geração de relatórios e cinco *managed beans*.

Também se fez necessário modificar todo o módulo de Achados e Perdidos, as regras de negócio da geração de visitas e incluir novas funcionalidades no cadastro e visualização de visitantes.

Algumas informações, como os detalhamentos das versões das tecnologias, os nomes das classes, seus métodos e atributos, ou dos arquivos de código, bem como a maior parte da documentação, a tela de atendimento (por possuir informais pessoais de indivíduos cadastrados no sistema), a tela de exibição de locais (que foi censurada) e o diagrama de classes não estarão presentes por questões de direitos autorais, pois, independente do desenvolvedor, todos os códigos fontes pertencem a Justiça Federal do RN, e parte deles foram cedidos para que esta monografia pudesse ser realizada. Além disso, todos os nomes de variáveis, métodos, atributos, objetos e qualquer informação pertinente não são os reais, tem apenas propósito de ilustrar e exemplificar os códigos.

3.1. Visão do sistema

O *Guest* é um sistema utilizado para auxiliar no controle de acessos de visitantes, o recolhimento, devolução de objetos perdidos bem como os atendimentos (com a inclusão do novo módulo), referente aos prédios da Justiça Federal situados no estado do Rio Grande do Norte. Sendo assim, para que isto funcione, existem três módulos que trabalham separadamente compartilhando de um mesmo banco de dados (denominado de BD) e de funções específicas do sistema.

3.1.1. Módulo de Visitas

O cadastro do visitante é composto pelas informações pessoais tais como: a) nome, b) data de nascimento, c) número de documento válido[4], d) unidade federativa (UF) em documentos nacionais, e) país de origem no que tange a documentos internacionais, f) tipo da matrícula - caso o documento seja emitido pelo Ministério da Fazenda -, g) número de telefone, e uma foto, que deverá ser capturada no momento do cadastro. Todas essas informações podem ser alteradas na página de gerenciamento de visitantes, todavia, elas são obrigatoriamente atualizadas a cada 365 dias.

Para iniciar a visita, é necessário definir o visitante, os locais que ele irá se dirigir e o crachá utilizado. Após isso, é registrada a entrada, e é feito o encaminhamento do visitante para os locais informados que são UAs, porém, não são a 3° vara, 7° vara, Turma Recursal e Central de atendimento, “comunicando-se” com o módulo de atendimento.

Os dados pessoais não são os únicos atributos a serem armazenados, se houver encaminhamento, da recepção ou de qualquer UA, os locais encaminhados e a hora do encaminhamentos são registrados, e existem as informações especiais. Esta classe é composta pela descrição, se é ou não relevante para a segurança, e um atributo do tipo *boolean*[5] de controle, para verificar se essa informação está ativa ou não (atributo incluído em conjunto com o módulo). As informações não relevantes são as que servem de apoio para atender melhor às necessidades dos visitantes, no entanto, existem informações que deverão ser chamadas de *relevantes* eis que são aquelas que necessitam de auxílio da segurança para adentrar a JFRN, com maior enfoque em pessoas agressivas ou perigosas.

3.1.2. Módulo de Achados e Perdidos

[4] Os tipo de documento permitidos são: Registro Geral (RG), número da carteira da Ordem de Advogados do Brasil (OAB), matrícula funcional (MF), passaporte nacional e internacional e RG estrangeiro (RGE)

[5] nome dado ao tipo de dados primitivo que possui apenas dois valores diferentes, podendo ser *true* (verdadeiro) e *felse* (falso), em algumas linguagens, ou 0 e 1, em outras.

Já o módulo de achados e perdidos é bem menor, e utiliza apenas 6 entidades e classes do sistema. Antes da inclusão, existia apenas uma tela de cadastro para incluir e devolver os objetos que eram entregues a segurança, de modo que se fazia necessário o nome do objeto, a descrição, o visitante - e se conseguisse ser identificado-, em uma imagem dele, totalmente opcional. Com a inclusão do módulo, foi possível incluir um fluxo aos objetos, onde eles podem ser incluídos pelas próprias unidades de atendimento, e ser, posteriormente, encaminhados ao setor da segurança, onde localiza o achados e perdidos. Dessa forma, ao adicionar um objeto, a tabela apresenta o detalhamento da peça, assim como a sua localização atual.

3.1.3. Módulo de Atendimento

O último módulo, o de atendimento dispõe de 12 entidades e 20 classes. Sua função é, basicamente, guardar os registros dos atendimentos realizados, efetuar encaminhamentos, editar dados específicos do visitante (nome, telefone e número do documento), adicionar informações especiais, bem como, ativá-las e desativá-las. Também é possível visualizar um histórico de visitas e seus atendimentos. Sua tela é dividida em dois espaços, sendo o espaço mais a esquerda para listar os visitantes encaminhados para a lotação do usuário atualmente logado no sistema, com exceção dos usuários da CA, para estes são expressos todos os visitantes atualmente em visita.

O sistema ainda é composto por telas de cadastro para todas as entidades de apoio aos módulos, para que essas não sejam incluídas pelo Sistema de Gerenciamento de Banco de Dados (SGBD). Porém, a maior parte dessas telas só são acessíveis pelos administradores do sistema, os servidores e estagiários do Núcleo de TI, pensando em possíveis gargalos gerados pelo aumento de usuários do sistema, as novas telas de apoio ao módulo de atendimento foram dinamizadas e criadas para serem operadas pelos diretores de cada uma das UAs.

3.1.4. Relatórios

Existem três tipos de relatórios, os relatórios por gráfico, por tabela e o feito com Jasper Reports. Cada relatório tem uma finalidade específica sobre as visitas, atendimentos e o visitante. Utilizando o *Linear Chart,* que pode ser modificado para *Bar Chart*, a preferência do usuário, para fazer os relatórios de atendimento por hora, por unidade de atendimento, por usuário e por tipo de atendimento, e os de visita por data e local. Esses dados são recuperados do banco devidamente ordenados utilizando seus respectivos *data transfer object* (DTO).

O relatório de atendimento e visita por data são criados em conjunto com o de hora (respectivo de cada um), formado por uma tabela, onde os dados também são agrupados e recebidos por um DTO. A tabela é exibida em ordem crescente pela quantidade de visitas/atendimentos no dia, sendo apresentado com a data, o dia da semana, e a quantidade. É possível reordenar utilizando os botões de ordenação.

O relatório individual foi solicitado pela segurança, e possui todas os dados de um visitante escolhido, dados pessoais, foto mais recente, suas informações especiais, o histórico de todas as suas visitas e de todos os seus atendimentos.

O sistema utiliza de permissões para garantir acesso às funcionalidades de cada módulo. O módulo de achados e perdidos pode ser usado por todos, entretanto, o gerenciamento de visitas e visitantes só pode ser utilizado pelos recepcionistas e pela equipe de segurança. Assim como a tela de atendimento só pode ser utilizada pelos servidores que realizam atendimento ao público e seus diretores.

3.2. Arquitetura

O padrão de arquitetura MVC é utilizado de forma bastante precisa, a camada de controle é a única responsável por transmitir os dados modelados para a exibição ao cliente. Entretanto, existe uma outra camada que faz a conexão do banco de dados com o modelador de dados, essa é a camada de acesso aos dados.

