UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE SISTEMAS DE INFORMAÇÃO – BACHARELADO

CONTROLE DE ROTA PARA VIGILANTES UTILIZANDO NFC PARA VALIDAÇÃO DE PRESENÇA

ANDRÉ FELIPE RAULINO

BLUMENAU
2016

ANDRÉ FELIPE RAULINO

CONTROLE DE ROTA PARA VIGILANTES UTILIZANDO NFC PARA VALIDAÇÃO DE PRESENÇA

Trabalho de Conclusão de Curso apresentado ao curso de graduação em Sistemas de Informação do Centro de Ciências Exatas e Naturais da Universidade Regional de Blumenau como requisito parcial para a obtenção do grau de Bacharel em Sistemas de Informação.

Prof. Alexander Roberto Valdameri - Orientador

BLUMENAU
2016

CONTROLE DE ROTA PARA VIGILANTES UTILIZANDO NFC PARA VALIDAÇÃO DE PRESENÇA

Por

ANDRÉ FELIPE RAULINO

Trabalho de Conclusão de Curso aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II pela banca examinadora formada por:

Presidente:	Prof. Alexander Roberto Valdameri – Orientador, FURB
Membro:	Prof. Adriano Gonçalves Polidoro – FURB
Membro:	Prof. Miguel Alexandre Wisintainer – FURB

Blumenau, dia 6 de Novembro de 2016

Dedico este trabalho aos meus pais, meus avós e a todos que possibilitaram eu estar aonde estou hoje.

AGRADECIMENTOS

À minha família que sempre esteve ao meu lado

Aos meus amigos que me ajudaram na realização deste trabalho

Ao meu orientador que me guiou do início ao fim nesta jornada

Nas grandes batalhas da vida, o primeiro passo para a vitória é o desejo de vencer

Mahatma Gandhi

RESUMO

Atualmente a segurança é um tema em evidência na sociedade. Em se tratando se segurança empresarial, existem diversos métodos para se proteger, seja através de câmeras, alarmes, vigias, controles de acesso, entre outros. Enquanto uns realizam a tarefa de forma mais automatizada outros são feitos de uma forma mais tradicional, como é o caso dos vigilantes de ronda. Este trabalho apresenta o desenvolvimento de um protótipo de sistema de controle de rotas de vigias que utiliza a tecnologia NFC para realizar a validação de ponto de checagem. O sistema possui dois módulos. O Módulo web que permite um administrador cadastrar, pesquisar e monitorar registros de vigias, rotas, pontos de checagem, execuções de rondas e validações de pontos de checagem. E o módulo para dispositivos móveis que serve como agente validador utilizado pelos vigias para realizar validações de pontos de checagem. O trabalho utiliza Mentawai Framework, Java e PostgreSQL. O resultado foi um sistema que realiza corretamente todos os requisitos propostos mas possui uma obrigatoriedade de conexão com a internet para seu funcionamento, o que pode apresentar alguns problemas para o uso de vigias em geral, o desenvolvimento também gerou conhecimentos sobre a utilização do NFC.

Palavras-chave: Segurança. Controle de presença. NFC.

ABSTRACT

Security is currently a hot topic in society. When it comes to business security, there are several methods to protect yourself, whether through cameras, alarms, lookouts, access controls, among others. While some perform the task in a more automated way, others done it in a more traditional way, as is the case with the patrol officers. This work presents the development of a prototype of a point watch system that uses NFC technology to perform checkpoint validation. The system has two modules. The web module that allows an administrator to register, search and monitor logs, lookups, routes, checkpoints, rounds, and checkpoint validations. And the mobile module that serves as a validating agent used by patrol officer to perform checkpoint validations. The work uses Mentawai Framework, Java and PostgreSQL. The result was a system that correctly performs all the proposed requirements but has an obligatory connection to the Internet for its operation, which may present some problems for the use of patrol officer in general, the development also generated knowledge about the use of NFC.

Key-words: Safety. Presence control. NFC.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

CSS - Cascading Style Sheets

DOM - Document Object Model

HTML - HyperText Markup Language

HTTP - HyperText Transfer Protocol

IDE - Integrated Develeopment Environment

JSP - Java Server Pages

MDL - Material Design Lite

MVC - Model View Controller

NFC - Near-field Communication

RFID - Radio Frequency Identification

XML - Extensible Markup Language

SUMÁRIO

1 INTRODUÇÃO

Atualmente o ser humano vive um período de sua história onde a liberdade é vista como um direito do cidadão, e qualquer pessoa pode fazer o que deseja desde que esteja dentro das leis de seu estado. A liberdade tornou-se parte crucial da convivência humana e com ela o homem aprendeu a crescer e a se expressar. Porém, conforme Bauman (2001, p. 26) “A liberdade não parece oferecer riscos enquanto as coisas obedientemente seguem o caminho que desejamos”. A existência de indivíduos que fazem o uso indecente de sua liberdade cria uma necessidade da aplicação de outro direito do ser humano, o da segurança.

Atualmente existem diversos métodos para uma empresa ou instituição garantir que suas propriedades estejam seguras, seja através de câmeras, alarmes, vigias, controles de acesso, entre outros. Enquanto uns realizam a tarefa de forma mais automatizada outros são feitos de uma forma mais tradicional, como é o caso dos vigilantes de ronda. Esses profissionais tem a tarefa de seguir uma rota predefinida que cobre os principais perímetros dentro de uma propriedade, checando de forma preventiva a existência de alguma situação anormal que possui a possibilidade de causar danos ou perda de propriedade do contratante (JANES, 2009).

O componente mais vulnerável em um sistema de segurança é o recurso humano, portanto é aquele que precisa de mais atenção (PRIOSTE, 2014). Muitos proprietários se preocupam com a possibilidade de seus vigias não estarem cumprindo a sua responsabilidade de realizar a ronda, deixando assim as suas propriedades desprotegidas. Para mitigar esta preocupação foram criados sistemas que fazem o monitoramento destes seguranças, garantindo assim que o proprietário possua conhecimento das ações de seus funcionários. As soluções oferecidas exigem equipamentos específicos que possuem a única finalidade de agir neste sistema, tornando a aplicação destes métodos uma atividade mais custosa.

Baseado nestas informações, a proposta deste trabalho é criar um sistema que utilize a tecnologia Near Field Communication (NFC) dos celulares atuais para realizar a validação de rota dos vigilantes, oferecendo uma alternativa mais barata e prática em relação às soluções atuais.

1.1 OBJETIVOS

O objetivo geral do trabalho proposto é o desenvolvimento de um protótipo de sistema de controle de rondas de vigias que utiliza o NFC para a validação de rota.

Os objetivos específicos do trabalho proposto são:

a) oferecer uma alternativa aos modelos convencionais de controle de rota de vigilantes;
b) oferecer segurança e tranquilidade para donos de empresas que desejam proteger seus patrimônios através de vigilantes;
c) explorar a tecnologia NFC e desvendar suas particularidades.

1.2 ESTRUTURA

Esta monografia está organizada em capítulos. No primeiro capítulo é apresentada a introdução ao assunto da pesquisa e descritos os objetivos do trabalho. O segundo capítulo demostra a importância de sistemas automatizados de segurança e segurança patrimonial. É apresentada uma introdução ao NFC e a forma que ele opera. O terceiro capítulo descreve o desenvolvimento do protótipo proposto, que inclui os seus requisitos, as ferramentas e técnicas utilizadas, assim como o detalhamento do desenvolvimento dos diferentes módulos do sistema. Por fim, o quarto capítulo descreve as conclusões sobre a pesquisa e apresenta alternativas para o desenvolvimento de trabalhos futuros relacionados a este.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo aborda assuntos relacionados ao trabalho, tais como sistemas automatizados de segurança, vigias de ronda, NFC, além de trabalhos correlatos.

2.1 SISTEMAS AUTOMATIZADOS DE SEGURANÇA PATRIMONIAL

Segurança significa o estado, qualidade ou condição de quem ou do que está livre de perigos, incertezas, assegurado de danos e riscos eventuais; situação em que nada há a temer (HOUAISS, 2009). Sendo assim, a segurança se torna vital para qualquer um que corra riscos em relação a fatores externos que podem afetar suas propriedades.

Segundo Cruz (2009, p.02), “A preocupação com segurança há tempos vem ganhando espaço significativo em empresas dos mais diversos segmentos, tornando-se parte do negócio.” Empreendedores investem cada vez mais em segurança para garantir a continuidade de seus negócios.

O maior motivador destes investimentos é o medo da violência urbana que se torna comum em nossa sociedade, a exposição à violência causa aos habitantes transtornos de ordem psicológica e fisiológica, motivados pelo medo constante (CARDIA, 2003). Aplicando isso a uma grande propriedade observa-se a relevância de tantos investimentos, a perda ocasionada por um eventual furto, assalto ou agressão dentro do perímetro da propriedade pode acarretar em grandes perdas financeiras e morais para o proprietário, o dano moral é perigoso por atingir diretamente os valores do indivíduo “desassossego, desconforto, medo, constrangimento, angústia, apreensão, perda da paz interior, sentimento de perseguição ou discriminação, desestabilização pessoal, profissional, social e financeira” (GONÇALVES, 2003, p. 220). Por isso a responsabilidade de garantir a segurança patrimonial é uma parte importante dos deveres do proprietário.

Segurança patrimonial ou privada é de acordo com Daliga (2012, p. 8), “A atividade desenvolvida por entes privados por meio de pessoas contratadas, devidamente habilitadas, visando proteger o patrimônio e as pessoas, transportar valores e garantir o transporte de cargas.”. Empresas especializadas podem ser contratadas para realizar esta segurança, estas empresas possuem os profissionais devidamente treinados e os equipamentos necessários para garantir que a propriedade esteja segura, as empresas também possuem sistemas automatizados que auxiliam os funcionários na realização da segurança.

Bayer, Eckhardt e Machado (2011, p. 15) definem automação como “[...] a tecnologia que se ocupa da utilização de sistemas mecânicos, eletroeletrônicos e

computacionais na operação e controle da produção". Aplicando esta definição na segurança patrimonial, pode-se observar diversos sistemas que são aplicados pelas empresas de segurança. São eles: sistemas de circuito fechado de TV, que em conjunto com câmeras garantem informações em tempo real para o operador; controle de acessos, que incluem controle de pessoas, cartão de ponto acesso, entre outros. Estes evitam acesso de pessoas que não possuem permissão de entrada em lugares específicos; segurança de dados, nos últimos anos com o grande avanço das tecnologias, a segurança digital se tornou tão crucial quanto a segurança de bens físicos; existem sistemas automatizados que gerenciam os processos realizados pelos próprios vigias de ronda, como demonstrado no próximo tópico.

2.2 VIGIAS DE RONDA

Os "Rondantes" – também conhecidos como "guardas noturnos", "agentes de segurança comunitária", "vigias noturnos", "vigilantes noturnos autônomos" ou simplesmente "vigias" ou "guardas de rua", são segundo Lopes (2011, p. 104) profissionais que possuem a tarefa de realizar rondas dentro de um determinado local. Estas rondas servem para cuidar da segurança patrimonial da propriedade, eles devem estar preparados para lidar com roubos, incêndios, invasão de estranhos, depredação ou qualquer outro acidente danoso que possa resultar em perda da propriedade do contratante.

Estes vigias seguem rotas pré-determinadas que passem por perímetros específicos, estes perímetros geralmente ficam próximos de localidades mais afastadas ou que possuem um maior potencial de acidentes. Os vigias também possuem horários para realizar esta rota, horários que podem oferecer um maior risco de acidente, como a noite para perímetros onde a chance de invasão é maior, ou em horários de pico para equipamentos que possam sofrer um superaquecimento (SOUSA, 2013).

Porém, quem vigia os vigilantes? Uma possibilidade levantada pelos donos de propriedade é a de que os vigias poderiam não estar realizando as suas rondas, então, para garantir que estes vigias realizem suas tarefas eles são submetidos a um monitoramento. Segundo Maximiano (2006), o monitoramento "consiste em acompanhar e avaliar a execução da estratégia", assim, os administradores possuem o controle necessário sobre as ações do vigia, tendo conhecimento de qualquer falha na ronda ou alteração suspeita na rotina.

Segundo Bateman (2000), para o funcionamento correto de uma estratégia os resultados precisam estar coerentes com os objetivos, e para isto sempre é necessário algum monitoramento, este é realizado através de relógios de vigia. Estes tem a tarefa de registrar o

exato momento que o vigia passou pelo ponto de verificação, desta forma o administrador tem todo o histórico da rota do vigia, sabendo se ele realmente passou pelo perímetro no horário correto ou até se ele mentiu em dizer que realmente realizou a rota.

Relógios de vigia são mecanismos automatizados que fazem este registro de horários, a versão analógica deste aparelho funciona através de chaves espalhadas nos pontos de verificação, o vigia utilizava esta chave no relógio para registrar o horário, o relógio então utilizava um sistema de impressão para guardar este horário em um disco gráfico de papel, que após a ronda seria entregue ao administrador para que então ele verifique a validade da ronda (SOUSA, 2013).

A modernização desta tecnologia criou os bastões de controle de ronda. Os bastões realizam uma tarefa semelhante a dos relógios de vigia, porém de forma digital, pois coletam os horários quando estes são conectados com os *iButtons*, que agem de uma forma parecida com as chaves do relógio de vigia. Os bastões enviam os dados diretamente para um computador que possui o sistema que mostra o relatório das rondas para o administrador (SOUSA, 2013).

Outra forma, pouco explorada, de realizar este controle de rota é através do NFC dos celulares, uma tecnologia que será apresentada a seguir.

2.3 NFC

Near-field Communication (NFC) é um padrão definido pelo NFC fórum, um consórcio global de empresas de hardware, software/aplicações, companhias de cartões de créditos, bancos, provedores de internet, e outros que estão interessados no avanço e padronização desta tecnologia promissora (ORITZ, 2008, tradução nossa).

> O NFC é uma extensão do RFID (Radio Frequency Identification), tecnologia patenteada por Charles Walton na década de 80. Esta tecnologia permite a partilha de dados entre vários dispositivos móveis, através de frequências de rádio sem haver a necessidade de autenticação. Com esta tecnologia será possível usar o telemóvel para efetuar pagamentos, partilhar contactos/ficheiros, jogos, etc. Antunes, Bernardes e Almeida (2011, p. 1).

O NFC se difere do Radio Frequency Identification (RFID) por possuir um limite de 20 centímetros de utilização. Este limite é imposto para garantir que o NFC seja usado a partir do contato, aumentando assim sua praticidade e sua segurança. O trabalho de Antunes, Bernardes e Almeida (2011) também cita dois tipos de funcionamentos da tecnologia: ativo e passivo.

O Ativo, onde ambos os dispositivos geram e recebem o sinal de rádio. Neste um smartphone recebe a informação de um terminal (emissor) e em seguida emite a confirmação para o mesmo equipamento;

O Passivo, que funciona quando um dos dispositivos emite o sinal de radiofrequência e o segundo apenas recebe a informação. É o caso de um smartphone (receptor) lendo uma das *tags* NFC (emissor). A comunicação aqui é de apenas uma via, pois a *tag* não consegue receber um sinal, apenas emitir (NASSAR; HORN, 2014).

As *tags* NFC são pequenos adesivos que possuem um micro mecanismo que atua como agente passivo de NFC. Elas podem conter pequenas sequências de dados que podem ser utilizados para obtenção e envio de informação que pode ser gravada previamente por um agente ativo. Gallo (2011, p.05, tradução nossa) explica o funcionamento de uma *tag* NFC:

> Os dados de uma aplicação guardados dentro de uma *Tag* NFC são primeiramente encapsulados em uma mensagem NDEF e então na estrutura de dados especificada pela plataforma de tipo de *tag* do NFC. A encapsulação da mensagem NDEF e da plataforma de tipo de *tag* do NFC é utilizada para identificar o tipo dos dados da aplicação, como uma URL, vCard ou imagem JPEG, e para garantir a interoperabilidade e a coexistência entre aplicações.

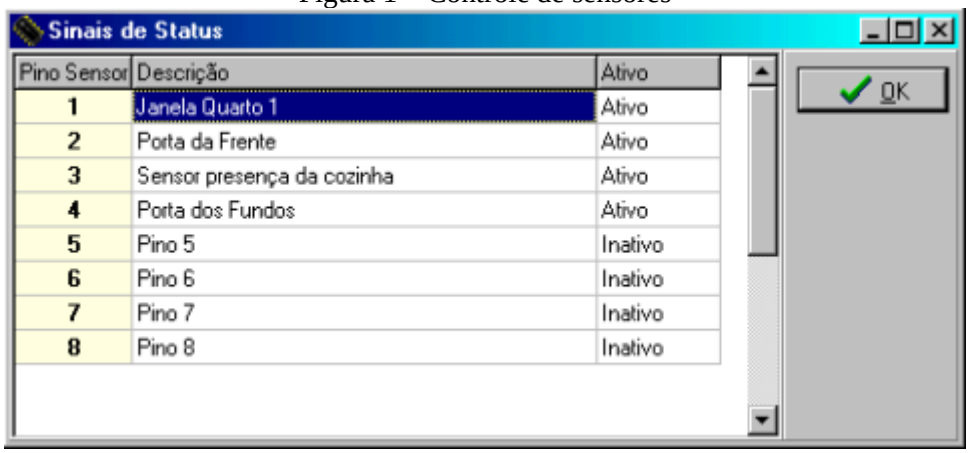
Desta forma, percebe-se que é possível à utilização de *tags* NFC como agentes validadores. Estas *tags*, mesmo tendo o funcionamento do tipo passivo podem trabalhar junto com um agente ativo de um celular e com um aplicativo previamente implementado com o intuito de gerar validações precisas e seguras.

2.4 TRABALHOS CORRELATOS

Podem-se citar como trabalhos correlatos as monografias realizadas pelos alunos Erasmo Krüger e Daniel Baumann para conclusão do curso de Ciência da Computação e Sistemas de Informação na Universidade Regional de Blumenau além do TopRonda que é um sistema de controle de vigilantes mantido pela TopData.

O trabalho de Krüger (2002) foi criar uma solução mais viável e flexível da utilização de técnicas de segurança residencial, especialmente quando aplicados em prédios, utilizando recursos da internet, o funcionamento de seu trabalho se baseava em cadastrar dispositivos que realizariam a segurança da casa. Na Figura 1 é apresentada a tela de controle de sensores, onde o usuário pode visualizar quais sensores estão ativos em seu sistema de segurança.

Figura 1 – Controle de sensores



Fonte: Krüger (2002).

A partir desses sensores, o SISM-NET de Krüger recebe todos os sinais enviados dos sensores para seus receptores, esse sinal então é enviado para o software, que gera um *log* com a data, hora e evento que foi detectada esta ocorrência. Na Figura 2 é apresentado o *log* gerado pelo SISM-NET, que pode ser salvo ou enviado por e-mail para o proprietário.

Figura 2 - Log gerado pelo SISM-Net

```
SISM-NET - Arquivo de LOG
Aplicativo desenvolvido por Erasmo Kruger para TCC - 2002/2
Orientador: Prof. Antonio Carlos Tavares
Data de Criação do arquivo: 07/11/02 23:19:48

07/11/02 23:19:48: Monitoramento Iniciado.
08/11/02 00:00:05: Programação por horário foi acionada: Alimentar Sensor de
Presença
08/11/02 06:58:33: Evento detectado no sensor: Monitoramento Quarto (Presença)
08/11/02 07:11:47: Evento detectado no sensor: Monitoramento Quarto (Presença)
08/11/02 07:18:41: Evento detectado no sensor: Monitoramento Quarto (Presença)
08/11/02 07:32:51: Evento detectado no sensor: Monitoramento Quarto (Presença)
08/11/02 07:43:28: Evento detectado no sensor: Monitoramento Quarto (Presença)
08/11/02 12:09:25: Evento detectado no sensor: Monitoramento Quarto (Presença)
08/11/02 12:58:58: Evento detectado no sensor: Monitoramento Quarto (Presença)
08/11/02 13:45:51: Evento detectado no sensor: Monitoramento Quarto (Presença)
08/11/02 14:07:45: Evento detectado no sensor: Monitoramento Quarto (Presença)
08/11/02 14:27:48: Evento detectado no sensor: Monitoramento Quarto (Presença)
08/11/02 14:57:08: Evento detectado no sensor: Monitoramento Quarto (Presença)
08/11/02 15:10:14: Evento detectado no sensor: Monitoramento Quarto (Presença)
08/11/02 15:16:26: Evento detectado no sensor: Monitoramento Quarto (Presença)
08/11/02 15:27:54: Evento detectado no sensor: Monitoramento Quarto (Presença)
08/11/02 15:56:12: Evento detectado no sensor: Monitoramento Quarto (Presença)
08/11/02 15:56:39: Monitoramento Finalizado.
```

Fonte: Krüger (2002).

