UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

TOBIAS HENRIQUE CALEGARI

SISTEMA WEB PARA GERENCIAMENTO DE MANIFESTAÇÕES DE UMA OUVIDORIA PÚBLICA MUNICIPAL

TRABALHO DE CONCLUSÃO DE CURSO

PATO BRANCO
2018

TOBIAS HENRIQUE CALEGARI

SISTEMA WEB PARA GERENCIAMENTO DE MANIFESTAÇÕES DE UMA OUVIDORIA PÚBLICA MUNICIPAL

Trabalho de Conclusão de Curso de graduação, apresentado à disciplina de Trabalho de Conclusão de Curso 2, do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco, como requisito parcial para obtenção do título de Tecnólogo.

Orientadora: Profa. Andreia Scariot Beulke

PATO BRANCO
2018

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Pato Branco
Departamento Acadêmico de Informática
Curso de Tecnologia em Análise e Desenvolvimento de Sistemas

TERMO DE APROVAÇÃO

TRABALHO DE CONCLUSÃO DE CURSO

SISTEMA WEB PARA GERENCIAMENTO DE MANIFESTAÇÕES DE UMA OUVIDORIA PÚBLICA MUNICIPAL

POR

TOBIAS HENRIQUE CALEGARI

Este trabalho de conclusão de curso foi apresentado no dia 28 de junho de 2018, como requisito parcial para obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas, pela Universidade Tecnológica Federal do Paraná. O acadêmico foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

Banca examinadora:

Profª MSc Andreia Scariot Beulke
Orientador

Prof. Esp. João Guilherme Brasil Pichetti

Profª Drª Beatriz Terezinha Borsoi

Prof. Dr. Edilson Pontarolo
Coordenador do Curso de Tecnologia em Análise e Desenvolvimento de Sistemas

Profª Drª Beatriz Terezinha Borsoi
Responsável pela Atividade de Trabalho de Conclusão de Curso

A Folha de Aprovação assinada encontra-se na Coordenação do Curso.

RESUMO

CALEGARI, Tobias Henrique. Sistema web para gerenciamento de manifestações de uma ouvidoria pública municipal. 2018. 58f. Monografia (Trabalho de Conclusão de Curso) - Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2018.

Uma Ouvidoria Pública representa um canal direto de comunicação entre o cidadão e o órgão governamental. Cabe a esse órgão a responsabilidade de que as manifestações dos cidadãos sejam encaminhadas para o setor responsável e que este responda ao cidadão de forma mais rápida e eficiente. Quando o processo de recebimento das manifestações ocorre de forma manual ou por meio de manipulação de arquivos individuais, o processo interno de encaminhamento e gerenciamento dos dados torna-se custoso e, muitas vezes, inoperante. Assim, o desenvolvimento de um sistema *web* que gerencie os dados dessas informações contribuirá para que as ações realizadas pelos usuários sejam efetuadas dinamicamente e os dados armazenados em um banco de dados, permitindo maior controle e eficiência das informações. O desenvolvimento do sistema proposto neste trabalho possibilitará a automatização do recebimento e encaminhamento das manifestações, bem como o acompanhamento *online* do seu *status*. O sistema também auxiliará o órgão público na obtenção de dados estatísticos, como, por exemplo, quais áreas da cidade possuem mais manifestações ou qual secretaria recebe mais manifestações. Este trabalho apresenta o desenvolvimento de um sistema *web* que gerencia o recebimento e trâmite de manifestações públicas municipais. O sistema apoiará esse processo, tanto para o órgão público, que ganha em agilidade e confiabilidade dos dados, quanto para o cidadão que poderá obter o retorno de sua manifestação de forma mais ágil.

Palavras-chave: Ouvidoria pública municipal. Gerenciamento de informações. Manifestações.

ABSTRACT

CALEGARI, Tobias Henrique. Web system for managing manifestations of a municipal public ombudsman. 2018. 58f. Monografia (Trabalho de Conclusão de Curso) - Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2018.

An Ombudsman Public represents a direct communication channel between the citizen and the government agency. It is the responsibility of this body that the manifestations of citizens will be sent to the responsible sector and that the individual responds to the citizen in a more agile way. When the process of receiving manifestations occurs manually or through manipulation of individual files, the internal process of routing and managing the data flow becomes expensive and often, ineffective. Thus, the development of a Web Information System (WIS) that manages the flow of this information will contribute to the actions performed by the users, by being dynamically performed and the data to be stored in a database, allowing greater control and efficiency in the flow of information. The development of the proposed system will enable the automation of the inbound and forwarding of the manifestations, as well as the online monitoring of their status. The system will also assist the public agency by obtaining statistical data such as which areas of the city have more manifestations or which sector receives more manifestations. This work presents the development of a web system that manages the reception and processing of municipal public manifestations. The system will support this process, both for the public agency, which gains agility and reliability of the data, and for the citizen who can obtain the return of his manifestation in a more agile way.

Keywords: Municipal public ombudsman. Information management. Manifestations.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTAGEM DE CÓDIGOS

LISTA DE SIGLAS

API	*Application Programming Interface*
CPF	Cadastro de pessoas físicas
CSS	*Cascading Style Sheets*
HTML	*HyperText Markup Language*
HTTP	*HyperText Transfer Protocol*
IP	*Internet Protocol*
MVC	*Model*, *View* e *Controller*
PDF	*Portable Document Format*
RF	Requisitos Funcionais
RNF	Requisitos Não Funcionais
SI	Sistemas de Informação
TCP	*Transmission Control Protocol*
URI	*Uniform Resource Identifier*
W3C	*World Wide Web Consortium*

SUMÁRIO

1 INTRODUÇÃO

Este capítulo apresenta as considerações iniciais, os objetivos e a justificativa da realização deste trabalho. No final do capítulo é apresentada a organização do texto que contém a estruturação dos seus capítulos.

1.1 CONSIDERAÇÕES INICIAIS

A interação entre o cidadão e o município é de crucial importância para uma gestão voltada ao interesse público. "A ouvidoria representa, assim, um *lócus* privilegiado para o exercício da democracia participativa, conquista da cidadania e controle social da *res* pública"[1] (CARDOSO, 2010, p.5).

A Ouvidoria Municipal permite a interação entre o cidadão do serviço público e o município. O Fala Cidadão é um exemplo de Ouvidoria Municipal que permite que a comunidade registre suas manifestações (reclamações, dúvidas, sugestões, solicitações de serviços e denúncias) e, assim, permite uma participação mais ativa na gestão pública. Nesse sentido, a Ouvidoria permite mapear a qualidade do serviço prestado na perspectiva da comunidade municipal.

De acordo com De Mario (2006) as informações que uma Ouvidoria acumula e o potencial que tem para trabalhar estas informações são fundamentais para o seu adequado funcionamento. Considerando que uma Ouvidoria Pública Municipal recebe diariamente muitas manifestações da comunidade, é importante que essas informações sejam gerenciadas adequadamente para permitir a qualidade no atendimento. Essa qualidade pode ficar comprometida se essas informações forem gerenciadas por um sistema de arquivos e de forma centralizada. A utilização de um sistema de arquivos pode gerar falhas na forma como os relatórios são apresentados, pois os dados podem se tornar inconsistentes e imprecisos devido à falta de confiabilidade que esse tipo de sistema pode trazer para o armazenamento e a manipulação dos dados.

A centralização dessas informações, por meio de sistema de arquivos individuais, dificulta o acompanhamento e o gerenciamento do chamado por cada cidadão. A coleta manual das manifestações acarreta prejuízos no desempenho dos serviços, interferindo no tempo de resposta para o usuário da Ouvidoria Municipal.

[1] Expressão latina que significa "coisa publica".

Diante do exposto, este trabalho apresenta o desenvolvimento de um sistema *web* que atende aos cidadãos e aos servidores municipais. O sistema permite o controle das informações recebidas pelo gestor por meio do registro *online* das manifestações direcionadas a cada setor municipal. O servidor responsável pelo controle dos registros validará a filtragem das informações por setores e os servidores terão acesso somente às informações pertinentes ao setor de sua responsabilidade. O sistema permite que o cidadão acompanhe o desenvolvimento do serviço solicitado por meio do *status* de gerenciamento da solicitação.

1.2 OBJETIVOS

A seguir são apresentados os objetivos gerais e específicos do sistema proposto. O objetivo geral está relacionado com o resultado principal da realização deste trabalho e os objetivos específicos complementam o geral em termos de funcionalidades do sistema.

1.2.1 Objetivo Geral

Desenvolver um sistema *web* que possibilite o gerenciamento das informações recebidas por meio de Ouvidoria Pública Municipal.

1.2.2 Objetivos Específicos

- Proporcionar eficiência e agilidade no atendimento das manifestações da comunidade municipal.
- Possibilitar o gerenciamento dos registros das manifestações da comunidade para os setores do órgão público.
- Possibilitar que cada cidadão acompanhe as manifestações que realiza.

1.3 JUSTIFICATIVA

Para que os cidadãos tenham participação na vida social e política de uma comunidade é importante que ele exerça sua cidadania por meio do engajamento em programas sociais, associação de bairros e outras formas de participação ativa na sociedade. Além disso, exercer a cidadania significa denunciar, exigir e cobrar seus direitos dos órgãos públicos, os serviços essenciais que assegurem dignidade e qualidade de vida. A cidadania pode estar também na participação ativa por meio de sugestões de solução para problemas e de melhorias, na manifestação de apoio ao que está sendo adequadamente sendo realizado e no oferecimento de ajuda.

Para atender esse vínculo com o cidadão, a Ouvidoria Pública Municipal é um órgão da administração pública que permite o diálogo entre o cidadão e o Governo e que visa promover a participação do cidadão na gestão pública por meio de reivindicações e do acompanhamento e avaliação dos serviços prestados. De acordo com De Mario (2011) o papel das Ouvidorias são os direitos sociais e as políticas voltadas para sua garantia, estes constituem o cerne das reclamações recebidas.

Para um bom desempenho dos serviços prestados à comunidade é necessário que o as informações recebidas diariamente pelas Ouvidorias sejam gerenciadas adequadamente, permitindo agilidade, segurança e integridade dos dados e isso é possível com o uso de Sistemas de Informação (SI).

Nesse sentido, o desenvolvimento deste trabalho se justifica pela necessidade do gerenciamento e controle das informações recebidas pelas Ouvidorias e para promover a comunicação entre o órgão público e o cidadão permitindo, dessa forma, que a comunidade acompanhe *online* o *status* de suas reivindicações.

O desenvolvimento de um sistema *web* obtido como resultado da realização deste trabalho visa facilitar a descentralização do sistema de arquivos que irá gerir os processos da Ouvidoria Municipal.

O sistema proposto neste trabalho será desenvolvido com o padrão *Model*, *View* e *Controller* (MVC) que garante a padronização do desenvolvimento do sistema, pois traz a separação da camada visual da aplicação da de domínio e do negócio (SILVEIRA *et* al., 2011). Em se tratando de aplicações desenvolvidas utilizando a linguagem Java, a camada do *model* é representada pelo JavaBeans ou banco de dados, a camada de *view* é representada por páginas JSP que mostram as informações para o usuário e a camada *controller* é um *servlet*" que controla as interações entre o *model* e a *view* (SAMPAIO, 2007).

Para o desenvolvimento *back-end* foi utilizado o *framework* Spring que fornece um modelo de programação e configuração para a criação de aplicações baseadas em Java (GUTIERREZ, 2014). O Spring baseia-se nas boas práticas e padrões de projeto e fornece suporte nativo às tecnologias de acesso a dados, como, por exemplo, o Hibernate que foi utilizado para a persistência de dados.

1.4 ESTRUTURA DO TRABALHO

Este texto está organizado em capítulos. O Capítulo 2 apresenta o referencial teórico. No Capítulo 3 são apresentados os materiais e o método utilizados para o desenvolvimento do trabalho. No Capítulo 4 está o resultado da realização do trabalho, que é a modelagem e o desenvolvimento do sistema. Por fim estão as considerações finais seguidas das referências utilizadas na composição do texto.

2 REFERENCIAL TEÓRICO

Este capítulo apresenta a fundamentação teórica deste trabalho, cujo conteúdo explana sobre os padrões e arquitetura de sistemas *web* e o modelo MVC que é utilizado para separar a arquitetura do sistema em camadas.

2.1 PADRÕES DE SISTEMAS WEB

O Consórcio *World Wide Web* (W3C) é uma comunidade internacional que desenvolve padrões para garantir que a *web* continue crescendo, composto por empresas, órgãos governamentais e organizações independentes. O W3C é responsável por desenvolver padrões, especificações e orientações técnicas, garantindo assim qualidades técnicas e editoriais (W3C*,* 2016).

Com o crescimento das aplicações *web* e a facilidade de implantação que elas possuem, necessitado somente de um web *browser* instalado para seu funcionamento, há uma necessidade de impor estes padrões e regras para o seu desenvolvimento. Com o uso desses padrões as aplicações *web* tornam-se ferramentas compatíveis com qualquer *browser* em qualquer sistema operacional. O *Hypertext Markup Language* (HTML), segundo Luckow e Melo (2010, p. 105), é utilizado para estruturar o conteúdo dos documentos web, juntamente com o *Cascading Style Sheets* (CSS) que é responsável por estruturar a parte estética da aplicação *web*, compõem o básico para a criação de uma página *web*.

Segundo o W3C, o HTML na versão 5, tem como principal objetivo facilitar a manipulação de elementos, possibilitando ao desenvolvedor modificar características dos objetos sem causar grande impacto ao usuário final. Também traz uma interação com CSS e JavaScript por meio de *Application Programming Interface* (API). Uma API fornece meios para a manipulação de características desses elementos e garante que a aplicação continue leve e funcional.

O CSS3 trouxe uma nova forma de trabalhar com o estilo de uma página *web*. Segundo Mazza (2012, p.61), a introdução de elementos na nova versão do CSS possibilitou a criação de estilos, inserção de cores e elementos, diretamente por linhas de código, sem o auxílio de programas externos. Foram reduzidas também a quantidade de arquivos de configuração do CSS e eliminadas as dependências de arquivos externos que impactam diretamente no desempenho de sites.

Um conceito aplicado às novas versões do CSS e HTML é a responsividade. Para Bryant e Jones (2012), responsividade é a capacidade de uma aplicação *web* adaptar-se às diversas resoluções em dispositivos diferentes. Uma técnica introduzida no CSS3 para implementar o conceito de responsividade são as *medias queries*. Por meio do elemento "@media" é possível criar condições para que o leiaute se adapte às diferentes resoluções de tela do computador ou dispositivo utilizado para acesso.

A Figura 1 mostra um exemplo do uso desse elemento. Nesse exemplo, a cor de fundo está condicionada pela resolução da tela.

```
@media only screen and (max-width: 500px) {
    body {
        background-color: lightblue;
    }
}
```

Figura 1 – Exemplo de código responsivo.
Fonte: W3CSchools

2.2 ARQUITETURA DE SISTEMAS WEB

Segundo o W3C a *World Wide Web* é um espaço para que as informações de interesse do usuário sejam identificadas por marcadores globais chamados de *Uniform Resource Identifier* (URI). Segundo a W3C a arquitetura básica da *web* é dividida em três bases principais: identificação, interação e formatos.

Na identificação são usadas URIs como identificadores de recursos. O recurso é o que a página *web* traz de informação para o usuário.

A interação é caracterizada por agentes *web* que se comunicam usando protocolos padronizados para trocas de mensagens que são definidas por uma sintaxe semântica. Inserindo uma URI no *browser,* o usuário solicita que ele entre em contato com o servidor do recurso. O navegador, por sua vez, envia uma solicitação *Hypertext Transfer Protocol* (HTTP) GET para o servidor por meio do TCP/IP na porta 80 e o servidor envia uma mensagem retornando o recurso que ele determina ser correspondente a URI solicitada.

Para que o recurso seja interpretado pelo *browser* é preciso que contenha os métodos necessários para a leitura em sua programação. Ao receber uma solicitação HTTP, o servidor processa as informações e envia partes do recurso por meio de dados da representação e metadados que são representados por meio do cabeçalho da mensagem. De acordo com a

W3C, o protocolo HTTP transmite normalmente um único octeto de dados acrescido do cabeçalho da mensagem e utiliza, também, alguns recursos como o "*Content-Type*" e o "*Content-Encoding*" para identificar o melhor formato da representação. O *browser* recebe a mensagem do servidor e está programado para interpretar a mensagem, este processo continua até que todo o recurso seja carregado na página do usuário.

2.3 MODELO MVC

Para facilitar o desenvolvimento de uma aplicação, são utilizados alguns padrões de arquitetura de desenvolvimento. De acordo com Hemrajani (2007, p. 109) o modelo MVC fornece uma separação perfeita entre apresentação e dados, é muito utilizado em projetos *web* e pode ser aplicado também em qualquer tipo projeto de software. Luckow e Melo (2010, p. 180) também descrevem o modelo MVC, como um padrão que determina a divisão do sistema em três camadas, chamadas de *model*, *view* e *controller*.

A camanda *model* é responsável pela camada de acesso a dados e também responsável por conter as regras de negócio.

A camada *view* compõe a interface do sistema, todas as informações geradas na camada *model* são exibidas na *view*. Também é nessa camada que ocorre a interação do usuário com o sistema.