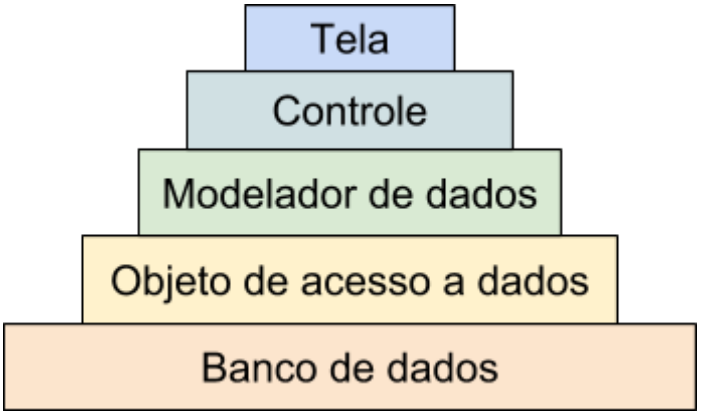


Figura 3 - Camadas da Arquitetura

A Figura 3 apresenta a divisão de camadas pela qual o dado deverá passar entre o banco de dados e a visualização do cliente. O objeto de acesso a dados (*Data Access Object - DAO*) se responsabiliza por transformá-los em instâncias de alguma classe. O modelador de dados será responsável por verificar as regras de negócio e moldar essas classes para que não ocorram futuros problemas, como o *NullPointerException* que é uma exceção ao tentar acessar um objeto nulo.

A camada de controle faz a conexão dos dados modelados com a tela, para organizar e tratar as requisições de maneira independente. A tela é a camada que expressa as informações em seu último estado. Porém, é também onde o cliente pode requerer alguma modificação ou inclusão de informação ao banco de dados, obrigando as camadas a fazerem todo o processo inverso, transformando os dados em estado final para um estado reconhecido pelo BD.

Para fazer essa comunicação, primeiramente, as telas são criadas utilizando a XHTML e são utilizados os componentes da biblioteca *primefaces*, por já terem o *ajax* integrado, o que torna mais simples a comunicação com os controles. Para que assim seja feito, é colocada a *tag* do primefaces, o nome do componente e o atributo *value* para fazer a comunicação dele com a variável na classe de controle, conforme o exemplo do *spinner* na Figura 4.

```
<p:spinner value="#{nomeDaClasse.variavel}" />
```

Figura 4 - Exemplo de Spinner

É obrigatória a utilização da *hashtag* (#) antes do colchete, assim como devem existir os *getters* e *setter*, métodos que retornam e modificam o valor da variável, respectivamente, conforme o padrão do Java na Figura 5:

```
public TipoDaVariavel getNomeDaVariavel(){
    ...
}

public void setNomeDaVariavel(TipoDaVariavel nomeDaVariavelAuxiliar){
    ...
}
```

Figura 5 - Exemplo de Get e Set

Pois o *ajax* os gera automaticamente, mantendo o encapsulamento dos dados. Entretanto, para que a classe de controle compreenda essa requisição e possa responder corretamente, é necessário a utilização da anotação *@ManagedBean(name ="nomeDaClasse")*, sendo o valor após a igualdade o nome que a tela fará as requisições, ou seja, o nome no *value.* O atributo *name* é opcional, por isso, caso não seja definido, o padrão é o nome da classe com a primeira letra minúscula, conforme apresentado na Figura 6.

```
//Componente da Tela
<p:dataTable value="#{mBExemplo.dadosModeladosDataModel}" lazy="true"/>

//Classe com anotação
package ...;

import javax.faces.bean.ManagedBean;
...
@ViewScoped
@ManagedBean
public class MBExemplo {

    private ModeloDataModel dadosModeladosDataModel;
    private ObjetoDeAcessoAosDadosHibernateDAO instanciaDeAcessoAosDados;
    ...
    public MBExemplo(){
    ...
    instanciaDeAcessoAosDados = inicialização;
    dadosModeladosDataModel = new ModeloDataModel(instanciaDeAcessoAosDados);
    ...
    }
    ...
    public void setDadosModeladosDataModel(ModeloDataModel dadosModeladosDataModel){
        this.dadosModeladosDataModel = dadosModeladosDataModel;
    }

    public ModeloDataModel getDadosModeladosDataModel(){
        return dadosModeladosDataModel;
    }
    ...
}
```

Figura 6 - Componente relacionado à atributo do Managed Bean

Entretanto, o *managed bean* não acessa, nem modela, os dados do BD. Para isso, é utilizado o objeto de acesso aos dados (*Data Access Object - DAO*) e os modelos de dados (*data model*). Onde o primeiro faz a comunicação com o banco de dados, obtendo as informações necessárias, e o outro modela esses dados conforme a necessidade, para assim, devolvê-los a camada de controle. Exemplificado na Figura 7.

```
//Modelo de Dados
package ...;

import org.primefaces.model.LazyDataModel;
...
public class ModeloDataModel extends LazyDataModel<NomeDaEntidade> {

    protected ObjetoDeAcessoAosDadosHibernateDAO dao;

    public ModeloDataModel(ObjetoDeAcessoAosDadosHibernateDAO dao) {
        this.dao = dao;
    }

    @Override
    public List<NomeDaEntidade> load(...) {
        ...
        return retorno;
    }
    ...
    public void setDao(ObjetoDeAcessoAosDadosHibernateDAO dao) {
        this.dao = dao;
    }

    public ObjetoDeAcessoAosDadosHibernateDAO(){
        return dao;
    }
    ...
}
```

Figura 7 - Exemplo de *Data Model*

Para fazer essa modelagem, é necessário a utilização de uma biblioteca específica do primefaces, a *org.primefaces.model.lazyDataModel*, e criar um *data model* que estende a classe *LazyDataModel.* Com isso, é feito requisições a um *DAO* previamente definido no *managed bean* que é passado como parâmetro no momento de sua construção, conforme demonstra a exemplificação abaixo.

3.3. Principais funcionalidades

Uma das primeiras funcionalidades que fora implementada foi o cadastro e a edição dos tipos de atendimento, eis que este é o único campo obrigatório para que o atendimento seja realizado, e uma UA só pode realizar um atendimento por vez, caso esteja associado a, ao menos um, tipo de atendimento. Inicialmente foi

criada a classe referente, seguida pela tela em conjunto com o *managed bean* e o *data model.*

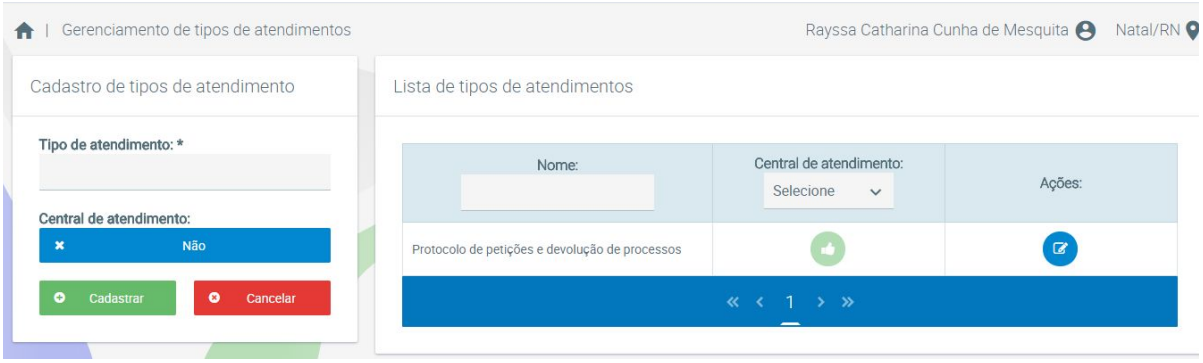


Figura 8 - Tela de gerenciamento de Tipos de Atendimento

A Figura 8 apresenta uma tela simples, todavia, muito importante para o desenvolvimento do projeto, logo que se refere ao gerenciamento de tipos de atendimento. Foi criado um *form* que contém os componentes *inputText* e *selectBooleanButton*, para tornar o campo de tipo de atendimento obrigatório, foi incluÍdo o atributo *required="true"* no *inputText*. O botão de confirmar chama um método no *managed bean* para a realização do cadastro, onde essas informações passaram a ser uma instância da classe e assim seguir o fluxo até serem salvas no banco de dados.