O foco dele, semelhante ao apresentado nesse trabalho, é garantir uma solução mais prática e barata do que as já apresentadas no mercado, oferecendo informações para proprietários por e-mail, fazendo simulações de presença ligando luzes dentro de casa criando a ilusão que há alguém na propriedade, mandar mensagens para o celular do proprietário, além de uma funcionalidade parecida com a proposta: a possibilidade de checar se vigias estão realizando suas rondas no horário correto a partir dos logs.

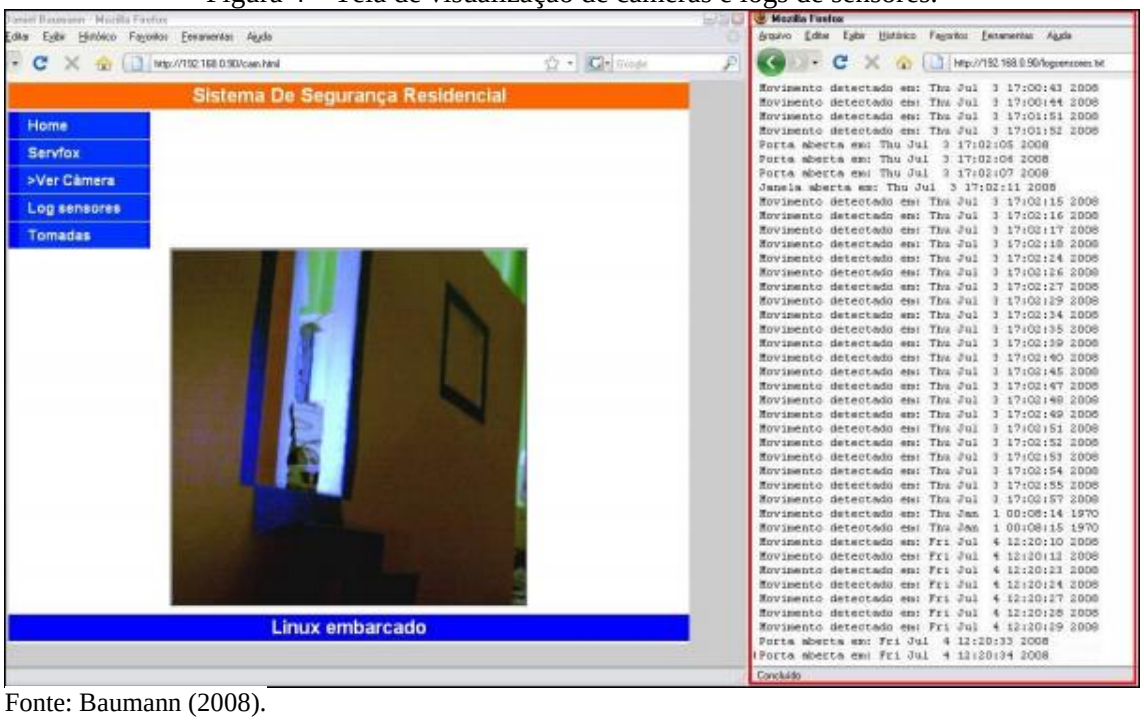
O trabalho de Baumann (2008) foi utilizar sistemas embarcados em Linux utilizando um hardware denominado Fox Board. Foi desenvolvido em uma versão nacional pelo professor e orientador Miguel Wisintainer. Na Figura 3 é apresentada a placa Fox Board nacional conectada à rede, câmera e circuito de tomadas e sensores, criado por Baumann(2008).

Figura 3 – Conexão de dispositivos a Fox Board nacional

Fonte: Baumann (2008).

Além deste dispositivo de segurança, Baumann (2008) também criou um protótipo de sistema web que acessa os dados retornados pelo dispositivo. Com este sistema é possível acessar em tempo real o conteúdo das câmeras, e aos *logs* de movimentação de cada uma delas. Na Figura 4 é apresentado o protótipo, demonstrando o acesso a uma câmera e o *log* de movimentação.

Figura 4 – Tela de visualização de câmeras e logs de sensores.

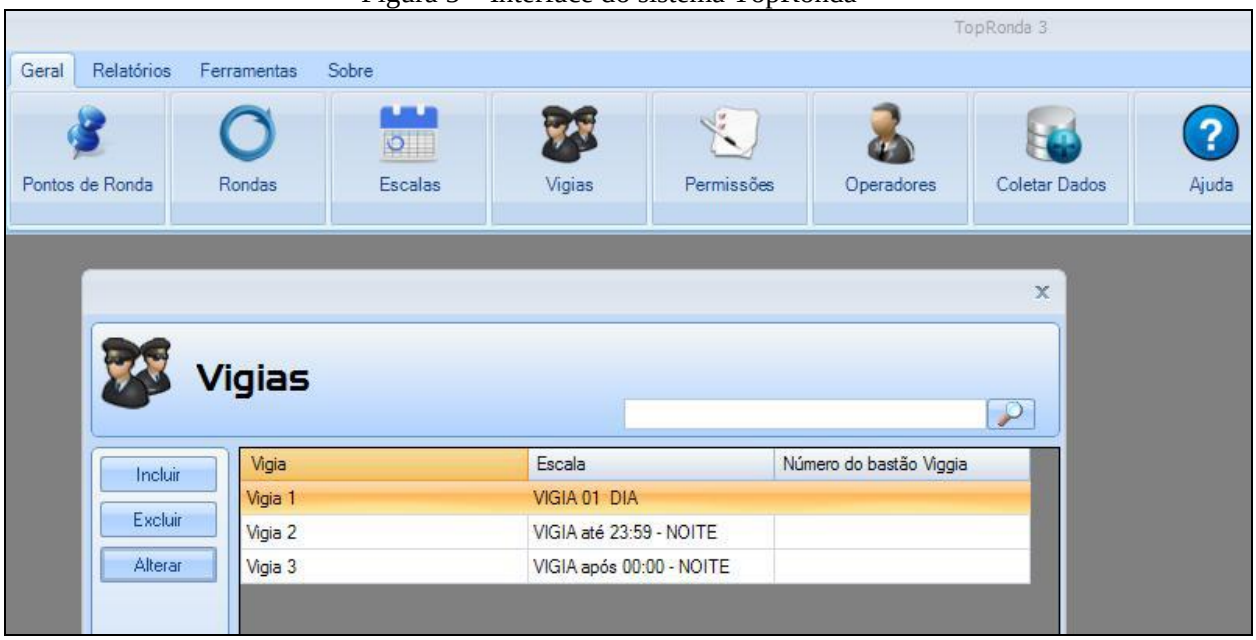


Fonte: Baumann (2008).

O foco de seu trabalho foi de apresentar uma versão alternativa para os sistemas de vigilância interna com o uso de câmeras, algo que já existia na época, mas, como apresentado neste trabalho, utiliza uma tecnologia diferente para atingir o mesmo objetivo.

O sistema de controle de vigias TopRonda que foi criado pela TopData é o software mais próximo do que o que é apresentado neste trabalho, ambos oferecem o mesmo controle de vigias, rondas e rotas, possuem funcionalidades semelhantes e utilizam um mesmo fluxo de trabalho que envolve a utilização de um aparelho especifico para realizar as validações realizadas pelos vigias. O TopRonda é um software instalável que é acessado pelo próprio computador, ao contrário do sistema web acessado pelo navegador que é apresentado neste trabalho. A Figura 5 mostra a interface do sistema TopRonda, é possível ver as opções disponíveis no menu e uma listagem dos vigias.

Figura 5 – Interface do sistema TopRonda



Fonte: Topdata (2016).

Outra diferença chave entre o TopRonda e o sistema desenvolvido é a tecnologia utilizada para a realização da validação do ponto de checagem pelo vigia, enquanto o sistema desenvolvido utiliza a tecnologia NFC de dispositivos móveis junto com *tags* NFC para validar, o TopRonda utiliza o bastão de ronda, este é uma tecnologia mais especifica e trabalhada para este caso e funciona em conjunto com botões inteligentes para realizar a validação. O bastão de ronda da TopData se chama bastão Viggia, ele é fabricado de forma durável e protegida, os botões inteligentes criados pela TopData se chamam i-buttons, o bastão realiza a validação por aproximação sem a necessidade de toque entre os dois, ele salva o horário das validações em uma memória interna que precisa ser descarregada em um

computador que possui o software através de uma porta USB encontrada no próprio bastão. A Figura 6 mostra o bastão Viggia e os i-buttons criador pela TopData.

Figura 6 – bastão Viggia e iButton

Fonte: Topdata (2016).

3 DESENVOLVIMENTO DO PROTÓTIPO

Neste capítulo são apresentadas as etapas de desenvolvimento dos módulos web e móvel do sistema. A seção 3.1 apresenta os Requisitos Funcionais (RF) e os Requisitos Não Funcionais (RNF) atendidos.

3.1 REQUISITOS

A seguir são detalhados os Requisitos Funcionais (RF) e Requisitos Não Funcionais (RNF) atendidos pelo protótipo desenvolvido.

a) permitir que o administrador faça log-in na aplicação web (RF);
b) permitir o administrador manter e monitorar um cadastro de vigias na aplicação web (RF);
c) permitir o administrador manter um cadastro de administradores na aplicação web (RF);
d) permitir o administrador manter e monitorar um cadastro de rotas na aplicação web (RF);
e) permitir o administrador manter e monitorar um cadastro de pontos de checagem na aplicação web (RF);
f) permitir o administrador monitorar uma listagem de execuções de ronda na aplicação web (RF);
g) permitir o administrador monitorar uma listagem de validações de pontos de checagem na aplicação web (RF);
h) permitir que o administrador defina permissões de acesso de outros administradores na aplicação web (RF);
i) permitir que o administrador defina um vigia como responsável por uma rota na aplicação web (RF);
j) permitir que o administrador adicione, ordene e delete pontos de checagem para rotas na aplicação web (RF);
k) permitir que o administrador pesquise nas listagens utilizando campos específicos dos cadastros na aplicação web (RF);
l) permitir que o administrador ordene de forma crescente ou decrescente as listagens a partir de um campo específico na aplicação web (RF);
m) permitir que o administrador gere um QR code na aplicação web para facilitar o log-in de um vigia ou administrador no aplicativo mobile (RF);

n) permitir que o administrador faça log-in informando os dados do servidor e seus dados pessoais no aplicativo mobile (RF);
o) permitir que o administrador faça log-in lendo um QR code gerado pelo servidor no aplicativo mobile (RF);
p) permitir que o administrador verifique os pontos de checagem que precisam ser preenchidos em uma *tag* NFC, no aplicativo mobile (RF);
q) permitir que o administrador selecione um ponto de checagem e preencha uma *tag* NFC ao aproximar o aparelho no aplicativo mobile (RF);
r) permitir que o vigia faça log-in informando os dados do servidor e seus dados pessoais no aplicativo mobile (RF);
s) permitir que o vigia faça log-in lendo um QR code gerado pelo servidor no aplicativo mobile (RF);
t) permitir que o vigia verifique as rotas sob sua responsabilidade que precisam ser executadas diariamente no aplicativo mobile (RF);
u) avisar ao vigia que uma de suas rotas está em atraso no aplicativo mobile (RF);
v) permitir o vigia iniciar uma execução de ronda em uma de suas rotas no aplicativo mobile (RF);
w) permitir o vigia verificar o atual ponto de checagem que precise ser validado no aplicativo mobile (RF);
x) permitir o vigia validar um ponto de checagem ao aproximar o aparelho de uma *tag* NFC no aplicativo mobile (RF);
y) permitir o vigia finalizar um execução de ronda ao validar seu último ponto de checagem no aplicativo mobile (RF);
z) permitir o vigia cancelar uma execução de ronda no aplicativo mobile (RF);
aa) utilizar Java para o *back-end* do servidor (RNF);
bb) utilizar o framework Mentawai para a utilização do padrão Model View Controller (MVC) (RNF);
cc) utilizar JavaScript, JQuery, HTML e CSS para implementar o *front-end* do servidor (RNF);
dd) utilizar o Banco de dados PostgreSql para armazenamento de dados no servidor (RNF);
ee) utilizar o Hibernate como framework de persistência para mapeamento das classes no servidor (RNF);
ff) utilizar LiteSQL para armazenar dados temporariamente no aplicativo móvel

(RNF);

gg) utilizar *JavaScript Object Notation* (JSON) para troca de informações entre o cliente e o servidor (RNF);

hh) executar o servidor no Apache Tomcat (RNF);

ii) permitir acesso ao servidor a partir de um navegador Chrome ou FireFox (RNF);;

jj) executar o cliente no sistema operacional Android (RNF);.

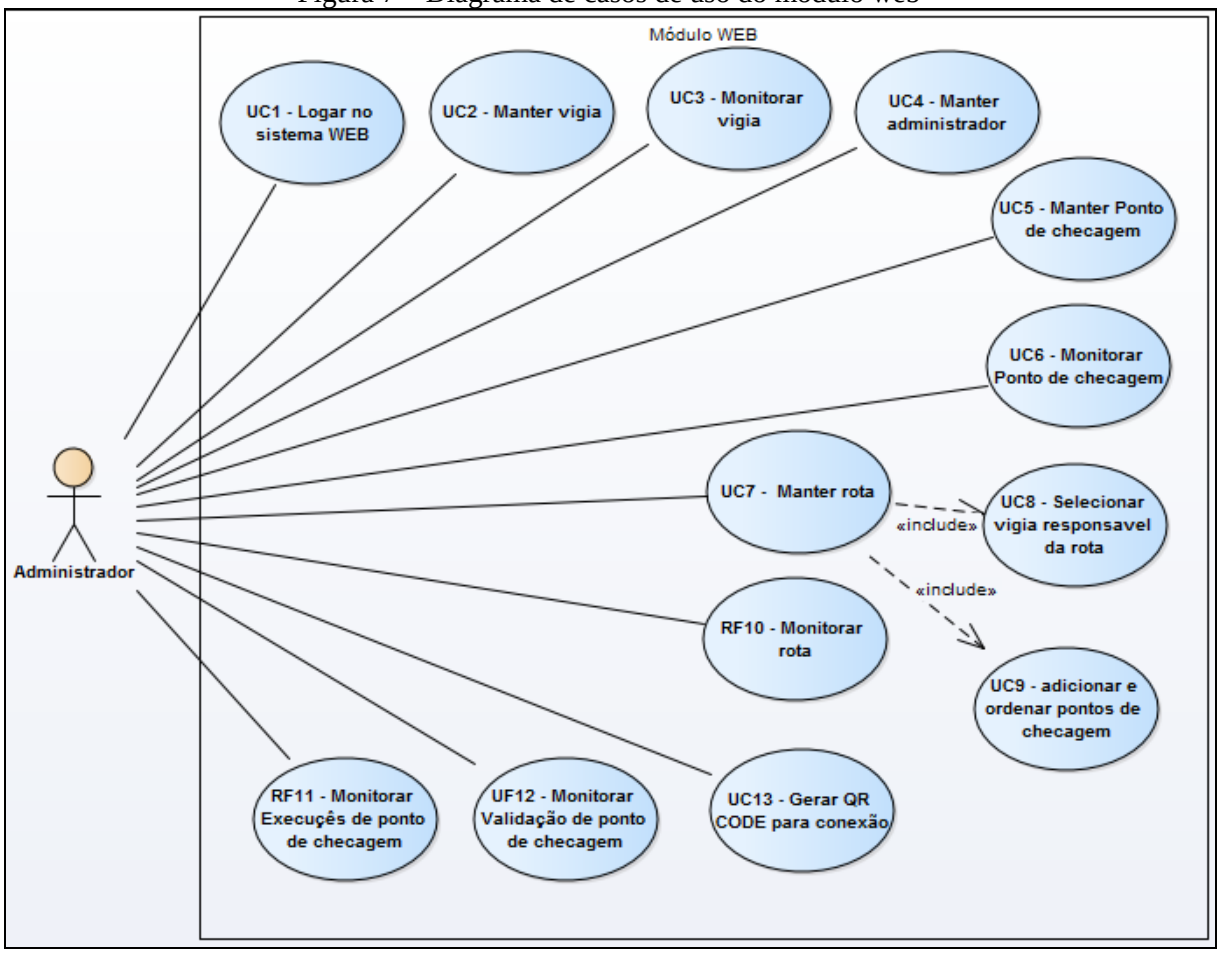
3.2 ESPECIFICAÇÃO

Esta seção apresenta os diagramas de casos de uso, bem como o diagrama de atividades e a matriz de rastreabilidade, sendo que a especificação dos principais casos de uso está descrita no Apêndice A.

3.2.1 Diagrama de casos de uso

Esta seção apresenta os casos de uso do servidor web e do cliente mobile, as descrições dos principias casos de uso encontram-se no Apêndice A. A Figura 7 apresenta as atividades realizadas pelo ator `administrador` no modulo web do sistema.

Figura 7 – Diagrama de casos de uso do modulo web

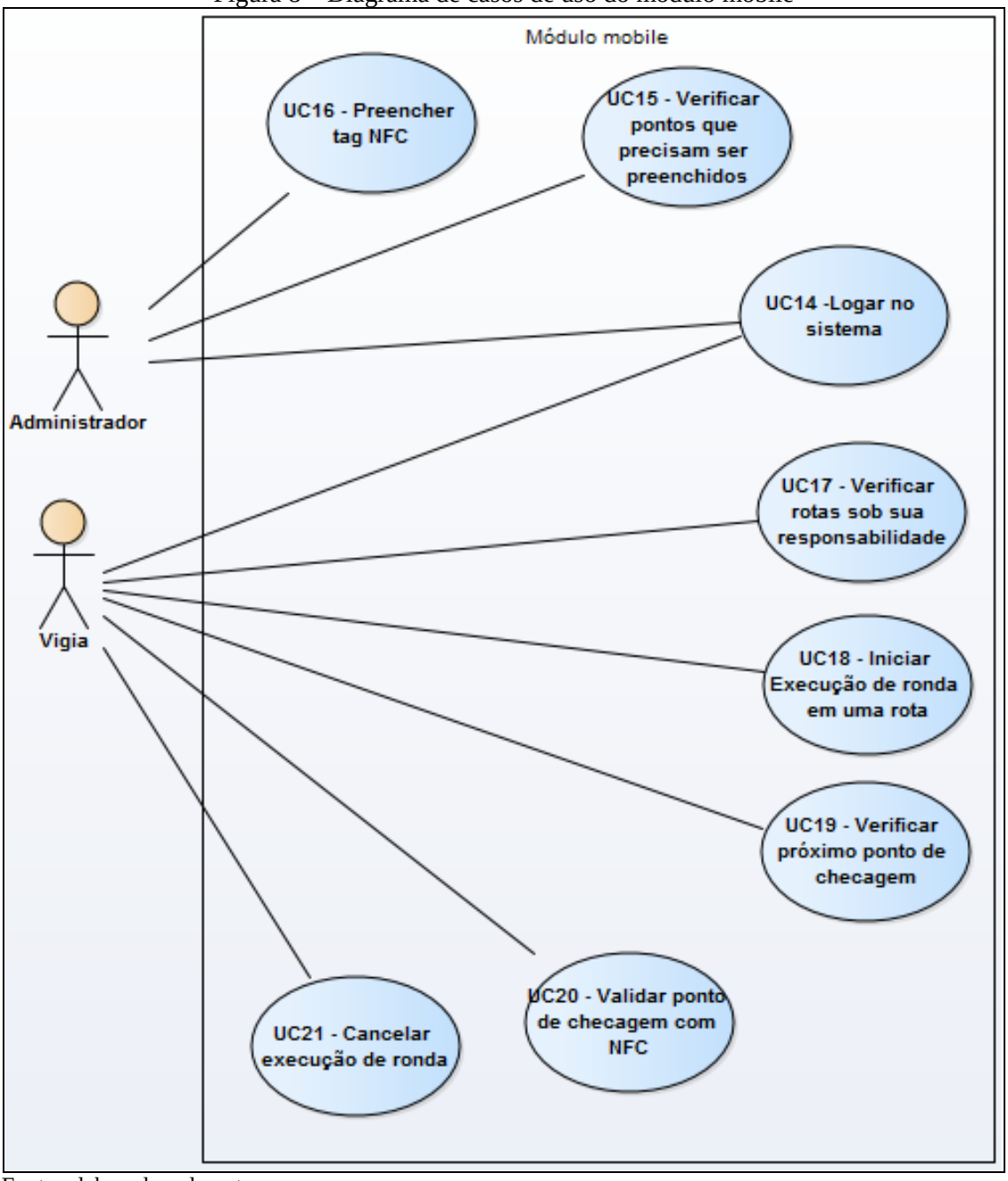


Fonte: elaborado pelo autor.