A camada *controller* é responsável por conectar a camada *model* com a camada *view*. O *controller* busca as informações da *model* para gerar a *view* e recebe as informações da *view* e envia para a *model*. A Figura 2 apresenta o modelo isquêmico de um modelo MVC.

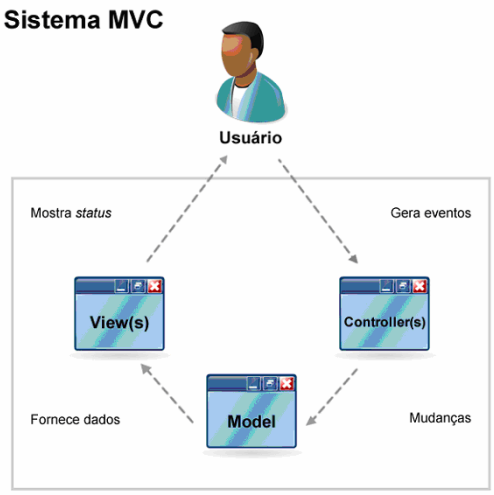

Figura 2 – Modelo MVC.
Fonte - wooxt (2014).

O MVC separa a comunicação entre as camadas, sendo que a *View* não pode visualizar a *Model* e esta não tem contato com a *View*. Assim, a comunicação entre ambas é intermediada pela camada *Controller*.

3 MATERIAIS E MÉTODO

Este capítulo apresenta os materiais e o método utilizados para o desenvolvimento deste trabalho. Os materiais elencam as ferramentas e as tecnologias utilizadas na modelagem, banco de dados, linguagem e *frameworks* para programação *front e back-end.* O método está relacionado a um modelo de processo prescritivo que visa ordenar e estruturar o desenvolvimento do aplicativo.

3.1 MATERIAIS

Ferramenta / Tecnologia	Versão	Finalidade
Bootstrap	3	*Framework* para estilização de conteúdos *web*.
Eclipse Mars	4.5.2	Programação em Java/*Web*
Hibernate	5.2.2	Persistência de dados
Java	8	Linguagem de programação
JQuery	3.1.0	*Framework front-end*
Lombok	1.18.0	*Framework para evitar repetição de códigos padrão*
Postgresql	9.3	Criação e gerenciamento do banco de dados
Spring MVC	3	*Framework back-end*
Visual Paradigma	13.1	Modelagem UML (casos de uso e diagrama de classes)

Quadro 1 - Ferramentas e tecnologias usadas no desenvolvimento do sistema

3.2 MÉTODO

O método utilizado neste trabalho foi o processo sequencial linear de Pressman e Maxim (2016). Nesse modelo as atividades e as tarefas ocorrem sequencialmente, com diretrizes de progresso definidas começando com a especificação dos requisitos do cliente e avançando pelas fases de modelagem, construção e disponibilização e culminando no suporte contínuo do software concluído (PRESSMAN, MAXIM, 2016, p. 42). Esse modelo foi utilizado porque o sistema não envolve muitos requisitos, embora a aplicação tenha implementação relativamente complexa em virtude das regras de negócio envolvidas.

Esse processo define as fases de comunicação, modelagem, construção e entrega. Neste trabalho são apresentadas as três primeiras etapas.

Na fase de comunicação ocorre o início do projeto com o levantamento de requisitos que neste trabalho foi realizado em dois momentos. No primeiro, foi utilizada a técnica de entrevista aberta. De acordo com Minayo (2010), essa técnica é utilizada para obter o maior

número possível de informações sobre determinado tema que, neste trabalho, são os requisitos funcionais e não funcionais do sistema proposto.

A entrevista foi realizada com um funcionário responsável pelo setor de "fala cidadão" da Prefeitura de Pato Branco, que é uma Ouvidoria Pública que visa receber manifestações da comunidade sobre os serviços prestados. Foi observado como as informações são captadas e distribuídas pela Ouvidoria. As informações obtidas na entrevista foram registradas em documento de texto.

No segundo momento, foram consultados alguns sistemas que oferecem serviços semelhantes ao proposto neste trabalho. Essa busca visou compreender a melhor forma de organizar a gestão de dados por meio do sistema. Foram encontrados sistemas que recebem protocolos para requisições de assuntos mais específicos, como, por exemplo, o envio de um projeto para aprovação no setor de engenharia. Porém esse sistema não abrange o modelo proposto para o recebimento de manifestações dos cidadãos de modo anônimo.

Após esta etapa foram definidos os requisitos funcionais e não funcionais do sistema. Os requisitos funcionais referem-se a uma função do sistema, como o sistema irá se comportar em relação a uma interação do usuário, por exemplo. De acordo com Wazlawick (2011, p. 25), "os requisitos não funcionais sempre aparecem ligados aos requisitos funcionais e podem ser de dois tipos: lógicos ou tecnológicos". Esse autor descreve que os requisitos não funcionais lógicos são as regras de negócio correspondentes ao requisito em que ele faz parte e os requisitos tecnológicos referem-se à tecnologia para a realização da função, como, por exemplo, a interface e restrições de segurança.

Na etapa de modelagem foi realizada a análise do projeto com a construção dos diagramas de caso de uso e os das principais atividades do sistema. A partir desses documentos foram geradas informações para a modelagem do banco de dados.

Posteriormente à elaboração dos casos de uso e dos diagramas de atividades, foi criado o diagrama de entidade e relacionamentos, e geradas as tabelas, com os tipos de campos, tamanhos e relacionamentos.

A última etapa realizada foi a implementação do sistema utilizando as ferramentas descritas no Quadro 1.

4 RESULTADOS

Este capítulo apresenta o resultado deste trabalho que é o desenvolvimento de um sistema *web* para o gerenciamento das informações recebidas por meio de Ouvidoria Pública Municipal.

4.1 ESCOPO DO SISTEMA

O sistema de gerenciamento dos dados de uma Ouvidoria Pública Municipal possibilitará o recebimento de manifestações por meio de um formulário *web*, a gerência e o encaminhamento dessas manifestações para seus devidos responsáveis e consulta *on-line* por meio de um protocolo gerado pelo sistema.

No formulário da manifestação será necessário informar nome, endereço, telefone, tipo de manifestação, para qual secretaria o cidadão deseja encaminhá-la e um breve comentário sobre o assunto. Será possível, caso o cidadão deseje, criar uma manifestação anônima, os campos obrigatórios para a manifestação anônima serão o tipo da manifestação, secretaria para qual deseja encaminhar e o comentário sobre a manifestação. Após o preenchimento dos campos e do envio da manifestação, o sistema exibirá um número de protocolo, para uma futura consulta e visualização do *status*. O tipo de manifestação “elogio” não gerará número de protocolo. O cidadão poderá por meio do número de protocolo consultar o *status* da sua manifestação utilizando uma página *web* que é parte do sistema.

O operador receberá todas as manifestações e ficará responsável por conferir se o assunto da manifestação é correspondente com a secretaria, escolhida pelo cidadão no preenchimento do formulário. Caso não seja, o operador alterará a secretaria e por fim encaminhará a manifestação para o setor responsável, também ficará responsável por cadastrar e alterar dados das secretarias quando necessário. O operador poderá consultar por nome, tipo de manifestação, meio de contato, data, secretaria.

O responsável pela secretaria receberá as manifestações correspondentes e não poderá visualizar os chamados de outros setores. O responsável fica encarregado de atualizar o *status* da manifestação e de finalizar e transmitir a resposta final ao cidadão. Assim como o operador, este usuário também poderá consultar por tipo de manifestação, meio de contato, e por *status* da manifestação, porém, ele não poderá consultar manifestações por secretarias.

O administrador ficará responsável por manter o tipo de manifestação, *status*, usuários e meio de contato. O tipo de manifestação será cadastrado apenas pelo nome, assim como *status* e meio de contato. No formulário de cadastro de usuários será necessário o preenchimento de dados pessoais como o Cadastro de Pessoas Física (CPF), nome completo, *e-mail*, setor e senha de acesso.

4.2 MODELAGEM DO SISTEMA

Os quadros numerados de 2 a 8 apresentam a descrição dos requisitos funcionais e os requisitos não funcionais do sistema. Nos quadros, as siglas RF e RNF são utilizadas para descrever os requisitos funcionais e requisitos não funcionais, respectivamente. Cada RF apresenta a descrição do requisito e seus respectivos RNF.

F1 – Registrar Manifestação	
Descrição: O sistema deve permitir que o cidadão cadastre uma manifestação de forma anônima ou não.	
Requisitos Não-Funcionais	
Nome	Restrição
NF1.1 - Preencher campos	Se a manifestação não for anônima, o cidadão deve preencher os campos de endereço, nome, telefone, tipo de manifestação, assunto e mensagem.
NF1.2 - Preencher campos anônimos	Se a manifestação for anônima, são de preenchimento obrigatório apenas os campos tipo de manifestação, assunto e mensagem.
NF1.3 - Validação	Os campos nome, endereço, telefone, assunto e mensagem serão validados por uma função do sistema.
NF1.4 - Selecionar meio de contato	Se o cidadão realizar a manifestação pessoalmente, por *e-mail* ou contato telefônico, o operador irá registrá-la no sistema e escolherá o meio pelo qual o cidadão realizou o contato.
NF1.5 - Definir contato online	Se o meio de contato for *online*, a opção de escolha do meio ficará oculta e o contato será por padrão definido como "*online"*.
NF1.6 - Inclusão da secretaria	O cidadão poderá selecionar a secretaria para qual ele deseja direcionar a manifestação ou deixar o campo em branco.
NF1.7 - Gerar número de protocolo	O sistema deve gerar um número único de protocolo para cada manifestação registrada. Esse número é informado ao final do registro. O protocolo é utilizado para acompanhamento do *status* da manifestação.
NF1.8 - Validar tipo de manifestação	Caso o tipo de manifestação seja "elogio" o sistema não gerará um protocolo para consulta. Se o usuário é identificado e houver *e-mail* no cadastro poderá ser enviada mensagem. De qualquer forma, uma mensagem padrão de agradecimento será apresentada na tela.

Quadro 2 - Registrar Manifestação

F2 – Receber Manifestação	
Descrição: O sistema deve permitir que o operador valide as informações de uma manifestação e encaminhe a sua respectiva secretaria.	
Requisitos Não-Funcionais	
Nome	Restrição
NF2.1 - Confirmar Secretaria	O operador poderá modificar o campo da secretaria para direcionar a manifestação à secretaria adequada.

NF2.2 - Alterar Secretaria	O operador poderá alterar a secretaria caso o assunto da manifestação não seja correspondente a que o cidadão cadastrou.
NF2.3 - Enviar Manifestação	Após conferir os dados o operador encaminhará a manifestação para a respectiva secretaria.

Quadro 3 - Receber Manifestação

F3 – Gerenciar Manifestação	
Descrição: O sistema deve permitir ao responsável que não possui privilégios administrativos, o recebimento, visualização, modificação de *status* e envio de respostas a manifestação que foi direcionada a sua respectiva secretaria.	
Requisitos Não-Funcionais	
Nome	Restrição
NF3.1 - Confidencialidade	O responsável só poderá visualizar a manifestação de sua própria secretaria.
NF3.2 - Pesquisar manifestação	O responsável poderá pesquisar a manifestação por data, nome do solicitante e endereço.
NF3.3 - Pesquisar manifestação	O operador poderá, além de pesquisar pelos itens do responsável comum, também filtrar por secretarias, e meio de contato.
NF3.4 - Atualizar Status	O responsável poderá atualizar a situação de sua respectiva manifestação, utilizando rótulo para o *status* (em análise, deferido, indeferido) e um comentário, caso necessário.

Quadro 4 – Gerenciar Manifestação

F4 – Consultar Manifestação	
Descrição: O sistema deve permitir que o responsável consulte a manifestação, por data e assunto. No caso do operador, também poderá ser feita uma consulta por secretaria.	
Requisitos Não-Funcionais	
Nome	Restrição
NF4.1 - Confidencialidade	O responsável só poderá consultar a manifestação da própria secretaria.
NF4.2 - Privilégios administrativos	O operador poderá consultar as manifestações de todas as secretarias.
NF4.3 - Consulta por protocolo	O cidadão poderá consultar o *status* das suas manifestações *online*, por meio do número de protocolo único fornecido no momento do cadastro de cada manifestação.

Quadro 5 - Consultar Manifestação

F5 – Manter usuários	
Descrição: O administrador poderá desativar, cadastrar novos usuários e editá-los no sistema.	
Requisitos Não-Funcionais	
Nome	Restrição
NF5.1 - Inclusão	O cadastro de usuário terá como campos obrigatórios CPF, nome completo e secretaria.
NF5.2 - Inclusão	O administrador escolherá qual secretaria o responsável estará vinculado e quais privilégios possuir.
NF5.3 - Validação	Os campos de nome, CPF e secretaria, serão validados por meio de uma função no sistema.
NF5.4 - Desativar usuários	O administrador poderá somente desativar usuários que foram desligados da secretaria.
NF5.5 - Editar	O administrador poderá editar o cadastro dos usuários.
NF5.6 - Trocar senha	O usuário terá a opção de trocar a senha do seu *login*.

Quadro 6 - Manter usuários

F6 – Alterar Secretarias	
Descrição: O operador poderá cadastrar novas secretarias no sistema, editar e desativá-las.	
Requisitos Não-Funcionais	
Nome	Restrição
NF6.1 - Inclusão de secretarias.	O administrador poderá incluir o nome da secretaria, o CPF e o nome do secretário responsável pela respectiva secretaria.
NF6.2 - Validação	Os campos nome da secretaria, CPF e nome do responsável pela secretaria serão validados por meio de uma função no sistema.
NF6.3 - Desativar secretarias	O administrador poderá desativar secretarias.
NF6.4 - Excluir	O operador não poderá excluir secretarias.
NF6.5 - Editar	O administrador poderá editar secretarias.

Quadro 7 - Alterar secretarias

F7 – Manter Meio de contato	
Descrição: O operador poderá cadastrar, alterar e excluir meios de contato.	
Requisitos Não-Funcionais	
Nome	Restrição
NF7.1 - Cadastrar	O meio de contato será cadastrado tendo como campo obrigatório único o nome.
NF7.2 - Validar Campos	O campo nome será validado por uma função no sistema.

Quadro 8 - Manter Meio de contato

A Figura 3 apresenta o diagrama de casos de uso do sistema. Esse diagrama contém as funcionalidades essenciais do sistema que são realizadas por seus atores representados pelo cidadão, responsável, operador e administrador. O administrador será responsável por manter meio de contato, usuários, *status* e tipo de manifestação. Ele terá acesso total ao sistema e poderá realizar cadastro de usuários, incluir novos tipos de manifestação, meios de contatos e *status*. O operador será responsável por registrar e receber as manifestações, manter secretarias. Ainda, o operador realizará a validação das manifestações e as encaminhará para secretarias correspondentes. O responsável receberá as manifestações direcionadas ao seu setor, atualizará o *status* da manifestação até que ela seja respondida e finalizada e poderá consultar as manifestações somente da sua secretaria. O cidadão acessará a tela de cadastro de manifestação e preencherá os dados solicitados para o que deseja. Por meio de um número de protocolo, que será fornecido pelo sistema após o término do registro, o cidadão poderá consultar a situação da sua manifestação.

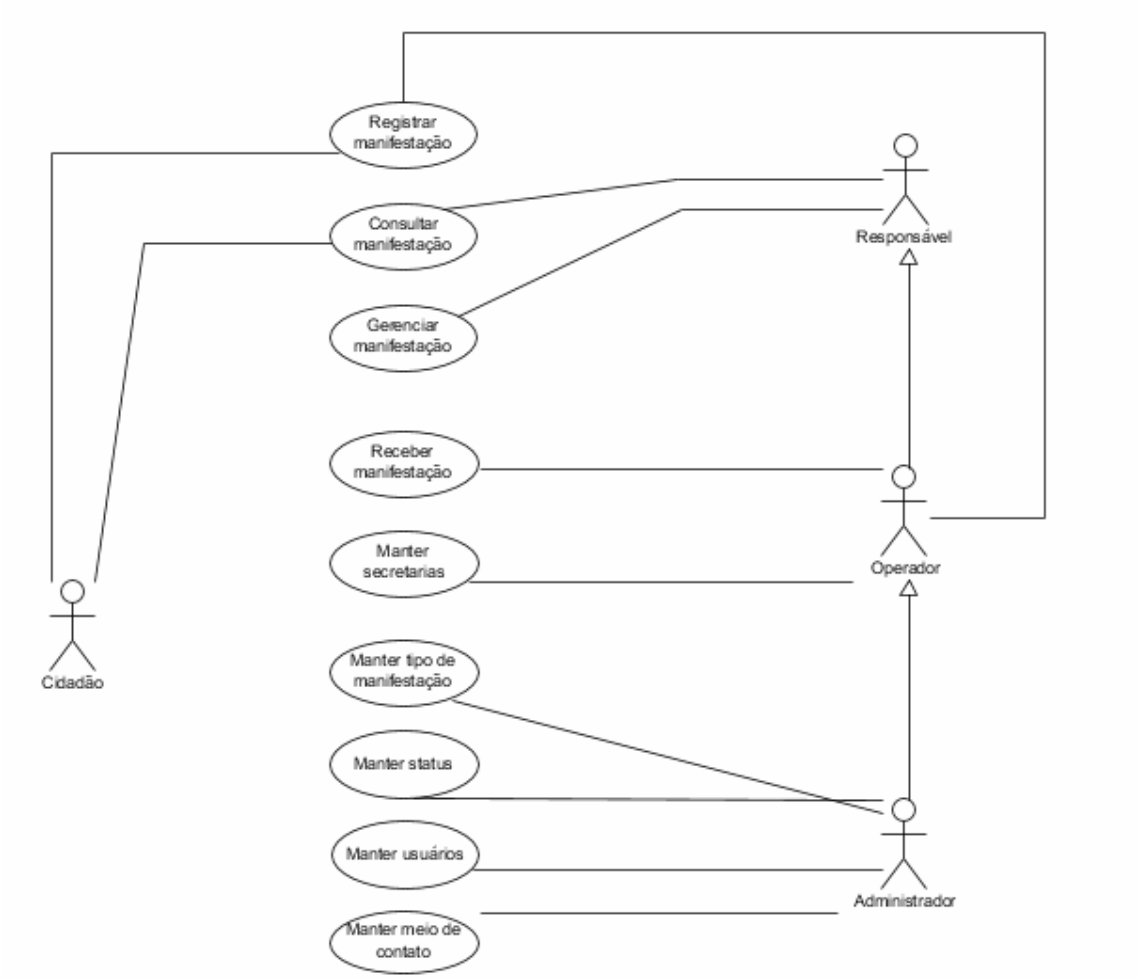

Figura 3 – Casos de uso

Os quadros numerados de 9 a 17 apresentam a descrição dos casos de uso apresentados na Figura 3.