Essas informações também podem ser editadas ao clicar no botão azul, na coluna de ações. As informações serão recuperadas e renderizadas no painel de cadastro, utilizando o componente da JSF setPropertyActionListener, que seta o valor passado em seu atributo *value* na variável definida no *Target.*

```
<h:form>
    <p:dataTable var="tipo" value="#{managedBean.listaDeTipos}"emptyMessage="Não há nada cadastrado"
        paginator="true" rows="5" lazy="true" paginatorPosition="bottom">
        ...
        <p:column headerText="Ações:">
            <p:commandButton actionListener="#{managedBean.antesDeEditar}" title="Editar tipo"/>
                <f:setPropertyActionListener target="#{managedBean.instanciaDaClasse}" value="#{ tipo }"
            </p:commandButton>
         </p:column>
         ...
    </p:dataTable>
</form>
```

Figura 9 - Exemplo de *Form*

Na Figura 9, o *target* é a variável a ser setada dentro do arquivo de controle, e é passado no *value* o valor que será incluído, que, neste caso, é o mesmo nome do atributo *var* no *data table*, que é o nome da variável utilizada dentro da tabela. Ou seja, ao clicar no botão, será chamado o método *antesDeEditar,* e será setado na variável *instanciaDaClasse* os valores daquela linha da tabela. Da mesma forma que é feito em todas as telas de gerenciamento de classes de apoio.

No entanto, a tela de gerenciamento de unidades de atendimento (expressa na Figura 11) não seguem a padrão do sistema, logo que fora realizado um reaproveitamento da classe de apoio referente aos locais (locais estes a serem visitados informados ao iniciar uma visita). Por isso, esta classe sofreu uma ligeira modificação, de modo que fora incluída uma variável que define se ela é ou não uma unidade de atendimento, que também restou incluída em seu formulário de cadastro. como pode ser visto na Figura 7. Desta forma, esta tela só pode ser acessada por administradores do sistema, desta forma, um local só poderá se tornar uma UA, caso seja requisitado ao NTI que assim deve ser feito.

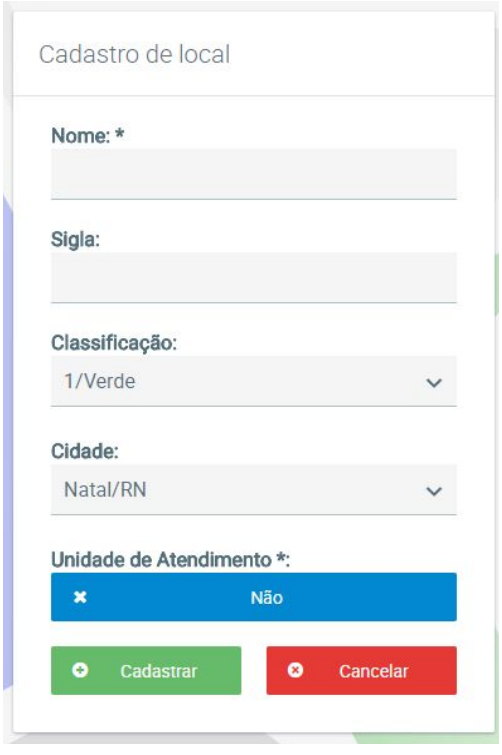


Figura 10 - Formulário de cadastro de locais

Além disso, para que se ocorra um atendimento, se faz necessária a existência de locais que estejam aptos a essa função, ou seja, estejam associados as TAs, por isso, fora criado um relacionamento entre as entidades de *locais e tipos de atendimento*. Para manter o módulo mais dinamizado, fora desenvolvido uma nova interface gráfica, com propósito específico de gerenciar as associações deste relacionamento pelos diretores das UAs, e não somente os administradores do sistema.

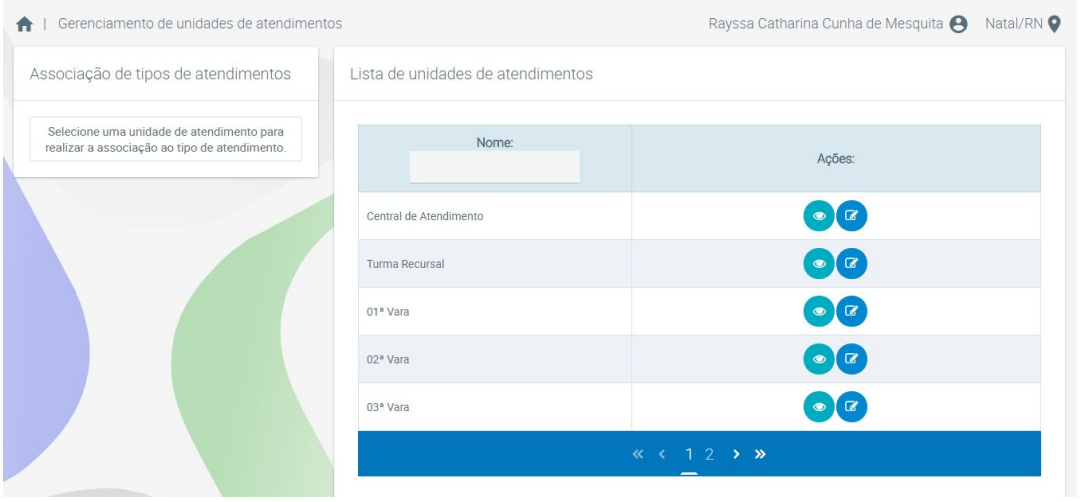


Figura 11 - Tela de gerenciamento de Unidades de Atendimento .

Para realizar associações ou às modificar, basta clicar no ícone azul, que pode ser visualizado na Figura 11, que irá utilizar dos mesmos recursos das outras telas de cadastro do sistema. O *panel* a esquerda será renderizado e lá será exibido o nome da UA que está sendo alterada, um *autocomplete* com *dropdown* para selecionar um tipo de atendimento não associado, e um botão que o acrescenta em uma *data table*. Essa tabela possui duas colunas, uma para o nome do tipo e outra para o botão que possibilita a desassociação desse tipo. Importante frisar que o algoritmo utiliza uma variável temporária que armazena o valor, e envia para a variável definitiva.

Outra função desta tela é exibir os tipos associados a determinada UA em uma *dialog* ao clicar no botão verde, que também utiliza o *setPropertyActionListener* para definir as informações que a irão popular.

A tela de atendimento é parte principal do módulo, onde a maior parte lógica está realizada, ela é composta por uma lista selecionável de visitantes, que pode ser filtrada por nome ou documento, voltada para a esquerda, e uma ficha de algum dos visitantes selecionados voltado a direta. Essa ficha contém as informações que podem ser editadas ao serem clicadas: *nome, telefone e número do documento*, bem como as que não são editáveis: *data de nascimento, local do documento* e *atualizado em*. O campo de *estado atual* é um campo mutável ao longo do fluxo, porém, não pode ser diretamente modificado por nenhum de seus usuários.