Apenas administradores tem acesso ao servidor, ali eles podem fazer cadastros de todas as tabelas do sistema, exceto execuções de rondas e validações de pontos de checagem, alguns desses cadastros necessitam de ações especiais como a definição de um vigia responsável e pontos de checagem para a rota, o administrador só terá acesso aos cadastros caso possua a permissão necessária para realizar esta ação.

O administrador pode também monitorar os cadastros de vigia, rota, ponto de checagem, execução de ronda e validação de ponto de checagem podendo assim acompanhar todas as ações que este cadastro realiza no sistema, é possível também gerar um QR code que pode ser lido pelo cliente para facilitar a conexão. A Figura 8 apresenta as ações realizadas pelo administrador e o vigia no módulo `mobile` do sistema.

Figura 8 – Diagrama de casos de uso do modulo mobile



Fonte: elaborado pelo autor.

Ambos, vigias e administradores, podem acessar o cliente mobile de duas formas, informando os dados da conexão e seus dados pessoais ou lendo o QR code gerado pelo servidor, o administrador só poderá se conectar se tiver a permissão concedida em seu cadastro. O administrador visualiza todos os pontos de checagem cadastrados pelo sistema que ainda não foram preenchidos e tem a opção de preencher uma *tag* NFC com um ponto de checagem selecionado. O vigia visualiza todos os as rotas sob sua responsabilidade e tem a opção de iniciar uma execução de ronda em uma destas rotas, durante a execução ele visualiza os pontos de checagem desta rota e tem a opção de validar um ponto de checagem ao

aproximar o aparelho de uma *tag*, ele também pode cancelar a execução de ronda que está sendo realizada. Estas informações são enviadas para o servidor para serem inseridas no banco de dados.

3.2.2 Matriz de rastreabilidade RF x UC

O Quadro 1 apresenta a matriz de rastreabilidade entre os requisitos da seção 3.1 e os casos de uso da Figura 7 e da Figura 8.

Quadro 1 - Matriz de rastreabilidade

Requisitos funcionais	**Casos de uso**
RF a: permitir que o administrador faça log-in na aplicação web	UC1
RF b: permitir o administrador manter e monitorar um cadastro de vigias na aplicação web	UC2 e UC3
RF c: permitir o administrador manter um cadastro de administradores na aplicação web	UC4
RF d: permitir o administrador manter e monitorar um cadastro de rotas na aplicação web	UC7 e RF10
RF e: permitir o administrador manter e monitorar um cadastro de pontos de checagem na aplicação web	UC5 e UC6
RF f: permitir o administrador monitorar uma listagem de execuções de ronda na aplicação web	UC11
RF g: permitir o administrador monitorar uma listagem de validações de pontos de checagem na aplicação web	UC12
RF h: permitir que o administrador defina permissões de acesso de outros administradores na aplicação web	UC4
RF i: permitir que o administrador defina um vigia como responsável por uma rota na aplicação web	UC8
RF j: permitir que o administrador adicione, ordene e delete pontos de checagem para rotas na aplicação web	UC9
RF k: permitir que o administrador pesquise nas listagens utilizando campos específicos dos cadastros na aplicação web	UC2, UC4, UC5 e UC7
RF l: permitir que o administrador ordene de forma crescente ou decrescente as listagens a partir de um campo específico na aplicação web	UC2, UC4, UC5 e UC7
RF m: permitir que o administrador gere um QR code na aplicação web para facilitar o log-in de um vigia ou administrador no aplicativo mobile	UC13
RF n: permitir que o administrador faça log-in informando os dados do servidor e seus dados pessoais no aplicativo mobile	UC14
RF o: permitir que o administrador faça log-in lendo um QR code gerado pelo servidor no aplicativo mobile	UC14
RF p: permitir que o administrador verifique os pontos de checagem que precisam ser preenchidos em uma *tag* NFC, no aplicativo mobile	UC15
RF q: permitir que o administrador selecione um ponto de checagem e preencha uma *tag* NFC ao aproximar o aparelho no aplicativo mobile	UC16
RF r: permitir que o vigia faça log-in informando os dados do servidor e seus dados pessoais no aplicativo mobile	UC14
RF s: permitir que o vigia faça log-in lendo um QR code gerado pelo servidor no aplicativo mobile	UC14
RF t: permitir que o vigia verifique as rotas sob sua responsabilidade que precisam ser executadas diariamente no aplicativo mobile	UC17
RF u: avisar ao vigia que uma de suas rotas está em atraso no aplicativo mobile	UC17
RF v: permitir o vigia iniciar uma execução de ronda em uma de suas rotas no aplicativo mobile	UC18
RF w: permitir o vigia verificar o atual ponto de checagem que precise ser validado no aplicativo mobile	UC19

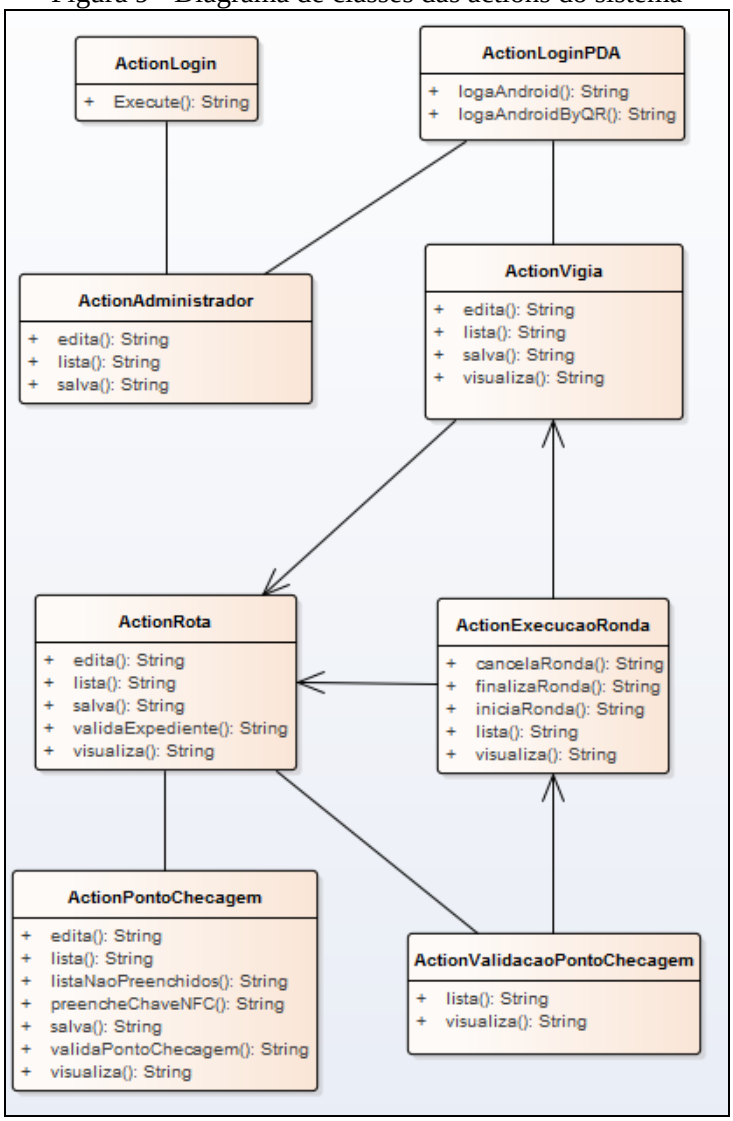
RF x: permitir o vigia validar um ponto de checagem ao aproximar o aparelho de uma *tag* NFC no aplicativo mobile	UC20
RF y: permitir o vigia finalizar um execução de ronda ao validar seu último ponto de checagem no aplicativo mobile	UC19
RF z: permitir o vigia cancelar uma execução de ronda no aplicativo mobile	UC21

Fonte: elaborado pelo autor.

3.2.3 Diagrama de classe

O protótipo desenvolvido neste trabalho utiliza o *framework* Mentawai para o gerenciamento da arquitetura MVC do sistema. Este *framework* utiliza o conceito de *actions* para controlar as ações de determinadas classes do sistema. Mais detalhes sobre o *framework* Mentawai será explicado na sessão do detalhamento da implementação. A Figura 9 demonstra o diagrama de classes e suas ações controladas pelas *actions*.

Figura 9 - Diagrama de classes das actions do sistema



Fonte: elaborado pelo autor.

Todos os métodos das classes do diagrama possuem retorno do tipo String, isto é devido a forma que o *framework* Mentawai controla os resultados das *actions*. As classes `ActionAdministrador`, `ActionVigia`, `ActionRota` e `ActionPontoChecagem` possuem os métodos genéricos de editar, listar e salvar, estes métodos são utilizados no servidor para facilitar as operações de cadastro e listagem dos dados. Segue uma breve descrição das classes listadas na figura NUMERO:

a) `ActionLogin`: utilizada para controlar a conexão dos administradores no servidor, o método Execute é criado automaticamente pelo Mentawai para facilitar a operação de login;
b) `ActionLoginPDA`: cuida das conexões feitas pelos dispositivos móveis ao servidor, ela possui métodos para conexão comum com login e senha, mas também possui a funcionalidade de fazer uma conexão a partir da leitura de um QR code pelo dispositivo móvel;
c) `ActionAdministrador`: controla operações realizadas com o cadastro de administrador;
d) `ActionVigia`: controla operações realizadas com o cadastro de vigia, também possui o método visualiza que permite o monitoramento das ações do vigia no sistema;
e) `ActionRota`: controla operações realizadas com o cadastro de rota, o método `validaExpediente` calcula o horário de início e termino durante o cadastro da rota e verifica se o horário informado respeita o horário de expediente do vigia, também possui o método visualiza que permite o monitoramento das ações realizadas nesta rota no sistema;
f) `ActionPontoChecagem`: controla operações realizadas com o cadastro de ponto de checagem, o método `listaNaoPreenchidos` é utilizado pelo administrador no dispositivo móvel para listar pontos de checagem que ainda não tiveram seu código NFC inserido em uma *tag*, o método `preencheChaveNFC` é chamado quando o administrador aproxima o dispositivo de uma *tag* vazia após selecionar a opção de preencher uma *tag* no dispositivo, o método `validaPontoChecagem` é chamado pelo vigia ao aproximar o dispositivo de uma *tag* enquanto executa uma ronda, também possui o método visualiza que permite o monitoramento das ações realizadas neste ponto de checagem no sistema;
g) `ActionExecucaoRonda`: possui um método para a listar as execuções realizadas no

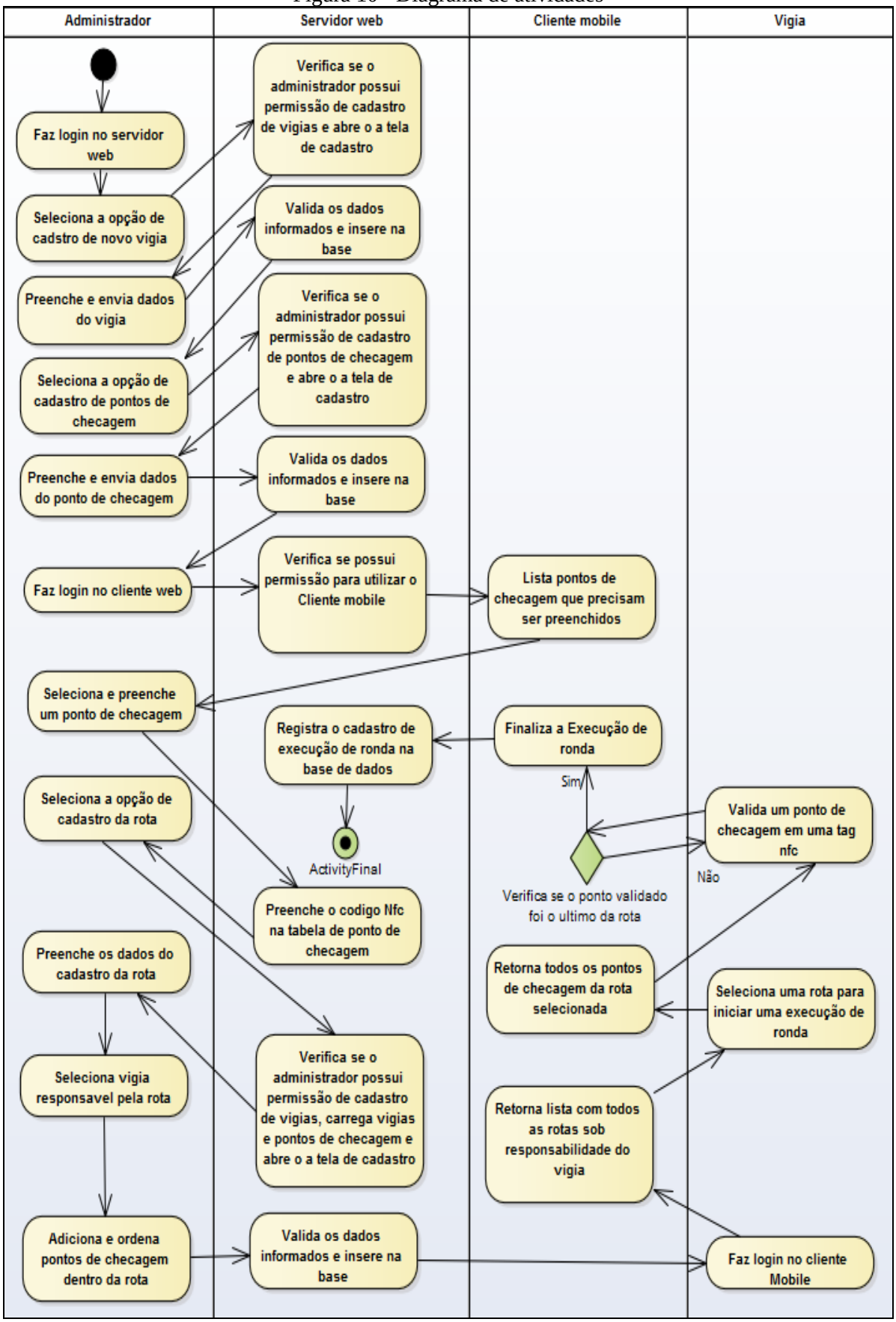
servidor, os métodos de controle da ronda `iniciaRonda` e `cancelaRonda` podem ser chamados pelo vigia através de seu dispositivo móvel, o método `finalizaRonda` é chamado automaticamente depois que o vigia valida o último ponto de uma ronda, também possui o método visualiza que permite o monitoramento das ações realizadas neste execução de ronda no sistema;

h) `ActionValidacaoPontoChecagem`: possui um método para a listar as validações realizadas no servidor, também possui o método visualiza que permite o monitoramento das validações de ponto checagem no sistema.

3.2.4 Diagrama de atividades

Nesta seção é apresentado o diagrama de atividades para a realização de uma execução de ronda em uma nova rota, incluindo cadastros necessários de vigias e pontos de checagem, passando pelos dois módulos do sistema. A Figura 10 apresenta o diagrama de atividades que descreve os processos para realizar uma execução de ronda.

Figura 10 - Diagrama de atividades



Fonte: elaborado pelo autor.

Inicialmente o administrador deve realizar o *login* no servidor web. O servidor web deve verificar se o administrador tem a permissão necessária para a criação de um novo vigia, caso ele possua o servidor deve mostrar *link* para o cadastro para o administrador. O administrador seleciona o *link* e então a opção para cadastrar um novo vigia. O administrador preenche os dados necessários e cadastra o vigia. O servidor web verifica se os dados foram preenchidos corretamente e então cria um novo vigia O administrador então acessa o cadastro de pontos de checagem, caso possua a permissão, e preenche os dados necessários. Em seguida deve selecionar a opção para gravar, o servidor web então verifica se os dados estão preenchidos corretamente e caso sim cria um novo cadastro de ponto de checagem.

O administrador então deve realizar *login* no aplicativo móvel. O aplicativo móvel realiza o *login* junto ao servidor e checa se o administrador possui a permissão necessária para acessar o aplicativo. O aplicativo móvel lista todos os pontos de checagem que precisam ser preenchidos em uma *tag* NFC. O administrador seleciona o ponto de checagem desejado e aproxima o aparelho da *tag*. O aplicativo móvel faz a inserção de um código na *tag* e envia este código para o servidor registrar no cadastro de ponto de checagem.

O administrador seleciona no servidor a opção de cadastro de rota, o servidor web verifica a permissão. Caso o administrador tenha a permissão, o servidor web carrega todos os vigias e pontos de checagem cadastrados no sistema. A partir disso apresenta a tela de cadastro de rota para o administrador. O administrador então preenche todos os dados necessários e seleciona um vigia. O administrador adiciona os pontos de checagem desejados para a rota. O administrador tem então a opção de ordenar estes pontos de checagem em qualquer ordem desejada. O administrador seleciona a opção de gravar a rota, onde então o servidor insere o cadastro na base de dados.

O vigia realiza *login* no aplicativo móvel. O aplicativo móvel retorna ao vigia uma listagem com todas as rotas sob sua responsabilidade. O vigia seleciona a rota desejada e inicia uma execução de ronda. O aplicativo móvel mostra o ponto de checagem que precisa ser validado. O vigia aproxima o aparelho da *tag* NFC. O aplicativo móvel registra a validação de ponto de checagem. Ao validar o último ponto de checagem o aplicativo móvel envia a confirmação de termino da execução da rota para o servidor.

3.3 IMPLEMENTAÇÃO

A seguir são mostradas as técnicas e ferramentas utilizadas, assim como a operacionalidade da implementação.

3.3.1 Técnicas e ferramentas utilizadas

O sistema foi desenvolvido e testado em um computador com o sistema operacional Windows 10. O Integrated Develeopment Environment (IDE) utilizado para a programação do modulo do servidor foi o Eclipse Mars release 4.5.2. A ferramenta de acesso ao banco de dados utilizada foi o pgAdmin 1.20. Para rodar o sistema de testes foi utilizado o servidor Apache Tomcat v7.0. O navegador utilizado para testes foi o Firefox versão 49.

Para o modulo móvel a IDE utilizada foi o Android Studio 2.1.2. Foi necessário a utilização de um aparelho Motorola Moto X geração 1 com Android 5.1 e duas *tags* NFC Smartrac para a realização dos testes. O aplicativo NFC tools foi utilizado para testar a leitura e a escrita realizada nas *tags*.

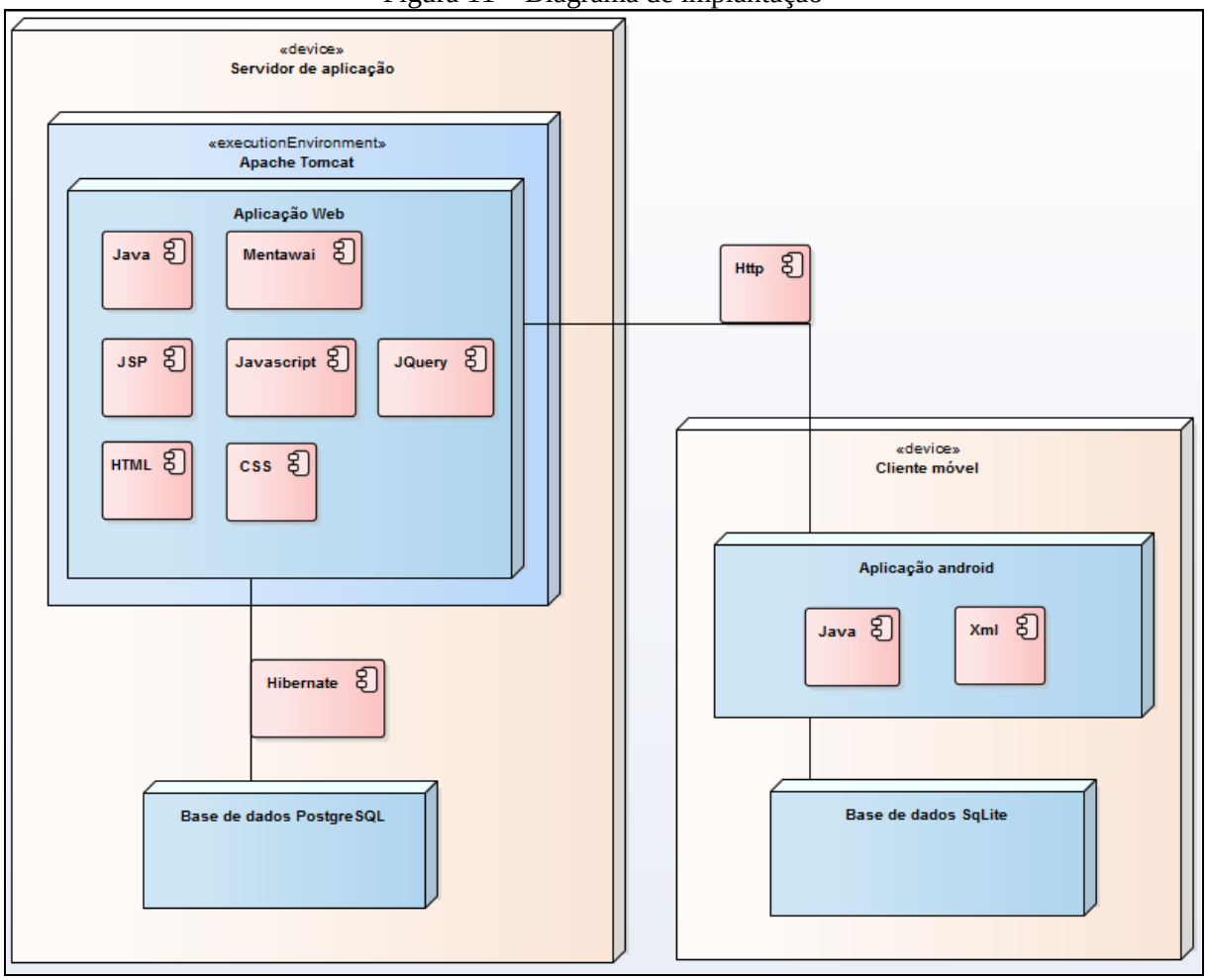
O *back-end* do modulo web foi desenvolvido em Java 8. O padrão MVC foi mantido para organizar a infraestrutura do sistema.Com vistas a facilitar a aplicação do MVC foi utilizado *framework* brasileiro Mentawai Framework. O mentawai foi escolhido para o desenvolvimento pela facilidade oferecida de acessar funções do sistema a partir de páginas web, sem necessidade de configurações via Extensible Markup Language (XML) como visto em outros *frameworks* semelhantes. Ainda, ele também oferece ferramentas úteis como controle de sessão e controle de permissão para acesso de páginas.