Caso de uso: Inserir manifestação (refere-se à operação de inclusão no caso de uso "Registrar Manifestação"). **Descrição:** Ator inclui dados no sistema. **Atores:** Cidadão, operador e administrador. **Pré-condição:** Não há. **Sequência de Eventos:** 1. Ator acessa a tela para cadastrar as informações solicitadas. 2. O sistema insere as informações no banco de dados e gera um número de protocolo para a consulta das informações. **Pós-Condição:** Registro inserido no banco de dados, protocolo de manifestação gerado.	
Nome do fluxo alternativo (extensão)	**Descrição**
1. Manifestação do tipo "elogio".	1.1. O cidadão seleciona o tipo de manifestação "elogio". 1.2 O cidadão preenche os demais campos do formulário e clica em salvar. 1.3 O sistema valida os campos preenchidos, porém não gera o número de

	protocolo.
2. Manifestação anônima.	2.1. O cidadão marca a manifestação como anônima, preenche os campos obrigatórios tipo, assunto e mensagem e clica em salvar. 2.2. O sistema valida as informações dos campos preenchidos, porém não gera o número de protocolo.
3. Campos obrigatórios não preenchidos.	3.1. O cidadão não preenche campos obrigatórios e clica em salvar. 3.2. O sistema valida as informações e exibe um alerta informando que campos obrigatórios não foram preenchidos, sem salvar informações no banco de dados. 3.3. O sistema retorna a tela de inclusão, apresentado os campos que já haviam sido preenchidos e destacando os campos sem preenchimento.
4. Campos preenchidos com formato inválido.	4.1. O cidadão preenche os campos de forma incorreta e clica em salvar. 4.2. O sistema valida as informações e exibe um alerta na tela, sem salvar os dados. 4.3. O sistema retorna a tela de inclusão, com os campos que já haviam sido preenchidos e destacando os campos com preenchimento inválido.

Quadro 9 – Operação "inserir" do caso de uso registrar manifestação

Caso de uso:
Inserir (refere-se à operação de inclusão nos casos de uso "manter").
Descrição:
Ator inclui dados no sistema.
Atores:
Operador e administrador.
Pré-condição:
Usuário deve estar logado no sistema.
Sequência de Eventos:
1. Ator acessa a tela para cadastrar as informações solicitadas.
2. O sistema insere as informações no banco de dados.

Pós-Condição:
Registro inserido no banco de dados.

Nome do fluxo alternativo (extensão)	**Descrição**
1. Campos obrigatórios não preenchidos.	1.1. O ator não preenche campos obrigatórios e clica em salvar. 1.2. O sistema valida as informações e exibe um alerta informando que campos obrigatórios não foram preenchidos, sem salvar informações no banco de dados. 1.3. O sistema retorna a tela de inclusão, com os campos que já haviam sido preenchidos e destacando os campos sem preenchimento.
2. Campos preenchidos com formato inválido.	2.1. O ator preenche os campos de forma incorreta e clica em salvar. 2.2. O sistema valida as informações e exibe um alerta na tela ao ator sem salvar os dados. 2.3. O sistema retorna à tela de inclusão, com os campos que já haviam sido preenchidos e destacando os campos com formato inválido.

Quadro 10 – Operação "inserir" dos casos de uso "manter"

Caso de uso:
Alterar (refere-se à operação de alteração nos casos de uso "manter").
Descrição:
Ator altera dados no sistema.
Atores:
Operador e administrador.
Pré-condição:
Dados cadastrados no sistema.
Sequência de Eventos:
1. O ator acessa a tela para visualização de dados já cadastrados.
2. O sistema apresenta o registro selecionado para a alteração.
3. Ator altera os dados do registro e clica em salvar.
4. O sistema valida as informações e salva as informações no mesmo registro.

Pós-Condição: Registro alterado no banco de dados.	
Nome do fluxo alternativo (extensão)	**Descrição**
1. Campos obrigatórios não preenchidos.	1.1. O ator não preenche campos obrigatórios e clica em salvar. 1.2. O sistema valida as informações e exibe um alerta informando que campos obrigatórios não foram preenchidos, sem salvar informações no banco de dados. 1.3. O sistema retorna a tela de alteração, com os campos que já haviam sido preenchidos e destacando os campos sem preenchimento.
2. Campos preenchidos com formato inválido.	2.1. O ator preenche os campos de forma incorreta e clica em salvar. 2.2. O sistema valida as informações e exibe um alerta na tela ao ator sem salvar os dados. 2.3. O sistema retorna à tela de inclusão, com os campos que já haviam sido preenchidos e destacando os campos com formato inválido.

Quadro 11 – Operação "alterar" dos casos de uso "manter"

Caso de uso: Excluir (refere-se à operação de exclusão nos casos de uso "manter"). **Descrição:** Ator solicita a exclusão de dados no sistema. **Atores:** Operador e administrador. **Pré-condição:** Dados cadastrados no sistema. **Sequência de Eventos:** 1. O ator acessa a tela para visualização de dados já cadastrados. 2. O sistema apresenta o registro selecionado para a exclusão. 3. Ator clica em excluir registro. 4. O sistema exclui as informações do banco de dados e exibe as informações do status do procedimento. **Pós-Condição:** Registro excluído no banco de dados.	
Nome do fluxo alternativo (extensão)	**Descrição**
1. Exclusão de registro que possui vínculos no sistema.	1.1. Ator solicita a exclusão do registro que possui vínculos no sistema. 1.2 O sistema verifica que o registro possui vínculos, não exclui o registro e exibe uma mensagem de alerta para o ator.

Quadro 12 – Operação "excluir" dos casos de uso "manter"

Caso de uso: Consultar (refere-se à operação de consulta nos casos de uso "manter"). **Descrição:** Ator solicita a consulta de dados cadastrados no sistema. **Atores:** Operador, administrador, responsável. **Pré-condição:** Dados cadastrados no sistema. **Sequência de Eventos:** 1. Ator acessa a tela para visualização de dados já cadastrados. 2. Ator indica que tipo de dados pretende consultar através de filtros. 3. O sistema exibe os dados da consulta ao usuário. **Pós-Condição:** Dados são exibidos aos usuários.

Quadro 13 – Operação "consultar" dos casos de uso "manter"

Caso de uso:
Consultar (refere-se à operação de consulta no caso de uso "consultar manifestação").
Descrição:
Ator solicita a consulta de uma manifestação no sistema.
Atores:
Cidadão, operador e administrador.
Pré-condição:
Manifestação cadastrada no sistema.
Sequência de Eventos:
1. O ator acessa a tela para consulta de manifestação através do protocolo.
2. O ator preenche o campo do protocolo e clica em pesquisar.
3. O sistema exibe na tela o *status* da manifestação consultada.

Pós-Condição:
Dados da consulta são apresentados na tela para o ator.

Nome do fluxo alternativo (extensão)	**Descrição**
1. Campo obrigatórios não preenchidos.	1.1. O ator não preenche o campo do protocolo e clica em pesquisar. 1.2. O sistema valida as informações, não apresenta o *status* da manifestação e exibe um alerta informando que o campo obrigatório não foi preenchido. 1.3. O sistema retorna à tela de pesquisa.
2. Campo preenchidos com formato inválido.	2.1. O ator preenche o campo do protocolo de forma incorreta e clica em pesquisar. 2.2. O sistema valida as informações e exibe um alerta na tela e não exibe o *status* da manifestação. 2.3. O sistema retorna à tela de pesquisa.

Quadro 14 – Operação "consultar" do caso de uso "consultar manifestação"

Caso de uso:
Consultar (refere-se à operação de consulta no caso de uso "gerenciar manifestação").
Descrição:
Ator solicita a consulta de uma manifestação no sistema.
Atores:
Responsável, operador e administrador.
Pré-condição:
Manifestação cadastrada no sistema.
Sequência de Eventos:
1. Ator acessa a tela para visualização de dados já cadastrados.
2. Ator indica que tipo de dados pretende consultar através de filtros.
3. O sistema exibe os dados da consulta ao usuário.

Pós-Condição:
Dados da consulta são apresentados na tela para o ator.

Nome do fluxo alternativo (extensão)	**Descrição**
1. Campo obrigatórios não preenchidos.	1.1. O ator não preenche campos obrigatórios para a consulta e clica em pesquisar. 1.2. O sistema valida as informações, não apresenta o relatório e exibe um alerta informando que o campo obrigatório não foi preenchido. 1.3. O sistema retorna à tela de pesquisa.
2. Campo preenchidos com formato invalido.	2.1. O ator preenche o campo de pesquisa de forma incorreta e clica em pesquisar. 2.2. O sistema valida as informações, não apresenta o relatório e exibe um alerta informando que o campo foi preenchido de forma inválida. 2.3. O sistema retorna à tela de pesquisa.
3. Consulta de dados que o ator não possui privilégios.	3.1. Ator solicita a consulta dos dados que não possui privilégio. 3.2 O sistema não apresenta os dados e exibe uma mensagem de que o ator não possui privilégios para acessá-la.

Quadro 15 – Operação "consultar" do caso de uso "gerenciar manifestação"

Caso de uso: Alterar (refere-se à operação de alteração no caso de uso "gerenciar manifestação"). **Descrição:** Ator altera *status* de uma manifestação no sistema. **Atores:** Responsável, operador e administrador. **Pré-condição:** Manifestação cadastrada no sistema. **Sequência de Eventos:** 1. O ator acessa a tela para visualização de dados já cadastrados. 2. O sistema apresenta o registro selecionado para a alteração. 3. Ator altera os dados do registro e clica em salvar. 4. O sistema valida as informações e salva as informações no mesmo registro. **Pós-Condição:** Registro alterado no banco de dados.	
Nome do fluxo alternativo (extensão)	**Descrição**
1. Campos obrigatórios não preenchidos.	1.1. O ator não preenche campo *status* e clica em salvar. 1.2. O sistema valida as informações e exibe um alerta informando que campos *status* não foi preenchido, sem salvar informações no banco de dados. 1.3. O sistema retorna à tela de alteração.

Quadro 16 – Operação "alterar" do caso de uso "gerenciar manifestação"

Caso de uso: Alterar (refere-se à operação de alteração no caso de uso "receber manifestação"). **Descrição:** Ator altera a secretaria de uma manifestação no sistema. **Atores:** Operador e administrador. **Pré-condição:** Manifestação cadastrada no sistema. **Sequência de Eventos:** 1. O ator acessa a tela para visualização de dados já cadastrados. 2. O sistema apresenta o registro selecionado para a alteração. 3. Ator altera os dados do registro e clica em salvar. 4. O sistema valida as informações e salva as informações no mesmo registro. **Pós-Condição:** Registro alterado no banco de dados.	
Nome do fluxo alternativo (extensão)	**Descrição**
1. Campos obrigatórios não preenchidos.	1.1. O ator não preenche campo secretaria e clica em salvar. 1.2. O sistema valida as informações e exibe um alerta informando que o campo não foi preenchido, sem salvar informações no banco de dados. 1.3. O sistema retorna à tela de alteração.

Quadro 17 – Operação "alterar" do caso de uso "receber manifestação"

A Figura 4 apresenta o diagrama de atividades do caso de uso "registrar manifestação". O sistema recebe os dados da manifestação e valida as informações para que os campos não estejam em branco ou com dados inválidos. Se algum dado estiver inválido o sistema exibe uma mensagem de erro e retorna à tela de registro, destacando os campos que não foram preenchidos corretamente. Após os dados serem validados com sucesso o sistema salva as informações no banco de dados.

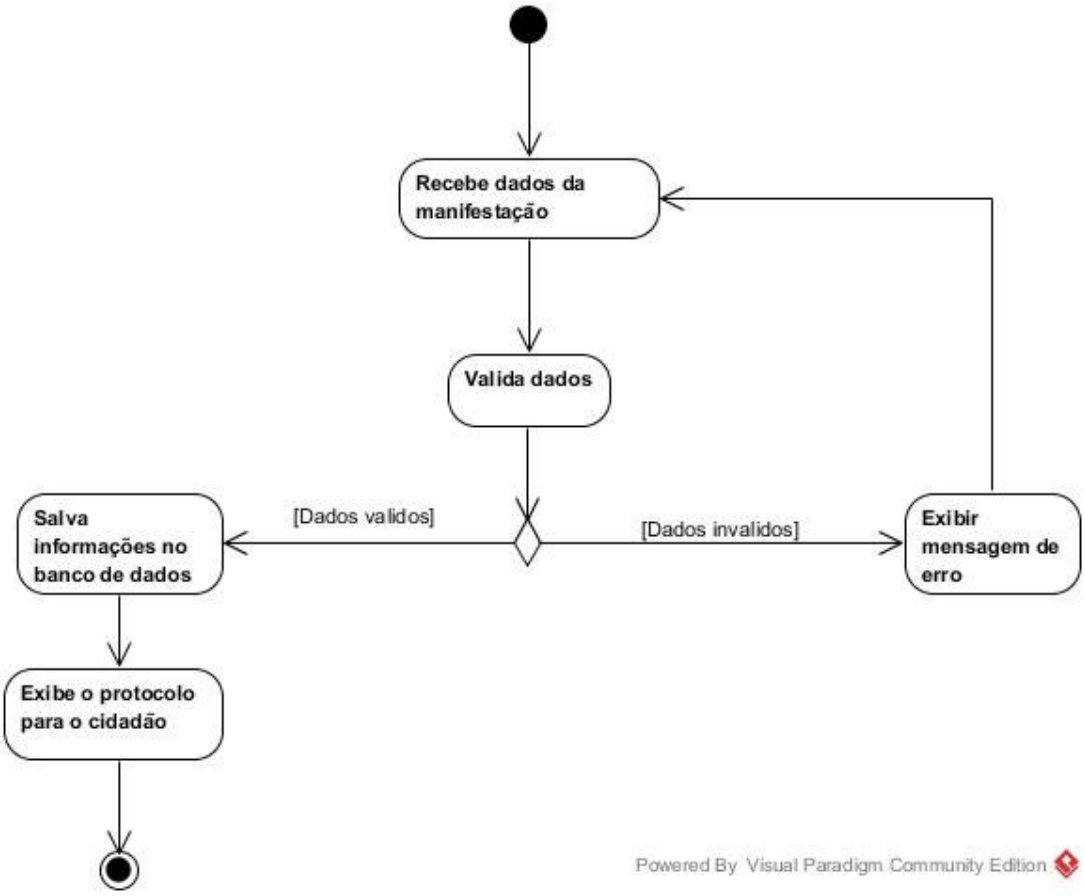

Figura 4 – Diagrama de atividades "registrar manifestação"

A Figura 5 apresenta o diagrama de atividades do caso de uso "receber manifestação". O operador recebe uma manifestação e confere se o assunto da manifestação será encaminhado para a secretaria que o cidadão selecionou ao registrar a sua manifestação. Caso o usuário tenha selecionado uma secretaria que não corresponde à manifestação registrada, o operador irá alterá-la, conferir as informações e enviar a manifestação para a secretaria correspondente.

