

FACULDADE PROFESSOR MIGUEL ÂNGELO DA SILVA SANTOS – FeMASS

CURSO DE GRADUAÇÃO EM SISTEMAS DE INFORMAÇÃO

DESENVOLVIMENTO DE SISTEMA WEB PARA SUPORTE À ZELADORIA URBANA

POR:

GUSTAVO ALBERTO DE SOUZA LEMOS

MACAÉ

2022

FACULDADE PROFESSOR MIGUEL ÂNGELO DA SILVA SANTOS – FeMASS

CURSO DE GRADUAÇÃO EMSISTEMAS DE INFORMAÇÃO

Gustavo Alberto de Souza Lemos

DESENVOLVIMENTO DE SISTEMA WEB PARA SUPORTE À ZELADORIA URBANA

Trabalho Final apresentado ao curso de graduação em Sistemas de Informação, da Faculdade Professor Miguel Ângelo da Silva Santos (FeMASS), para obtenção do grau de BACHAREL em Sistemas de Informação.

Professor Orientador: Professor Dr. Alan Carvalho Galante

MACAÉ/RJ

2022

GUSTAVO ALBERTO DE SOUZA LEMOS

DESENVOLVIMENTO DE SISTEMA WEB PARA SUPORTE À ZELADORIA URBANA

Trabalho de Conclusão de Curso apresentado ao curso de graduação em Sistemas de Informação, da Faculdade Professor Miguel Ângelo da Silva Santos (FeMASS), para obtenção do grau de BACHAREL em Sistemas de Informação.

Aprovada em \_\_\_ de \_\_\_\_\_\_\_ de 20\_\_\_

BANCA EXAMINADORA

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Prof. Dr. Alan Galante Carvalho

Faculdade Professor Miguel Ângelo da Silva Santos (FeMASS)

1º Examinador

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Prof. Afonso Carlos Tavares Pinheiro

Faculdade Professor Miguel Ângelo da Silva Santos (FeMASS)

2º Examinador

CÓPIA DA ATA ASSINADA APÓS A BANCA

**AGRADECIMENTO**

Agradeço à minha família pela compreensão e pelo suporte durante esta etapa. Ao corpo docente da FeMASS, cada professor conseguiu me transmitir conhecimentos e experiências que vão além da vida acadêmica. Agradeço ao professor Doutor Alan Galante Carvalho por ter aceitado me orientar, pela confiança em minhas capacidades e responsabilidades, além de ter perseverado em me ajudar a enfrentar este projeto. Também quero agradecer ao Laboratório de Inovação em Gestão Pública, que me contagiou quando conheci pessoas determinadas a tornar a vida da sociedade melhor. Sem a ajuda de cada integrante este trabalho não seria possível.

**RESUMO**

Na era do conhecimento e da informação as mudanças tornaram-se ainda mais constantes,assim, é possível dizer que quem não se adaptar a esta nova realidade não conseguirá lidar com os desafios atuais. À medida que a sociedade evolui, o governo precisa acompanhar o mesmo ritmo, renovando a forma como funciona e presta serviços à sua população. A cidade de Macaé, no Rio de Janeiro, percebeu isso e está se esforçando para melhorar a qualidade de seus serviços públicos. Pensando nisso, o presente trabalho desenvolveu um protótipo de software para apoiar a comunicação do serviço de zeladoria urbana entre cidadãos e colaboradores, aplicando métodos ágeis, arquiteturas de software,tais como cliente-servidor, distribuída, mvc e mvvm, além de utilizar algumas ferramentas do ecossistema Spring e tecnologias *frontend*, como Angular e Ionic.

Palavras-chave: Serviço Público. Web. Macaé. Tecnologia da Informação

**ABSTRACT**

In the era of knowledge and information changes became even more constant, those who donot adapt themselves to this new reality will not be able to handle current challenges. As society evolves the government has got to keep up with at the same pace by renewing the way it works and provides services to its population. The city of Macaé, in Rio de Janeiro, realized that and is applying some effort to improve its public services quality. With this idea in mind, the present work develops a software prototype to support urban janitorial service communication between citizens and public agents by applying agile methods and software architectures, such as client-server, multi-tier, multi-layered, mvc and mvvm, besides by using some tools of the Spring ecosystem and frontend technologies, like Angular and Ionic.