As páginas que compõem o *front-end* do módulo web o sistema foram feitas com Java Server Pages (JSP) e Cascading Style Sheets (CSS) para estruturação da página e javascript para rodar os scripts necessários, para o controle dos elementos Document Object Model (DOM) da página foram utilizados o jQuery e o jQuery UI. A interface foi criada seguindo o padrão Material design criado pelo Google, para ajudar na aplicação do Material design foi utilizado o Material Design Lite (MDL) que é um *template front-end* utilizado para facilitar a implementação da interface, foram usados plug-ins do jQuery para criação de mascaras em campos e para geração de QR code.

O aplicativo móvel foi desenvolvido em Java utilizando o padrão de *activities* comum do android. A interface foi feita em XML, utilizando uma base de dados temporária em LiteSql. A conexão dos dois módulos do sistema foi feita através de mensagens JSON enviadas via HyperText Transfer Protocol (HTTP).

O sistema desenvolvido necessita de pelo menos um computador para agir como servidor de aplicação e um dispositivo móvel para atuar como agente ativo de validação NFC. Vários dispositivos móveis conectando no mesmo servidor de aplicações não devem atrapalhar o desempenho do sistema em geral, mas uma conexão de dados fraca durante uma execução de ronda pode interferir drasticamente o tempo de envio e recebimento das requisições HTTP tornando o sistema lento. Na Figura 11 demonstra-se a partir de um diagrama de implantação a interação entre os módulos do servidor web e o cliente móvel e as tecnologias empregadas em cada um deles.

Figura 11 – Diagrama de implantação



Fonte: elaborado pelo autor.

O Mentawai é um *framework open-source* e se baseia na utilização de *actions* para gerenciar as chamadas HTTP vindas das páginas web e convertendo em uma ação dentro das classes java do sistema. A utilização do Mentawai depende da utilização de uma classe chamada `ApplicationManager`, dentro desta classe é necessária a configuração da *actions* das classes que serão controladas pelo Mentaway. O Quadro 2 mostra a configuração de uma *action* dentro do `ApplicationManager`.

Quadro 2 – Configuração de *action* mentawai

```
32  action("/Rota", ActionRota.class, "lista")
33      .on(SUCCESS, fwd("/listagens/listaRotas.jsp"));
```

Fonte: do autor.

Esta ação utilizara o método `lista()` dentro da classe Java `ActionRota` assim que o navegador fizer uma requisição para a página `/Rota.lista.mtw`. O Quadro 3 mostra a forma que a requisição de uma *action* deve ser feita na página web.

Quadro 3 – Chamada da listagem

```
210 <mtw:form name="pesquisaForm" action="Rota.lista.mtw" method="post">
211     <tr>
212         <td style="padding: 0px;">
213             <table>
214                 <tr>
215                     <td style="padding: 0px;">
216                         <div class="mdl-textfield mdl-js-textfield mdl-textfield--floating-label">
217                             <mtw:select klass="mdl-textfield__input"
218                                 id="pesquisar" name="pesquisar" list="listaPesquisa"
219                                 extra="onchange='validateTipPes()'"/>
220                             <label class="mdl-textfield__label"
221                                 for="pesquisar">Pesquisar por</label>
222     <!--  .......  -->
```

Fonte: do autor.

A *tag* `<mtw:form action="Rota.lista.mtw" method="post">` é uma *tag* disponibilizada pelo próprio Mentawai, com ela é possível fazer uma chamada HTTP para uma *action* através do atributo `action`, o atributo `method` permite o envio por GET ou POST, todos os inputs que estão dentro desta *tag* se tornam parâmetros para envio para a `ActionRota`. O Quadro 4 mostra a implementação do método `lista()` dentro da `ActionRota`.

Quadro 4 - implementação do método `lista()` dentro da `ActionRota`

```
public String lista() {
    String pesquisaPor = input.getString("pesquisar");

    RotaJpaDAO rotaDAO = RotaJpaDAO.getInstance();

    ArrayList<Rota> rotas= new ArrayList<>();

    if(pesquisaPor == null || pesquisaPor.equals("todos"))
        rotas = (ArrayList<Rota>) rotaDAO.findAll();
    else{
        output.setValue("pesquisaPor", pesquisaPor);

        if(pesquisaPor.equals("idrota")){
            if(input.getString("valorInt").equals(""))
                rotas = (ArrayList<Rota>) rotaDAO.findAll();
            else
                rotas = (ArrayList<Rota>) rotaDAO.findAllByParameterInt(pesquisaPor, input.getString("valorInt"));
        }else if(pesquisaPor.equals("horarioiniciorota") || pesquisaPor.equals("horariofimrota")){
            rotas = (ArrayList<Rota>) rotaDAO.findAllByParameterInt(pesquisaPor,
                    Time.valueOf(input.getString("valorHor" )+":"+ input.getString("valorMin")+":00").toString());
        }else if(pesquisaPor.equals("vigiaresponsavelrota_idVigia")){
            VigiaJpaDAO vigiaDAO = VigiaJpaDAO.getInstance();
            Vigia vigia = vigiaDAO.getById(Long.parseLong(input.getString("valorRes")));
            rotas = (ArrayList<Rota>) rotaDAO.getByResponsavel(vigia);
        }else
            rotas = (ArrayList<Rota>) rotaDAO.findAllByParameter(pesquisaPor, input.getString("valorTxt"));
    }

    VigiaJpaDAO vigiaDAO = VigiaJpaDAO.getInstance();

    ArrayList<Vigia> vigias = (ArrayList<Vigia>) vigiaDAO.findAll();

    output.setValue("listaHoras",horaSimpleList());
    output.setValue("listaMinutos",minutoSimpleList());
    output.setValue("listaPesquisa", campoPesquisaItem());

    SimpleListData sld = new SimpleListData();

    for(Vigia vig : vigias){
        sld.add(vig.getIdVigia().toString(), vig.toString());
    }

    output.setValue("listaVigias", sld);

    output.setValue("rotas", rotas);

    output.setValue("usuario", (Administrador) getSessionObj());
    return SUCCESS;
}
```

Fonte: do autor.

O retorno desta classe precisa ser a constante `SUCCESS` para que o usuário seja redirecionado para a página `/listagens/listaRotas.jsp`. O Mentawai permite o envio e o recebimento de parâmetros via HTTP para essas classes a partir dos comandos `input.getString("nome do campo")` e `output.setValue("nome do campo", objeto)`.

As *actions* de listagem utilizadas possuem a tarefa de listar todos os objetos e pesquisar eles de acordo com filtros disponibilizados, o comando `input.getString("pesquisar")` é o responsável por buscar um parâmetro enviado pela página web que informa qual dos campos deve ser pesquisado de acordo com o valor informado.

A partir desta estrutura o Mentawai consegue fazer qualquer tipo de interação entre as classes Java e as páginas web. Outros exemplos da utilização de *actions* no sistema são as

ações de envio de dados no cadastro e de monitoramento das classes. O Quadro 5 mostra o método `salva()` dentro da `ActionPontoChecagem`.

Quadro 5 - implementação do método `salva()` dentro da `ActionPontoChecagem`

```
103    public String salva() {
104        PontoChecagem pontoChecagem;
105        PontoChecagemJpaDAO pontoChecagemDAO = PontoChecagemJpaDAO.getInstance();
106
107        if(input.getString("id").isEmpty()){
108            pontoChecagem = new PontoChecagem();
109        }else{
110            Long id = Long.parseLong(input.getString("id"));
111            pontoChecagem = pontoChecagemDAO.getById(id);
112        }
113
114        String nome = input.getString("nome");
115        String descricao = input.getString("descricao");
116        String observacao = input.getString("observacao");
117
118        pontoChecagem.setNomePontoChecagem(nome);
119        pontoChecagem.setDescricaoPontoChecagem(descricao);
120        pontoChecagem.setObservacaoPontoChecagem(observacao);
121
122        pontoChecagemDAO.merge(pontoChecagem);
123
124        output.setValue("usuario", (Administrador) getSessionObj());
125        return SUCCESS;
126    }
```

Fonte: do autor.

O método mostrado é responsável pelo cadastro de novos pontos de checagem ou a atualização de um já cadastrado. O Quadro 6 mostra o método `visualiza()` dentro da `ActionPontoChecagem`.

Quadro 6 - implementação do método `visualiza ()` dentro da `ActionPontoChecagem`

```
public String visualiza() {

    if(input.getString("id") != null && !input.getString("id").isEmpty()){
        Long id = Long.parseLong(input.getString("id"));

        PontoChecagemJpaDAO pontoChecagemDAO = PontoChecagemJpaDAO.getInstance();
        PontoChecagem pontoChecagem = pontoChecagemDAO.getById(id);

        RotaPontoChecagemJpaDAO pontoRotaChecagemDAO = RotaPontoChecagemJpaDAO.getInstance();
        List<Rota> rotaPontoChecagem = pontoRotaChecagemDAO.findAllByPonto(pontoChecagem);

        ValidacaoPontoChecagemJpaDAO validacaoPontoChecagemDAO
            = ValidacaoPontoChecagemJpaDAO.getInstance();
        ArrayList<ValidacaoPontoChecagem> validacaoPontoProvisa
            = (ArrayList<ValidacaoPontoChecagem>) validacaoPontoChecagemDAO.findAll();
        ArrayList<ValidacaoPontoChecagem> validacaoPontoChecagem = new ArrayList<>();

        for(ValidacaoPontoChecagem val : validacaoPontoProvisa){
            if(val.getPontoValidado().getPontoChecagem().getIdPontoChecagem()
                    .equals(pontoChecagem.getIdPontoChecagem())){
                validacaoPontoChecagem.add(val);
            }
        }

        output.setValue("id", pontoChecagem.getIdPontoChecagem());
        output.setValue("nome", pontoChecagem.getNomePontoChecagem());
        output.setValue("descricao", pontoChecagem.getDescricaoPontoChecagem());
        output.setValue("observacao", pontoChecagem.getObservacaoPontoChecagem());
        output.setValue("rotas", rotaPontoChecagem);
        output.setValue("validacoesPontoChecagem", validacaoPontoChecagem);
    }

    output.setValue("usuario", (Administrador) getSessionObj());
    return SUCCESS;
}
```

Fonte: do autor.

Este método é responsável por retornar os dados de um ponto de checagem junto com todas as validações de ponto de checagem e rotas vinculadas com o mesmo. Todas os *outputs* são enviadas para a página web, não importando o tipo do dado. O próprio Mentawai consegue manter o formato dos dados nos dois lados da aplicação.

Além do controle das *actions*, o Mentawai oferece outras utilidades para as páginas WEB, uma delas é o controle de sessão que foi utilizado no sistema web, para utilizar esta funcionalidade foi necessário utilizar um tipo especial de *action*, chamado de `ActionLogin`. O Quadro 7 mostra a configuração da `ActionLogin` dentro do `ApplicationManager`.

Quadro 7 – Configuração da `ActionLogin`

```
28  action("/Login", ActionLogin.class)
29  .on(SUCCESS, redir("/Login.goHome.mtw"))
30  .on(ERROR, fwd("/login.jsp"));
```

Fonte: do autor.

Após a configuração da *action* foi necessária a implementação de um filtro, um *filter* é um equivalente de um *action* na infraestrutura do Mentawai, porem o *filter* sempre é executado independente de chamadas HTTP, o mentawai já possui um filtro de autenticação

implementado, mas foi preciso configurar ele no `ApplicationManager`. O Quadro 8 mostra a configuração do filtro de autenticação dentro do `ApplicationManager`.

Quadro 8 – Configuração do filtro de autenticação

```
19 @Override
20 public void loadFilters() {
21     filter(new AuthenticationFilter());
22     on(LOGIN, redir("/login.jsp"));
23 }
```

Fonte: do autor.

Com o filtro configurado todas as requisições das *actions* são bloqueadas e redirecionadas para a página de *login*. A partir desta tela o usuário poderá informar seus dados de acesso. Caso os dados sejam válidos um novo objeto é enviado para a sessão, enquanto este objeto está na sessão o mentawai permite acesso aos objetos. O Quadro 9 mostra como foi implementado este passo.

Quadro 9 – início de sessão

```
AdministradorJpaDAO administradorDAO = AdministradorJpaDAO.getInstance();
Administrador administrador = null;
try {
    administrador = administradorDAO.getByLogin(user, pass);
} catch (Exception e) {
    addError("Usuário não encontrado!");
    return ERROR;
}
```

Fonte: do autor.

Devido a forma que os filtros trabalham no Mentawai, identificou-se um problema nas ações que devem ser acessadas exclusivamente pelo dispositivo móvel. Como o dispositivo usa um controle de sessões próprio do Android, o Mentawai não identifica a chamada como uma sessão válida e retorna a página de *login* para o dispositivo, gerando um problema de conexão. Felizmente existe uma forma de bloquear a autenticação do filtro que é a adição do método `bypassAuthentication()` nas *actions* destinadas ao módulo android. O Quadro 10 mostra a utilização do método `bypassAuthentication()`.

Quadro 10 – método `bypassAuthentication()`

```
67 action("/Vigia", ActionVigia.class, "retornaListaRotas")
68     .bypassAuthentication()
69     .on(SUCCESS, ajax(new JsonRenderer()));
```

Fonte: do autor.

Outra funcionalidade do Mentawai que foi utilizada é o controle de acesso, o administrador tem a opção de definir permissões de acesso para um novo administrador criado no sistema, estas permissões podem bloquear o novo administrador se ele tentar entrar em uma área do sistema que ele não deve acessar. O Mentawai faz este controle na forma de grupos de acesso que são definidos na hora do *login* junto com a sessão, e depois lidos na

página web para então verificar se o usuário logado atualmente possui o grupo de permissão necessário para acessar a página. O Quadro 11 mostra a implementação que foi feita na `LoginAction()` para carregar os grupos de permissões que o administrador possui.

Quadro 11 – Carrega grupos de permissão

```
62 if(administrador.isPodeVigiaAdministrador())
63     vigia = "vigia";
64 if(administrador.isPodeAdministradorAdministrador())
65     administradorStr = "administrador";
66 if(administrador.isPodeRotaAdministrador())
67     rota = "rota";
68 if(administrador.isPodePontoAdministrador())
69     ponto = "ponto";
70 if(administrador.isPodeExecucaoAdministrador())
71     execucao = "execucao";
72 if(administrador.isPodeValidadoAdministrador())
73     validado = "validado";
74 if(administrador.isPodePDAAdministrador())
75     PDA = "PDA";
76
77 setSessionGroups(vigia, administradorStr , rota , ponto , execucao , validado , PDA);
```

Fonte: do autor.

Para garantir que o usuário não possa acessar uma área do sistema, a retirada dos *links* dos menus foi um passo importante. Para isso foi necessário utilizar *tags* que fazem a verificação dos grupos de permissão definidos no *login* Essas *tags* são usadas para bloquear algum conteúdo específico dentro de uma página, neste caso foram bloqueados apenas os *links* de acesso. O Quadro 12 mostra a implementação do menu lateral e as *tags* utilizadas para ocultá-los caso o usuário não possua permissão.

Quadro 12 – Menu lateral e *tags* de acesso

```
<nav class="demo-navigation mdl-navigation mdl-color--blue-grey-800">
    <a class="mdl-navigation__link"
        style="color: rgba(255, 255, 255, 0.56)" href="Login.goHome.mtw"">Página
        inicial</a> <mtw:hasAuthorization groups="vigia">
        <a class="mdl-navigation__link"
            style="color: rgba(255, 255, 255, 0.56)" href="Vigia.lista.mtw"><i
            class="mdl-color-text--blue-grey-400 material-icons"
            role="presentation">person</i>Vigias</a>
    </mtw:hasAuthorization> <mtw:hasAuthorization groups="administrador">
        <a class="mdl-navigation__link"
            style="color: rgba(255, 255, 255, 0.56)"
            href="Administrador.lista.mtw"><i
            class="mdl-color-text--blue-grey-400 material-icons"
            role="presentation">supervisor_account</i>Administradores</a>
    </mtw:hasAuthorization> <mtw:hasAuthorization groups="rota">
        <a class="mdl-navigation__link"
            style="color: rgba(255, 255, 255, 0.56)" href="Rota.lista.mtw"><i
            class="mdl-color-text--blue-grey-400 material-icons"
            role="presentation">playlist_play</i>Rotas</a>
    </mtw:hasAuthorization> <mtw:hasAuthorization groups="ponto">
        <a class="mdl-navigation__link"
            style="color: rgba(255, 255, 255, 0.56)"
            href="PontoChecagem.lista.mtw"><i
            class="mdl-color-text--blue-grey-400 material-icons"
            role="presentation">check_circle</i>Pontos de checagem</a>
    </mtw:hasAuthorization> <mtw:hasAuthorization groups="execucao">
        <a class="mdl-navigation__link"
            style="color: rgba(255, 255, 255, 0.56)"
            href="ExecucaoRonda.lista.mtw"><i
            class="mdl-color-text--blue-grey-400 material-icons"
            role="presentation">playlist_add_check</i>Execução de Ronda</a>
    </mtw:hasAuthorization> <mtw:hasAuthorization groups="validado">
        <a class="mdl-navigation__link"
            style="color: rgba(255, 255, 255, 0.56)"
            href="ValidacaoPontoChecagem.lista.mtw"><i
            class="mdl-color-text--blue-grey-400 material-icons"
            role="presentation">done</i>Validados</a>
    </mtw:hasAuthorization> <mtw:hasAuthorization groups="PDA">
        <a class="mdl-navigation__link"
            style="color: rgba(255, 255, 255, 0.56)"
            href="Login.goQR.mtw"><i
            class="mdl-color-text--blue-grey-400 material-icons"
            role="presentation">adb</i>QR code</a>
    </mtw:hasAuthorization> <a class="mdl-navigation__link"
        style="background: rgb(198, 40, 40); color: rgba(255, 255, 255, 0.56)"
        href="Logout.mtw" onclick="return confirma()">Sair</a>
</nav>
```

Fonte: do autor.

A verificação de acesso dos grupos é feita a partir da *tag* `<mtw:hasAuthorization>`. O grupo passado para ela como parâmetro precisa estar escrito na sessão para que o conteúdo dentro desta *tag* seja mostrado. Apesar desta estratégia bloquear a visibilidade de um *link* para o usuário, este ainda pode realizar uma requisição sem o *link*, escrevendo o *Uniform Resource Locator* (URL) da *action* e executando a ação manualmente. Porém o Mentawai disponibiliza um bloqueio na própria *action* garantindo que qualquer acesso não desejado seja redirecionado. O Quadro 13 mostra este bloqueio implementado.

Quadro 13 – Bloqueio de acesso da *action*

```
87  action("/Rota", ActionRota.class, "lista")
88      .authorize("rota")
89      .on(ACCESSDENIED, redir("/Login.goHome.mtw"))
90      .on(SUCCESS, fwd("/listagens/listaRotas.jsp"));
```

Fonte: do autor.