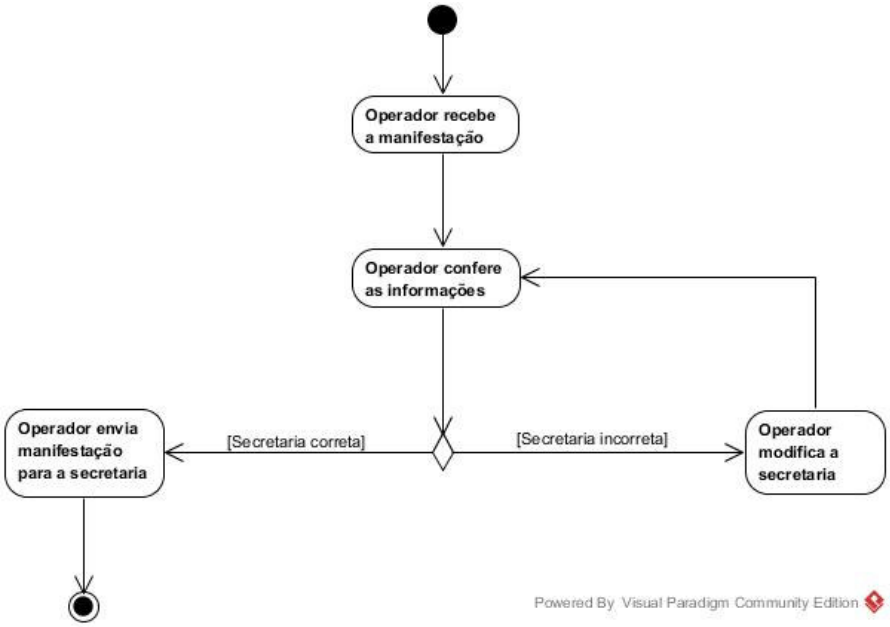

Figura 5 – Diagrama de atividade receber manifestação

A Figura 6 apresenta o diagrama de atividades do caso de uso "gerenciar manifestação". O responsável acessa as informações de uma manifestação e se ela ainda não foi respondida, ele atualiza o *status* da manifestação. Assim que uma manifestação for atendida o seu *status* é atualizado para, por exemplo, finalizado e escreve a resposta da manifestação em um campo de texto.

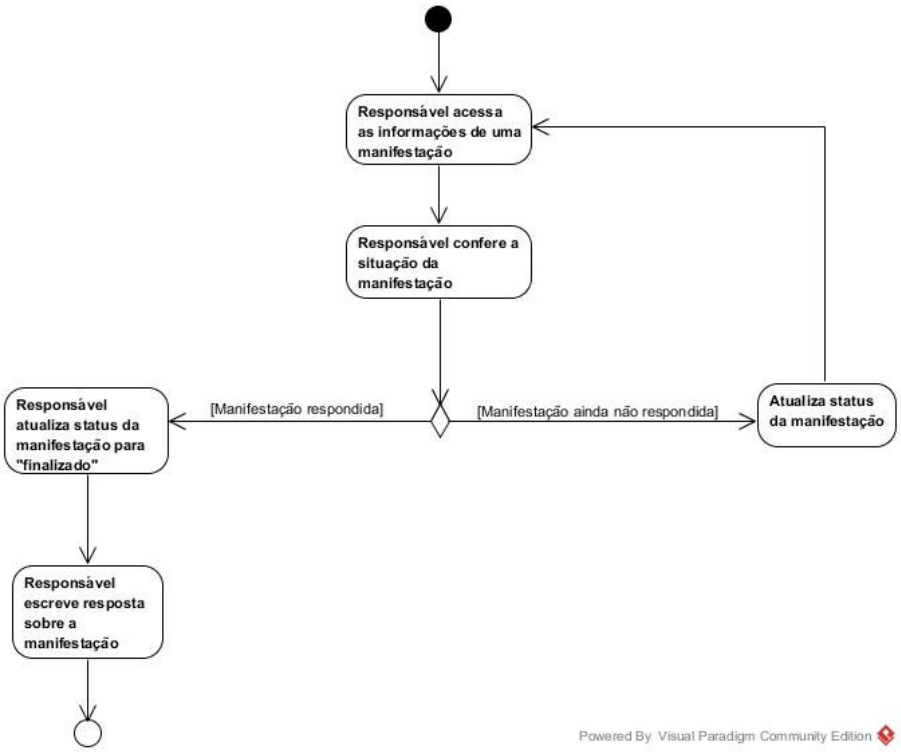

Figura 6 – Diagrama de atividade gerenciar manifestação

A Figura 7 representa o diagrama de entidade e relacionamento que representa o banco de dados da aplicação proposta.

A entidade denominada “Informação pessoais” armazena dados pessoais, de contato e da manifestação realizada pelo cidadão. A entidade denominada “Manifestação” contém os dados do registro da sugestão e/ou reclamação realizada pelo cidadão como, data, comentário, meio de contato e tipo de manifestação. Essa entidade se relaciona com as entidades denominadas “Informações Pessoais” e “Protocolo”. A tabela “Protocolo” refere-se ao protocolo gerado pelo sistema para que o cidadão possa acompanhar os encaminhamentos de sua manifestação. Essa entidade se relaciona com as denominadas “Usuário” e “Secretaria”. A entidade “Secretaria” armazena dados de nome da secretaria, responsável e secretário. A entidade “Usuário” é responsável por armazenar informações pessoais dos usuários para atribuir as permissões de acesso ao sistema. A entidade “Secretario” armazena o nome do secretário e as datas de entrada e saída, armazenando o período de tempo em que o secretário ocupou o cargo.

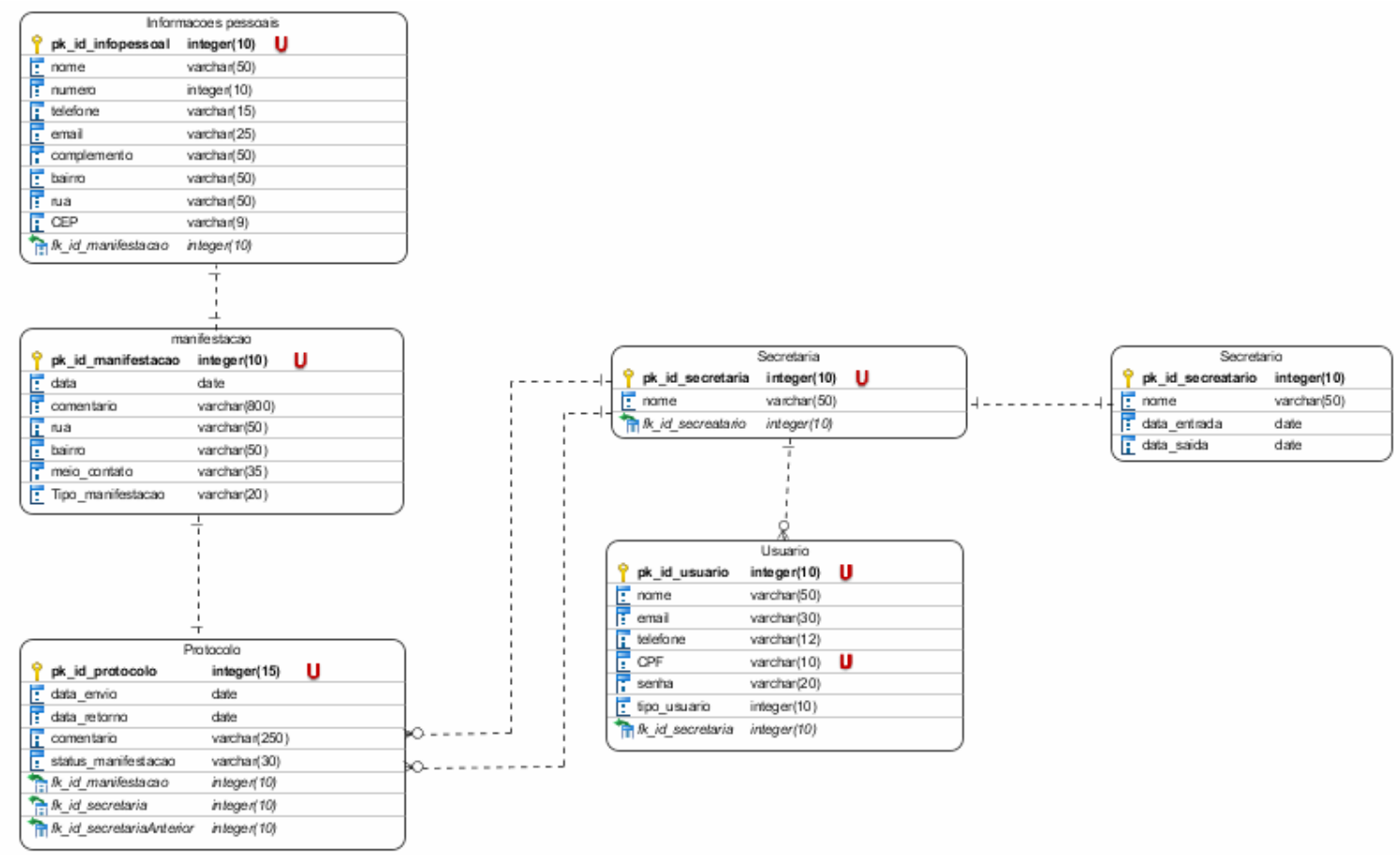

Figura 7 - Diagrama de entidade e relacionamento

4.3 APRESENTAÇÃO DO SISTEMA

O sistema é composto por dois leiautes padrão, um para a área comum e outro para a área administrativa. A área comum é destinada a usuários não autenticados e possui na guia superior os *links* para criar e consultar uma manifestação. Também possui um *link* para acesso à área restrita. A Figura 8 ilustra a estrutura da página inicial do sistema.

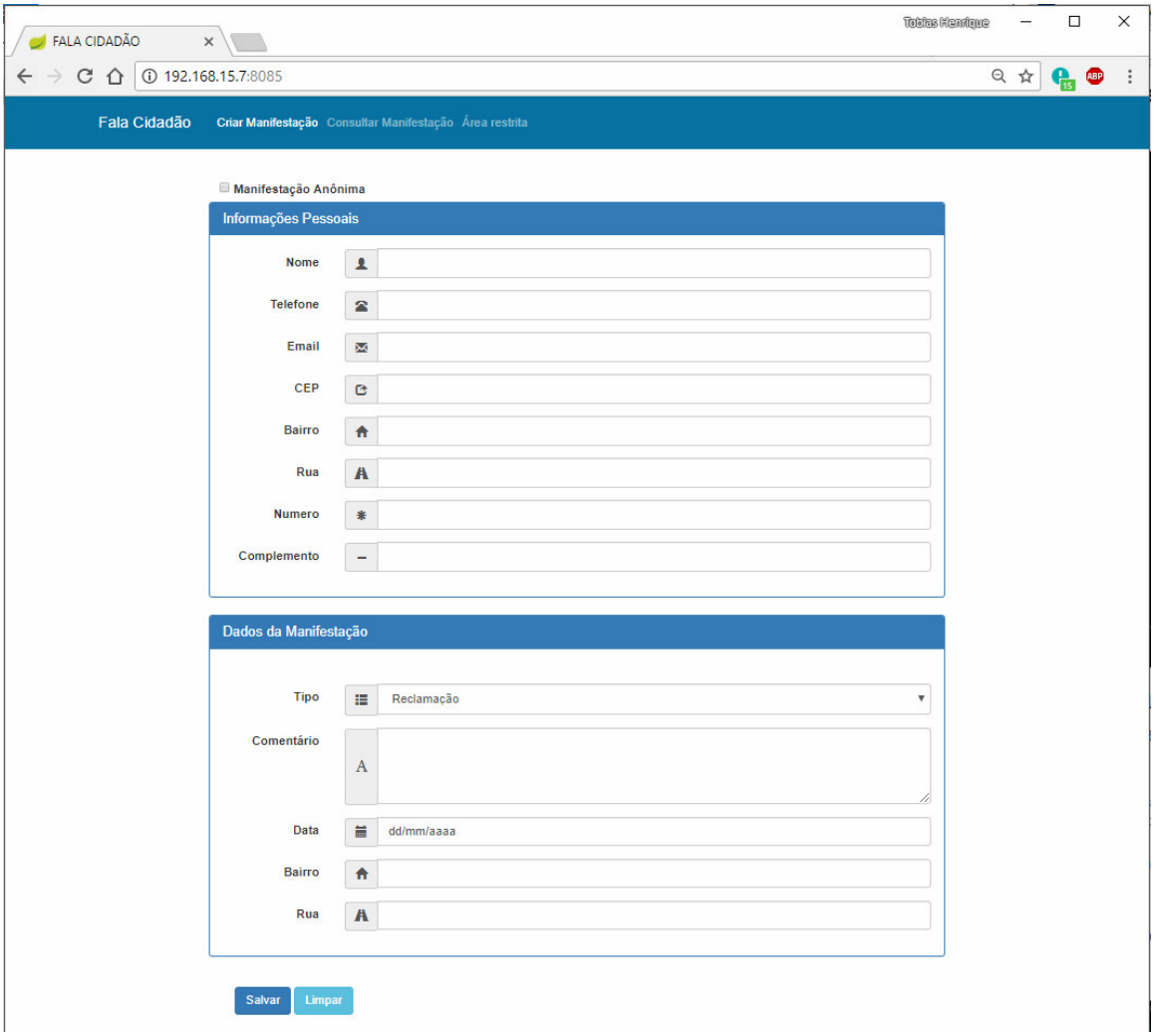

Figura 8 – Tela Inicial

Ao cadastrar uma manifestação o usuário poderá optar por criar uma manifestação anônima. Para isso, é necessário marcar a opção “manifestação anônima” e os campos da guia “informações pessoais” ficarão ocultos e apenas a guia “dados da manifestação” ficará habilitada para preenchimento. A Figura 9 mostra como a tela de cadastro ficará caso o *checkbox* “manifestação oculta” for marcado.

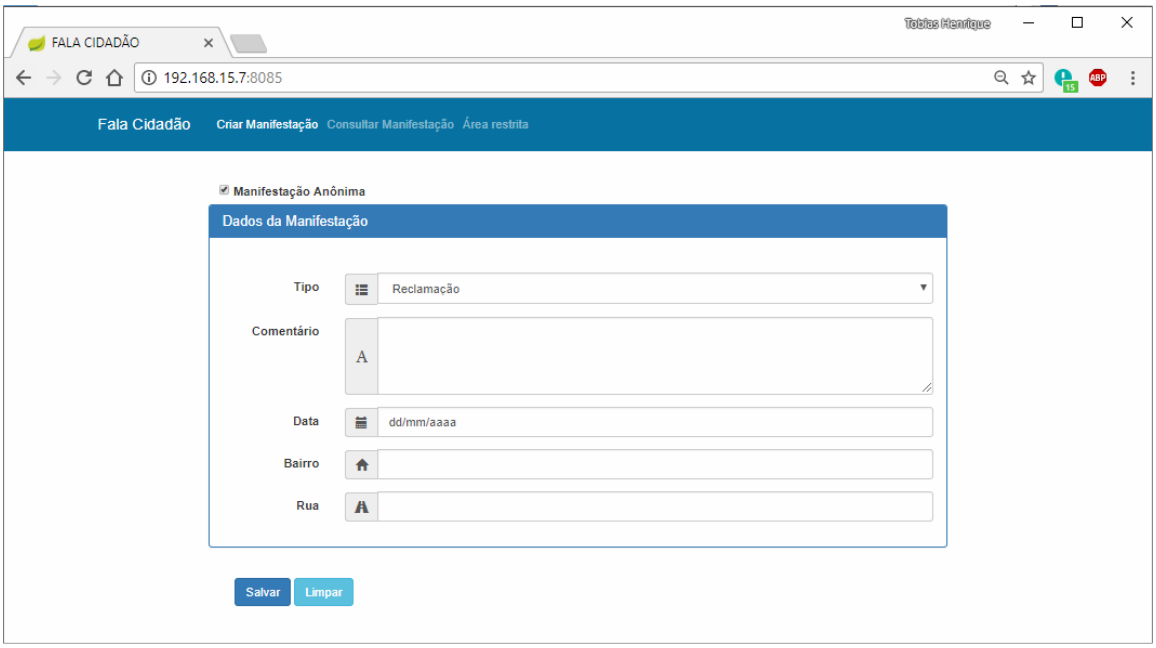

Figura 9 - Tela inicial manifestação anônima

Ao terminar de incluir os dados da manifestação e clicar no botão “Salvar”, o usuário é direcionado para a tela que apresenta o número de protocolo da manifestação gerada. Essa tela é exibida é representada na Figura 10.