A lista de visitantes exibe informações diferente para usuário de UAs diferentes. No caso da *Central de Atendimento*, serão exibidos todas as pessoas que estão atualmente em visita, ordenadas pelo nome, independente de ter sido encaminhada pela recepção ou não. Para as demais unidades de atendimentos serão apresentados todos os visitantes que foram encaminhados para elas, ordenadas por ordem de encaminhamento, do mais antigo ao mais novo.

```
1  //Lista
2  SELECT DISTINCT *
3  FROM tabela_de_visitantes as t_vtnt
4      INNER JOIN tabela_de_visitas as t_vis ON (t_vtnt.visita_id = t_vis.id)
5  WHERE (t_vnt.nome LIKE %info_do_filtro% OR t_vnt.documento LIKE %info_do_filtro%)
6      AND t_vis.registro_saida IS NULL
7  ORDER BY t_vnt.nome ASC
8  OFFSET inicio ROWS FETCH NEXT tamanho_da_pagina ROWS ONLY
9
10 //Quantidade
11 SELECT DISTINCT COUNT(*)
12 FROM tabela_de_visitantes as t_vtnt
13     INNER JOIN tabela_de_visitas as t_vis ON (t_vtnt.visita_id = t_vis.id)
14 WHERE (t_vnt.nome LIKE %info_do_filtro% OR t_vnt.documento LIKE %info_do_filtro%)
15     AND t_vis.registro_saida IS NULL
```

Figura 12 - Exemplo de consulta para CA

Para capturar a lista da CA foi feita uma busca no BD foi feito um *inner join* da tabela de visitantes com a tabela de visitas, capturando apenas os visitantes que contenham a informação descrita no filtro da lista e estejam com a informação de saída nula, como mostra o exemplo na Figura 12. Deste modo, é feita uma ordenação por nome, limitando a quantidade de resultados por um valor inicial até o tamanho da página da tabela (neste caso formado por 10 linhas). Sendo assim,

já sido feita uma segunda requisição ao banco para saber a quantidade total de visitantes que deverão aparecer e definir a quantidade de páginas. Gerando solicitações ao BD parecidas com as expostas acima. Para as demais, necessário se fez criar dois atributos que representam o estado atual e a data do último encaminhamento, essas informações estarão salvas no banco de dados, bem como serão recuperadas pela camada de dados apenas os visitantes que possuem o *status* como remetido para aquela UA específica, formulando assim consultas parecidas com as apresentadas anteriormente, porém, com alguma modificações marcantes como a necessidade de outro *inner join* com a tabela de locais para saber se há algum visitante encaminhado, e são ordenados pela data do encaminhamento, que é um *timestamp,* que é um tipo de dado que representa um ponto específico na linha do tempo e leva em consideração o fuso horário em questão*,* como exemplificado na Figura 13.

```
1  //Lista
2  SELECT DISTINCT *
3  FROM tabela_de_visitantes as t_vtnt
4      INNER JOIN tabela_de_visitas as t_vis ON (t_vtnt.visita_id = t_vis.id)
5      INNER JOIN tabela_de_locais as t_loc ON (t_vis.status_atendimento = t_loc
6  WHERE (t_vnt.nome LIKE %info_do_filtro% OR t_vnt.documento LIKE %info_do_filt
7      AND t_loc = local_do_usuario
8  ORDER BY t_vnt.data_do_encaminhamento DESC
9  OFFSET inicio ROWS FETCH NEXT tamanho_da_pagina ROWS ONLY
10
11 //Quantidade
12 SELECT DISTINCT COUNT(*)
13 FROM tabela_de_visitantes as t_vtnt
14     INNER JOIN tabela_de_visitas as t_vis ON (t_vtnt.visita_id = t_vis.id)
15     INNER JOIN tabela_de_locais as t_loc ON (t_vis.status_atendimento = t_loc
16 WHERE (t_vnt.nome LIKE %info_do_filtro% OR t_vnt.documento LIKE %info_do_filt
17     AND t_loc = local_do_usuario
```

Figura 13 - Exemplo de consulta para UA

Perceba que, para diferenciar as varas que tem primeiro atendimento na CA das outras, se fez necessária a criação de uma lista com as informações pertencentes às UAs que não possuem o primeiro atendimento especializado. Ao iniciar uma visita, é verificado se algum dos locais encaminhados está presente nesta lista, se sim, modificar os novos atributos especificados acima para obter os dados necessários para que possam ser atendidos.

Entretanto, esta regra de *negócio* acima, não se aplica a visitante do tipo advogado, pois estes têm acesso aos processos, de modo que não devem

necessitar de encaminhamento para serem atendidos por quaisquer uma das UAs. Desta forma, o referido visitante será encaminhado para todas as unidades de atendimento, conforme pode ser melhor explicitado na Figura 14.

```
public ClasseDeVisitante encaminharVisitante(ClasseDeVisita visita){
    ClasseDeVisitante visitante = visita.pegarVisitante();
    if(!visitante.pegarTipo().pegarDescricao().equals("Advogado")) {
        for(ClasseLocal l : visita.pegarLocais()){
            //Verifica se local é unidade de atendimento e chama a função para verificar tabela
            if(...){
                //Verifica se a lista de encaminhamento está vazia
                if() {
                    //Inicializa a lista
                }
                //Adiciona o local l a lista
                //Inclue o horário do encaminhamento
            }
        }
    }else {
        //Encaminha o advogado para todas as Unidades de Atendimento
    }
    return visitante;
}

public boolean verificacaoDeTabela(ClasseLocal l){
    List<ListaDeLocaisPrimeiroAtendimentoLivre> lista = new ArrayList<ListaDeLocaisPrimeiroAtendimentoLivre>()
    lista = pegaAsInformacoesDoDAO;
    for(ListaDeLocaisPrimeiroAtendimentoLivre objeto : lista){
        //Verifica se o id de l é igual ao id de objeto
        if(...)
            return true;
    }
    return false;
}
```

Figura 14 - Exemplo de encaminhamento de visitante

Além dessas funcionalidades, a edição de informações diretamente no label foi implementada utilizando um componente do *PrimeFaces* chamado *cellEdit*. Apenas ao clicar na célula, é possível editá-la e salvar a informação diretamente no banco de dados.

As informações especiais, como visto na Figura 15, que também aparecem no módulo de visita. Para que fossem melhor aproveitadas, se fez a inclusão de um atributo de desativação em casos de erro humano. Desta forma, existem botões que trabalham sobre tais dados, de modo que são habilitados de acordo com a permissão do usuário. Sendo assim, todos podem cadastrar informações, apenas diretores de unidades de atendimento e seguranças podem desativá-las, e somente os seguranças podem reabilitá-las, também pela mesma possibilidade de equívoco provocado seja do diretor ou da segurança. Vale ressaltar que todo este fluxo fora configurado em uma reunião com os servidores representantes.

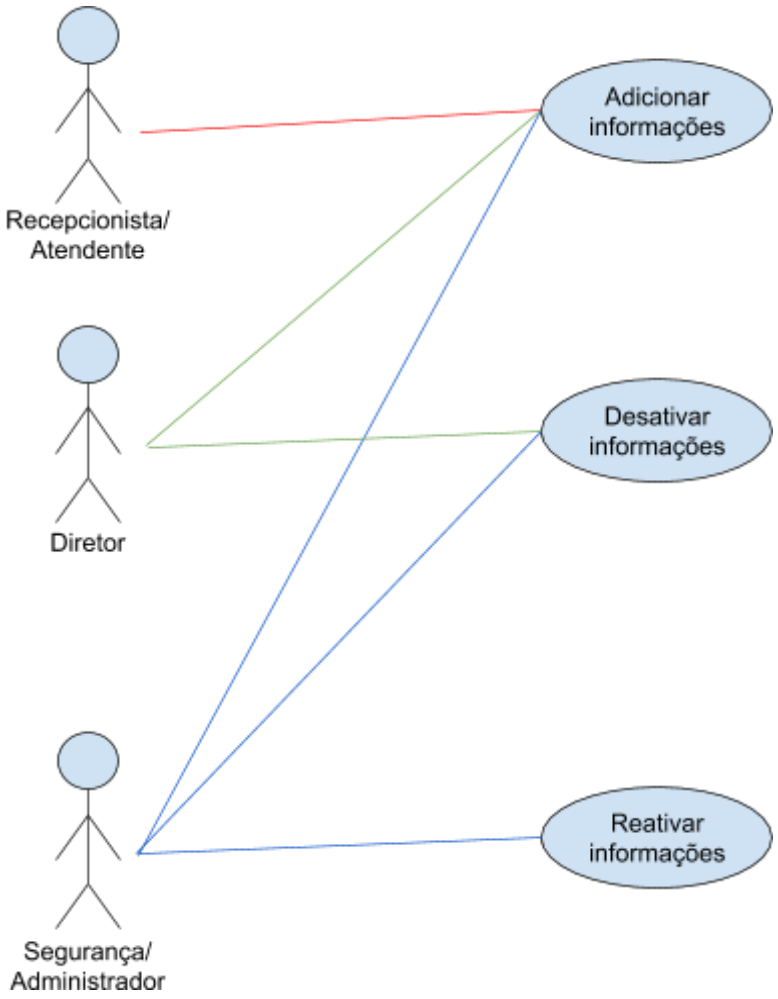


Figura 15 - Caso de Uso das Informações

. Ademais, é possível visualizar o histórico do visitante que é exibido em forma de tabela, em uma *dialog*, expressando todas as vezes que o visitante selecionado frequentou qualquer prédio da Justiça Federal que utiliza o *guest*, desde que o sistema foi implementado. Ele apresenta as informações de data, locais visitados e se houve algum atendimento. Havendo atendimento, aparecerá, também, o ícone do *rowExpansion*. Ao clicar nela, todos os atendimentos daquele dia serão exibidos em outra tabela, onde serão explanadas a data e hora do atendimento, o nome do usuário que o realizou, o local do ocorrido, o número do processo e se houve encaminhamento.