Para facilitar a construção das listas de dados no sistema foi utilizada uma biblioteca de *tags* chamada Display tag library 1.2, com essa livraria é possível integrar listas vindas do Mentawai para as páginas web utilizando apenas um atributo. O display *tag* é altamente configurável e controla automaticamente diversos aspectos importantes das listagens do sistema, como paginação, ordenação crescente de decrescente e criação de *links* que criam chamadas HTTP.

Durante o cadastro de rotas o usuário precisa selecionar um vigia para ser o responsável pela rota e um horário de início e término diário para esta rota. A partir de então o sistema precisa validar se o horário informado encaixa com o horário de expediente do vigia selecionado. Assim, o horário de início da rota não pode ser menor do que o horário de início de expediente do vigia e o horário de término da rota não pode ser maior do que o horário de término do expediente do vigia. O problema ocorre pois, ambos os horários da rota e do vigia são salvos sem uma data definida pois eles devem ser horários diários. Para resolver a questão foi feita uma função para checar se o horário da rota está entre o horário de início e fim do expediente do vigia, porém, considerando que existiram vigias noturnas, que iniciam em um dia e termina em outro, os horários podem acabar se confundindo na hora do cálculo que define se um é maior que o outro. A solução encontrada foi o uso de dois cálculos, um para testar se o horário do início da rota está entre o início do expediente e o fim do expediente, e outro para testar se o horário de fim da rota está entre o horário de início da rota e o fim do expediente. Ao passar por estas duas condições a função retorna um valor que é tratado e interpretado pela página web como um sinal para continuar suas operações normalmente. Caso algum dos horários falhe em alguma das condições a função retorna uma mensagem de erro que deve ser mostrada pela página web ao usuário. O Quadro 14 mostra a implementação da função de validação de expediente.

Quadro 14 – Função de validação de expediente

```
338    public String validaExpediente() {
339        VigiaJpaDAO vigiaDAO = VigiaJpaDAO.getInstance();
340        Vigia vigia = vigiaDAO.getById(Long.parseLong(input.getString("vigia")));
341
342        String horaInicio = input.getString("horaInicio");
343        String horaFim = input.getString("horaFim");
344        String minutoInicio = input.getString("minutoInicio");
345        String minutoFim = input.getString("minutoFim");
346
347        String inicioRota = horaInicio+":"+minutoInicio+":00";
348        String fimRota = horaFim+":"+minutoFim+":00";
349
350        String inicioExpediente = vigia.getHorarioInicioExpedienteString();
351        String fimExpediente = vigia.getHorarioFimExpedienteString();
352
353        try {
354            if(isTimeBetweenTwoTime(inicioExpediente, fimExpediente, inicioRota)){
355                if(isTimeBetweenTwoTime(inicioRota, fimExpediente, fimRota)){
356                    output.setValue("retorno", "false");
357                    return SUCCESS;
358                }else{
359                    output.setValue("retorno", "O horário informado não "
360                            + "respeita o horário de expediente de "+ vigia.getNomeVigia() +" "
361                            + "("+vigia.getHorarioInicioExpedienteString() +" - "
362                            + vigia.getHorarioFimExpedienteString() +")");
363                    return SUCCESS;
364                }
365            }else{
366                output.setValue("retorno", "O horário informado não respeita o horário de expediente de "
367                    + vigia.getNomeVigia() +" ("+vigia.getHorarioInicioExpedienteString()
368                    +" - "+ vigia.getHorarioFimExpedienteString() +")");
369                return SUCCESS;
370            }
371        } catch (ParseException e) {
372            output.setValue("retorno", "O horário informado não respeita o horário de expediente de "
373                    + vigia.getNomeVigia() +" ("+vigia.getHorarioInicioExpedienteString() +" - "
374                    + vigia.getHorarioFimExpedienteString() +")");
375            return SUCCESS;
376        }
377    }
```

Fonte: do autor.

Para a persistência de dados no banco de dados PostgreSQL foi utilizado o *framework* Hibernate. O Hibernate é uma solução de mapeamento objeto-relacional. Sua configuração é feita a partir de um arquivo XML chamado `persistence.xml`. Dentro deste arquivo e preciso preencher dados como o *driver* JDBC, endereço de conexão ao banco de dados, nome da base de dados a ser acessada, *login* e senha para conexão. O Quadro 15 mostra o `persistence.xml` criado para configuração do sistema implementado.

Quadro 15 – Configuração do `persistence.xml`

```
<?xml version="1.0" encoding="UTF-8" ?>
 <persistence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
       http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
     version="2.0" xmlns="http://java.sun.com/xml/ns/persistence">
     <persistence-unit name="crudHibernatePU" transaction-type="RESOURCE_LOCAL">
         <provider>org.hibernate.ejb.HibernatePersistence</provider>
         <properties>
             <property name="hibernate.dialect" value="org.hibernate.dialect.PostgreSQLDialect" />
             <property name="javax.persistence.jdbc.driver" value="org.postgresql.Driver" />
             <property name="javax.persistence.jdbc.url"
                   value="jdbc:postgresql://localhost:5432/TCControleVigias" />
             <property name="javax.persistence.jdbc.user" value="postgres" />
             <property name="javax.persistence.jdbc.password" value="senhas" />

             <property name="hibernate.show_sql" value="true" />
             <property name="hibernate.format_sql" value="false" />
             <property name="hibernate.use_sql_comments" value="false" />
             <property name="hibernate.jdbc.wrap_result_sets" value="false" />
             <property name="hibernate.hibernate.cache.use_query_cache" value="true" />
             <property name="hibernate.hbm2ddl.auto" value="update" />
         </properties>
     </persistence-unit>
 </persistence>
```

Fonte: do autor.

Apesar de facilitar o uso do banco de dados o Hibernate não oferece as operações necessárias para o sistema. Assim, foram criados *Data Access Objects* (DAOs) para cada um dos objetos do sistema com as operações básicas de criação, atualização e leitura do banco de dados. Decidiu-se que o sistema não teria operações de remoção de dados, pois todos os registros podem ser reutilizados na operação de monitoramento.

Foi utilizada a biblioteca Hibernate Annotations, que permitiu a configuração do banco de dados a partir de anotações das próprias *models* do Java em vez da utilização de *scripts*. Foi preciso definir um *model* como entidade usando a marcação `@Entity` e `@Table`. Para as colunas, o próprio Hibernate identifica o tipo de dado a ser inserido na base, mas foi preciso informar alguns parâmetros dentro da marcação `@Column` para configuração dos campos como o tamanho dos campos texto, se ele pode ser *null* ou não, etc. Colunas utilizadas como identificadores das tabelas possuem a marcação especial `@Id`. Para geração de um campo automaticamente o Hibernate possui uma marcação chamada `@GeneratedValue`, essa marcação foi utilizada para definir o valor gerado do id como automático, criando assim um efeito de auto increment na coluna. O Quadro 16 mostra a configuração de uma tabela a partir de um *model* via Hibernate Annotations.

Quadro 16 – Configuração de tabela Hibernate

```
15 @Entity
16 @Table(name = "vigia")
17 public class Vigia {
18
19     @Id
20     @GeneratedValue(strategy=GenerationType.AUTO)
21     private Long idVigia;
22     @Column
23     private String nomeVigia;
24     @Column
25     private String rgVigia;
26     @Column
27     private String cpfVigia;
28     @Column
29     private String enderecoVigia;
30     @Column
31     private String bairroVigia;
32     @Column
33     private String cidadeVigia;
34     @Column
35     private String telefoneVigia;
36     @Column
37     private String celularVigia;
38     @Column
39     private String loginVigia;
40     @Column
41     private String senhaVigia;
42     @Column
43     private boolean ativoVigia;
44     @Column
45     private Time horarioInicioExpediente;
46     @Column
47     private Time horarioFimExpediente;
48     // .......
```

Fonte: do autor.

Geralmente os *outputs* gerados pelo Mentawai são suficientes para que o Javascipt e o jQuery consigam montar a página com os parâmetros enviados pelo *backend*. Porém, em alguns casos como listas dinâmicas os dois têm problemas de compatibilidade. Neste caso, existem soluções para realizar esta integração de forma automática, mas como este foi um caso único no sistema desenvolvido, decidiu-se criar uma integração manual. O Quadro 17 mostra a integração manual criada entre o Mentawai e o jQuery neste caso de uma lista dinâmica.

Quadro 17 – Integração manual Mentawai/jQuery

```
<!-- Integração manual mentawai/Jquery -->
<mtw:list value="listaPontosChecagemRota">
    <mtw:loop var="ponto">
        <script type="text/javascript">addPontosChecagemSelected(
                "<mtw:out value='ponto.idRotaPontoChecagem'/>",
                "<mtw:out value='ponto.pontoChecagem.idPontoChecagem'/>",
                "<mtw:out value='ponto.pontoChecagem'/>");</script>
    </mtw:loop>
</mtw:list>
<!-- Integração manual mentawai/Jquery -->

</html>
```

Fonte: do autor.

As *tags* `<mtw:list>`, `<mtw:loop>` e `<mtw:out>` são as que gerenciam o *output* do Mentawai para a página web. A função `addPontosChecagemSelected` é uma função do javacript que utiliza o jQuery para criar uma lista dinâmica.

A conexão cliente-servidor realizada entre o sistema web e o aplicativo anrdoid foi realizada através de conexões HTTP, utilizando JSON como o meio de troca de informações. Essa troca foi realizada a partir de chamadas realizadas a partir do dispositivo móvel em direção a uma *action* gerenciada pelo Mentawai. Oo Mentawai possui um *renderer* chamado `JsonRenderer`. Esse *renderer* transforma todas as saídas da *action* em um único objeto JSON e reenvia os dados para o dispositivo móvel. O Quadro 18 mostra a aplicação do `JsonRenderer` na saída de um *action* dentro do `ApplicationManager`.

Quadro 18 – aplicação do JsonRenderer em uma *action*

```
action("/PontoChecagem", ActionPontoChecagem.class, "listaNaoPreenchidos")
.bypassAuthentication()
.on(SUCCESS, ajax(new JsonRenderer()));
```

Fonte: do autor.

Devido à natureza simples do JSON não é possível enviar objetos muito complexos por ele, por isso foi utilizado o conceito de *Data Transfer Object* (DTO). O DTO é basicamente uma versão simplificada dos *views* do sistema, apenas com campos importantes do sistema e com relacionamentos resumidos ao uso de ids. Existem métodos para converter um objeto comum em DTO em todas as classes do sistema que são enviadas via JSON para o dispositivo móvel e existem métodos no dispositivo móvel que transformam este JSON em um objeto pronto para uso. O Quadro 19 mostra um exemplo de um método que transforma um objeto como em um DTO.

Quadro 19 – Método de transformação do objeto em DTO

```
129  public ExecucaoRondaDTO getDTO(){
130      ExecucaoRondaDTO dto = new ExecucaoRondaDTO();
131      dto.setIdRondaExecutada(idRondaExecutada);
132      dto.setDiaExecucaoRota(diaExecucaoRota.toString());
133      dto.setHorarioExecucaoRota(horarioExecucaoRota.toString());
134      dto.setRotaRondaExecutada(rotaRondaExecutada.getDTO());
135      dto.setStatusRondaExecutada(statusRondaExecutada);
136      return dto;
```

Fonte: do autor.

Normalmente a classe `ExecucaoRonda` possui uma rota em sua estrutura, essa rota por sua vez possui vários campos. Transformar todos estes dados em um JSON seria utilizar muitos recursos para atingir um mesmo objetivo. O método `getDTO()` busca apenas os campos que serão utilizados na aplicação móbile, todos os objetos que são filhos da classe também são transformados em DTO e, dessa forma, é possível manter a mesma estrutura orientada a objetos nos dois módulos do sistema sem comprometer muito a performance. O Quadro 20 mostra a forma que esse DTO é recebido na aplicação mobile.

Quadro 20 – Conversão de um DTO JSON em objeto

```
public Ronda(JSONObject rondaObject) {
    try {
        this.idRondaExecutada = rondaObject.getLong("idRondaExecutada");
        this.diaExecucaoRota = Date.valueOf(rondaObject.getString("diaExecucaoRota"));
        this.horarioExecucaoRota = Time.valueOf(rondaObject.getString("horarioExecucaoRota"));
        this.rotaRondaExecutada = new Rota(rondaObject.getJSONObject("rotaRondaExecutada"));
        this.statusRondaExecutada = StatusExecucaoRonda
                .valueOf(rondaObject.getString("statusRondaExecutada"));
    } catch (JSONException e) {
        e.printStackTrace();
    }
}
```

Fonte: do autor.

Quando uma nova execução de ronda é iniciada pelo vigia no dispositivo móvel, o aplicativo busca no servidor dados sobre a rota que será executada. Na busca obtém-se dados como horários de início e termino da rota, e todos os pontos de checagem que precisam ser executados em ordem. O aplicativo Android trabalha com o modelo de *activities* para controlar suas telas. As *activities* precisam de uma forma de trocar dados entre si para realizarem suas funções. Geralmente essa troca de dados é feita através de *intents*, porém esse processo é pouco eficiente quanto ao envio de muitas informações entre todas as *activities* que o sistema utiliza, além de a perda de dados caso o usuário feche o aplicativo no meio da utilização.

Para resolver este problema foi utilizada uma base de dados LiteSql no aplicativo Android para guardar dados enquanto uma ronda é executada, essa base de dados facilita a troca de dados entre as *activities* de forma que quando esta execução de ronda é terminada e

enviada para ser persistida no servidor, a base temporária é limpa e o sistema espera uma nova execução de ronda ser iniciada. Para a utilização dessa base foi necessária a configuração de um SQLiteOpenHelper para manter a configuração da base e a definição das tabelas. O Quadro 21 mostra parte da configuração da base de dados criada com essa classe.

Quadro 21 – Configuração da database LiteSql

```
public class DatabaseHelper extends SQLiteOpenHelper {
    // Logcat tag
    private static final String LOG = "DatabaseHelper";
    // Database Version
    private static final int DATABASE_VERSION = 11;
    // Database Name
    private static final String DATABASE_NAME = "controleDeVigias";
    // Table Names
    private static final String TABLE_RONDA = "ronda";
    private static final String TABLE_ROTA = "rota";
    private static final String TABLE_ROTAPONTOCHECAGEM = "rotaPontoChecagem";

    private static final String ID_RONDA_EXECUTADA = "idRondaExecutada";
    private static final String DIA_EXECUCAO_ROTA = "diaExecucaoRota";
    private static final String HORARIO_EXECUCAO_ROTA = "horarioExecucaoRota";
    private static final String ROTA_RONDA_EXECUTADA = "rotaRondaExecutada";
    private static final String STATUS_RONDA_EXECUTADA = "statusRondaExecutada";
    private static final String ID_ROTA = "idrota";
    //.......
    private static final String CREATE_TABLE_RONDA = "CREATE TABLE " + TABLE_RONDA
                    +"("+
                        ID_RONDA_EXECUTADA +" bigint NOT NULL PRIMARY KEY, " +
                        DIA_EXECUCAO_ROTA +" date, " +
                        HORARIO_EXECUCAO_ROTA +" time without time zone, " +
                        STATUS_RONDA_EXECUTADA +" character varying(255), " +
                        ROTA_RONDA_EXECUTADA +" bigint, " +
                        "FOREIGN KEY("+ROTA_RONDA_EXECUTADA+") REFERENCES "
                            + TABLE_ROTA +"("+ID_ROTA+")"
                    +")";
    @Override
    public void onCreate(SQLiteDatabase db) {

        // creating required tables
        db.execSQL(CREATE_TABLE_RONDA);
        //.......
    }
    //.......
```

Fonte: do autor.

Foram criados DAOs com operações de inserção e leitura dos dados para todos os objetos do sistema. Apenas uma inserção é realizada ao início de uma execução de ronda. Esta inserção cria os registros para todas as tabelas, a leitura da base é feita sempre que é necessário o acesso de alguma informação das tabelas.

Para permitir que o aplicativo tenha a funcionalidade de NFC, foi utilizada a biblioteca `NfcAdapter` que é nativa do android. Com ela é possível fazer com que uma *activity* Android aceite a aproximação do aparelho de uma *tag* NFC como uma forma de entrada de dados. O Quadro 22 mostra o trecho de código necessário para ativar este *adapter*, o qual deve ser implementado dentro do `onCreate`() da activity.

Quadro 22 – Ativando o NfcAdapter em uma activity android

```
adapter = NfcAdapter.getDefaultAdapter(this);
pendingIntent = PendingIntent.getActivity(this, 0,
        new Intent(this, getClass()).addFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP), 0);
IntentFilter tagDetected = new IntentFilter(NfcAdapter.ACTION_TAG_DISCOVERED);
tagDetected.addCategory(Intent.CATEGORY_DEFAULT);
writeTagFilters = new IntentFilter[]{tagDetected};

if (adapter == null) {
    new AlertDialog.Builder(MainActivity.this)
            .setIcon(android.R.drawable.ic_dialog_alert)
            .setTitle("Erro")
            .setMessage("Seu dispositivo não suporta NFC")
            .setPositiveButton("Ok",null)
            .show();
}else if(!adapter.isEnabled()) {
    new AlertDialog.Builder(MainActivity.this)
            .setIcon(android.R.drawable.ic_dialog_alert)
            .setTitle("Erro")
            .setMessage("O NFC esta desativado, " +
                    "ative no menu de configurações deste aparelho")
            .setPositiveButton("Ok",null)
            .show();
}
```

Fonte: do autor.

É possível verificar se o aparelho possui NFC ou se o mesmo está ativado fazendo duas condições como mostrado na figura. Após definir que uma *activity* pode utilizar uma *tag*, o Android trata o contato do aparelho com a *tag* NFC como uma nova *intent* da *activity*. É preciso então interceptar essa *intent* e utilizá-la da forma desejada pelo sistema.

Existem três formas diferentes que a aplicação trata esta aproximação. A primeira ocorre quando o vigia ou administrador aproxima o celular de uma *tag* enquanto a aplicação está em uma *activity* que não utiliza NFC, desta forma a aproximação é tratada, mas nada e executado. A segunda forma ocorre quando o administrador aproxima o aparelho para preencher o ponto de checagem em uma *tag* NFC, neste caso o sistema gera uma chave NFC e insere na *tag*. Após a inserção essa chave é enviada para o servidor para ser preenchida no campo `keyNfcPontoChecagem` da tabela de pontos de checagem. O Quadro 23 mostra a

forma que a *intent* de aproximação de *tag* é tratada quando utilizada pelo administrador para preencher um ponto de checagem.

Quadro 23 – Preenchendo *tag* NFC.

```
@Override
protected void onNewIntent(Intent intent){
    if(NfcAdapter.ACTION_TAG_DISCOVERED.equals(intent.getAction())){
        mytag = intent.getParcelableExtra(NfcAdapter.EXTRA_TAG);

        try {
            if(mytag==null){
                Toast.makeText(ctx, "Erro detectado", Toast.LENGTH_LONG ).show();
            }else{

                Long nfcKey = generateNFCkey();

                write(nfcKey + "",mytag);

                new AsyncWriteKEY().execute(pontoChecagem.getIdPontoChecagem()+"", nfcKey+"");

                Toast.makeText(ctx, "escrita ok", Toast.LENGTH_LONG ).show();
            }
        } catch (IOException e) {
            Toast.makeText(ctx, "erro na escrita", Toast.LENGTH_LONG ).show();
            e.printStackTrace();
        } catch (FormatException e) {
            Toast.makeText(ctx, "erro na escrita" , Toast.LENGTH_LONG ).show();
            e.printStackTrace();
        }
    }
}
```

Fonte: do autor.

O método generateNFCkey() é o responsável por gerar a chave que é gravada na *tag* e então enviada para o servidor a partir da classe assíncrona AsyncWriteKEY(). Esta chave é gerada a partir do horário atual convertido em milissegundos somado com um valor aleatório gerado no momento da criação, dessa forma há uma garantia de que duas chaves nunca serão iguais. O Quadro 24 mostra o código responsável por gerar a chave NFC descrito acima.

Quadro 24 – Gerador da chave NFC

```
private long generateNFCkey(){
    long time = System.currentTimeMillis();
    Random randomGenerator = new Random();
    return time + randomGenerator.nextInt(99999);
}
```

Fonte: do autor.

A terceira maneira de tratar da aproximação da *tag* ao dispositivo ocorre quando um vigia faz a validação de um ponto de checagem enquanto realiza um execução de ronda. Neste caso a *intent* é tratada para ler a chave NFC que foi inserida em uma *tag* por um administrador. O Quadro 25 mostra a forma que a *intent* de aproximação de *tag* é tratada quando utilizada pelo vigia para ler o conteúdo um ponto de checagem.