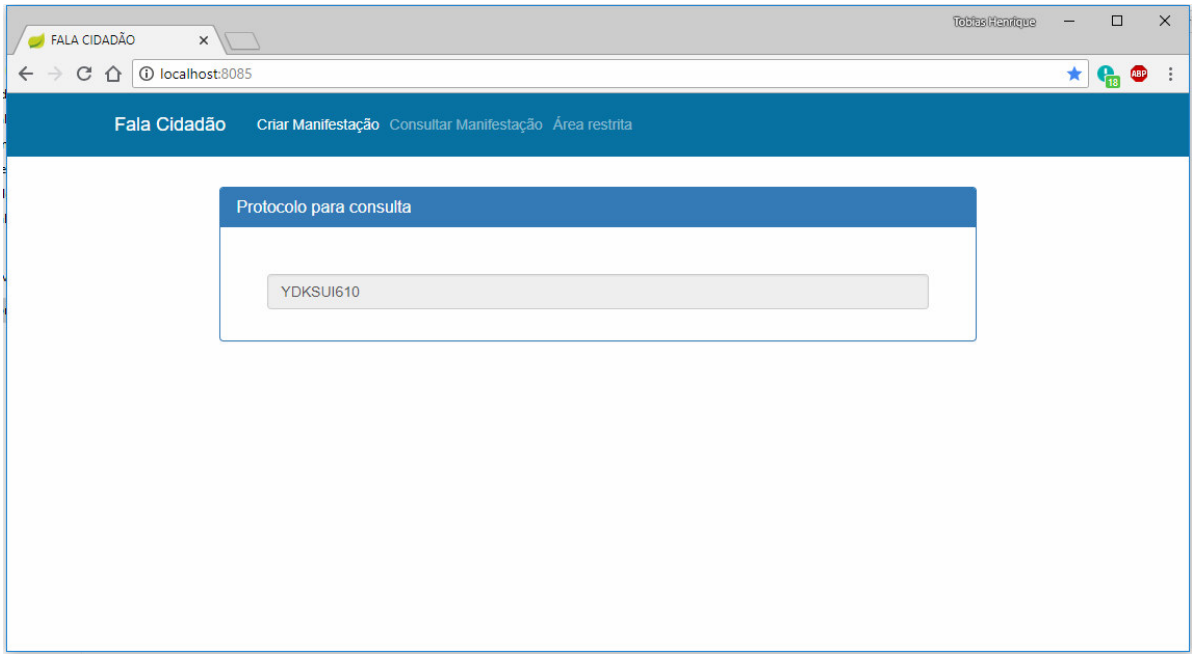

Figura 10 - Exibição de protocolo para pesquisa

A Figura 11 apresenta a tela em que o usuário é direcionado após consultar o protocolo da sua manifestação. Essa tela contém informações referentes ao *status* da manifestação, último dia que ela foi movimentada no sistema e a resposta obtida.

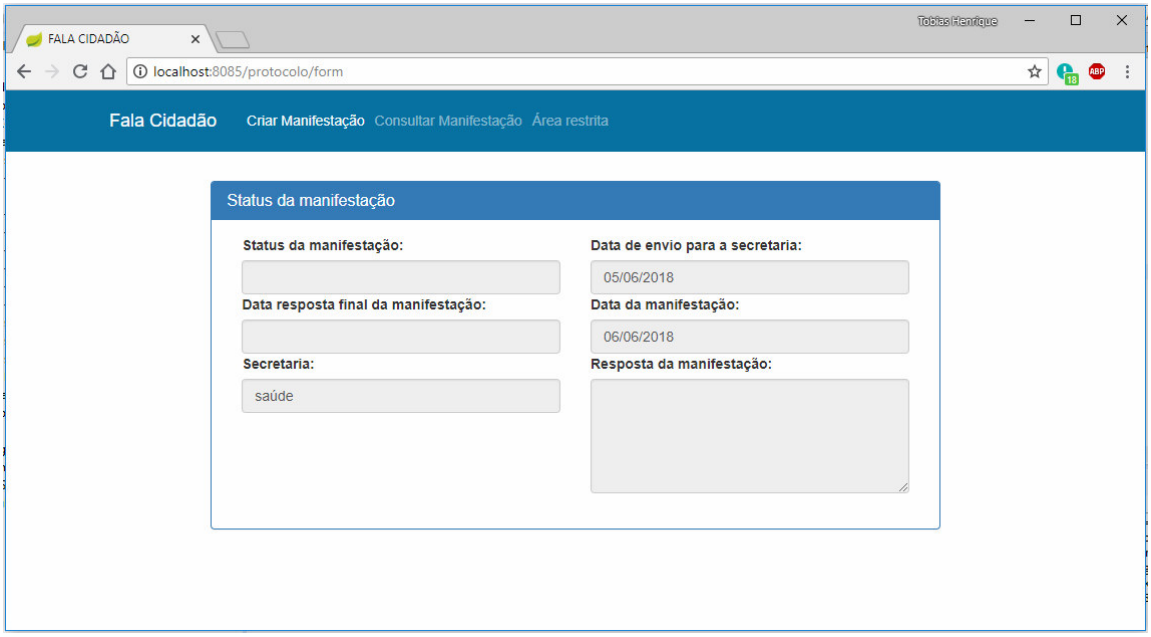

Figura 11 - Status da manifestação.

A Figura 12 apresenta exemplo da tela de *login* para acesso à área restrita do sistema, na qual são gerenciadas as manifestações registradas na área comum.

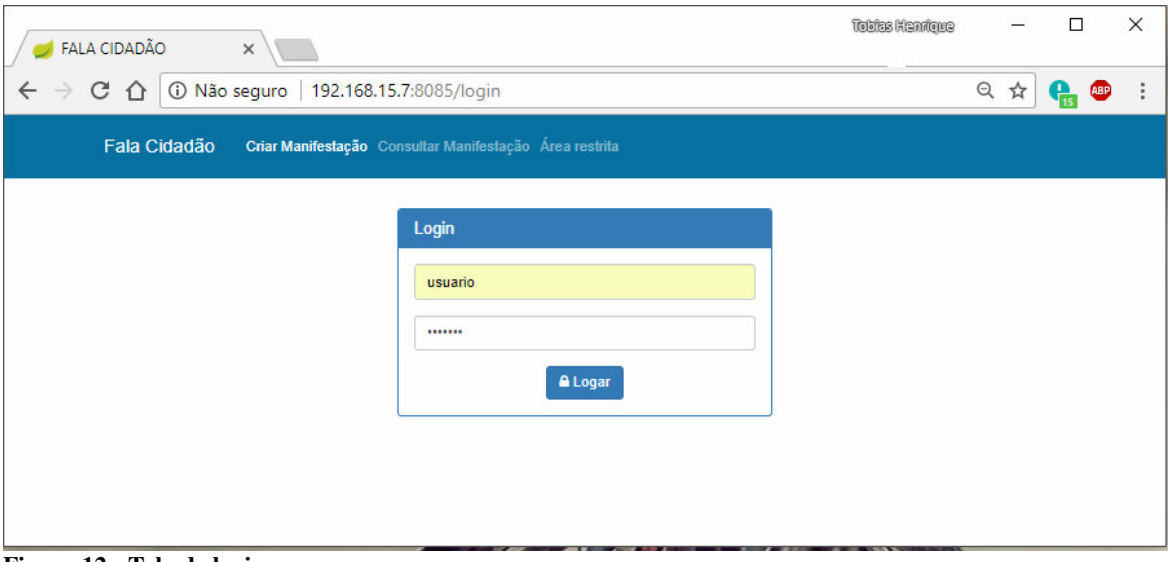

Figura 12 - Tela de login

Ao se autenticar no sistema, o usuário é direcionado para a página inicial. O leiaute da área administrativa é composto por três partes principais: o menu lateral que apresenta todas as opções do sistema, a barra superior que apresenta as informações do usuário

autenticado na sessão e o painel principal com as informações do sistema. O usuário é direcionado para a guia “Gerenciar manifestação” que exibe as manifestações referentes à secretaria que o usuário está vinculado são exibidas. A Figura 13 apresenta a tela para gerenciar a manifestação.

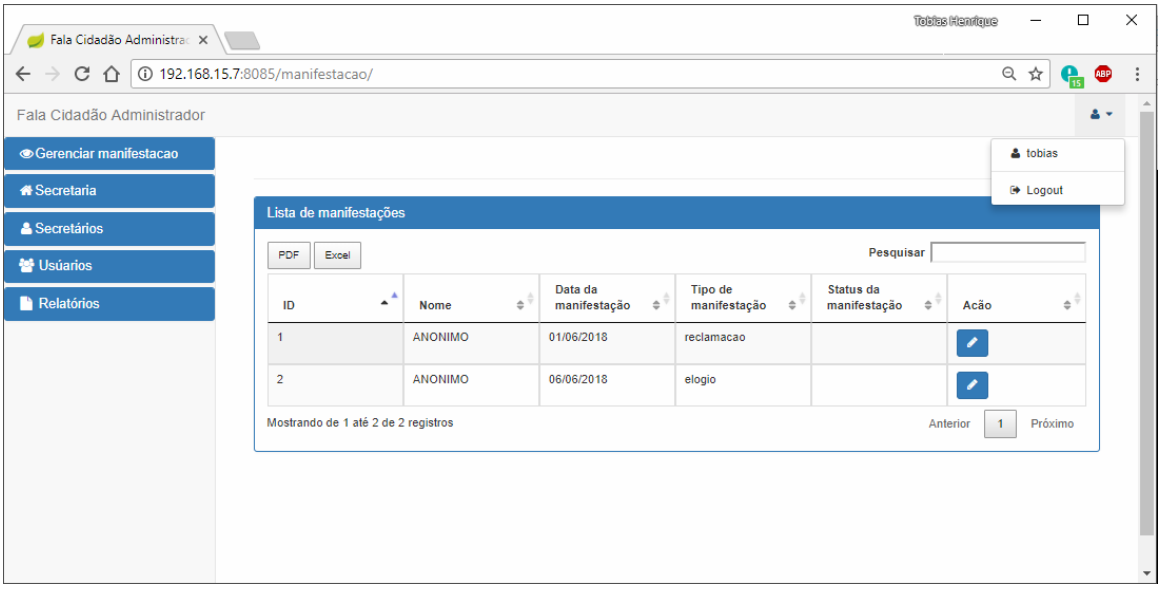

Figura 13 - Gerenciar Manifestação

O leiaute da tela se adapta de acordo com a permissão de cada usuário, a Figura 13 mostra o usuário administrador do sistema, que tem acesso a todas as informações do sistema. As Figuras 14 e 15 mostram, respectivamente, como o sistema é exibido caso um operador e o responsável realizem, respectivamente, a autenticação no sistema.

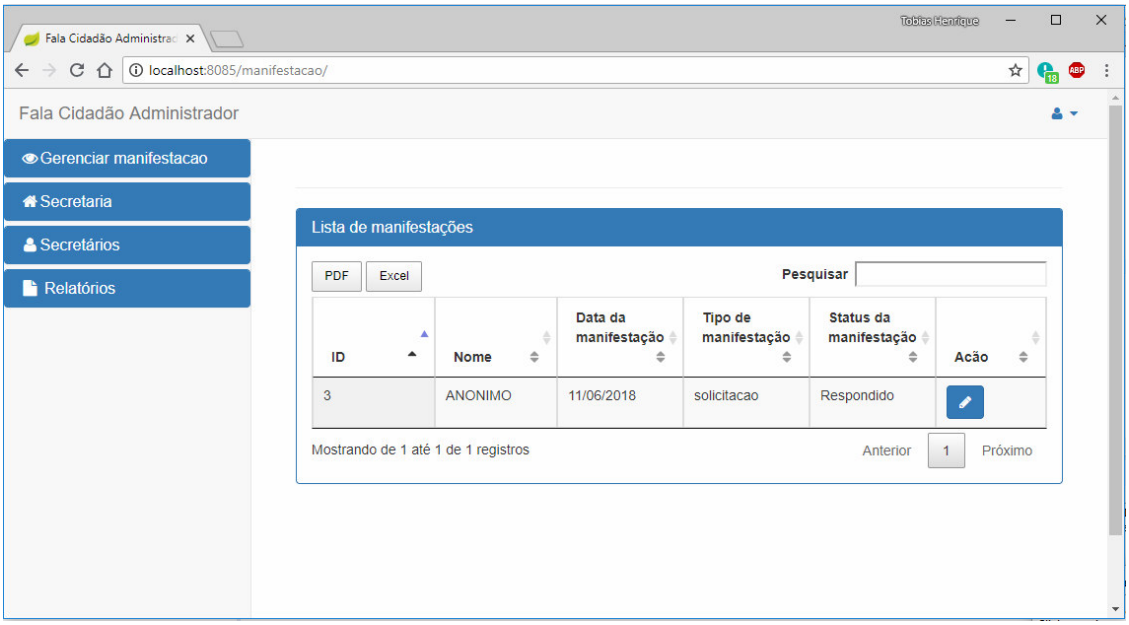

Figura 14 - Leiaute Operador

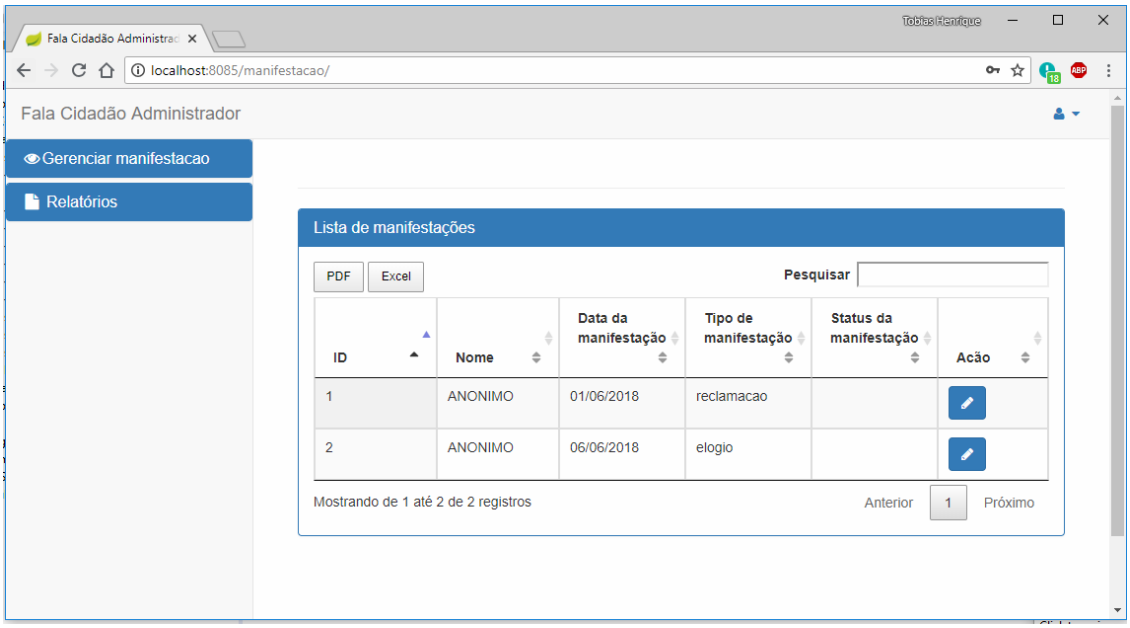

Figura 15 - Leiaute Usuário

Ao clicar no botão de ação da página gerenciar manifestação, o usuário é direcionado para a página na qual as informações do cidadão que realizou a manifestação é mostrada. Na guia “Dados da manifestação” são exibidas as informações que foram preenchidas pelo cidadão na tela inicial. Na última guia, de protocolo de resposta, são exibidos os campos para que o usuário autenticado poderá encaminhar a manifestação para outra secretaria, atualizar o

status e a data de resposta da manifestação. A Figura 16 apresenta a tela de atualização do *status* da manifestação.

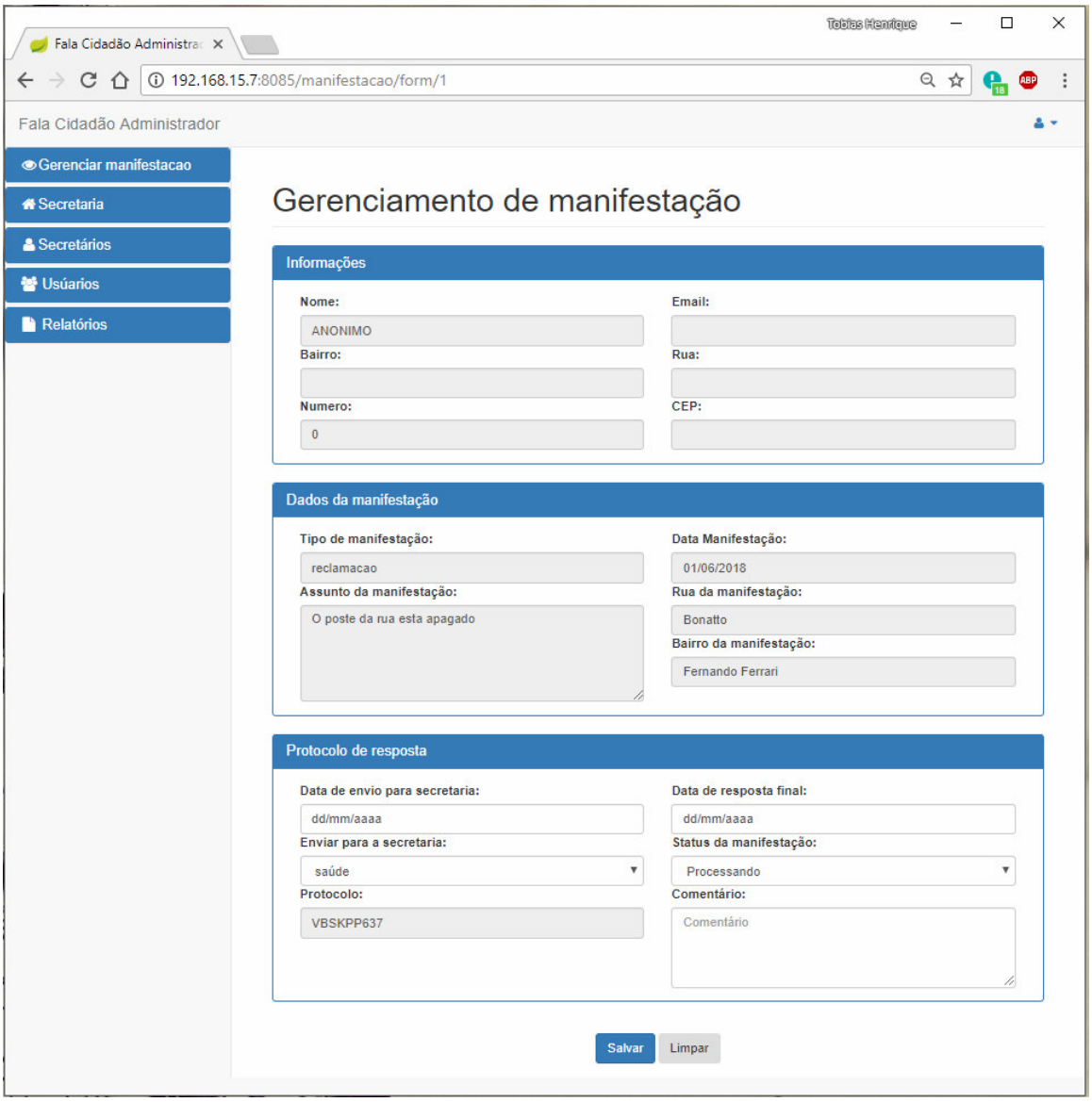

Figura 16 - Atualizar Status da manifestação

A Figura 17 mostra um exemplo de como funcionam os cadastros da área administrativa. O usuário cadastra as informações no formulário e clica no botão “Salvar”.