Key words: Public Service. Macaé. Web. Information Technology

**LISTA DE FIGURAS**

[Figura 1 Modelo de classes 40](#_Toc108161389)

[Figura 2 Estrutura de pastas do servidor de credenciais 42](#_Toc108161390)

[Figura 3 Estrutura de pastas do servidor de recursos 43](#_Toc108161391)

[Figura 4 Dados estáticos no arquivo *application.yml* 44](#_Toc108161392)

[Figura 5 Código do arquivo *ResourceServerConfig*.java 45](#_Toc108161393)

[Figura 6 Conteúdo do arquivo *CitizenRepository*.java 46](#_Toc108161394)

[Figura 7 Conteúdo do arquivo *CitizenService*.java 47](#_Toc108161395)

[Figura 8 Conteúdo do arquivo *CitizenDTO*.java 48](#_Toc108161396)

[Figura 9 Conteúdo do arquivo *CitizenController*.java 49](#_Toc108161397)

[Figura 10 Protótipo da tela de cadastro utilizando o Figma 50](#_Toc108161398)

[Figura 11 Tela de acesso do cidadão 50](#_Toc108161399)

[Figura 12 Alterações no diagrama de classes 52](#_Toc108161400)

[Figura 13 Alterações em *ResourceServerConfig*.java 53](#_Toc108161401)

[Figura 14 Conteúdo de *DistrictTableSeeder*.java 54](#_Toc108161402)

[Figura 15 Conteúdo da classe de domínio *Call*.java 55](#_Toc108161403)

[Figura 16 Lista de categorias de serviço 56](#_Toc108161404)

[Figura 17 Lista de serviços 56](#_Toc108161405)

[Figura 18 Formulário da ocorrência 57](#_Toc108161406)

[Figura 19 Acesso do colaborador 57](#_Toc108161407)

[Figura 20 Lista de ocorrências 58](#_Toc108161408)

[Figura 21 Formulário de ocorrência 58](#_Toc108161409)

[Figura 22 Formulário de atendimento 59](#_Toc108161410)

[Figura 23 Diagrama de classes corrigido 60](#_Toc108161411)

[Figura 24 Estrutura do arquivo *FileStorageService.*java 61](#_Toc108161412)

[Figura 25 Organização de imagens no sistema de arquivos 62](#_Toc108161413)

[Figura 26 Visualizador de imagens construído para formulário de ocorrências 63](#_Toc108161414)

[Figura 27 Trecho de código *frontend* que gerencia imagens compactadas 64](#_Toc108161415)

[Figura 28 Lista de atendimentos da ocorrência 65](#_Toc108161416)

[Figura 29 Diagrama de classes corrigido 65](#_Toc108161417)

[Figura 30 Método da classe *CallController*.java 66](#_Toc108161418)

[Figura 31 Recurso para indicação de trote 67](#_Toc108161419)

[Figura 32 Formulário de atendimento 67](#_Toc108161420)

[Figura 33 Histórico de atendimentos pelo setor do colaborador 68](#_Toc108161421)

[Figura 34 Opções da conta na aplicação de interface para o colaborador 68](#_Toc108161422)

[Figura 35 Formulário de dados pessoais do colaborador 69](#_Toc108161423)

[Figura 36 Formulário de alteração de senhas 69](#_Toc108161424)

[Figura 37 Relatório gráfico: ocorrências x atendimentos por secretaria 70](#_Toc108161425)

[Figura 38 Lista de secretarias 71](#_Toc108161426)

[Figura 39 Formulário de criação de secretaria 71](#_Toc108161427)

[Figura 40 Lista de colaboradores 72](#_Toc108161428)

[Figura 41 Formulário de cadastro do colaborador 72](#_Toc108161429)

[Figura 42 Formulário de edição do colaborador 73](#_Toc108161430)

**SUMÁRIO**

[1 INTRODUÇÃO.................................................................................................................... 11](#_Toc108138027)

[2 CONCEITOS E FERRAMENTAS APLICADOS NO DESENVOLVIMENTO DO SISTEMA................................................................................................................................. 14](#_Toc108138028)

[**2.1 Métodos ágeis................................................................................................................ 14**](#_Toc108138029)

[**2.2 UML............................................................................................................................... 16**](#_Toc108138030)

[**2.3 HTTP............................................................................................................................. 16**](#_Toc108138031)

[2.3.1 URL...................................................................................................................... 17](#_Toc108138032)

[2.3.2 Métodos HTTP..................................................................................................... 17](#_Toc108138033)

[2.3.3 Status HTTP......................................................................................................... 18](#_Toc108138034)

[2.3.4 Estrutura da mensagem HTTP............................................................................. 18](#_Toc108138035)

[**2.4 Arquitetura de software............................................................................................... 19**](#_Toc108138036)

[2.4.1 Padrões de arquitetura.......................................................................................... 20](#_Toc108138037)