Quadro 25 – Lendo *tag* NFC.

```
@Override
protected void onNewIntent(Intent intent) {
    if (NfcAdapter.ACTION_TAG_DISCOVERED.equals(intent.getAction())) {
        mytag = intent.getParcelableExtra(NfcAdapter.EXTRA_TAG);

        try {
            if (mytag == null) {
                Toast.makeText(ctx, "Erro detectado", Toast.LENGTH_LONG).show();
            } else {
                readFromTag(intent, mytag);
            }
        } catch (Exception e) {
            Toast.makeText(ctx, "Erro na leitura", Toast.LENGTH_LONG).show();
            e.printStackTrace();
        }
    }
}
```

Fonte: do autor.

Após a realização do código de leitura o sistema obtém a chave NFC contida na *tag* validada, a chave é então comparada com a chave do registro de ponto de checagem que precisa ser validado no momento. Caso as chaves não possuam o mesmo valor o sistema mostra uma mensagem de erro para o usuário informando que a *tag* aproximada não é válida. Caso as duas chaves sejam equivalentes, o sistema marca o status do ponto de checagem como validado e incrementa o contador do ponto de checagem atual.

O contador do ponto de checagem atual é uma variável numérica que inicia em zero e é auto incrementada toda a vez que um ponto de checagem é validado. A variável é zerada quando a execução de rota é finalizada, esse contador serve para acompanhar os pontos de checagem que são executados em uma rota, toda a rota possui os pontos de checagem que precisam ser validados e cada um desses pontos possui um valor que define a ordem que ele deve ser validado. O sistema utiliza o valor do contador para saber qual ponto deve ser validado e quais já foram validados, além de saber quando uma rota chega ao fim, esse método foi implementado para mitigar um dos maiores problemas da solução desenvolvida que é diminuir a quantidade de dados trafegando entre o dispositivo e o servidor, combinado com a base de dados que é criada ao iniciar uma execução de ronda, esse método remove a necessidade do servidor manter o controle dos pontos de checagem e enviar objetos pela rede. O Quadro 26 mostra a implementação que incrementa o contador e valida se o fim da rota foi alcançado.

Quadro 26 - Incrementando e validando o `contadorValidacao`

```
Toast.makeText(RondaActivity.this, "Validado com sucesso", Toast.LENGTH_LONG).show();

contadorValidacoes++;
if (contadorValidacoes == contadorValidacoesTotal) {
    new AsyncFinalizaRonda().execute(rondaAtual.getIdRondaExecutada() + "");
}

SharedPreferences.Editor editor = sharedpreferences.edit();

editor.putString("contadorValidacao", contadorValidacoes + "");
editor.commit();

RondaActivity.this.recreate();
```

Fonte: do autor.

Para realizar a leitura de QR code no aplicativo do dispositivo móvel foi utilizada a biblioteca `IntentIntegrator` que é nativa do android. Para realizar a função de conexão via QR code foi adicionada uma opção no menu da tela de *login*, quando selecionada essa opção é iniciado o *scan* a partir da câmera do aparelho, o resultado desse *scan* é então tratado e dele são retirados os dados emitidos pelo servidor, esses dados então são usados para realizar uma função de *login*. O Quadro 27 mostra o código necessário para realizar a leitura do QR code, iniciando pelo click do usuário na opção do menu.

Quadro 27 – Ativando o NfcAdapter em uma *activity* android

```
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle item selection
    switch (item.getItemId()) {
        case R.id.leQr:
            new IntentIntegrator(this).initiateScan();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}

public void onActivityResult(int requestCode, int resultCode, Intent intent) {
    IntentResult scanResult = IntentIntegrator
            .parseActivityResult(requestCode, resultCode, intent);
    if (scanResult != null) {

        try{
            String[] arra = scanResult.getContents().toString().split(" SEPARA ");

            String id = arra[0].split(" - ")[0];

            new AsyncLoginByQR().execute(arra[1], arra[2] , id);
        }catch (Exception e ){
            new AlertDialog.Builder(MainActivity.this)
                    .setIcon(android.R.drawable.ic_dialog_alert)
                    .setTitle("Erro")
                    .setMessage("QR code não identificado")
                    .setPositiveButton("Ok",null)
                    .show();
        }
    }
}
```

Fonte: do autor.

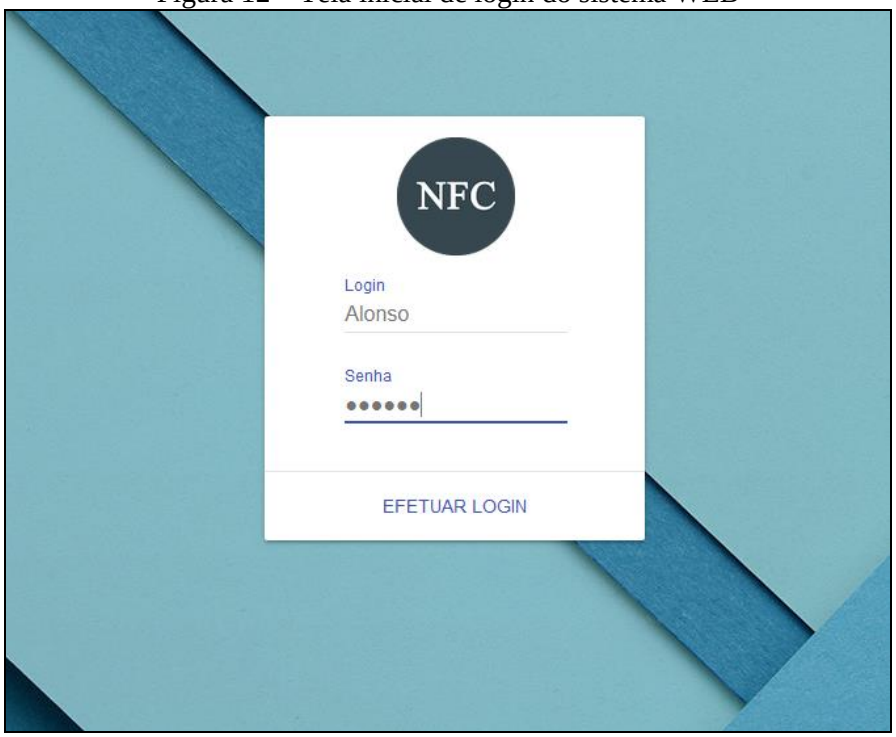
3.3.2 Operacionalidade da implementação

Esta seção mostrara a solução implementada, ela será dividida em duas subseções, uma demonstra as funcionalidades do servidor WEB quando acessado por um administrador, a segunda seção demonstra as funcionalidades do aplicativo móvel quando acessado por um administrador ou por um vigia.

3.3.2.1 Funcionalidades do servidor WEB

Ao acessar o sistema web o administrador é redirecionado para uma página de *login*, o administrador deve então informar corretamente suas credenciais para poder acessar o sistema. A Figura 12 mostra a tela de login do sistema web.

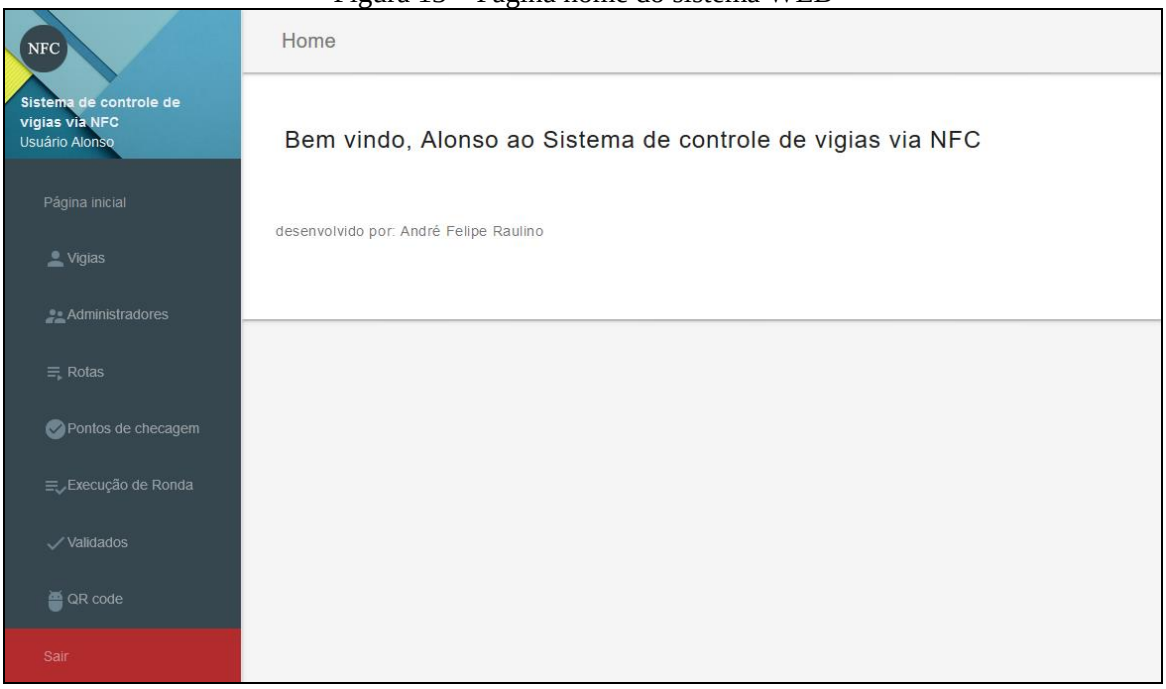
Figura 12 – Tela inicial de login do sistema WEB



Fonte: do autor.

Após efetuar o *login* com sucesso, é apresentada ao administrador a pagina inicial do sistema. A partir deste ponto se torna visível um menu lateral que oferece acesso às demais localidades do sistema, o menu também possui um botão `sair` que fecha a sessão do administrador logado atualmente. A Figura 13 mostra a tela home e o menu lateral.

Figura 13 – Pagina home do sistema WEB



Fonte: do autor.

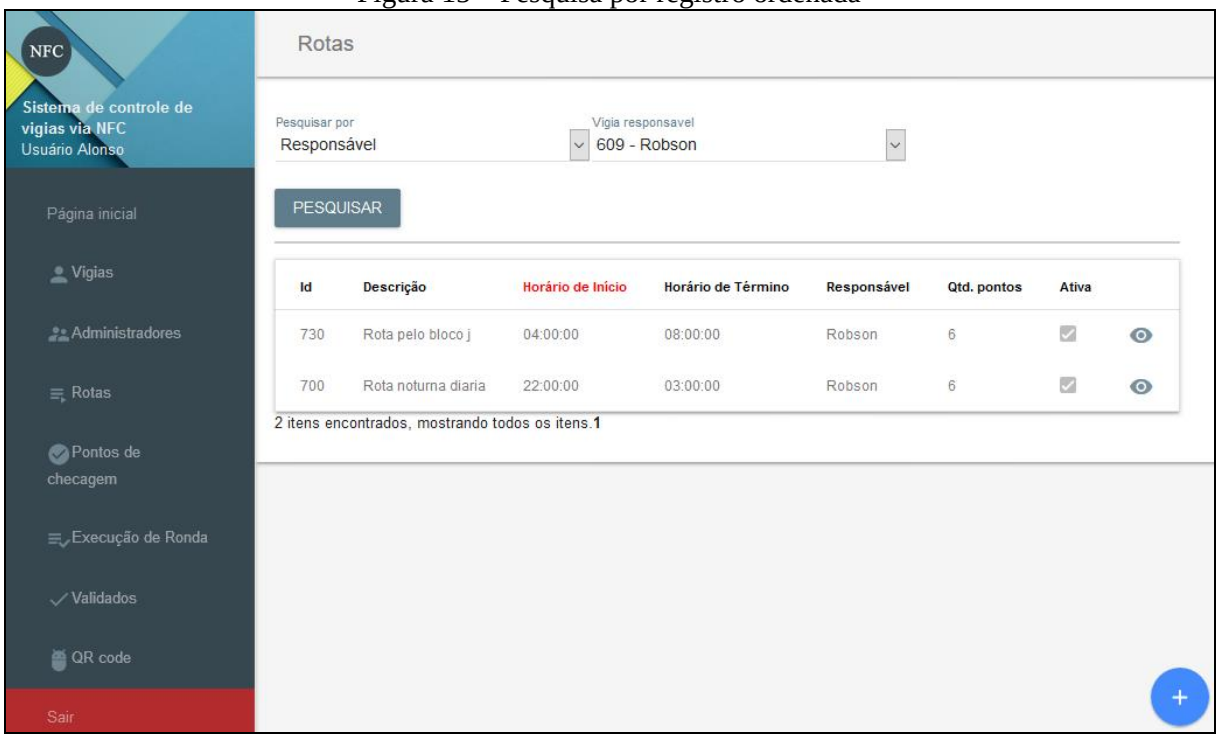
Ao escolher a opção de listagem de vigias, o administrador entra na tela de listagem. A partir desta tela o administrador pode realizar pesquisas com campos específicos, ordenar de forma crescente ou decrescente qualquer tipo de dado da tabela, monitorar um registro especifico ou adicionar um novo registro a partir do botão `mais`. Os registros de execução de ronda e pontos de checagem não possuem o botão `mais` pois podem apenas serem adicionados a partir do dispositivo móvel. A Figura 14 mostra a tela de listagem de vigias.

Figura 14 – listagem de vigias.

Fonte: do autor.

A interface de pesquisa oferece a possibilidade de pesquisar nas tabelas utilizando campos específicos. Quando o administrador entra na página de listagem a caixa `Pesquisa por` estará sempre com o valor `Todos` selecionado. Para realizar uma pesquisa é preciso selecionar nesta caixa o campo que deseja-se pesquisar, um input dinâmico aparecerá a direita da caixa que permitirá que o administrador informe o valor que deseja pesquisar. Após informar o valor o administrador então seleciona a opção `pesquisar`. É possível também realizar pesquisas por registros em tabelas que possuem relacionamentos com outras, por exemplo, é possível pesquisar por responsável na tabela de rotas informando um vigia especifico. A Figura 15 mostra uma pesquisa de rotas, utilizando o vigia responsável para filtrar os resultados e ordenada por horário de início.

Figura 15 – Pesquisa por registro ordenada

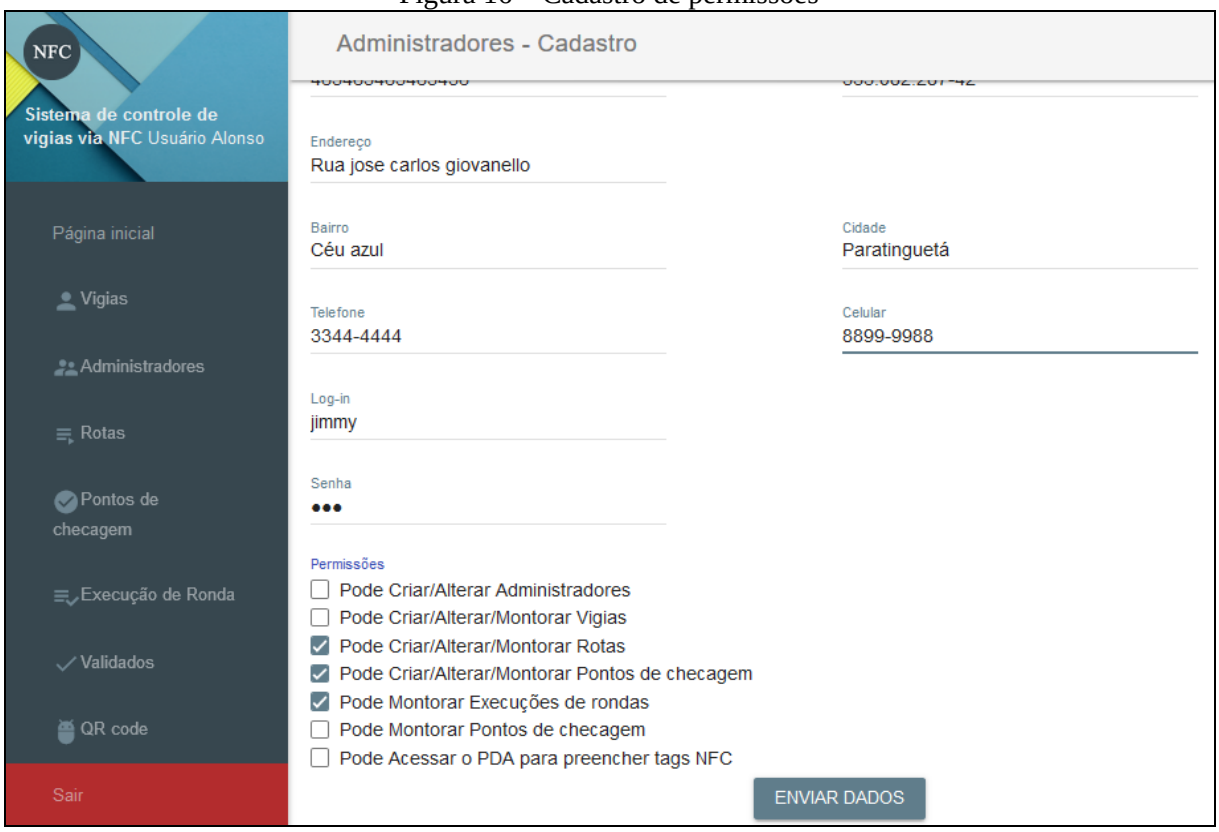


Fonte: do autor.

Ao selecionar a opção `mais` ou selecionar um dos registros da tabela o usuário é redirecionado para a tela de cadastro do registro. Então mostrado ao usuário um formulário contendo os campos que o registro possui, caso o usuário tenha selecionado um registro da tabela estes campos já estarão preenchidos com os seus dados, o usuário informa os dados do formulário e então seleciona a opção `enviar dados`. O sistema faz uma validação dos dados informados e verifica se eles estão preenchidos corretamente, caso algum campo esteja preenchido incorretamente o sistema informa o problema ao usuário e não permite que a ação continue. Caso os dados estejam corretos o registro informado é cadastrado ou editado na base de dados do sistema.

O cadastro de administradores possui uma área especial que define as permissões concedidas ao administrador cadastrado. A Figura 16 mostra a área do cadastro de permissões dentro do cadastro dos administradores.

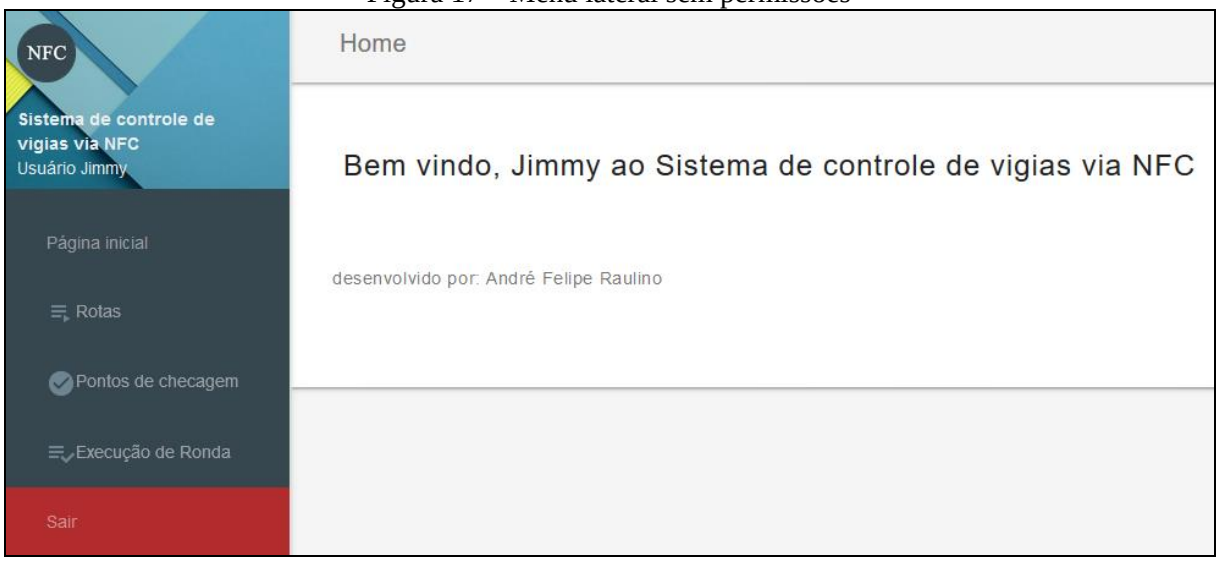
Figura 16 – Cadastro de permissões



Fonte: do autor.

Quando uma permissão não é concedida a um administrador, quando este realizar *login* no sistema ele não verá no menu lateral a opção para acesso da área do sistema que ele não possui acesso. A Figura 17 mostra o menu lateral deste administrador.