Para realizar um atendimento, necessário se faz clicar no botão verde localizado abaixo da imagem do visitante selecionado, então, será exposta uma *dialog* com um formulário a ser preenchido, que podem ser incluídas as seguintes

informações: *Tipo de atendimento, número do processo, encaminhar, descrição e observação*. O *tipo de atendimento* é sempre obrigatório, já o *número do processo* só não é obrigatório para a CA. Caso o atendimento seja para um advogado, o campo de encaminhar será desabilitado utilizando o atributo *disable="#{managedBean.algoritmo}"*. Onde o *algoritmo* se trata de um método que verifica se o visitante é um advogado, se sim, retorna *true*, se não, *false;*

O campo de número do processo poderá já vir preenchido em casos de encaminhamento pela central de atendimento para qualquer uma das demais unidades em que o servidor informou durante o atendimento ao formulário. Para que assim ocorra, sempre que será iniciado um novo atendimento, será feita uma busca ao banco de dados, pelo último atendimento referente a essa visita. Após isso, é verificado se o resultado da busca é nulo, se não, é analisado se veio da central de atendimento, se for, é armazenado o valor na variável do *managed bean* responsável pelo número do processo. Se qualquer uma dessas informações não estiverem representadas dessa forma, o campo estará vazio. O código abaixo, na Figura 16, demonstra melhor como é feito esse método.

```
1  public String getNProcesso() {
2      //Verifica se o visitante selecionado é diferente de nulo
3      if(...) {
4          //Armazena em uma variavel temporaria o último atendimento da visita atual
5          ...
6          if(atendimento != null){
7              //Verifica se o atendimento ocorreu na central
8              if(...)
9                  nProcesso = atendimento.getNumeroProcesso();
10         }
11     }
12     return nProcesso;
13 }
```

Figura 16 - Exemplo de Algoritmo Sobre Número de Processo

No campo de encaminhamento, 3ª vara, 7ª vara, Turma Recursal e o local onde está ocorrendo o atendimento são serão exibidos. Ao finalizar um atendimento, e ser definido um local, os atributos referentes ao estado são modificados. Caso não haja, eles serão setados com seus respectivos valores nulos.

Há, em sua composição, três estados atuais em que o visitante poderá apresentar. São eles: "*Nenhum atendimento pendente", "Encaminhado para múltiplas unidades de atendimento*" e "*Encaminhada para 'nome da UA'*".

Ao finalizar um visita, mesmo que uma pessoa esteja encaminhada para um atendimento, o sistema limpa essas variáveis de encaminhamento, por conseguinte, os visitantes desaparecem da lista de encaminhamento nas tela de atendimento.

Ademais, existem três tipos de relatórios, os relatórios por gráfico, por tabela e o feito com Jasper Reports. Cada relatório tem uma finalidade específica que foram conversados entre os representantes das unidades de atendimento e o representante da segurança.

Sendo assim, fora utilizado o *Linear Chart,* que pode sofrer modificações para *Bar Chart*, a escolha é do usuário, para fazer os relatórios de atendimento por hora, por unidade de atendimento, por usuário e por tipo de atendimento. Foi criado apenas um managed bean para suportar os quatro relatórios, para reproduzir os dados é utilizada a biblioteca *org.primefaces.model.chart*, porém os dados são recuperados do banco devidamente ordenados utilizando seus respectivos objetos de transferência de dados (DTO), e cada um possui uma página diferente.

O relatório de atendimento por data é criado em conjunto com o de hora, contudo, ele é criado em uma tabela, onde os dados também são agrupados e recebidos por um DTO. A tabela é exibida em ordem crescente pela quantidade de visitas no dia, sendo apresentado com a data de maior incidência de atendimento, seguida do dia da semana, e ao lado a quantidade de visitas. É possível reordenar utilizando os botões de ordenação do *PrimeFaces*.

O *Jasperreports* é utilizado apenas para o relatório individual, este foi solicitado pela segurança, e possui todas os dados de um visitante escolhido, informações pessoais, foto mais recente, suas informações especiais, o histórico de todas as suas visitas e de todos os seus atendimentos.

Para que os outros módulos pudessem suportar e conversar com o novo módulo, foi necessário algumas modificações. No módulo de visita surgiu os algoritmos de encaminhamento de visitantes, a inclusão da desativação e ativação

das informações especiais, e o botão de relatório individual na *dialog* que exibe o visitante.

O achados e perdidos foi totalmente repaginado, todo um novo fluxo foi implementado para monitorar melhor os objetos deixados nas UAs ou na recepção, para fazer isso foram alguns dados a classe. Um dos atributos se refere a quem encontrou o objeto, uma informação requisitada pela equipe da segurança. Outra informação incluída foi o local atual do objeto, para casos de cadastramento por unidades de atendimento, esse local pode ser modificado para “Em transporte para a segurança”, e para “Segurança”. Ao ser transportado para a segurança, o nome da pessoa que fará o transporte é informado. Por fim, ao devolver o objeto, o nome do visitante que o recebeu é adicionado ao registro.

Por fim, as permissões foram totalmente modificadas, antes específicas para ações, passaram a ser específicas de usuários. Anteriormente, permissão x autorizava a funcionalidade 1 e 2, e a permissão y concedia acesso a funcionalidade 4 e 5. Dessa forma, se um usuário precisasse das funcionalidades 2 e 5, obrigatoriamente deveria ter as permissões x e y. Com a modificação, os quatro tipos de usuários do sistema tem suas próprias permissões, que garantem a utilização de todas as funções necessárias. Por exemplo, para um diretor de UA, existe a permissão a, que tem as funcionalidades 1, 2 e 3, que são as funcionalidades que todos os diretores obtém acesso.

Quem controla a permissão dos usuários, é o spring security, que utiliza os *roles* para garantir ou negar o acesso às telas do sistema. Sendo necessário apenas um documento xml que associa uma tela a determinadas permissões.

3.4. Artefatos de software

Os artefatos de software são todos aqueles documentos produzidos durante a criação e o desenvolvimento do sistema. Por isso, são explicitados aqui alguns dos principais diagramas UML que ajudaram a fundamentar o sistema, e construir todo o sistema baseado em regras pré-definidas.

O diagrama de atividade é um diagrama de estados em que eles representam a execução de uma ação ou atividade. O diagrama representa todos

os estados que um visitante pode passar durante sua estadia dentro JFRN. (STADZISZ, 2002)

Foram representadas quatro raias: Visita, recepção, central de atendimento e unidade de atendimento. A raia de visita apresenta os estados que não envolvem atendimentos. A raia de recepção trata dos possíveis estados de um visitante enquanto se encontra na recepção.

O mesmo ocorre paras as raias de CA e UA, são seus possíveis estados enquanto estão na central de atendimento, e na unidade de atendimento, respectivamente. Os quadrados se tratam dos estados, e os losangos são as escolhas que definem qual o próximo estado do visitante.