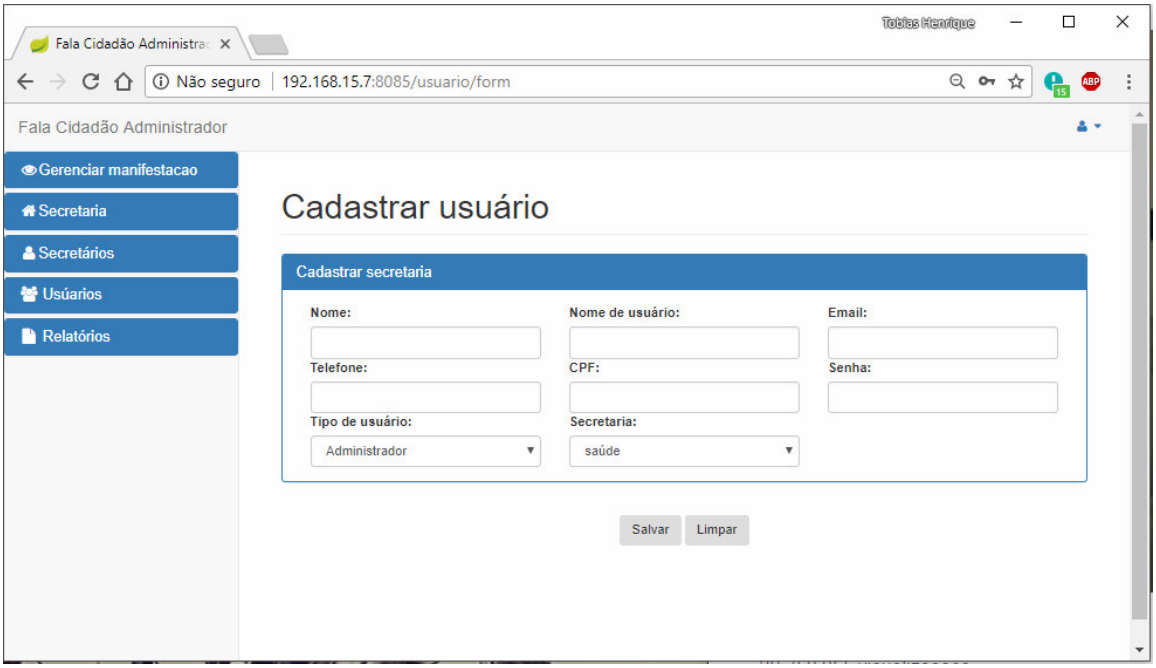

Figura 17 - Cadastro de Usuário

Se as informações foram cadastradas com sucesso uma mensagem de alerta é mostrada na tela, conforme apresentado na Figura 18.

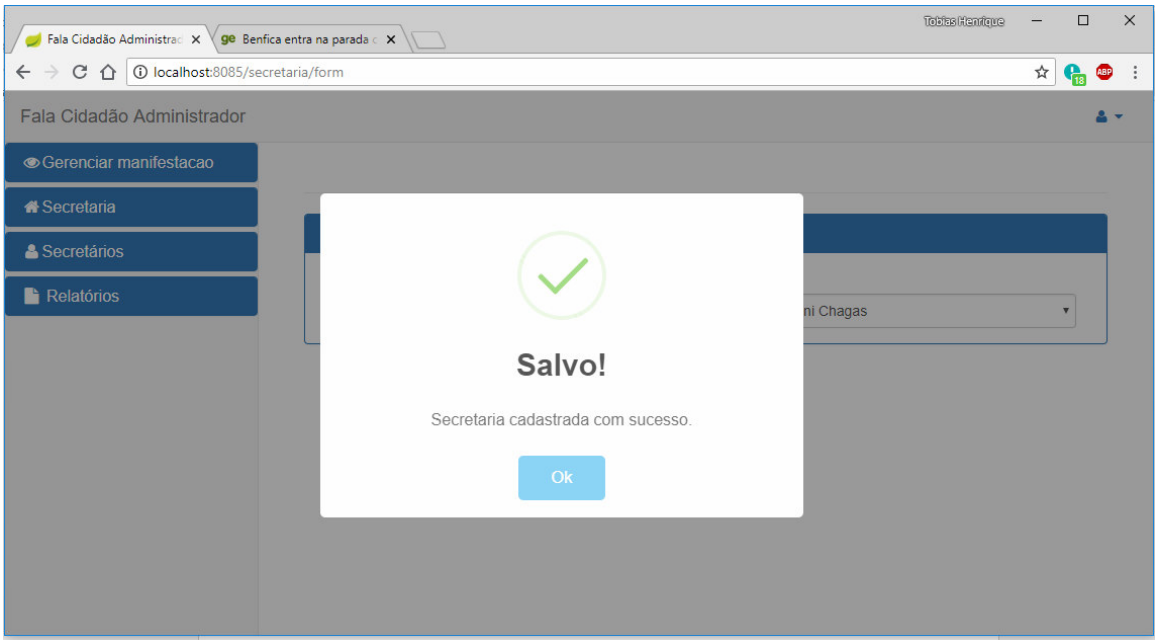

Figura 18 - Alerta de cadastro

As informações cadastradas nos formulários são exibidas em listas de acordo com as informações de cada cadastro. Um exemplo de lista é apresentado na Figura 19.

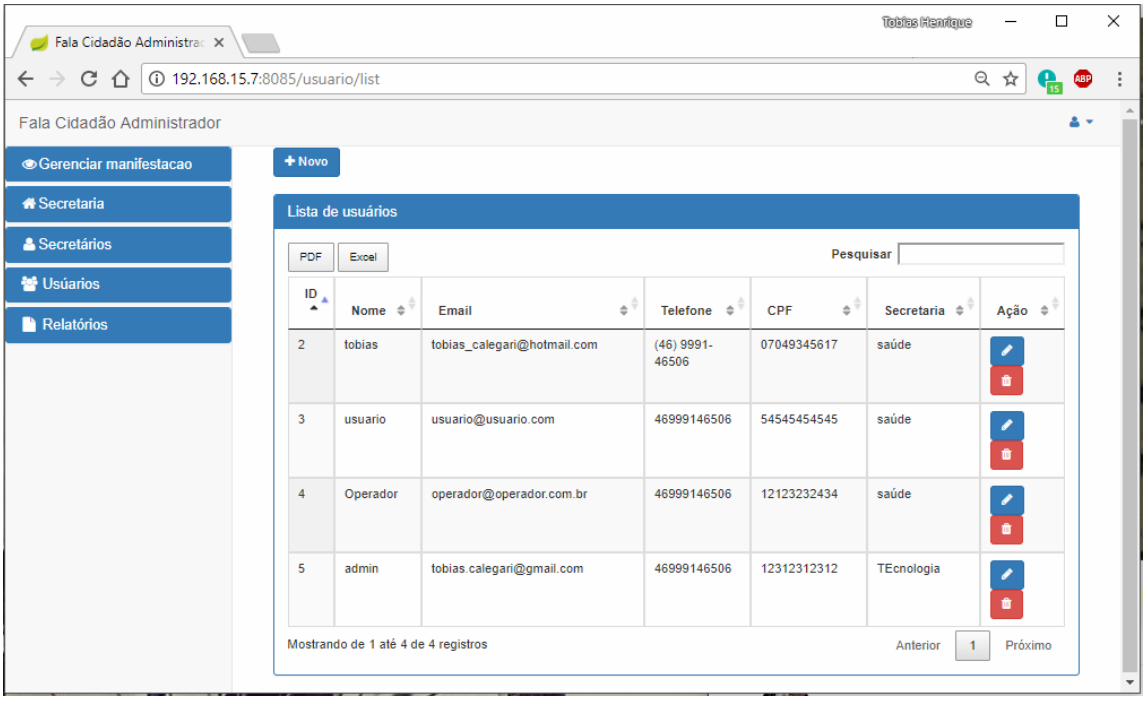

Figura 19 - Exemplo de lista

4.4 IMPLEMENTAÇÃO DO SISTEMA

A fim de otimizar o desenvolvimento das interfaces e evitar repetição de código, foram utilizados dois leiautes padrão. O primeiro compreende a parte do sistema que não necessita de autenticação. Esse leiaute é chamado de “template.tag” (Listagem 1) e contém os menus de acesso para as páginas de *login*, pesquisa e criar uma manifestação.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>
<%@ attribute name="scriptsEspecificos" fragment="true" required="false"%>

<!DOCTYPE html>
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  <title>FALA CIDADÃO</title>
  <link rel="stylesheet" type="text/css"
href="//netdna.bootstrapcdn.com/bootstrap/3.0.0/css/bootstrap-glyphicons.css">
  <link rel="stylesheet" type="text/css" href="<c:url value=" /assets/css/zabuto_calendar.css " />"/>
  <link type="text/css" href="<c:url value=" /assets/css/sweetalert.css " />" rel="stylesheet" />
  <!-- Custom styles for this template -->
  <link type="text/css" href="<c:url value=" /assets/css/style.css " />" rel="stylesheet" />
  <link type="text/css" href="<c:url value=" /assets/css/style-responsive.css " />" rel="stylesheet"
/>
```

```
  <link type="text/css" href="<c:url value=" /assets/css/estiloProprio.css " />" rel="stylesheet" />
  <link type="text/css" href="<c:url value=" /assets/css/bootstrap.css " />" rel="stylesheet" />
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
integrity="sha384-BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u"
    crossorigin="anonymous">
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-
awesome.min.css">
  <style>
    body {
      padding-top: 54px;
    }
    @media (min-width: 992px) {
      body {
        padding-top: 56px;
      }
    }
  </style>
</head>
<body>
  <!-- Navigation -->
  <nav class="navbar navbar-expand-lg navbar-dark bg-dark navbar-fixed-top fundo">
    <div class="container">
      <a class="navbar-brand" href="/">Fala Cidadão</a>
      <div class="collapse navbar-collapse" id="navbarResponsive">
        <ul class="navbar-nav ml-auto">
          <li class="nav-item active">
            <a class="nav-link" href="/">Criar Manifestação
            </a>
          </li>
          <li class="nav-item">
            <a class="nav-link" href="/protocolo/form">Consultar Manifestação</a>
          </li>
          <li class="nav-item">
            <a class="nav-link" href="/login">Área restrita</a>
          </li>
        </ul>
      </div>
    </div>
  </nav>
  <!--main content start-->
  <jsp:doBody/>
  <script src="<c:url value=" /assets/js/sweetalert.min.js " />"></script>
  <!--Import jQuery before export.js-->
  <script type="text/javascript" src="https://code.jquery.com/jquery-2.1.1.min.js"></script>
  <!--Data Table-->
  <script type="text/javascript" src="
https://cdn.datatables.net/1.10.13/js/jquery.dataTables.min.js"></script>
  <script type="text/javascript" src="
https://cdn.datatables.net/buttons/1.2.4/js/dataTables.buttons.min.js"></script>
```

```
  <!--Export table buttons-->
  <script type="text/javascript"
src="https://cdnjs.cloudflare.com/ajax/libs/jszip/2.5.0/jszip.min.js"></script>
  <script type="text/javascript"
src="https://cdn.rawgit.com/bpampuch/pdfmake/0.1.24/build/pdfmake.min.js"></script>
  <script type="text/javascript"
src="https://cdn.rawgit.com/bpampuch/pdfmake/0.1.24/build/vfs_fonts.js"></script>
  <script type="text/javascript"
src="https://cdn.datatables.net/buttons/1.2.4/js/buttons.html5.min.js"></script>
  <script type="text/javascript"
src="https://cdn.datatables.net/buttons/1.2.1/js/buttons.print.min.js"></script>
  <jsp:invoke fragment="scriptsEspecificos"></jsp:invoke>
</body>
</html>
```

Listagem 1 - Leiaute padrão área comum

A Listagem 1 foi desenvolvida utilizando os padrões de páginas *Java Server Pages* (JSP) e o *framework* Bootstrap para desenvolvimento de páginas *web*. O menu que utiliza a classe *navbar* cria o menu que compõe todos os *links* para acesso ao sistema. A classe *collapse* esconde os itens do menu caso o sistema seja acessado por um dispositivo com tela menor. A tag *jsp:doBody* indica o local que o conteúdo da página será carregado quando a página for montada no servidor.

A Listagem 2 apresenta o leiaute da área administrativa, com o nome de "*templateAdmin.tag*". Essa página contém todos os menus da área administrativa e também utiliza padrões JSP e o framework *Bootstrap*.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib uri="http://www.springframework.org/tags" prefix="t"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>
<%@ taglib prefix="sec" uri="http://www.springframework.org/security/tags" %>
<%@ attribute name="scriptsEspecificos" fragment="true" required="false"%>
<!DOCTYPE html>

<head>
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta name="description" content="">
    <meta name="author" content="">
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

    <title>Fala Cidadão Administrador</title>

    <link type="text/css" href="<c:url value=" /assets/css/bootstrap.min.css "/>" rel="stylesheet" />
    <link type="text/css" href="<c:url value=" /assets/css/metisMenu.min.css "/>" rel="stylesheet" />
  <link type="text/css" href="<c:url value=" /assets/css/sb-admin-2.min.css "/>" rel="stylesheet" />
  <link type="text/css" href="<c:url value=" /assets/css/font-awesome.min.css "/>" rel="stylesheet" />
  <link type="text/css" href="<c:url value=" /assets/css/dataTables.tableTools.min.css "/>"
rel="stylesheet" />
  <link type="text/css" href="<c:url value=" /assets/css/sweetalert.css " />" rel="stylesheet" />
  <link rel="stylesheet" type="text/css"
href="https://cdn.datatables.net/1.10.13/css/jquery.dataTables.min.css">
  <link type="text/css" rel="stylesheet"
href="https://cdn.datatables.net/buttons/1.2.4/css/buttons.dataTables.min.css">
  <link type="text/css" href="//maxcdn.bootstrapcdn.com/font-awesome/4.2.0/css/font-awesome.min.css"
```

```
rel="stylesheet">
</head>

<body>
    <div id="wrapper">
      <nav class="navbar navbar-default navbar-static-top" role="navigation" style="margin-bottom: 0">
            <div class="navbar-header">
            <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-
collapse">
                    <span class="sr-only">Toggle navigation</span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>
                <a class="navbar-brand" href="/manifestacao/">Fala Cidadão Administrador</a>
            </div>
            <ul class="nav navbar-top-links navbar-right">
                <li class="dropdown">
                    <a class="dropdown-toggle" data-toggle="dropdown" href="#">
                        <i class="fa fa-user fa-fw"></i>
                        <i class="fa fa-caret-down"></i>
                    </a>
                    <ul class="dropdown-menu dropdown-user">
                        <li>
                            <a href="#">
                                <i class="fa fa-user fa-fw"></i>
                                <sec:authentication property="principal.nome" />
                            </a>
                            <li class="divider"></li>
                            <li>
                                <a href="/logout"><i class="fa fa-sign-out fa-fw"></i>Logout</a>
                            </li>
                    </ul>
                    </li>
            </ul>
                <div class="navbar-default sidebar" role="navigation">
                <div class="sidebar-nav navbar-collapse">
                    <div class="panel-group" id="accordion" role="tablist" aria-
multiselectable="true">
                        <div class="panel panel-primary">
                            <div class="panel-heading" role="tab" id="headingZero">
                                <h4 class="panel-title">
                                    <a role="button" data-toggle="collapse" data-parent="#accordion"
href="#collapseZero" aria-expanded="true" aria-controls="collapseZero">
                                        <a href="/manifestacao/">
                                            <i class="fa fa-eye fa-fw"></i>Gerenciar manifestacao</a>
                                    </a>
                                </h4>
                            </div>
                            <div id="collapseZero" class="panel-collapse collapse in" role="tabpanel"
aria-labelledby="headingZero">
                            </div>
                        </div>
                        <sec:authorize access="hasAnyRole('ROLE_ADMIN', 'ROLE_OPERADOR')">
                            <div class="panel panel-primary">
                                <div class="panel-heading" role="tab" id="heading1">
                                    <h4 class="panel-title">
                                        <a class="collapsed" role="button" data-toggle="collapse"
data-parent="#accordion" href="#collapse1" aria-controls="collapse1">
                                            <i class="fa fa-home fa-fw"></i>Secretaria
                                    </h4>
                                </div>
                                <div id="collapse1" class="panel-collapse collapse" role="tabpanel"
aria-labelledby="heading1">
                                    <div class="panel-body">
                                        <ul class="nav nav-second-level">
                                            <li>
                                                <a href="/secretaria/form">Cadastrar Secretaria</a>
                                            </li>
                                            <li>
                                                <a href="/secretaria/list">Listar Secretaria</a>
                                            </li>
                                        </ul>
                                    </div>
```

```
                            </div>
                        </div>
                        <div class="panel panel-primary">
                            <div class="panel-heading" role="tab" id="heading2">
                                <h4 class="panel-title">
                                    <a class="collapsed" role="button" data-toggle="collapse"
data-parent="#accordion" href="#collapse2" aria-controls="collapse2">
                                        <i class="fa fa-user fa-fw"></i>Secretários
                                </h4>
                            </div>
                            <div id="collapse2" class="panel-collapse collapse" role="tabpanel"
aria-labelledby="heading2">
                                <div class="panel-body">
                                    <ul class="nav nav-second-level">
                                        <li>
                                            <a href="/secretario/form">Cadastrar Secretário</a>
                                        </li>
                                        <li>
                                            <a href="/secretario/list">Listar Secretários</a>
                                        </li>
                                    </ul>
                                </div>
                            </div>
                        </div>
                    </sec:authorize>
                    <sec:authorize access="hasRole('ROLE_ADMIN')">
                        <div class="panel panel-primary">
                            <div class="panel-heading" role="tab" id="headingTree">
                                <h4 class="panel-title">
                                    <a class="collapsed" role="button" data-toggle="collapse"
data-parent="#accordion" href="#collapseTree" aria-controls="collapseTree">
                                        <i class="fa fa-users fa-fw"></i> Usúarios
                                </h4>
                            </div>
                            <div id="collapseTree" class="panel-collapse collapse" role="tabpanel"
aria-labelledby="headingTree">
                                <div class="panel-body">
                                    <ul class="nav nav-second-level">
                                        <li>
                                            <a href="/usuario/form">Cadastrar Usúarios</a>
                                        </li>
                                        <li>
                                            <a href="/usuario/list">Listar Usúarios</a>
                                        </li>
                                    </ul>
                                </div>
                            </div>
                        </div>
                    </sec:authorize>
                    <div class="panel panel-primary">
                        <div class="panel-heading" role="tab" id="headingFour">
                            <h4 class="panel-title">
                                <a class="collapsed" role="button" data-toggle="collapse" data-
parent="#accordion" href="#collapseFour" aria-controls="collapseFour">
                                    <i class="fa fa-file fa-fw"></i> Relatórios
                            </h4>
                        </div>
                        <div id="collapseFour" class="panel-collapse collapse" role="tabpanel"
aria-labelledby="headingFour">
                            <div class="panel-body">
                                <ul class="nav nav-second-level">
                                    <li>
                                        <a href="/relatorio/porNome">Por Nome</a>
                                    </li>
                                </ul>
                            </div>
                        </div>
                    </div>
                </div>
            </div>
            <!-- /.sidebar-collapse -->
        </div>
        <!-- /.navbar-static-side -->
    </nav>
```

```
        <section id="main-content">
            <jsp:doBody />
        </section>
        <script src="<c:url value=" /assets/js/sweetalert.min.js " />"></script>
        <!--Import jQuery before export.js-->
        <script type="text/javascript" src="https://code.jquery.com/jquery-2.1.1.min.js"></script>
        <!--Data Table-->
        <script type="text/javascript" src="
https://cdn.datatables.net/1.10.13/js/jquery.dataTables.min.js"></script>
        <script type="text/javascript" src="
https://cdn.datatables.net/buttons/1.2.4/js/dataTables.buttons.min.js"></script>
        <!--Export table buttons-->
        <script type="text/javascript"
src="https://cdnjs.cloudflare.com/ajax/libs/jszip/2.5.0/jszip.min.js"></script>
        <script type="text/javascript"
src="https://cdn.rawgit.com/bpampuch/pdfmake/0.1.24/build/pdfmake.min.js"></script>
        <script type="text/javascript"
src="https://cdn.rawgit.com/bpampuch/pdfmake/0.1.24/build/vfs_fonts.js"></script>
        <script type="text/javascript"
src="https://cdn.datatables.net/buttons/1.2.4/js/buttons.html5.min.js"></script>
        <script type="text/javascript"
src="https://cdn.datatables.net/buttons/1.2.1/js/buttons.print.min.js"></script>
        <script type="text/javascript" src="/assets/js/bootstrap.min.js"></script>
        <jsp:invoke fragment="scriptsEspecificos"></jsp:invoke>
</body>
</html>
```