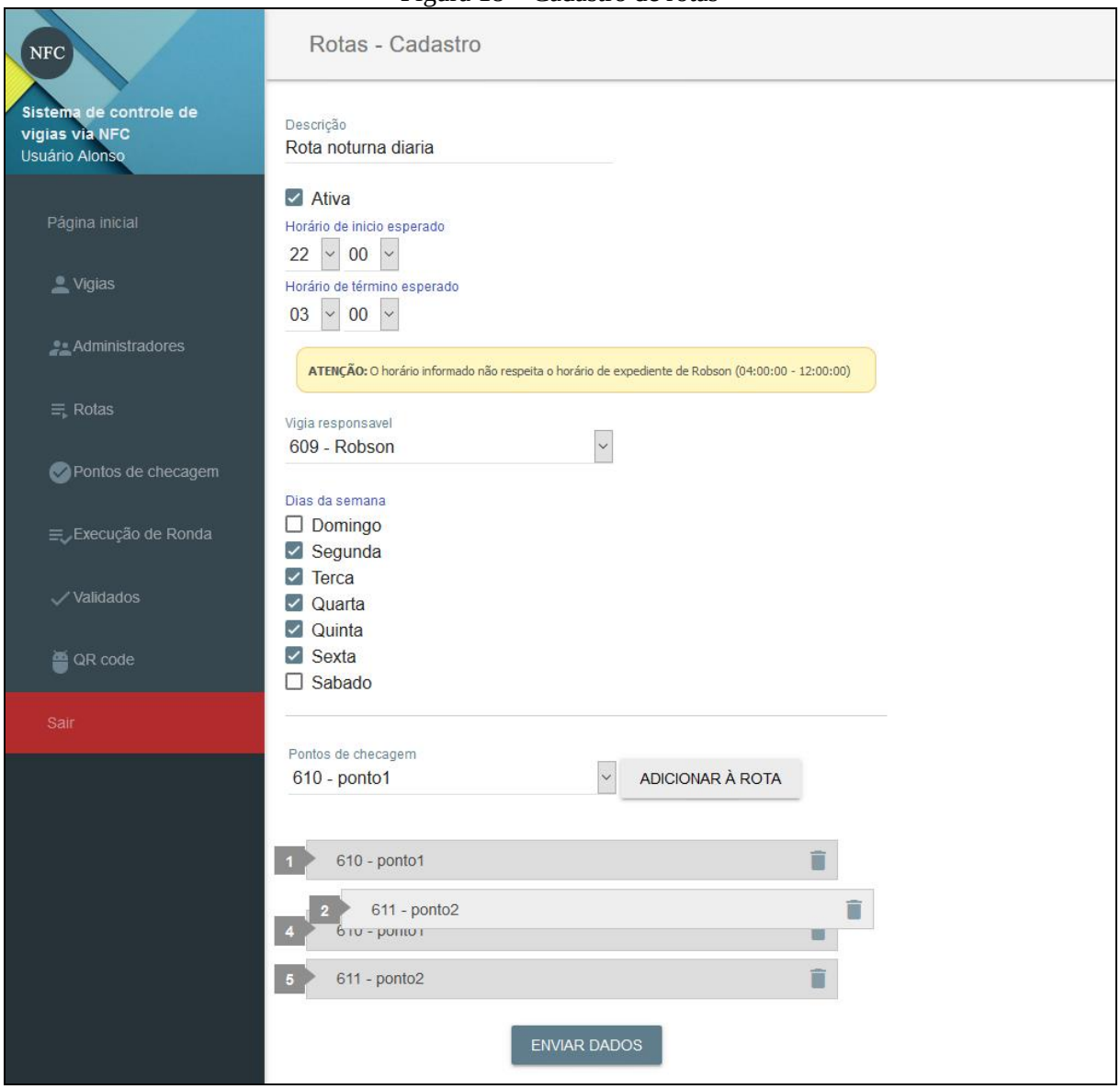
Figura 17 – Menu lateral sem permissões



Fonte: do autor.

O cadastro de rotas possui diversos casos especiais, quando o administrador define um horário para a rota ou seleciona um vigia para ser o responsável. O sistema faz uma validação para checar se o horário informado respeita o horário de expediente do vigia, caso não respeite uma mensagem é mostrada ao administrador que pode então selecionar outro horário ou responsável. O cadastro possui também uma área para definição dos dias da semana que esta rota será executada. Para adicionar um novo ponto de checagem a ser validado na rota o administrador deve escolher o ponto de checagem desejado na caixa `Pontos de checagem` e então selecionar a opção `adicionar à rota`, o administrador então tem a opção de ordenar ou deletar os pontos de checagem. A Figura 18 mostra o cadastro de rotas e todos os seus casos especiais.

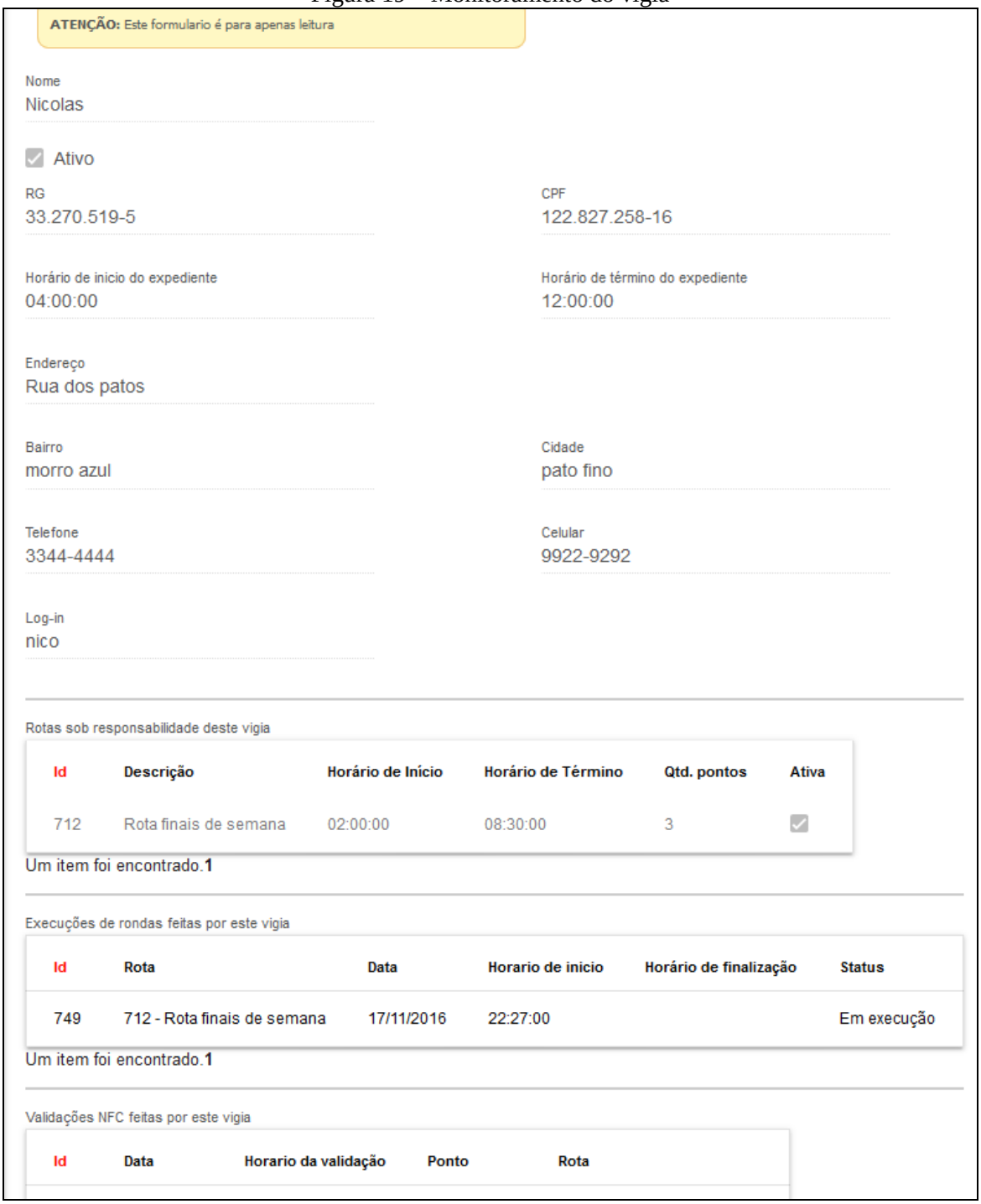
Figura 18 – Cadastro de rotas



Fonte: do autor.

As listagens possuem uma opção de visualização representada por um ícone de um olho, ao selecionar essa opção o administrador entra na tela de monitoramento. Essa tela traz todos os dados relacionados ao registro para consulta, esta tala também mostra todos os relacionamentos que o registro possui. A Figura 19 mostra a tela de monitoramento de um vigia e todas as rotas que ele é responsável, execuções de ronda que ele realizou e suas validações NFC.

Figura 19 – Monitoramento do vigia



Fonte: do autor.

Ao selecionar a opção de `QR code` o administrador é levado a uma tela que possui todos os cadastros de vigias e administradores para serem selecionados. Ao selecionar um desses cadastros e então apertar no botão `gerar`, é gerado então um QR code a partir dos dados do vigia ou administrador selecionado e dos dados de conexão ao servidor. Ao ler este QR code no aplicativo Android a conexão é realizada automaticamente sem necessidade de informar o *Internet Protocol* (IP) e a porta de conexão além dos dados pessoais de *login* e senha. A Figura 20 mostra a tela de geração do QR code no servidor.

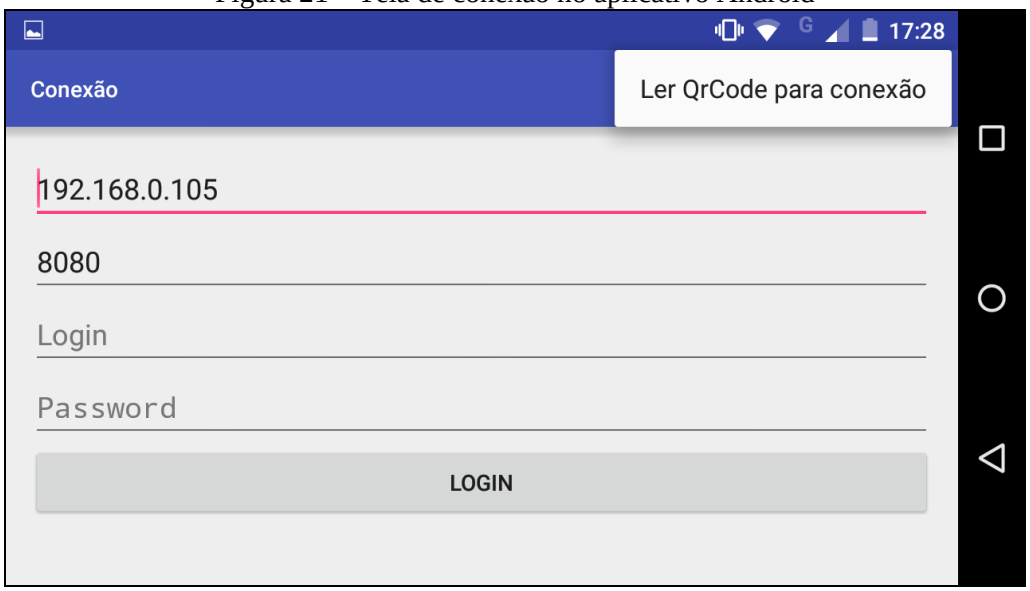
Figura 20 – tela de geração de QR code

Fonte: do autor.

3.3.2.2 Funcionalidades do dispositivo móvel

Ao acessar o aplicativo no dispositivo Android, o usuário é levado a uma tela onde tem a opção de realizar o *login* normalmente ou ler o QR code gerado pelo servidor para realizar a conexão. A Figura 21 mostra a tela de conexão.

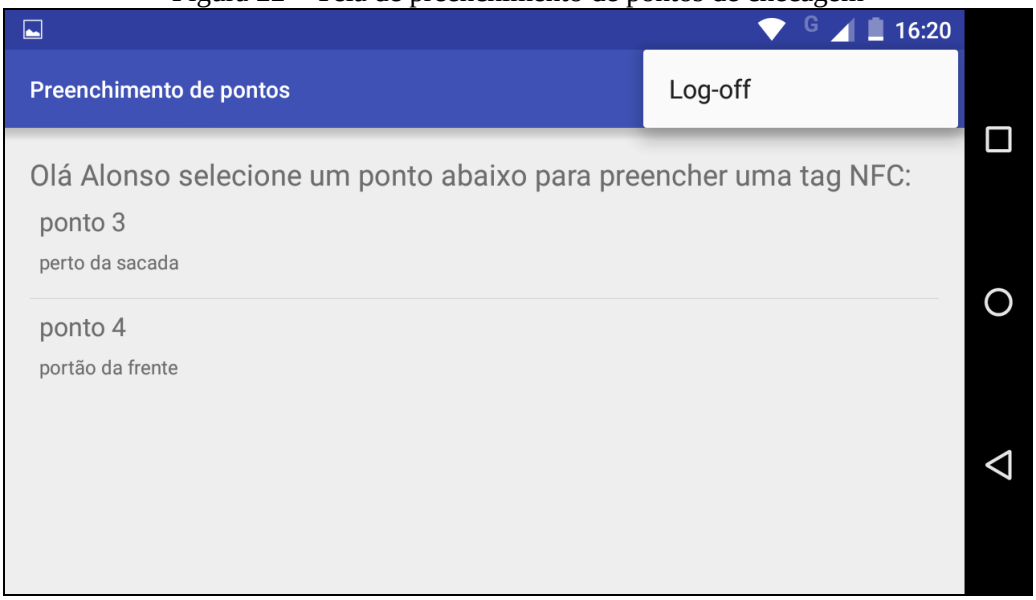
Figura 21 – Tela de conexão no aplicativo Android



Fonte: do autor.

Após realizar *login* o administrador é levado a uma listagem de pontos de checagens que precisam ser inseridos dentro de uma *tag* NFC.O administrador então deve selecionar qual ponto de checagem deseja preencher. A Figura 22 mostra a tela de preenchimento de pontos de checagem.

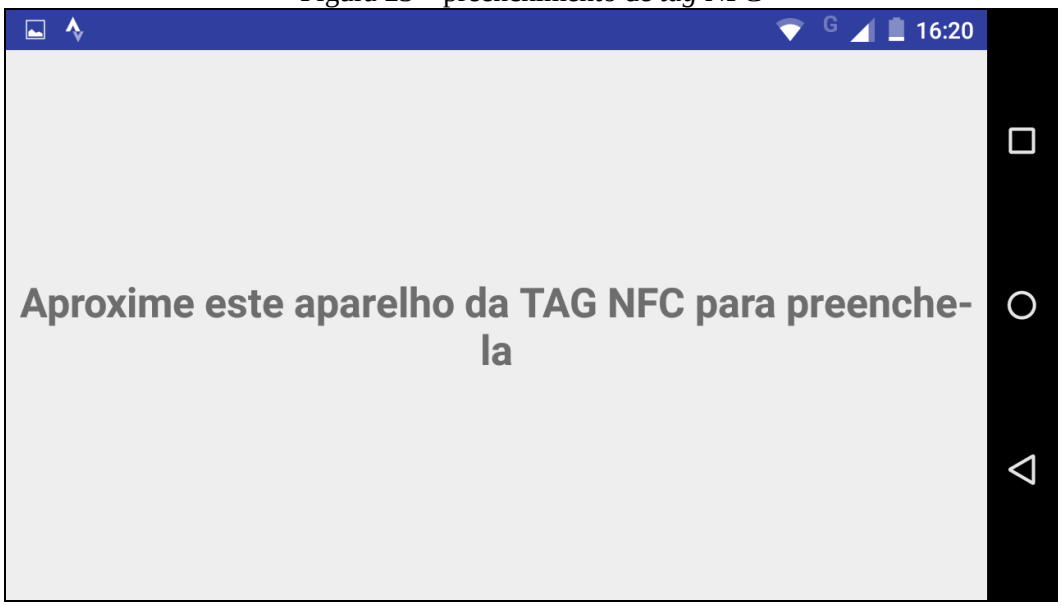
Figura 22 – Tela de preenchimento de pontos de checagem



Fonte: do autor.

Após selecionar um dos pontos para ser preenchido, o administrador é enviado para uma tela cuja única ação é a aproximação do aparelho a uma *tag* NFC vazia. Ao realizar esta ação o sistema então preenche a *tag* com a chave NFC. A Figura 23 mostra a tela de preenchimento NFC.

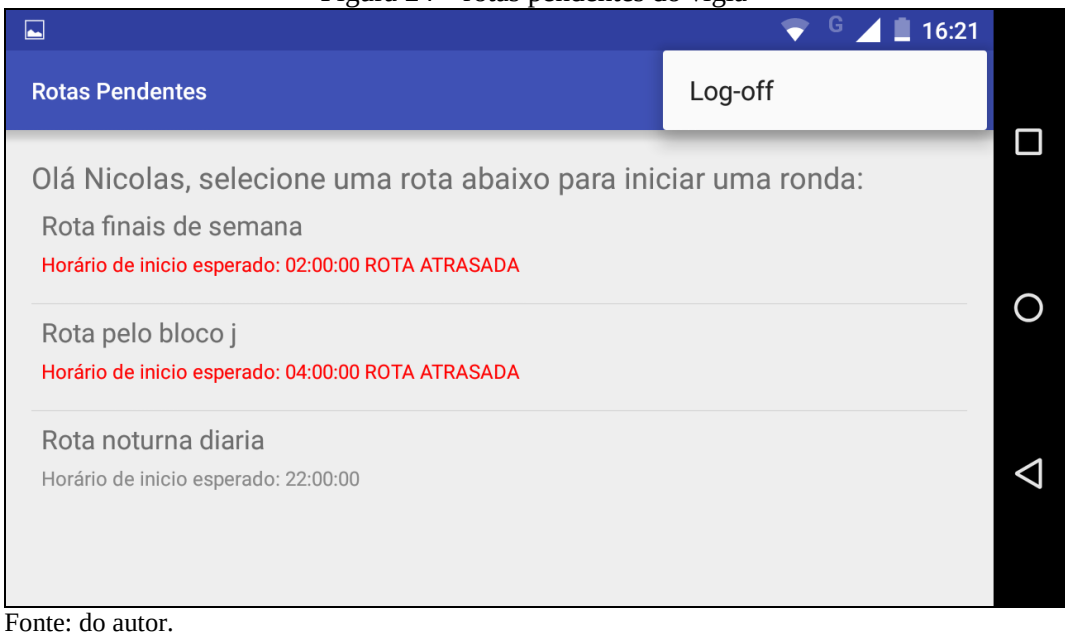
Figura 23 – preenchimento de *tag* NFC



Fonte: do autor.

Caso o usuário que tenha realizado o *login* seja um vigia, ele é redirecionado para uma tela que mostra todas as rotas sob sua responsabilidade, são apresentadas apenas rotas que ainda não foram executadas no dia. Ao selecionar uma destas rotas o vigia inicia um execução de ronda. A Figura 24 mostra a tela de rotas pendentes do vigia.

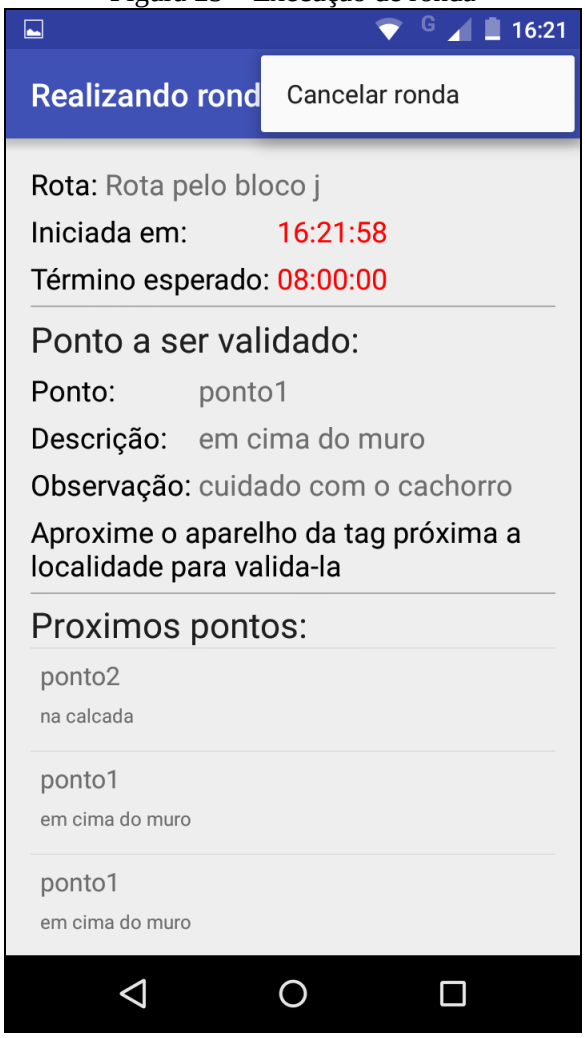
Figura 24 – rotas pendentes do vigia



Fonte: do autor.