Na Figura 17, é possível visualizar o diagrama de atividade, que tem como o primeiro estado o qual o visitante pode se encontrar, onde ocorrer a indagação sobre os locais a serem frequentados. Será verificado se o local informado é uma Unidade de atendimento. Se a resposta for negativa, o sistema o encaminhará para os locais pertinentes, e nenhum encaminhamento será realizado, o visitante entrará no estado “Em Visita” e assim deverá se manter até o fim da mesma. Em caso de resposta positiva, outra questão será levantada, se o local é 3ª Vara, 7ª Vara ou Turma Recursal. Sendo um deles, o visitante será encaminhado para a CA, de outra forma, será encaminhado para a respectiva UA.

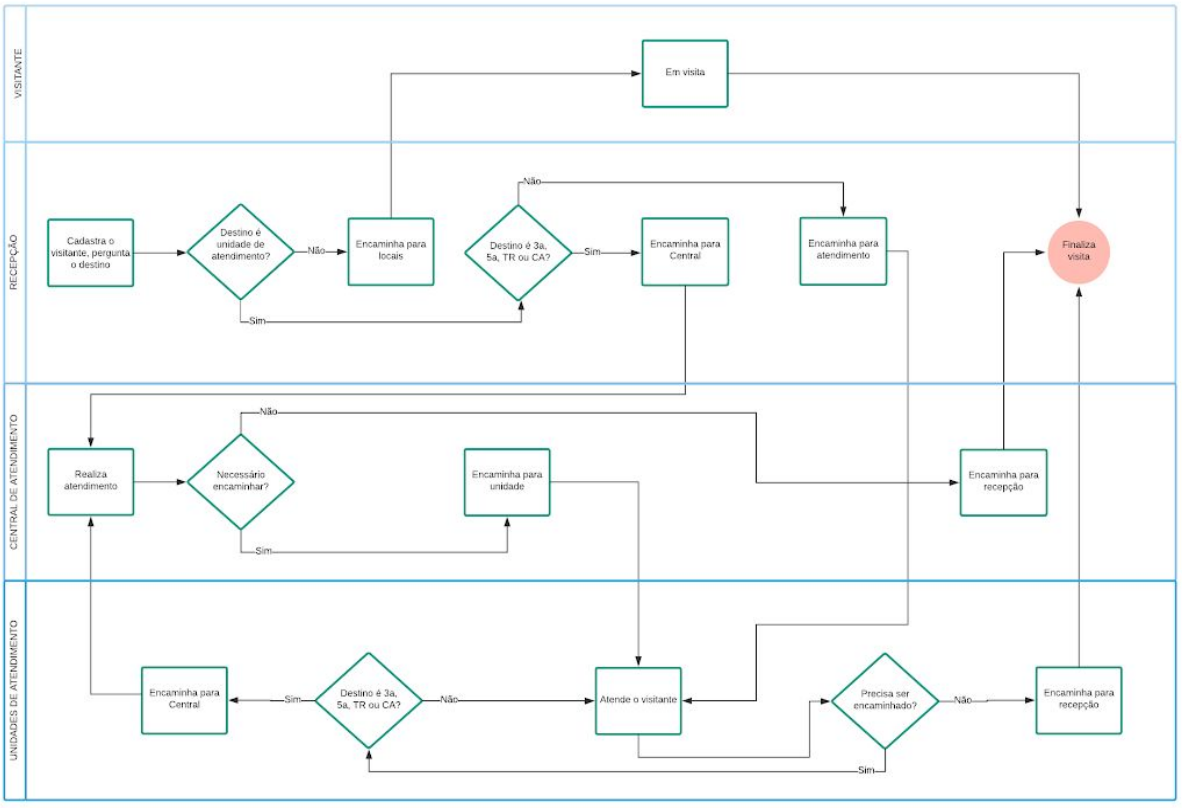


Figura 17 - Diagrama de Atividade

Apesar de estarem em raias diferentes, o fluxo seguido após o estado de atendimento do visitante é praticamente o mesmo. Vai ser verificado se há necessidade de encaminhamento, em caso positivo, segue para a mesmo estado de indagação, se o local é a 3ª ou 7ª vara, ou Turma Recursal, voltando a um estado anterior no fluxo, gerando um *loop* até que a resposta do encaminhamento seja negativa, o que levará o visitante a voltar a recepção e finalizar o atendimento.

Na Figura 18, é possível o diagrama de caso-de-uso do módulo, de acordo com STADZISZ (2002, p. 3), esse diagrama é “eficiente para determinação e documentação dos serviços a serem desempenhados pelo sistema”, além de ajudar na comunicação entre o desenvolvedor e o cliente. Neste diagrama existem dois elementos principais, os atores, que são entidades externas que a atuam diretamente no sistema, e o caso-de-uso, que descreve algum serviço oferecido pelo sistema.

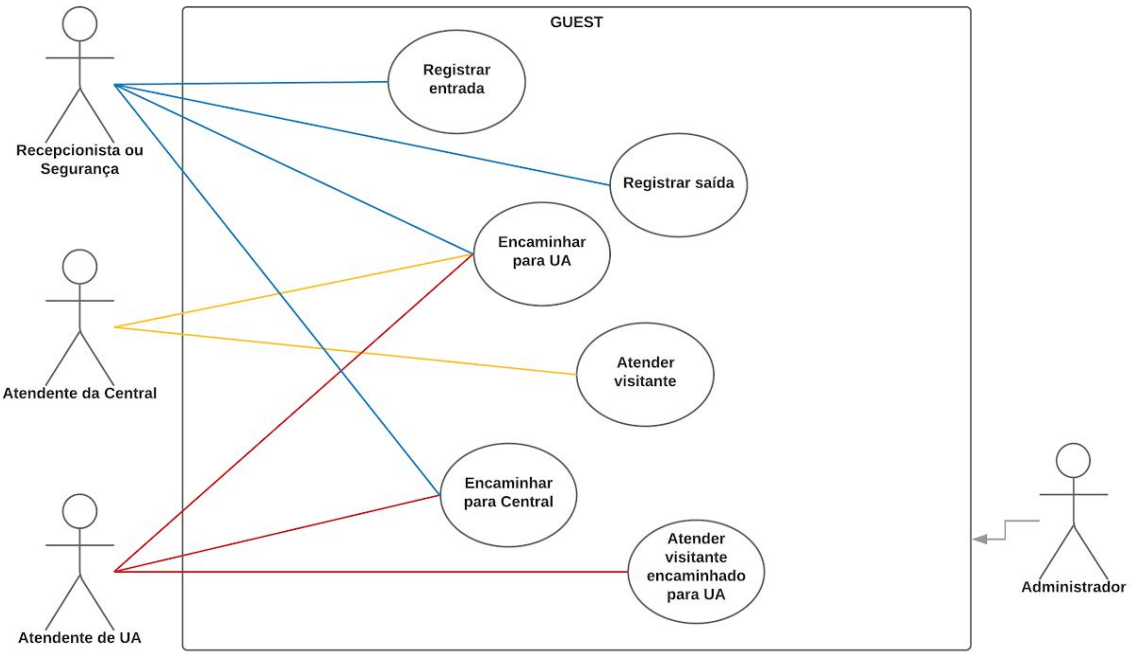


Figura 18 - Diagrama de Caso-de-Uso

Existem três tipos de relacionamento, o ator com ator, o caso de uso com caso de uso, e o ator com caso de uso. Este último sendo o único que foi utilizado para desenvolver o diagrama.

O referido é bem simples, existem quatro atores, cada um representa um tipo de usuário do sistema, e existem seis casos de uso. O administrador tem relacionamento com todos os casos de uso, ou seja, ele abrange todas as funcionalidades.