[2.4.2 Arquitetura multicamadas.................................................................................... 20](#_Toc108138038)

[2.4.3 Arquitetura cliente-servidor................................................................................. 21](#_Toc108138039)

[2.4.4 Webservices......................................................................................................... 22](#_Toc108138040)

[2.4.5 REST.................................................................................................................... 22](#_Toc108138041)

[2.4.6 Representações do recurso................................................................................... 24](#_Toc108138042)

[2.4.7 REST API............................................................................................................. 24](#_Toc108138043)

[2.4.8 Arquitetura MVC................................................................................................. 25](#_Toc108138044)

[2.4.9 Arquitetura MVVM.............................................................................................. 25](#_Toc108138045)

[**2.5 Segurança...................................................................................................................... 26**](#_Toc108138046)

[2.5.1 Autenticação e autorização................................................................................... 26](#_Toc108138047)

[2.5.2 JWT...................................................................................................................... 27](#_Toc108138048)

[2.5.3 Protocolo OpenAuthorization............................................................................. 28](#_Toc108138049)

[**2.6 Ferramentas.................................................................................................................. 29**](#_Toc108138050)

[3 APRESENTAÇÃO DO PROBLEMA DE PESQUISA.................................................... 32](#_Toc108138051)

[**3.1 Motivação...................................................................................................................... 32**](#_Toc108138052)

[4 PROCESSO DE CONSTRUÇÃO DO SISTEMA DE ZELADORIA URBANA.......... 34](#_Toc108138053)

[**4.1 Definição das *personas*................................................................................................. 34**](#_Toc108138054)

[**4.2 Descrição dos cenários................................................................................................. 34**](#_Toc108138055)

[**4.3 Levantamento de requisitos........................................................................................ 35**](#_Toc108138056)

[**4.4 Definição da arquitetura.............................................................................................. 37**](#_Toc108138057)

[**4.5 Definição do fluxo de desenvolvimento...................................................................... 38**](#_Toc108138058)

[4.5.1 Sprint 1................................................................................................................. 39](#_Toc108138059)

[4.5.2 Sprint 2................................................................................................................. 51](#_Toc108138060)

[4.5.3 Sprint 3................................................................................................................. 59](#_Toc108138061)

[4.5.4 Sprint 4................................................................................................................. 68](#_Toc108138062)

[5 CONSIDERAÇÕES FINAIS.............................................................................................. 74](#_Toc108138063)

[REFERÊNCIAS...................................................................................................................... 76](#_Toc108138063)

[ANEXO A – Declaração de correção gramatical................................................................. 79](#_Toc108138063)

# 1 INTRODUÇÃO

A emenda constitucional nº 19, artigo 37, parágrafo 3, incentiva a promoção de meios de participação do cidadão, especialmente referente ao acesso aos registros administrativos e às informações sobre as atividades governamentais. Bem como à prestação de serviços dos órgãos públicos em forma de solicitação e fiscalização.

De acordo com Sant’ana (2009), a adoção de tecnologia da informação e comunicação (TIC) é uma alternativa viável para a participação ativa do usuário cidadão referente à área de administração pública. Isso pois, especificamente em prestação de serviços públicos, existe uma forte demanda tanto nas formas de solicitar atendimento, por parte do cidadão, quanto nas formas de receber e responder um pedido de atendimento, por parte dos servidores públicos.

Sabendo que a Constituição de 1988, por intermédio do artigo 18, garante certo grau de autonomia tanto aos estados quanto aos municípios, algumas cidades já se mobilizaram em relação às formas de interatividade entre cidadão e governo. Dentre os exemplos pode-se referenciar a cidade de Volta Redonda, no Rio de Janeiro, que implantou o aplicativo Fiscaliza VR para tornar seu atendimento mais acessível e dinâmico à população.

Macaé, município do Rio de Janeiro, também decidiu inovar sua interação com seus habitantes. Conforme o artigo 2º do Decreto Municipal nº 97, de 17 de julho de 2019, inaugurou-se o laboratório de inovação, sob a responsabilidade da secretaria de planejamento, batizado de Inova Macaé, cujo vasto escopo de atividades desempenhadas pela equipe inclui as práticas de melhorias no atendimento ao usuário.

Em menos de um ano, dentre várias outras atividades desempenhadas, como treinamentos e palestras, os integrantes do Inova Macaé também desenvolveram um aplicativo para dispositivos móveis chamado de Macaé App. Este pode ser obtido através das lojas de aplicativos móveis, como Google Play ou Apple Store.