Listagem 2 - Leiaute área administrativa

A Listagem 2 apresenta um padrão para construção do menu da página administrativa, a classe "*dropdown-menu dropdown-user*" constrói o menu que mostra o nome de usuário e botão para *logout* do sistema. Para mostrar o nome do usuário da sessão atual, foi utilizada a *tagLib* do *springSecurity* que foi nominada de "sec". Utilizando a função *<sec:authentication property="principal.nome" />* é possível acessar as informações do usuário da sessão.

Na construção dos menus para acesso as informações, foram utilizadas as classes *collapsed* e a *data-parent accordion*. Aplicando essas classes, combinado com a classe *nav nav-second-level*, as opções ficam recolhidas dentro do menu principal, podendo ser acessadas por meio da expansão do menu. Esse padrão é utilizado para apresentar todos os menus do sistema.

Também é possível visualizar que a *tagLib* "*sec*" é utilizada novamente, em conjunto com a função *authorize*. Essa função limita o acesso de usuários autenticados de acordo com a permissão que foi dada a ele, caso o usuário não possua permissão, parte do leiaute não é carregado.

A página inicial do sistema foi desenvolvida na página "*index.jsp*" (Listagem 3). É possível observar o usuário da taglib "*layout*", que faz o uso dos leiautes padrões, no caso da página *index*, foi utilizado o de nome *template.* Também é possível notar a utilização de uma função JavaScript para controlar o *checkbox* de manifestação anônima. A função denominada

“bloqueio” tem o objetivo de esconder parte do leiaute, representado pela div “dadosPessoais” caso o cidadão marque a opções anônimo.

```
<%@ page language="java" contentType="text/html; charset=utf-8" pageEncoding="utf-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>
<%@ taglib tagdir="/WEB-INF/tags/layout" prefix="layout"%>
<layout:template>
    <jsp:attribute name="scriptsEspecificos">

    </jsp:attribute>
    <jsp:body>
        <section class="container col-md-8 col-md-offset-2 margem-top">
            <form id="frm" action="/" method="post">
                <input type="hidden" id="id" name="id" value="${manifestacao.id}" class="form-control"
/>
                <div class="form-check">
                    <label class="form-check-label">
                        <input type="checkbox" id="checkbox" name="checkbox" value="checkbox"
onclick="bloqueio()"> Manifestação Anônima
                    </label>
                </div>
                <div id="dadosPessoais" class="form-horizontal">
                    <div class="panel panel-primary">
                        <div class="panel-heading">
                            <h3 class="panel-title">Informações Pessoais</h3>
                        </div>
                        <div class="panel-body">
                            <div class="form-group">
                                <label for="nome" class="control-label col-md-2">Nome</label>
                                <div class="col-md-10">
                                    <div class="input-group">
                                        <span class="input-group-addon glyphicon glyphicon-
user"></span>
                                        <input type="text" required="required" id="nome" name="nome"
value="${informacaoesPessoais.nome}" class="form-control" />
                                    </div>
                                </div>
                            </div>
                            <div class="form-group">
                                <label for="telefone" class="control-label col-md-2">Telefone</label>
                                <div class="col-md-10">
                                    <div class="input-group">
                                        <span class="input-group-addon glyphicon glyphicon-phone-
alt"></span>
                                        <input type="text" required="required" id="telefone"
name="telefone" value="${informacaoesPessoais.telefone}" class="form-control"
                                        />
                                    </div>
                                </div>
                            </div>
                            <div class="form-group">
                                <label for="email" class="control-label col-md-2">Email</label>
                                <div class="col-md-10">
                                    <div class="input-group">
                                        <span class="input-group-addon glyphicon glyphicon glyphicon-
envelope"></span>
                                        <input type="email" required="required" id="email"
name="email" value="${informacaoesPessoais.email}" class="form-control"
                                        />
                                    </div>
                                </div>
                            </div>
                            <div class="form-group">
                                <label for="cep" class="control-label col-md-2">CEP</label>
                                <div class="col-md-10">
                                    <div class="input-group">
                                        <span class="input-group-addon glyphicon    glyphicon
glyphicon-map-marker"></span>
                                        <input type="text" required="required" id="cep" name="cep"
value="${informacaoesPessoais.cep}" class="form-control" />
                                    </div>
                                </div>
```

```
                            </div>
                            <div class="form-group">
                                <label for="bairro" class="control-label col-md-2">Bairro</label>
                                <div class="col-md-10">
                                    <div class="input-group">
                                        <span class="input-group-addon glyphicon    glyphicon
glyphicon-home"></span>
                                        <input type="text" required="required" id="bairro"
name="bairro" value="${informacaoesPessoais.bairro}" class="form-control"
                                        />
                                    </div>
                                </div>
                            </div>
                            <div class="form-group">
                                <label for="rua" class="control-label col-md-2">Rua</label>
                                <div class="col-md-10">
                                    <div class="input-group">
                                        <span class="input-group-addon glyphicon  glyphicon glyphicon-
road"></span>
                                        <input type="text" required="required" id="rua" name="rua"
value="${informacaoesPessoais.rua}" class="form-control" />
                                    </div>
                                </div>
                            </div>
                            <div class="form-group">
                                <label for="numero" class="control-label col-md-2">Numero</label>
                                <div class="col-md-10">
                                    <div class="input-group">
                                        <span class="input-group-addon glyphicon glyphicon-asterisk
"></span>
                                        <input type="text" required="required" id="numero"
name="numero" value="${informacaoesPessoais.numero}" class="form-control"
                                        />
                                    </div>
                                </div>
                            </div>
                            <div class="form-group">
                                <label for="complemento" class="control-label col-md-
2">Complemento</label>
                                <div class="col-md-10">
                                    <div class="input-group">
                                        <span class="input-group-addon glyphicon glyphicon-
minus"></span>
                                        <input type="text" required="required" id="complemento"
name="complemento" value="${informacaoesPessoais.complemento}" class="form-control"
                                        />
                                    </div>
                                </div>
                            </div>
                        </div>
                    </div>
                    <div class="panel panel-primary">
                        <div class="panel-heading">
                            <h3 class="panel-title">Dados da Manifestação</h3>
                        </div>
                        <div class="panel-body">
                            <div id="dadosManifestacao" class="form-horizontal">
                                <div class="form-group">
                                    <p class="h4"></p>
                                </div>
                                <div class="form-group">
                                    <label for="complemento" class="control-label col-md-2">Tipo</label>
                                    <div class="col-md-10">
                                        <div class="input-group">
                                            <span class="input-group-addon glyphicon glyphicon-th-
list"></span>
                                            <select class="form-control select" name="tipoManifestacao"
id="tipoManifestacao" required>
                                                <option value="reclamacao">Reclamação</option>
                                                <option value="solicitacao">Solicitação</option>
                                                <option value="denuncia">Denuncia</option>
                                                <option value="elogio">Elogio</option>
                                            </select>
```

```
                                    </div>
                                </div>
                            </div>
                            <div class="form-group">
                                <label for="comentario" class="control-label col-md-
2">Comentário</label>
                                <div class="col-md-10">
                                    <div class="input-group">
                                        <span class="input-group-addon glyphicon glyphicon-
font"></span>
                                        <textarea class="form-control" required="required"
id="assunto" name="assunto" value="${manifestacao.assunto}" rows="4"></textarea>
                                    </div>
                                </div>
                            </div>
                            <div class="form-group">
                                <label for="data" class="control-label col-md-2">Data</label>
                                <div class="col-md-10">
                                    <div class="input-group">
                                        <span class="input-group-addon glyphicon glyphicon-
calendar"></span>
                                        <input type="date" required="required" id="data" name="data"
value="${manifestacao.data}" class="form-control" />
                                    </div>
                                </div>
                            </div>
                            <div class="form-group">
                                <label for="bairro" class="control-label col-md-2">Bairro</label>
                                <div class="col-md-10">
                                    <div class="input-group">
                                        <span class="input-group-addon glyphicon    glyphicon
glyphicon-home"></span>
                                        <input type="text" required="required" id="mbairro"
name="mbairro" value="${manifestacao.mbairro}" class="form-control" />
                                    </div>
                                </div>
                            </div>
                            <div class="form-group">
                                <label for="rua" class="control-label col-md-2">Rua</label>
                                <div class="col-md-10">
                                    <div class="input-group">
                                        <span class="input-group-addon glyphicon  glyphicon glyphicon-
road"></span>
                                        <input type="text" required="required" id="mrua" name="mrua"
value="${manifestacao.mrua}" class="form-control" />
                                    </div>
                                </div>
                            </div>
                            <input type="hidden" id="meioContato" name="meioContato" value="sistema"
class="form-control" />
                        </div>
                    </div>
                </div>
                <div class="">
                    <div class="panel-body">
                        <div class="form-group">
                            <div class="col-md-1">
                                <input type="submit" class="btn btn-primary" value="Salvar">
                            </div>
                            <div class="col-md-1">
                                <input type="reset" class="btn btn-info" value="Limpar">
                            </div>
                        </div>
                    </div>
                </div>
            </form>
        </section>
    </jsp:body>
</layout:template>
```

Listagem 3 - Página *Index*

A *taglib* jsp:body insere o conteúdo da página no leiaute padrão. A estrutura da página foi dividida em duas *div* principais, denominadas “dadosPessoais” e “dadosManifestacao”, ambas utilizando a classe “*form-horizontal*”. Os campos de cadastro foram inseridos dentro dessas partições principais, respeitando sempre um padrão, após a estruturação da *tag* “form” que demonstra que a página é uma de formulário.

Os campos de cadastro são compostos por uma *div* representada pela classe “form-group”, uma *label* para o campo, *divs* para controle do tamanho do campo como, por exemplo, a *div* de classe *col-md-10*. A composição do campo também apresenta a *tag spam* que insere uma representação gráfica do campo no canto esquerdo do cadastro, por último a campo *input*, que recebe as informações digitadas. Esse padrão é seguido em todos os formulários do sistema, salvo a *tag spam*, que é utilizada somente nesse *form*. Por fim, são incluídos os botões de salvar (*submit*) e de limpar os campos do formulário.

A Listagem 4 apresenta como são estruturadas as listas que exibem os dados no sistema. Inicialmente é possível observar que a página utiliza o leiaute padrão “templateAdmin”.

```
<%@ page language="java" contentType="text/html; charset=utf-8" pageEncoding="utf-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>
<%@ taglib tagdir="/WEB-INF/tags/layout" prefix="layout"%>

<layout:templateAdmin>
    <jsp:attribute name="scriptsEspecificos">
        <script src="<c:url value=" /assets/js/secretario.js "/>"></script>

    </jsp:attribute>
    <jsp:body>
        <section class="wrapper">
            <!-- Page Content -->
            <div id="page-wrapper">
                <div class="container-fluid">
                    <div class="row">
                        <div class="col-lg-12">
                            <h1 class="page-header"></h1>
                        </div>
                        <!-- /.col-lg-12 -->
                    </div>
                    <section class="wrapper">

                        <a class="btn btn-primary" href="<c:url value=" /secretario/form "/>"
id="novo">
                            <i class="fa fa-plus" aria-hidden="true"></i> Novo</a>
                        <br />
                        <br />
                        <div class="spacer"></div>

                        <div class="panel panel-primary">
                            <div class="panel-heading">
                                <h3 class="panel-title">Lista de secretários</h3>
                            </div>
                            <div class="panel-body">

                                <table id=secretario class="table table-striped table-bordered
display" style="width: 100%">
                                    <thead>
                                        <tr>
                                            <th class="col-md-2">ID</th>
```

```
                                        <th class="col-md-2">Nome</th>
                                        <th class="col-md-2">Data de entrada</th>
                                        <th class="col-md-2">Data de Saída</th>
                                        <th class="col-md-2">Ação</th>
                                    </tr>
                                </thead>
                                <c:forEach var="secretario" items="${secretarios}">
                                    <tr id="row_${secretario.id}">
                                        <td>${secretario.id}</td>
                                        <td>${secretario.nome}</td>
                                        <td>${secretario.data_entrada}</td>
                                        <td>${secretario.data_saida}</td>
                                        <td>
                                            <a class="edit btn btn-primary" title="Editar"
href="<c:url value=" /secretario/form/${secretario.id} "/>">
                                                <i class="fa fa-pencil" aria-hidden="true"></i>
                                            </a>
                                            <a class="remove btn btn-danger" title="Remover"
href="javascript:remover(${secretario.id})">
                                                <i class="fa fa-trash" aria-hidden="true"></i>
                                            </a>
                                        </td>
                                    </tr>
                                </c:forEach>
                            </table>
                        </div>
                    </div>
                </section>
            </div>
            <!-- /.container-fluid -->
        </div>
        <!-- /#page-wrapper -->
    </section>
</jsp:body>
</layout:templateAdmin>
```

Listagem 4 - Exemplo de tabela de dados

Na Listagem A *tag jsp:attribute* é responsável por carregar os códigos JavaScript que carregam a estrutura do *datatable e* também os botões para importação da *table* formatos de PDF ou XSLX. Os rótulos da tabela são inseridos na *table* por meio das *tags tr* e *th.* Mais abaixo é possível observar a *taglib c:forEach* que é utilizada para carregar as informações que são enviadas do *controller* a variável *var* recebe uma lista de dados e é carregada linha a linha até o fim da lista. Todas as tabelas de dados do sistema são carregadas seguindo o mesmo padrão de estrutura.