A recepcionista, ou o segurança, só podem registrar entrada e saída de visitantes, e, indiretamente, encaminhar para UA. O atendente da Central só poderá atender visitantes e encaminhar para alguma UA, e o atendente de UA tem um mesmo casos de uso do atendente da Central, que é a possibilidade de encaminhar para UA, com inclusão de dois novos serviços, sendo um deles atender visitantes que foram encaminhados para sua UA e o outro, encaminhar para a Central de Atendimento.

4. Conclusão

O desenvolvimento do presente módulo possibilitou realmente a evolução do sistema *Guest,* permitindo, não só criar os principais requisitos do sistema, como também, os requisitos de menor importância. Foi efetivamente implementado o módulo de atendimento, o fluxo do achados e perdidos, os diferentes tipos de encaminhamentos, e, ainda, os relatórios sobre os atendimentos.

O objetivo geral foi alcançado conforme o esperado, logo que foi criado um novo módulo que gerencia todos os atendimentos realizados por UAs, bem como o módulo de achados e perdidos possui uma estrutura mais robusta.

Também foi possível atingir todos os objetivos específicos, tais quais estão parcialmente exposto neste trabalho de conclusão de curso. A coleta de requisitos pode ser verifica no documento de requisitos no apêndice D, a fundamentação teórica apresenta a pesquisa de aprendizagem anterior a produção do módulo, também em apêndices estão os protótipos de telas do sistema e a criação do módulo está explicitado no capítulo de desenvolvimento.

A versão estável foi concluída e está, atualmente, em processo de inclusão e adaptação na JFRN, sendo utilizado apenas por algumas unidades de atendimento.

Apesar de tudo, a JFRN tem como propósito continuar evoluindo continuamente esse sistema. Para isso, é necessário que todas as tecnologias utilizadas se mantenham atualizadas para melhorar a manutenção do sistema para os próximos estagiários que virão a evoluí-lo.

Com o aumento da quantidade de usuários, o sistema pode implementar um botão de pânico para solicitar o auxílio do pessoal da segurança imediato, em casos de extrema urgência.

Assim, constatamos que as orientações feitas durante o decorrer do curso foram aproveitadas durante todo o desenvolvimento deste trabalho, tendo contribuído com elevado conhecimento na integração da teoria com a prática.

Referências

LUCIANO, J.; ALVES, W. Padrão de arquitetura MVC: Model-View-Controller. **EPeQ Fafibe,** São Paulo, 3ª. Ed., Vol. 01, 2011.

FARIA, F. **Java EE 7 com JSF, PrimeFaces e CDI**. Minas Gerais, AlgaWorks Softwares, Treinamentos e Serviços Ltda, 2013.

STADZISZ, P. **Projeto de Software usando a UML**. Centro Federal de Educação Tecnológica do Paraná, Paraná, 2002.

BALDUÍNO, T.; RUFINO, R. **Alto desempenho utilizando framework Hibernate e padrão Java Persistence Api**. Universidade Paranaense, Paraná, 2014.

BAWISKAR, A.; SAWANT, P.; KANKATE, V.; MESHRAM, B. **Spring Framework: A Companion to JavaEE**. IJCEM International Journal of Computational Engineering & Management, Vol. 15 Issue 3, 2012.

DIKANSKI, A.; STEINEGGER, R.; ABECK, S. **Identification and Implementation of Authentication and Authorization Patterns in the Spring Security Framework.** SECURWARE: The Sixth International Conference on Emerging Security Information, Systems and Technologies, 2012.

ANDRADE, T. PrimeFaces: Uma visão geral da tecnologia e do mercado, AlgaWorks Blog. Nov. 2016. Disponível em: <http://blog.algaworks.com/tecnologia-e-mercado-do-primefaces/>. Acesso em: 3 mar. 2018.

OTÁVIO, J. Introdução ao Scrum, DevMedia. Nov. 2015. Disponível em: <https://www.devmedia.com.br/introducao-ao-scrum/33724/>. Acesso em: 7 jul. 2018.

MARTINS, M. Relatórios em Java – JasperReports e iReport. K19 Treinamentos. Nov. 2010. Disponível em: <k19.com.br/artigos/relatorios-em-java-jasperreports-e-irepor/>

SOUSA, A.; OLIVEIRA, A.; GONÇALO, E.; MENDONÇA, J.; GALVÃO, S. **Manual Normativo de Trabalhos de Conclusão de Curso da UERN**, Universidade do Estado do Rio Grande do Norte, Mossoró, 2015.

MENESES, E. **Sistema de controle de clínica médica.** 2017. 168 f. Monografia. Universidade Cândido Mendes. São Paulo.

LINO, R. NOVO SGA 1.1.4: DOCUMENTAÇÃO - SOBRE, Novo SGA, Jul. 2013. Disponível em <http://doc.novosga.org/1.0/about.html>

Anderson, C. NOVO SGA 1.1.4: DOCUMENTAÇÃO - USANDO O NOVO SGA, Novo SGA, Jul. 2013. Disponível em <http://doc.novosga.org/1.0/using.html>

FOWLER, M.; RICE, D.; FOEMMEL, M.; HIEATT, E.; MEE, R.; STAFFORD, R. **Patterns of Enterprise Application Architecture.** Estados Unidos, Canadá. Addison-Wesley. 2003.

Apêndice A - Tabelas do Documento de Requisitos

1. Requisitos Funcionais

Iden.	Descrição	Prioridade
RF-01	Será necessário incluir dois novos tipos de usuário: ● Diretores ● Atendentes	essencial
RF-02	Diretores podem realizar atendimentos, cadastrar e desativar informações especiais, gerenciar UAs e TAs, e manipular objetos perdidos.	essencial
RF-03	Os atendentes tem como função apenas manipular objetos perdidos, realizar atendimentos e cadastrar informações especiais.	essencial
RF-04	O sistema deverá armazenar as seguintes informações de atendimento: ● Tipo do Atendimento ● Descrição do atendimento ● Unidade de Atendimento onde foi realizado ● O usuário que o realizou ● A data e a hora do atendimento ● Se houver, encaminhamento.	essencial
RF-05	Criação de novos atributos para a classe de Achados e Perdidos: ● Local atual do objeto ● A pessoa que recebeu o objeto no ato da devolução	essencial
RF-06	Encaminhamento para UAs pela recepção	essencial
RF-07	Apresentar um histórico de visitas e atendimentos	essencial
RF-08	Renderizar na tela de atendimento apenas os visitantes encaminhados para a lotação do usuário	essencial
RF-09	Renderizar na tela de atendimento de usuários lotados na CA, todos os visitantes atualmente em visita	essencial
RF-10	Tela de gerenciamento de UAs.	importante
RF-11	Tela de gerenciamento de TAs.	importante
RF-12	Ativação e desativação de informações especiais	desejável

RF-13	Filtro da lista de visitantes encaminhados por nome e documento	desejável
RF-14	Encaminhar visitante que são advogados para todas as UAs assim que uma visita for iniciada	desejável

2. Requisitos não-funcionais

2.1. Requisitos de Segurança

Iden.	Descrição	Prioridade
RNF-01	O visitante só pode ser atendido por uma UA diferente da CA, caso tenha sido encaminhado para tal..	essencial
RNF-02	Não permitir que ocorram primeiros atendimentos na 3ª Vara, 7ª Vara e Turma Recursal.	essencial
RNF-03	Não permitir que UAs possam realizar encaminhamento de advogados	desejável

2.2. Requisitos de Interface

Iden.	Descrição	Prioridade
RNF-04	O sistema deverá ser simples e intuitivo,	desejável
RNF-05	Deverá conter menus e botões de acesso que beneficiem a navegabilidade do usuário	desejável
RNF-06	O design da interface deve ser responsivo, se adaptando aos diferentes tamanhos de tela.	desejável