Diante desse contexto, o Macaé App viabiliza uma aproximação entre o município e os cidadãos de maneira que a prefeitura tenha uma participação mais ativa no cotidiano do cidadão macaense e torne-se mais eficiente em tomar conhecimento das necessidades da população. Tais necessidades podem ser impercebíveis quando obtidas através de processos burocráticos herdados em gerações passadas.

O aplicativo serve como uma interface da prefeitura para o cidadão e suas funcionalidades são agrupadas em segmentos como mobilidade urbana, educação, participação cidadã, saúde, Procon, zeladoria urbana, entre outros. A cada segmento de serviço é designado um sistema de software para lidar com a sua respectiva complexidade do escopo.

Por sua vez, o sistema de software referente ao segmento de zeladoria urbana, conhecido como Macaé Fiscaliza, apresenta uma considerável relevância por assegurar uma comunicação rápida, eficiente e econômica. Através do Macaé Fiscaliza os usuários são capazes de notificar o município sobre irregularidades, problemas ou solicitação de serviços nas áreas de infraestrutura, segurança, saúde, saneamento, meio ambiente, urbanismo, energia, transporte público, limpeza e conservação, mobilidade urbana, dentre outros.

Apesar de seu funcionamento ser regular, o sistema interno do Macaé Fiscaliza apresenta fragilidades porque foi desenvolvido por ferramentas que não necessitam de codificação, o que o caracteriza não somente como “engessado”, pela impossibilidade de adaptação às necessidades dos usuários, como dependente de ferramentas de terceiros. Outra preocupação são as queixas dos funcionários da prefeitura, os quais se submetem ao retrabalho durante sua utilização.

Nesse contexto, como objetivo geral, o presente estudo se compromete em desenvolver um protótipo de sistema capaz de atender às necessidades dos *stakeholders* (funcionários e cidadãos) aplicando uma arquitetura web e aspectos de segurança, como autenticação e autorização.Para garantir a efetividade desse compromisso, destacam-se como objetivos específicos: Projetar o sistema que contemple a lógica do processo de comunicação entre cidadãos e colaboradores para a prestação de serviços públicos através de metodologias ágeis, diagramas UML e *user stories* que qualifiquem as atividades exercidas pelos funcionários; aplicar a arquitetura de *webservice* na aplicação *backend*, tornando possível a comunicação com diversos dispositivos utilizando o ecossistema *Spring*;construir uma aplicação *backend* responsável por centralizar a autenticação dos usuários, obedecendo o protocolo *oauth*2.1, para que seja utilizada em múltiplas aplicações;construir uma aplicação *backend* que forneça os dados e processamento na prestação de serviços de zeladoria urbana; implementar uma aplicação que possa servir de interface para o cidadão, permitindo ser acessada através de qualquer dispositivo;e, por fim, implementar uma aplicação que sirva de interface para o servidor público.

A investigação abordada tem o cunho de propor uma solução para os problemas inicialmente apresentados e demonstrá-la em prática. Esta pesquisa foi classificada obedecendo à taxonomia estabelecida por Prodanov e Freitas (2013), assim sendo, trata-se de uma “pesquisa-ação”, para a qual, conforme o contexto apresentado, foi necessária uma abordagem qualitativa e quantitativa. Ainda, menciona-se que, para a coleta de dados, a pesquisa se submeteu a sucessivas entrevistas informais com a equipe Inova junto aos técnicos da prefeitura.

Relativo à estruturação do projeto, este capítulo serviu como uma apresentação introdutória das justificativas, do problema, objetivos geral e específicos, além da metodologia utilizada. Em seguida, no segundo capítulo, será discutido o referencial teórico sobre as ferramentas de engenharia de software e gerência de projetos que foram utilizados no sistema de software em questão; arquiteturas web; contexto sobre metodologias ágeis; processo de autenticação e autorização; tecnologias utilizadas no desenvolvimento (*frameworks*, linguagens e banco de dados).

No terceiro capítulo será apresentado o contexto do problema da pesquisa, abordando o funcionamento do aplicativo de zeladoria urbana Macaé Fiscaliza. No quarto capítulo serão explorados a construção do sistema e os possíveis obstáculos encontrados. Por fim, o quinto capítulo apresentará a conclusão e as propostas de trabalhos futuros.

# 2 CONCEITOS E FERRAMENTAS APLICADOS NO DESENVOLVIMENTO DO SISTEMA

O presente capítulo expõe elementos teóricos, práticos e o ferramental necessário para construir o sistema. Sua progressão parte de elementos mais abstratos e brandos para elementos mais específicos.