A Listagem 5 apresenta um exemplo de como os *models* do sistema são estruturados. A anotação *@Entity* é responsável por mapear a classe no banco de dados, levando em consideração os atributos da classe.

```
package br.edu.utfpr.model;

import java.io.Serializable;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import lombok.Data;

@Entity
@Data
public class Manifestacao implements Serializable {
```

```
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String data;

    private String assunto;

    private String mrua;

    private String mbairro;

    private String tipoManifestacao;
}
```

Listagem 5 - Exemplo de model padrão

A anotação *@Data* é importada do projeto lombok que tem o objetivo evitar a repetição de códigos padrões na aplicação, a anotação é responsável por criar todos os *getters* e *setters* da classe. A anotação *@Id* e *@GeneratedValue* são responsáveis por sinalizar que o atributo id da classe é único e gerado automaticamente pelo sistema.

A Listagem 6 mostra um exemplo de como são estruturados os *repositorys* do projeto. O *repository* é uma interface que estende a classe *JpaRepository* importado do *springframework* e é responsável por controlar as transações com o banco de dados.

```
package br.edu.utfpr.repository;

import java.util.List;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import br.edu.utfpr.model.Manifestacao;
import br.edu.utfpr.model.Protocolo;

public interface ProtocoloRepository extends JpaRepository<Protocolo, Long> {

    @Query(value = "\r\n" +
            "select manifestacao.data as dataManifestacao, protocolo.comentario as
protocoloComentario, " +
            "protocolo.status_manifestacao, secretaria.nome from manifestacao inner join " +
            "protocolo on protocolo.manifestacao_id = manifestacao.id inner join " +
            "secretaria on protocolo.secretaria_id = secretaria.id where codigo like :cod",
nativeQuery = true)
    public List<Object[]> findByCodigo(@Param("cod") String codigo);

    public Protocolo findByCodigoOrderByCodigo(String codigo);

    public Protocolo findByManifestacao(Manifestacao manifestacao);

    public Protocolo findBySecretariaNome(String nome);

    public Protocolo findProtocoloByCodigo(String codigo);

}
```

Listagem 6 - Exemplo de *repository*

As consultas no padrão são declaradas de forma simples, apenas o parâmetro que será retornado pela consulta, o nome da consulta e o parâmetro. Já a anotação *@Query,* juntamente com o atributo, “nativeQuery = true”, utiliza o padrão jpql que é uma linguagem

própria usada para criar consultas fora do padrão do *spring framework*. Para passar um parâmetro por meio da consulta é utilizada a anotação *@Param*.

Os *controllers* do sistema são responsáveis por enviar e receber informações pelas páginas do sistema. A Listagem 7 mostra o *controller* responsável pela página *index* do sistema.

```
package br.edu.utfpr.controller;

import java.text.DateFormat;
import java.util.Calendar;
import java.util.Date;
import java.util.Random;
import javax.validation.Valid;
import org.json.JSONObject;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import br.edu.utfpr.model.InformacoesPessoais;
import br.edu.utfpr.model.Manifestacao;
import br.edu.utfpr.model.Protocolo;
import br.edu.utfpr.repository.InformacoesPessoaisRepository;
import br.edu.utfpr.repository.ManifestacaoRepository;
import br.edu.utfpr.repository.ProtocoloRepository;

@Controller
@RequestMapping("/")
public class IndexController {

    @Autowired
    private InformacoesPessoaisRepository informacoesPessoaisRepository;

    @Autowired
    private ProtocoloRepository protocoloRepository;

    @Autowired
    private ManifestacaoRepository manifestacaoRepository;

    @RequestMapping(value = "/index", method = RequestMethod.GET)
    public String list(Model model) {
        return "index";
    }

    @RequestMapping(value = "", method = RequestMethod.GET)
    public String index(Model model) {
        return "index";
    }

    public String retornaProtocolo(Model model) {
        model.addAttribute("protocolo", protocoloRepository.findAll());
        return "/protocolo/protocolo";

    }

    @RequestMapping(value = "/", method = RequestMethod.POST, produces = "application/json")
    public String salvar(@Valid Manifestacao manifestacao, InformacoesPessoais informacoesPessoais,
Model model,
            BindingResult erros) {
        JSONObject retorno = new JSONObject();
        Protocolo protocolo = new Protocolo();
        Date data = new Date();
        try {

            if (erros.hasErrors()) {
                retorno.put("situacao", "ERRO");
                retorno.put("mensagem", "Falha ao salvar registro!");
            } else {
```

```
                String dStr = java.text.DateFormat.getDateInstance(DateFormat.MEDIUM).format(data);
                protocolo.setData_envio(dStr);
                manifestacao.setData(dStr);
                protocolo.setCodigo(geraProtocolo());
                protocolo.setManifestacao(manifestacao);
                informacoesPessoais.setManifestacao(manifestacao);
                manifestacaoRepository.save(manifestacao);
                informacoesPessoaisRepository.save(informacoesPessoais);
                protocoloRepository.save(protocolo);

                retorno.put("situacao", "OK");
                retorno.put("mensagem", "Registro salvo com sucesso!");
                retorno.put("id", manifestacao.getId());

                model.addAttribute("protocolo", protocoloRepository.findByManifestacao(manifestacao));
                return "/protocolo/protocolo";
            }

        } catch (Exception ex) {
            retorno.put("situacao", "ERRO");
            retorno.put("mensagem", "Falha ao salvar registro! - <br /> " + ex.getMessage());
            ex.printStackTrace();
        }
        return retorno.toString();
    }

    private String geraProtocolo() {
        String protocolo = "";
        String letras = "ABCDEFGHIJKLMNOPQRSTUVYWXZ1234567890";
        Random caracteres = new Random();
        Calendar cal = Calendar.getInstance();
        int aux = cal.get(Calendar.MILLISECOND);
        String armazenaChaves = "";
        int index = -1;
        for (int i = 0; i < 6; i++) {
            index = caracteres.nextInt(letras.length());
            armazenaChaves += letras.substring(index, index + 1);
        }

        protocolo = armazenaChaves + aux;

        return protocolo;

    }
}
```

Listagem 7 - Index controller

A anotação *@Controller* declarada na classe sinaliza que ela é responsável por controlar determinadas *urls* do sistema, a anotação *@RequestMapping* seguida do parâmetro ("/") configura qual *url* o *controller* irá responder. Em seguida as anotações *@Autowired* injetam instâncias dos objetos no *controller* para que eles possam ser usados quando necessário.

A estrutura para que um método responda a uma requisição de página é exemplificada na linha acima do método *index "@RequestMapping(value = "", method = RequestMethod.GET)",* sendo que o parâmetro de valor do *@RequestMapping* representa a página que o método irá responder, também é possível observar o uso da expressão "GET" indica que o método envia informações para a página. Também é possível utilizar o método "POST" para receber informações da página.

O método “retornaProtocolo” representa um exemplo de como as informações são enviadas utilizando um método do *controller* para a página. O parâmetro “Model” é inserido no método passando uma lista do objeto que vem do *repository* pesquisado. A linha de código “model.addAttribute("protocolo", protocoloRepository.findAll())” carrega uma lista de protocolos na chamada “protocolo”, a linha “return "/protocolo/protocolo"” carrega a lista na página especificada.

O método “salvar” que é um método POST, recebe informações de página e salva essas informações no banco de dados. A anotação *@Valid* recebe instâncias de “informaçõesPessoais” e “manifestacao”, com essas instâncias o método trabalha dos dados de data. Após o tratamento do formato das datas, o sistema utiliza um método criado para gerar o número de protocolo. O método geraProtocolo() utiliza um *random* de caracteres somados a um número auxiliar gerado pelos milissegundos em que a aplicação estava no momento do processamento. Por fim, as informações são salvas em seus respectivos *repositorys,* utilizando o método save e retorna a página de protocolo.

A Listagem 8 apresenta outro *controller* que utiliza os mesmos conceitos do *index*, utiliza também anotações como *@Controller, @RequestMapping, @Autowired*.

```
package br.edu.utfpr.controller;

import javax.validation.Valid;

import org.json.JSONObject;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.annotation.AuthenticationPrincipal;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseBody;
import br.edu.utfpr.model.InformacoesPessoais;
import br.edu.utfpr.model.Manifestacao;
import br.edu.utfpr.model.Protocolo;
import br.edu.utfpr.model.Secretaria;
import br.edu.utfpr.model.Usuario;
import br.edu.utfpr.repository.InformacoesPessoaisRepository;
import br.edu.utfpr.repository.ManifestacaoRepository;
import br.edu.utfpr.repository.ProtocoloRepository;
import br.edu.utfpr.repository.SecretariaRepository;

@Controller
@RequestMapping("/manifestacao")
public class ManifestacaoController {

    @Autowired
    private ManifestacaoRepository manifestacaoRepository;

    @Autowired
    private SecretariaRepository secretariaRepository;

    @Autowired
    private InformacoesPessoaisRepository informacoesPessoaisRepository;

    @Autowired
    private ProtocoloRepository protocoloRepository;
```

```
    @RequestMapping(value = "/form/{id}", method = RequestMethod.GET)
    public String form(@PathVariable Long id, Model model, @AuthenticationPrincipal Usuario usuario) {
        InformacoesPessoais info = new InformacoesPessoais();
        info = informacoesPessoaisRepository.findOne(id);
        Long idManifestacao = info.getManifestacao().getId();
        model.addAttribute("secretarias", secretariaRepository.findAll());
        model.addAttribute("informacoesPessoais", informacoesPessoaisRepository.findOne(id));
        model.addAttribute("manifestacao", manifestacaoRepository.findOne(idManifestacao));
        model.addAttribute("protocolo", protocoloRepository.findOne(id));
        return "/manifestacao/form";
    }

    @RequestMapping(value = "/", method = RequestMethod.GET)
    public String list(Model model, @AuthenticationPrincipal Usuario usuario) {
        Secretaria secretaria = secretariaRepository.findOne(usuario.getSecretaria().getId());
        if (secretaria.getNome() == "admin") {
            model.addAttribute("manifestacoes",
                    manifestacaoRepository.findByManifestacaoAndProtocoloAndSecretaria());
        } else {
            model.addAttribute("secretaria", secretaria);
            model.addAttribute("manifestacoes",
                    manifestacaoRepository.
                    findManifestacaoAndProtocoloAndSecretariaBySecretariaId(secretaria.getId()));
        }
        return "/manifestacao/list";
    }

    @RequestMapping(value = "/form/{id}", method = RequestMethod.POST, produces = "application/json")
    public String salvar(@Valid Manifestacao manifestacao, Protocolo protocolo, BindingResult erros,
Model model) {
        JSONObject retorno = new JSONObject();

        try {

            if (erros.hasErrors()) {
                retorno.put("situacao", "ERRO");
                retorno.put("mensagem", "Falha ao salvar registro!");
                return "/manifestacao/";
            } else {

                protocolo.setManifestacao(manifestacao);
                protocoloRepository.save(protocolo);
                retorno.put("situacao", "OK");
                retorno.put("mensagem", "Registro salvo com sucesso!");
                retorno.put("id", manifestacao.getId());
                return "/manifestacao/";

            }
        } catch (Exception ex) {
            retorno.put("situacao", "ERRO");
            retorno.put("mensagem", "Falha ao salvar registro! - <br /> " + ex.getMessage());
            ex.printStackTrace();
        }
        return retorno.toString();
    }

    @RequestMapping(value = "/delete/{id}", produces = "application/json", method = RequestMethod.GET)
    @ResponseBody
    public String excluir(@PathVariable Long id) {
        JSONObject retorno = new JSONObject();

        try {
            manifestacaoRepository.delete(id);
            retorno.put("situacao", "OK");
            retorno.put("mensagem", "Registro removido com sucesso!");
        } catch (Exception ex) {
            retorno.put("situacao", "ERRO");
            retorno.put("mensagem", "Falha ao remover registro!");
        }
        return retorno.toString();
    }
}
```

Listagem 8 - Controller Manifestação

Além dos conceitos utilizados no *index, manifestacaoController* utiliza a anotação *@AuthenticationPrincipal* que "captura" as informações do usuário dentro da sessão. Essas informações são necessárias para o método *list*, no qual as informações são carregadas de acordo com a secretaria de cada usuário.

O método *delete* é responsável com receber uma informação por meio do parâmetro {id} e excluir o registro correspondente. Esse padrão de método é utilizado em todos os *controllers* em que é permitido excluir um registro.

5 CONCLUSÃO

Este trabalho teve como objetivo o desenvolvimento de uma aplicação *web* para facilitar a comunicação do cidadão com o órgão público, permitindo que o cidadão tenha melhor controle das manifestações enviadas. Foram utilizados neste trabalho a linguagem Java para desenvolvimento do *back-end*, para desenvolvimento do *front-end* foram utilizadas tecnologias como HTML, JavaScript, CSS.

Para facilitar o desenvolvimento da aplicação foram empregados o uso de *frameworks* tanto para o *front-end* quanto para o *back-end*. Para a implementação do lado servidor foi utilizado o Spring MVC que é um conjunto de *frameworks* que facilita o desenvolvimento de uma aplicação Java. O Spring é divido em vários módulos, que fornecem soluções para diferentes etapas do desenvolvimento, como, por exemplo, o Spring Data JPA, que tem como função, agilizar e facilitar o acesso aos dados do banco de dados ou o Spring Security que facilita a configuração de segurança do sistema.

O Bootstrap foi utilizado para estilizar a apresentação do conteúdo. Esse *framework* fornece varias opções de estilização de páginas HTML como, por exemplo, o sistema de *grids* que auxilia no desenvolvimento de aplicações *web* responsivas, ou seja, a interface se adapta em diferentes resoluções. Esse recurso é importante em uma aplicação *web*, pois muitos usuários a acessam por meio de dispositivos diversificados como, celulares, *notebooks*, *tablets*, entre outros.

Dentre as funcionalidades do sistema, como trabalho futuro, pode-se desenvolver a emissão de relatórios das localidades que mais possuem manifestações, das secretarias que mais recebem manifestações, entre outros. Ainda, pode-se desenvolver o controle de fluxo das manifestações, para que o cidadão acompanhe, de forma gradativa, o registro de sua manifestação e quem a recebeu e/ou analisou e o resultado dessa análise.

REFERÊNCIAS

BRYANT, Jay; JONES, Mike. **Pro HTML5 Performance.** New York: Apress, 2012.

CARDOSO, Antonio Semeraro Rito. **Ouvidoria pública como instrumento de mudança**, 2010. Disponível em: <http://repositorio.ipea.gov.br >. Acesso em: 28 ago. 2016.

DE MARIO, Camila Gonçalves. **Ouvidorias públicas em debate: possibilidades e desafios.** Jundiaí: Paco Editorial, 2011.

DE MARIO, Camila Gonçalves. **Ouvidorias públicas municipais no Brasil**. Dissertação (Mestrado) Pontifícia Universidade Católica de Campinas, 2006.

GUTIERREZ, Felipe. **Introducing spring framework: a primer**. Rio de Janeiro, Apress, 2014.

HEMRAJANI, Anil. **Desenvolvimento ágil em Java com Spring, Hibernate e Eclipse**. São Paulo, Pearson, 2007.

LUCKOW, Décio Heinzelmann; MELO, Alexandre Altair de. **Programação Java para a Web, São Paulo,** 2010.

MAZZA, Lucas; **HTML5 e CSS3**: domine a web do futuro. São Paulo; editora Casa do Código, 2012.

PRESSMAN, Roger; MAXIM, Bruce. **Engenharia de Software**, 8 ed.. McGraw Hill Brasil, 2016.

SAMPAIO, Cleuton. **Guia do java enterprise edition 5**: desenvolvendo aplicações corporativas. Rio de Janeiro: Brasport, 2007.

SILVEIRA, Guilherme; KUNG, Fabio; MOREIRA, Guilherme. **Introdução à arquitetura de design de software**: uma visão sobre a plataforma Java. Rio de Janeiro: Elsevier, 2011.

WAZLAWICK, Raul Sidnei. **Análise e projeto de sistemas de informação orientados a objetos.** 2 ed. Rio de Janeiro, RJ: Elsevier, 2011.

W3C - World Wide Web Consortium Disponível em: < https://www.w3.org/standards/webarch/> Acesso em: 01 jul. 2018a.