Ao selecionar uma das rotas pendentes o vigia inicia uma execução de ronda, o sistema grava o horário de início e disponibiliza ao vigia um horário máximo para a execução da ronda. Em destaque é demonstrado o ponto de checagem que precisa ser verificado no momento e alguns dados sobre ele. É mostrada também uma lista com os próximos pontos que o procedem na rota atual. Enquanto visualiza essa tela o vigia deve se locomover até o ponto em questão e realizar a aproximação do aparelho à *tag* NFC para realizar a validação do ponto, este ponto então some da listagem e o vigia precisa repetir esse processo com o próximo ponto, ao realizar a validação do último ponto a execução de ronda é terminada e o vigia volta para a listagem de rotas pendentes. O vigia também tem a opção de cancelar a ronda em qualquer momento, mas esta não será marcada como terminada, a rota ainda aparecera na listagem de rotas pendentes para o vigia. A Figura 25 mostra a tela de execução de ronda.

Figura 25 – Execução de ronda



Fonte: do autor.

3.4 RESULTADOS E DISCUSSÕES

O protótipo desenvolvido contempla todos os requisitos funcionais e não funcionais apresentados. A utilização da tecnologia NFC foi aplicada corretamente e sem grandes desafios no software e todos os componentes funcionaram corretamente nos testes realizados. Porém, os testes de funcionalidade foram realizados de forma muito controlada e por isso é impossível afirmar se o sistema agiria corretamente em um ambiente mais caótico.

Tanto o software web quanto o aplicativo para dispositivo móvel apresentam uma performance satisfatória, porém a conexão entre o cliente e o servidor precisa ser consistente para que o sistema criado possua uma boa usabilidade. A interface desenvolvida para o software web oferece um visual limpo que facilita o uso do sistema. Foi utilizada também a tecnologia QR code para facilitar a conexão de um vigia à aplicação Android.

Em comparação com o software correlato TopRonda desenvolvido pela TopData, o sistema desenvolvido neste trabalho mostra algumas vantagens como uma melhor interface para o controlador, a possibilidade do monitoramento das ações realizadas por um determinado registro dentro do sistema, um aparelho que oferece um controle de seção e mais opções para o vigia além da transmissão imediata dos dados via internet. As desvantagens incluem a necessidade de um aparelho celular com a tecnologia especifica de NFC, a fragilidade de um celular comum em comparação a um bastão Viggia da TopData, um software controlador inferior ao utilizado pelo TopRonda, a falta da possibilidade da geração de relatórios e a constante necessidade de conexão com a internet para o funcionamento das validações do vigia. O Quadro 28 mostra um comparativo entre as funcionalidades do sistema desenvolvido e do TopRonda da TopData.

Quadro 28 – Comparação do sistema desenvolvido com o TopRonda

	TopRonda	**Sistema desenvolvido**
Permite o controle de vigias, pontos de controle, administradores e rotas	Sim	Sim
Permite o controle de escalas	Sim	Não
Possui a necessidade da utilização de um aparelho especifico para a validação de pontos de checagem	Sim	Não
Realiza em tempo real a validação do ponto de checagem na base de dados ao realizar a validação	Não	Sim
Necessita de internet para a realização da validação dos pontos de checagem	Não	Sim
Geração de relatórios	Sim	Não
Permite o monitoramento de registros	Não	Sim
Utilização de QR code para facilitar acesso do vigia ao sistema	Não	Sim

Fonte: elaborado pelo autor.

4 CONCLUSÕES

O principal objetivo do trabalho, que era a construção de um sistema de monitoramento de vigias utilizando NFC para validação dos pontos de checagem foi atingido, a tecnologia NFC foi aplicada sem grandes dificuldades. Todos os módulos do sistema apresentaram pequenos desafios, mas ao fim tudo foi realizado corretamente.

O maior problema que o desenvolvimento apresentou foi a conexão entre os dois módulos do sistema, Isso se deve a grande quantidade de dados que são trocados entre o sistema Web e o aplicativo do dispositivo móvel e a necessidade constante de conexão com a internet, medidas foram tomadas para mitigar os problemas que isso possa gerar, mas ainda não a um nível totalmente satisfatório. Este problema poderia ter sido resolvido na etapa de estruturação, mas as soluções consideradas envolviam a perda da característica de envio instantâneo das validações de ponto de checagem para o servidor, foi decidido então manter a necessidade da conexão com a internet.

As ferramentas utilizadas se mostraram adequadas para o desenvolvimento do software, principalmente o *framework* Mentawai, que facilitou muito o gerenciamento de ações e conexões HTTP entre o servidor e o dispositivo móvel. A pesquisa para a aplicação do NFC foi baseada nas próprias ferramentas disponibilizadas pelo Android, as quais ajudaram muito o entendimento da funcionalidade da tecnologia.

A decisão de criar uma forma mais simples de conexão do vigia utilizando QR code foi tomada apenas no fim do desenvolvimento motivada por um desejo de utilizar a tecnologia de escrita e leitura de QR code na aplicação. Para isso a utilização do jQuery se mostrou bem proveitosa pois permitiu o desenvolvimento do gerador do QR code no servidor de forma bem simples pois utiliza apenas uma livraria para realizar a ação, o Android foi mais uma vez útil ao disponibilizar uma classe que permite a leitura do QR code.

Ao finalizar este trabalho concluiu-se que é possível a utilização do NFC para diversas aplicações, até mesmo para metodologias já consolidadas no mercado. Espera-se que esse trabalho seja uma porta para muitos outros que desejam desenvolver com a mesma tecnologia e que esta ganhe cada vez mais utilidades com o passar do tempo.

4.1 EXTENSÕES

As sugestões para possíveis extensões desse trabalho são enumeradas abaixo:

a) apresentar alternativas para a realização da conexão entre os módulos, de forma que não seja necessário o uso constante da internet, mas sem perder a característica

de envio rápido das validações ao servidor;

b) expandir a estrutura de dados que são gerenciados pelo sistema. Adicionar registros de dispositivos e de filiais, respectivamente expandindo assim o controle do administrador sobre os dispositivos dos vigias e permitir o uso do sistema em diversas localidades físicas ao separar vigias e pontos de checagem por filial;
c) criar um modulo para geração de relatórios impressos e envio de e-mails dos registros que estão sendo monitorados.

REFERÊNCIAS

BAUMAN, Daniel. **Protótipo de um Sistema de Segurança Residencial com Linux Embarcado**. 2008. Trabalho de conclusão de curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

BAUMAN, Zygmunt. **Comunidade:** A busca por segurança no mundo atual. Rio de Janeiro: Zahar, 2003.

BAYER, Fernando Mariano; ECKHARDT, Moacir; MACHADO, Renato (2011). **Automação de Sistemas.** Disponível em: <estudio01.proj.ufsm.br/cadernos_automacao/oitava_etapa/automação_sistemas_2012.pdf >. Acesso em: 04 abr. 2016.

BERNARDES, Manuel; ANTUNES, Pedro; ALMEIDA, Mário (2011). **NFC - Near Field Communication.** Trabalho da disciplina de Seminário de Sistemas e Tecnologias da Informação I. Universidade Atlântica, Portugal. Disponível em: <http://ssti1-1112.wikidot.com/nfc-near-field-communication>. Acesso em: 28 mar. 2016.

CARDIA, Nancy; ADORNO, Sérgio Adorno; Poleto, Frederico. **Homicídio e violação de direitos humanos em São Paulo**, 2003. Disponível em: <http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0103-40142003000100004>. Acesso em: 28 mar. 2016.

CLARK, Sarah. **London Underground to accept NFC payments**, 2014 Disponível em: <http://www.nfcworld.com/2014/07/25/330604/london-underground-accept-nfc-payments/>. Acesso em: 28 mar. 2016.

CRUZ, Aparecido. **Tecnologia de atendimento em recepção e portaria**. Guarulhos: Unifesp, 2009;

DALIGA, Carlos. **Segurança Patrimonial**. Planalitna: Clube de autores, 2012.

FAVORETTO JUNIOR, José Domingos. **O que falta para o NFC decolar no Brasil?**, 2012. Disponível em: <www.mobiletime.com.br/11/07/2012/o-que-falta-para-o-nfc-decolar-no-brasil-/287897/news.aspx>. Acesso em: 02 abr. 2016.

FERRAZ FILHO, Onildo Luciano de Souza. **Comunicação NFC (Near Field Communication) entre Dispositivos Ativos**. 2010. Trabalho graduação em engenharia da computação – Centro Informática, Universidade Federal de Pernambuco, Pernambuco.

GALLO, Francesco. **NFC Tags:** A technical introduction, applications and products, 2011 Disponível em: <http://www.nxp.com/documents/other/R_10014.pdf>. Acesso em: 28 mar. 2016.

GONÇALVES, Daniel Itokazu. **Aspectos relevantes do dano moral trabalhista.** COAD Doutrina e Jurisprudência, n. 27, p. 217-221, 06 jul. 2003.

HOUAISS, Antônio**. Dicionário Eletrônico Houaiss**. Rio de Janeiro, Ed. Objetiva, 2009.

JANES, Ricardo. **Estudo sobre sistemas de segurança em instalações elétricas automatizadas**. 2009.Disponivel em: < http://www.teses.usp.br/teses/disponiveis/3/3143/tde-29062009-181507/publico/Dissertacao_Ricardo_Janes.pdf>. Acesso em: 03 abr. 2016.

KRÜGER, Erasmo. **Protótipo de sistema de segurança predial através de monitoramento utilizando recursos da internet**. 2002. Trabalho de conclusão de curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

LOPES, Cleber da silva. **Como se vigia os vigilantes: O Controle da polícia federal sobre a segurança privada**, 2010. Disponível em: <http://www.scielo.br/pdf/rsocp/v19n40/08.pdf>. Acesso em: 28 mar. 2016.

MAXIMIANO, A. C. A. **Introdução á administração**. 6. ed. São Paulo: Atlas, 2006.

NASSAR, Victor; HORN, Vieira Milton Luiz. **A Internet das coisas com as tecnologias RFID e NFC** São Paulo: Blucjer, 2014.

ORITZ, C. Enrique. **An Introduction to Near-Field Communication and the Contactless Communication API**, 2008. Disponível em: <http://www.oracle.com/technetwork/articles/javame/nfc-140183.html>. Acesso em: 28 mar. 2016.

PRIOSTE, Erasmo. **A prevenção é a melhor precaução**, 2014. Disponível em: <http://www.de-seguranca.com.br/a-prevencao-e-a-melhor-precaucao/>. Acesso em: 19 mai. 2016.

SOUSA, Emanuel Tiago Abreu de. **Sistema de Gestão de Rondas: Aplicação android para empresas de segurança privada**, 2013. Disponível em: <digituma.uma,.pt/bitstream/10400.13/659/1/MestradoTiagoSousa.pdf>. Acesso em: 02 abr. 2016.

TOPDATA. **Sistema para controle de rondas.** Disponível em: <https://www.topdata.com.br/>. Acesso em: 24 nov. 2016.

APÊNDICE A – Descrição dos Casos de Uso

Nos quadros abaixo são apresentados os detalhamentos dos principais casos de uso previstos nos diagramas apresentados na seção 3.2.1.

No Quadro 29 apresenta-se o caso de uso `Logar no sistema WEB` do módulo do servidor.

Quadro 29 - Descrição dos casos de uso UC1

Caso de uso	UC1- Efetuar login
Descrição	Utiliza os dados de identificação do administrador para iniciar uma sessão no sistema web
Ator	Administrador
Pré-condição	O administrador deve estar cadastrado no banco de dados
Fluxo principal	a) O administrador informa seus dados de identificação b) O sistema valida os dados informados pelo administrador c) O sistema redireciona o administrador para a pagina inicial do sistema
Fluxo alternativo	a) Os dados de identificação do administrador foram preenchidos incorretamente b) Alerta com a mensagem “O usuário e a senha foram informados incorretamente” é apresentada
Pós-condição	O administrador inicia uma sessão no sistema web

Fonte: elaborado pelo autor.

No Quadro 30 apresenta-se o caso de uso `Manter vigia` do módulo do servidor.

Quadro 30 - Descrição dos casos de uso UC2

Caso de uso	UC2- Manter vigia
Descrição	Permite ao administrador o cadastro de novos vigias ou a alteração e exclusão de vigias já existentes.
Ator	Administrador
Pré-condição	a) A sessão do administrador deve estar iniciada

Fluxo principal	b) O administrador seleciona a opção de gerenciamento de vigias. c) O sistema apresenta uma tela com todos os vigias já cadastrados e seus dados básicos como nome, rg e cpf d) O administrador acessa a opção de novo vigia, ou seleciona um dos vigias já cadastrados para alterar.
Fluxo alternativo	a) Campo(s) obrigatório(s) não preenchido(s). b) Alerta com a mensagem “Campo obrigatório não foi informado” é apresentada
Cenário - Inclusão	a) O sistema mostra uma tela de cadastro com os dados dos vigias a serem preenchidos. b) O administrador informa todos os campos obrigatórios. c) O administrador seleciona a opção incluir. d) O sistema inclui o cadastro de vigia e apresenta a mensagem “Novo vigia cadastrado com sucesso”.
Cenário - Alteração	a) O sistema mostra todos os vigias já cadastrados. b) O administrador seleciona a opção de alteração de algum dos vigias. c) O sistema abre uma tela de alteração de vigias com os dados do vigia selecionado já preenchidos. d) O administrador realiza as alterações necessárias e) O administrador seleciona a opção alterar. f) O sistema altera o cadastro de vigia e apresenta a mensagem “Vigia alterado com sucesso”.
Pós-condição	O administrador criou ou alterou um vigia.

Fonte: elaborado pelo autor.

No Quadro 31 apresenta-se o caso de uso `Manter rota` do módulo do servidor.

Quadro 31 - Descrição dos casos de uso UC7

Caso de uso	UC7- Manter rota
Descrição	Permite ao administrador o cadastro de novas rotas ou a alteração de rotas já existentes.
Ator	Administrador
Pré-condição	A sessão do administrador deve estar iniciada

Fluxo principal	a) O administrador seleciona a opção de gerenciamento de rota. b) O sistema apresenta uma tela com todas as rotas já cadastradas e seus dados básicos como nome, número de pontos de validação e vigia responsável. c) O administrador acessa a opção de nova rota, ou seleciona uma das rotas já cadastradas para alterar.
Fluxo alternativo	a) Campo(s) obrigatório(s) não preenchido(s). b) Alerta com a mensagem “Campo obrigatório não foi informado” é apresentada
Cenário - Inclusão	a) O sistema mostra uma tela de cadastro com os dados das rotas a serem preenchidos. b) O administrador informa todos os campos obrigatórios. c) O administrador seleciona a opção incluir. d) O sistema inclui o cadastro de rota e apresenta a mensagem “Nova rota cadastrada com sucesso”.
Cenário - Alteração	a) O sistema mostra todas as rotas já cadastradas. b) O administrador seleciona a opção de alteração de alguma das rotas. c) O sistema abre uma tela de alteração de rotas com os dados da rota selecionada já preenchidos. d) O administrador realiza as alterações necessárias e) O administrador seleciona a opção alterar. f) O sistema altera o cadastro de rota e apresenta a mensagem “Rota alterada com sucesso”.
Pós-condição	O administrador criou ou alterou uma ronda.

Fonte: elaborado pelo autor.

No Quadro 32 apresenta-se o caso de uso `Selecionar vigia responsável da rota` do módulo do servidor.

Quadro 32 - Descrição dos casos de uso UC8

Caso de uso	UC8- Selecionar vigia responsável da rota
Descrição	Permite ao administrador a definição de um vigia como o responsável por uma ronda.

Ator	Administrador
Pré-condição	a) A sessão do administrador deve estar iniciada b) Um vigia deve estar cadastrado c) Uma rota deve estar cadastrada
Fluxo principal	a) O administrador seleciona a opção de gerenciamento de rotas. b) O sistema apresenta uma tela com todas as rotas já cadastradas e seus dados básicos como nome, número de pontos de validação e vigia responsável. c) O administrador acessa a opção de definição de cadastro de rotas. d) O sistema mostra uma lista com todos os vigias disponíveis para se tornarem responsáveis pela rota. e) O administrador seleciona a opção de gravação f) O sistema salva o vigia como responsável pela rota e apresenta a mensagem “Vigia vinculado com sucesso”.
Fluxo alternativo	a) Campo(s) obrigatório(s) não preenchido(s). b) Alerta com a mensagem “Campo obrigatório não foi informado” é apresentada
Pós-condição	O administrador definiu um vigia responsável para uma rota.

Fonte: elaborado pelo autor.

No Quadro 33 apresenta-se o caso de uso `Logar no sistema` do módulo do dispositivo móvel.

Quadro 33 - Descrição dos casos de uso UC14

Caso de uso	UC14 - Logar no sistema
Descrição	Utiliza os dados de identificação do vigia ou administrador para iniciar uma sessão no sistema do dispositivo móvel
Ator	Vigia ou Administrador
Pré-condição	O vigia ou administrador deve estar cadastrado no banco de dados

Fluxo principal	a) O vigia ou administrador informa seus dados de identificação b) O sistema valida os dados informados pelo vigia c) O sistema redireciona o vigia para a tela inicial do sistema
Cenário – Efetuar *login* via QR code	a) O vigia ou administrador seleciona a opção de efetuar *login* ao ler QR code b) O vigia ou administrador lê o QR code com o aparelho c) O sistema redireciona o vigia para a tela inicial do sistema
Fluxo alternativo	a) Os dados de identificação do vigia foram preenchidos incorretamente b) Alerta com a mensagem "O usuário e a senha foram informados incorretamente" é apresentada
Pós-condição	O vigia inicia uma sessão no aplicativo do dispositivo móvel

Fonte: elaborado pelo autor.

No Quadro 34 apresenta-se o caso de uso `Preencher tag NFC` do módulo do dispositivo móvel.

Quadro 34 - Descrição dos casos de uso UC16

Caso de uso	UC16 - Preencher *tag* NFC
Descrição	Permite ao administrador a vinculação de um agente passivo de NFC a um ponto de checagem
Ator	Administrador
Pré-condição	a) O administrador deve estar cadastrado no banco de dados b) Um ponto de checagem deve estar cadastrado
Fluxo principal	a) O administrador seleciona a opção de vinculação de ponto de checagem. b) O administrador seleciona um ponto de checagem a ser vinculado. c) O sistema mostra a mensagem "Aproxime o aparelho da *Tag* a ser vinculada" d) O administrador aproxima o dispositivo do agente

	passivo e) O sistema passa as informações de identificação do ponto de checagem do dispositivo para o agente passivo.
Pós-condição	O agente passivo esta vinculado com um ponto de checagem

Fonte: elaborado pelo autor.

No Quadro 35 apresenta-se o caso de uso `Verificar próximo ponto de checagem` do módulo do dispositivo móvel.

Quadro 35 - Descrição dos casos de uso UC19

Caso de uso	UC19 - Verificar próximo ponto de checagem
Descrição	Permite ao vigia verificar qual será o próximo ponto de checagem que precisa ser verificado em sua ronda atual.
Ator	Vigia
Pré-condição	a) O vigia deve estar cadastrado no banco de dados b) Uma execução de ronda deve estar sendo executada
Fluxo principal	a) O vigia seleciona a opção de verificação de seu próximo ponto de checagem. b) O sistema informa ao vigia qual é o próximo ponto de checagem a ser validado, junto com uma breve descrição da localização.
Pós-condição	O sistema informa ao vigia seu próximo ponto de checagem a ser realizado dentro da rota.

Fonte: elaborado pelo autor.

No Quadro 36 apresenta-se o caso de uso `Validar ponto de checagem com NFC` do módulo do dispositivo móvel.

Quadro 36 - Descrição dos casos de uso UC20

Caso de uso	UC20- Validar ponto de checagem com NFC
Descrição	Permite ao vigia realizar a validação de um ponto de checagem utilizando o NFC.
Ator	Vigia
Pré-condição	a) O vigia deve estar cadastrado no banco de dados b) Uma execução de ronda deve estar sendo executada

Fluxo principal	a) O vigia seleciona a opção de realizar a validação de ponto de checagem b) O sistema mostra a mensagem “Aproxime o aparelho a *tag* a ser validada” c) O vigia aproxima o aparelho ao agente passivo d) O sistema verifica as informações de identificação do ponto de checagem do dispositivo com o agente passivo. e) O sistema realiza a verificação do ponto de checagem
Pós-condição	O ponto de checagem é devidamente validado pelo sistema

Fonte: elaborado pelo autor.