## 2**.1 Métodos ágeis**

Nas palavras de Pressman (2011, p. 82): “Em muitas situações, não se conseguirá definir completamente os requisitos antes que se inicie o projeto. É preciso ser ágil o suficiente para dar uma resposta ao ambiente de fluidos negócios”. Obedecendo a essas condições, os métodos ágeis podem ajudar no desenvolvimento de aplicações de software. Esses métodos são mais objetivos se comparados aos seus antecessores, como o cascata, dedicando a utilização de recursos principalmente na construção de software ao invés de alocá-los em quantidade considerável na preparação do projeto.

A metodologia ágil é aplicável em sistemas de pequeno a médio porte cujos requisitos podem mudar constantemente, focando nas etapas de desenvolvimento e teste. Sommerville (2016) acrescenta que os métodos ágeis são incrementais, nos quais cada incremento é pequeno e novas versões são disponibilizadas aos usuários a cada duas ou três semanas. O autor também declara que cada incremento deve apresentar um pequeno número de funcionalidades do software(SOMMERVILLE, 2021).

Sommerville (2016) explica que os métodos ágeis são norteados pelos valores do manifesto ágil. Ainda, o autor elenca aqueles que se aplicam na maioria dos métodos, sendo estes: engajamento do cliente; adesão às mudanças; entregas incrementais; manutenção da simplicidade; e, por último, pessoas acima de processos. Além do mais, Sommerville (2021) reitera que existem diversas técnicas para identificar o processo e a forma que os usuários trabalham, dentre elas a entrevista e a etnografia, porém análises e conversas informais também são efetivas e menos custosas.

A partir do momento que há motivação de criar um novo sistema voltado a automatizar um processo é necessário identificar os potenciais usuários, os interessados (*personas*), os possíveis cenários e os requisitos do sistema (expressos em *user stories*). Em resumo, *personas*são *stakeholders* que operam o sistema. Ademais, por meio da obra de Sommerville (2021), pudemos constatar que *personas* são ilustrações de usuários que auxiliam a identificar os propósitos do software e como usá-los.

Quanto ao cenário, caracteriza-se como uma narrativa que descreve as circunstâncias de uso do sistema em potencial e que apresenta um problema junto da possível solução para resolvê-lo. Para o cenário detalha-se apenas o que for considerado necessário, e por esta razão o cenário se distingue de uma especificação minuciosa.

Por sua vez, através da definição dada pelo Project Management Institute (2017), *userstory* é uma breve descrição de um valor, ou funcionalidade, que pode ser entregue a um usuário específico. Sommerville (2021) adiciona que esse valor deve expressar necessidades indivisíveis, contudo, uma *user story* que não se restringe a uma única unidade de trabalho é chamada de *epic* ou épico. Em outras palavras, um épico é uma descrição de requisitos em formato de *user story* que se desmembra em tarefas menores.

De maneira sucinta, os requisitos do sistema formatados em *user stories* se expressam como enunciados que indicam a *persona*, a funcionalidade que o sistema deve fornecer para essa *persona* e, quando necessário, a razão de existência da funcionalidade. A vantagem em adotar esses conceitos se encontra na facilidade de comunicação e entendimento entre os desenvolvedores e os usuários, contribuindo para a concepção de um software harmônico com os processos e indivíduos que participam da execução do negócio.

A informalidade proposta pelo conjunto de métodos é justificada por cumprir com os objetivos de maneira a evitar desperdício de recursos em formalidades irrelevantes. Por isso, *user stories* e cenários não seguem uma especificação normativa. Com base nas considerações de Sommerville (2021), essas práticas são ferramentas com função de explorar a criatividade semelhante ao propósito do *brainstorming* por exemplo.

Após obter um esclarecimento aceitável sobre o sistema com a coleta de requisitos necessários para o sistema (possivelmente em formato de *user stories*) e de esboçar um projeto inicial (seja através de protótipos gráficos ou diagramas), iniciam-se as etapas de desenvolvimento. Não obstante, mantendo o planejamento em paralelo, ainda que, nas palavras de Sommerville (2021), o planejamento deva ser informal e com mínima documentação.

## 2.2 UML

A base conceitual da UML é o paradigma orientado a objetos que promove a perspectiva de que a realidade deve ser expressa como um conjunto de objetos que se relacionam entre si. Este paradigma se difundiu na engenharia de software tanto acadêmica quanto comercial. Inclusive, é possível associar tal paradigma tanto à teoria das ideias, desenvolvida por Platão, quanto à teoria da forma e substância de Aristóteles. Analogamente, as ideias ou formas abstratas de Platão são equivalentes ao termo da orientação a objetos conhecido como “classe”, enquanto a forma e substância de Aristóteles equivalem ao termo “objeto” e, portanto, o objeto é uma instância da classe.