2.3. Requisitos de Operacionais

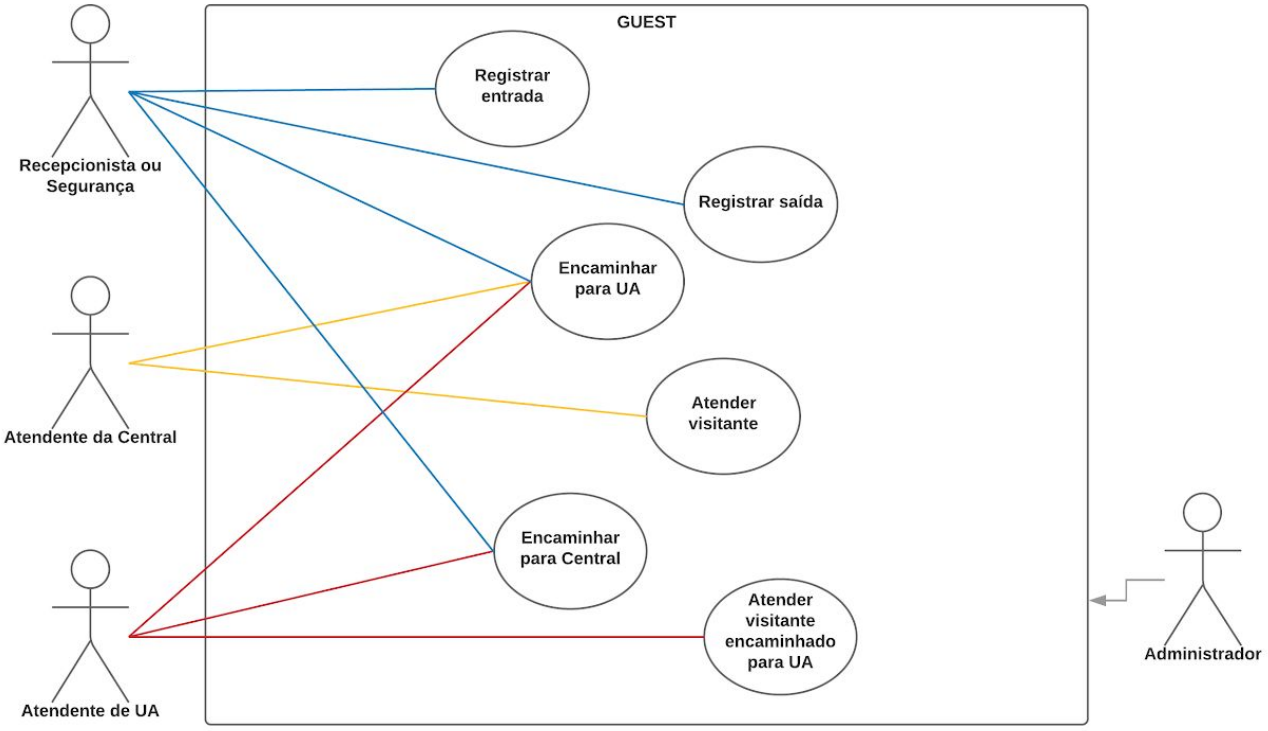
Iden.	Descrição	Prioridade
RNF-07	O sistema deverá ser desenvolvido na linguagem JAVA.	essencial
RNF-08	O sistema deverá ser desenvolvido em uma arquitetura em camadas (MVC).	essencial
RNF-09	O sistema deverá respeitar os padrões utilizados pelo *Guest*	importante

RNF-10	A camada de aplicação deve ser compatível com os principais navegadores.	desejável

2.4. Requisitos de Confiabilidade

Iden.	Descrição	Prioridade
RNF-11	O sistema deve estar disponível das 7 horas da manhãs às 18 horas, de segunda a sexta.	essencial
RNF-12	As manutenções e inserções de novas funcionalidades devem ser realizadas na versão de teste (homologação) e só depois de validadas serem disponibilizadas na versão para o público (produção).	desejável

Apêndice B - Explicação do Caso-de-Uso



- Registrar Entrada: É o ato de iniciar uma visita no sistema *Guest*. Só pode ser feito por recepcionistas, seguranças e administradores do sistema..

- Registrar Saída: É o ato de finalizar uma visita no sistema *Guest.* Só pode ser feito por recepcionistas, seguranças e administradores do sistema.

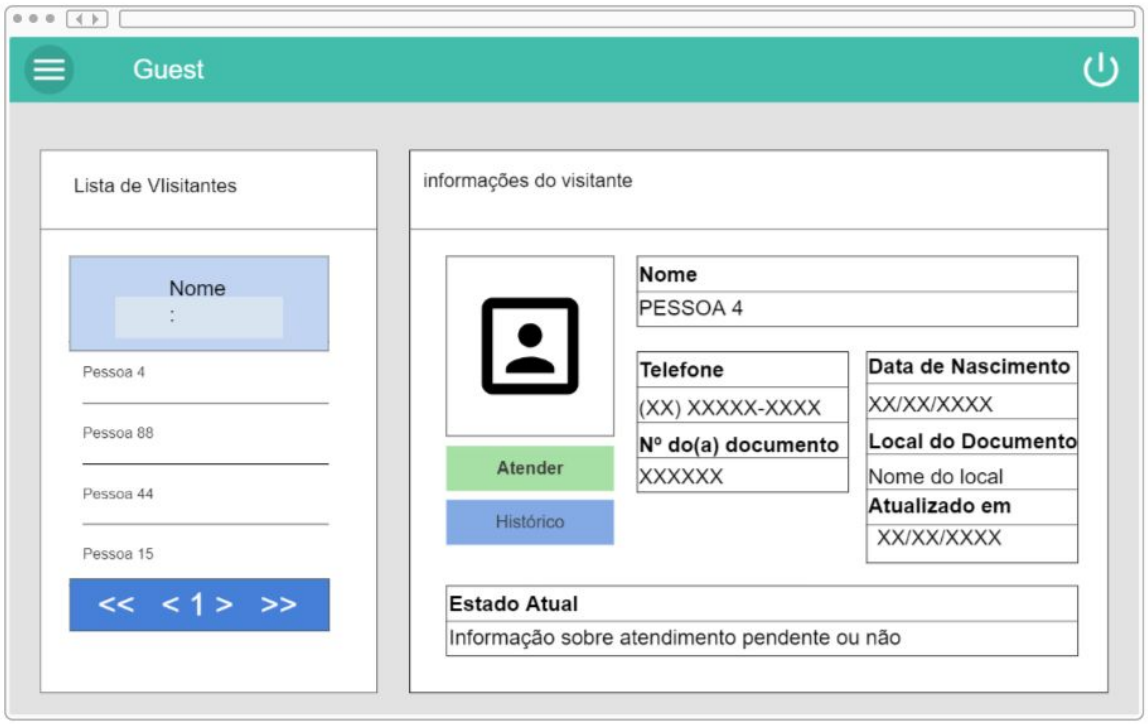
- Encaminhar para UA: É o ato de encaminhar um visitante para uma Unidade de Atendimento, permitindo que seu nome apareça na lista de espera. Pode ser realizado de forma automática, sem que os usuários tenham consciência, pelos seguranças, recepcionistas e administradores. E de forma totalmente explícita pelos atendentes da Central de Atendimento, atendentes de Unidades de Atendimento, e administradores.

- Atender visitante: É o ato de atender um visitante que não foi encaminhado para nenhum lugar. Só pode ser realizado por atendentes da Central de Atendimento e administradores do sistema.

- Encaminhar para Central: É o ato de encaminhar um visitante para a Central de Atendimento, podendo ser feito de forma inconsciente pelos seguranças, recepcionistas e administradores do sistema. E de forma proposital por atendentes das unidades de atendimentos e administradores.

- Atender visitante encaminhado para UA: É o ato de atender um visitante que foi encaminhado para uma Unidade de Atendimento. Pode ser realizado por atendentes de Unidades de Atendimento e Administradores.

Apêndice C - Protótipo de Tela de Atendimento



Apêndice D - Protótipo de Tela de Gerenciamento de UA

Apêndice E - Protótipo de Tela de Gerenciamento de TA