Retomando, a sigla UML significa *Unified Modeling Language*, que se traduz em um conjunto de diagramas comportamentais e estruturais que ilustram um projeto de software em diferentes perspectivas. Dentre os diagramas estruturais, o mais relevante é o diagrama de classes pois, através de Bernhard Rumpe (2017), expressa o cerne do desenvolvimento da maioria dos sistemas de software.

O diagrama de classes deixa explícito como os dados devem se estruturar e como as estruturas devem se relacionar, haja vista que é um artefato de suma importância que não pode ser ignorado (ao menos em relação ao contexto do presente projeto). Desse modo, o diagrama de classes pode ser usado como documentação mínima para complementar os métodos ágeis.

## 2.3 HTTP

*HyperText Transfer Protocol*, HTTP, é um protocolo de comunicação que atua na camada de aplicação do padrão de rede de computadores conhecido como *Transmission Control Protocol/Internet Protocol*, TCP/IP. O HTTP é descrito como base para a rede mundial de computadores e denominado como “linguagem nativa” dos servidores web (TANENBAUM; WETHERALL, 2010). Tal comunicação é baseada no envio de requisições através do computador, denominado cliente HTTP, por serviços ou dados armazenados em um servidor HTTP que, por sua vez, processa a requisição e retorna o conteúdo solicitado vinculado a uma resposta (GOURLEY; TOTTY, 2002).

O fato de o protocolo ser aplicado à rede mundial de computadores deixa implícito que um conteúdo fornecido pelo servidor HTTP pode ser acessível por clientes HTTP em qualquer parte do mundo. Ademais, tecnicamente, implica que a comunicação cliente-servidor é intermediada por outros computadores, conforme descrito pela MDN Web Docs (2022): “Entre a solicitação e a resposta existem várias entidades, designadas coletivamente como *proxies*, que executam operações diferentes e atuam como *gateways*(intermediários) ou *caches*”.

Para esclarecimento, o termo “*cache*”, apresentando no parágrafo anterior, pode ser empregado para identificar métodos de armazenamento temporário de conteúdos com alta frequência de acessos. Sendo assim, a utilização de *cache* visa reduzir o tempo de acesso.

### 2.3.1 URL

Qualquer conteúdo ou serviço que possa ser disponibilizado por um servidor específico é um recurso web. Ainda, para que um recurso seja rastreável é adotado um mecanismo conhecido como*Unified Resource Locator*, ou seja, URL (GOURLEY; TOTTY, 2002). A URL é um tipo de URI, *Unified Resource Identifier*, cuja estrutura se inicia pelo identificador do protocolo de comunicação utilizado e, logo em seguida, pelo endereço do servidor e do caminho (PATH) para acessar o recurso referente ao endereço do servidor.

### 2.3.2 Métodos HTTP

Os métodos HTTP são parte integrante de toda requisição responsável por indicar ao servidor qual ação deve ser executada. De acordo com Richardson e Ruby (2007), dentre o conjunto de métodos disponíveis,*GET*, *POST*, *PUT* e *DELETE* são utilizados nas operações mais comuns.

O método *GET* é usado em situações que o cliente solicita uma busca por dados. Por sua vez,*POST* é utilizado para a postagem e criação de um novo conteúdo no servidor. *PUT* serve para a alteração de dados já existentes no servidor. Enquanto isso,*DELETE* exclui algum recurso armazenado no servidor. Em complemento, Tanenbaum e Wetherall (2010) afirmam que os métodos devem ser declarados em letras maiúsculas.

### 2.3.3 Status HTTP

Transmitido pelo servidor, o status HTTP é a forma de informar se a requisição foi devidamente computada. Caso contrário, é indicada a ocorrência de falhas durante o recebimento ou processamento. Conforme a MDN Web Docs(2022), os status são representados em código numérico de três dígitos agrupados em cinco classes, sendo estas: indicadores de informação (100 a 199); indicadores de sucesso (200 a 299); indicadores de redirecionamento (300-399); indicadores de falha no cliente (400 a 499); e indicadores de erro no servidor (500 a 599).

### 2.3.4 Estrutura da mensagem HTTP

Gourley e Totty (2002) descrevem que tanto uma requisição HTTP quanto uma resposta HTTP podem ser generalizadas com uma mensagem. Também abordam que as mensagens trocadas entre cliente e servidor apresentam a mesma estrutura dividida em três partes: linha inicial ou *start line*; campos de cabeçalho ou *header fields*; e corpo ou *body*.

A linha inicial é parte integrante obrigatória, cujo propósito varia de acordo com o tipo de mensagem. Para uma requisição, essa parte deve descrever o método HTTP e o caminho do recurso relativo ao servidor requisitado, enquantopara a resposta, deve ser dedicada a informar o status HTTP.

Os campos de cabeçalho são parte opcional da mensagem, a qual geralmente é utilizada para auxiliar em seu detalhamento. Cada item do cabeçalho é composto por nome e valor. Um aspecto é que cada item deve ser separado por ponto e vírgula (;). Os cabeçalhos podem conter dados sobre o cliente, o servidor, o conteúdo transmitido, as credenciais do usuário e muitas outras opções. Richardson e Ruby (2007) comparam a estrutura de uma mensagem HTTP com um envelope de correspondências. Desse modo, os cabeçalhos se comportam como carimbos que acrescentam informações, as quais, no caso das correspondências, são sobre o seu trajeto.Por fim, o corpo é a parte responsável por armazenar o recurso solicitado, e sua obrigatoriedade é situacional.

## 2.4 Arquitetura de software

Conforme Bass, Clements e Kazma(2003*apud* PRESSMAN, 2011, p. 230): “A arquitetura de software de um programa ou sistema computacional é a estrutura ou estruturas do sistema, que abrange os componentes de software, as propriedades externamente visíveis desses componentes e as relações entre eles”. Ou seja, sem grande rigor, é a forma que um sistema de software é organizado e estruturado.

Sabendo que um sistema de software pode ser uma composição de diversos programas, e cada programa é um conjunto de instruções de máquina sequenciadas de forma lógica para exercer alguma ação específica. Portanto, torna-se necessário orquestrar os componentes do sistema, permitindo a devida interação entre estes. Dessa forma, é indispensável elaborar um projeto de arquitetura de software nos estágios iniciais do desenvolvimento.

Bezerra (2015) traduz arquitetura de software como uma arquitetura lógica de um sistema computacional. Enquanto isso, Hofmeister, Nord e Soni (2000 *apud* SOMMERVILLE, 2006) destacam que a arquitetura de software pode servir como uma maneira de estruturar as discussões com os indivíduos envolvidos e negociar os requisitos a serem atendidos pelo sistema. Além disso, Bosch (2000 *apud* SOMMERVILLE, 2006)completa sugerindo que a arquitetura do sistema impacta no desempenho, consistência, distribuição e manutenção do sistema. Assim, o conceito de arquitetura de software pré-determina categorias ou modelos.

Desenvolver um software é uma tarefa mais dinâmica que desenvolver um edifício por exemplo. Justamente pelo software ser subjetivo e virtual é que os programas de computador não obedecem às leis da física como os demais projetos de engenharia e, assim, não se submetem a tantas restrições. Tal façanha deve ser mérito da ciência da computação em tornar essa subjetividade algo computável.

Com isso, o sistema pode ser elaborado utilizando mais de uma arquitetura de software ou ainda compilar características de diversas arquiteturas. De acordo com Sommerville(2006, p.246, tradução nossa), “em alguns casos a arquitetura geral do sistema pode ser composta por uma combinação de diferentes arquiteturas”.

### 2.4.1 Padrões de arquitetura

O desenvolvimento de um sistema de software é suscetível a problemas.Ingeno (2018) manifesta o padrão de arquitetura de software como solução para uma “família” de problemas recorrentes pertencentes a um contexto específico. Complementando com uma síntese sobre a definição de Raj, Ramam e Subramanian (2017), um padrão de arquitetura serve como molde para a organização estrutural de sistemas complexos de forma a solucionar problemas recorrentes das circunstâncias pré-definidas às quais o softwaredeve se submeter.

Além do mais, através de Fielding (2000), entende-se que a arquitetura desoftware remete ao comportamento do software durante sua execução sob uma “macro” perspectiva ao invés de ser uma estrutura que se resume no código de programação que gera o sistema. Ainda, no que lhe diz respeito, Ingeno (2018) explica que os padrões de arquitetura podem ser aplicados no sistema como um todo ou parcialmente. Comenta também que mais de um padrão pode ser aplicado no mesmo software, pois um problema pode ser resolvido com uma combinação de soluções.

Acerca da aplicação em um projeto, Raj, Ramam e Subramanian (2017) também defendem o uso de múltiplas arquiteturas, afirmando que é uma atitude recomendável por especialistas e personalidades do ramo para acelerar e facilitar o desenvolvimento caso necessário. Ademais, Fielding (2000) esclarece que um sistema pode ser composto em níveis hierárquicos de abstração, onde cada nível pode apresentar uma arquitetura de software desde que respeite o nível superior. Quanto ao conceito de abstração, nas palavras de Shvets (2021, p.15, tradução nossa): “Abstração é um modelo de um fenômeno ou objeto do mundo real que representa todos os detalhes relevantes ao contexto, ou perspectiva, e omite os demais detalhes”.

### 2.4.2 Arquitetura multicamadas

Antes de prosseguir, é fundamental definir o significado de separação de conceitos. Baseado em Ingeno (2018), é um princípio que um sistema atinge quando se estrutura em componentes independentes para gerenciar suas funcionalidades de forma a reduzir sua complexidade.

A arquitetura multicamadas decompõe o sistema ou aplicação em elementos menores, em que cada um é responsável por parte do conjunto de funcionalidades da aplicação, que remete à separação de conceitos e torna o sistema adaptável a mudanças. Devido a questões de tradução, uma arquitetura multicamadas pode ser *multi-layer* ou *multi-tier*, o que as distingue são suas abordagens e o elemento decomposto.

Uma arquitetura *multi-layer* separa uma aplicação de software em componentes lógicos, por sua vez uma arquitetura *multi-tier* segrega um sistema em aplicações (ou programas) distintas, as quais podem se localizar em mais de uma máquina física (INGENO, 2018). Em outros termos, enquanto uma arquitetura *multi-layer* segrega logicamente, a arquitetura *multi-tier* segrega topologicamente.É importante esclarecer que um sistema *multi-layer* organiza seus elementos em camadas hierárquicas, onde cada camada fornece serviços para a sua superior (FIELDING, 2000).

### 2.4.3 Arquitetura cliente-servidor

O modelo cliente-servidor é uma das arquiteturas de software voltadas para sistemas distribuídos. Com relação a isso, Sommerville (2006, p.267, tradução nossa) define que “um sistema distribuído é um sistema onde a informação é distribuída através de diversos computadores ao invés de estar confinada em uma única máquina”.O referido autor reforça que os principais componentes desse modelo são: um conjunto de servidores, responsável por oferecer um serviço; um conjunto de clientes, que realizam as chamadas dos serviços ofertados pelos servidores; e a rede, que é o meio que permite o acesso do cliente ao servidor.

Porém, na perspectiva de projeto e desenvolvimento de software, Richards e Ford (2020) explicam que um sistema baseado na arquitetura cliente-servidor segrega suas funcionalidades técnicas em duas classes de aplicação: *frontend* e *backend*.Seguindo essa vertente, para Ingeno (2018), o *frontend* age como meio de acesso ao usuário do sistema e o *backend* atua como servidor que pode englobar o banco de dados e geralmente se responsabiliza por monitorar mensagens recebidas do *frontend*.

Quando a lógica do *frontend* se restringe a aspectos visuais a aplicação é denominada *thin-client*, desse modo todo o processamento da lógica é atribuído ao *backend*. Entretanto, existem situações em que grande parte do processamento é designada ao *frontend*, que passa a ser nomeado como *fat*-*client*. Nesse caso o servidor se restringe ao gerenciamento do banco de dados.

Dentre as vantagens de um modelo derivado de uma arquitetura de um sistema distribuído estão: compartilhamento de recursos (hardware ou software); intercambialidade de recursos mediante as padronizações não autorais; processamento concorrente; escalabilidade e tolerância a falhas (COULOURIS; DOLLIMORE; KINDBERG, 2001*apud* SOMMERVILLE, 2006).

### 2.4.4 *Webservices*

No início da internet o acesso aos recursos disponibilizados pelo servidor era viável apenas através de um navegador de internet instalado na máquina do usuário. Tal restrição impedia o aproveitamento de funcionalidades de servidores já existentes em novas aplicações *backend*.Sommerville (2016) sugere que os *webservices* nasceram da impraticabilidade na comunicação entre servidores e, logo em seguida, caracteriza um *webservice* como uma aplicação que atende a uma interface capaz de representar os recursos computacionais de maneira uniforme, independente das entidades envolvidas na comunicação, seja ela 'tanto entre cliente e servidor quanto entre servidores distintos.

Sommerville (2016) pontua que a experiências adquiridas com a adesão de *webservices* culminaram em um conjunto de especificações industriais denominadas como “WS-\* *standards*”,as quais, apesar de certificarem a qualidade do software, acarretam em alta complexidade e na sobrecarga no uso de recursos computacionais. Essas desvantagens impulsionaram as corporações na adoção de uma abordagem alternativa aos WS-\* *standards*, denominada REST.

### 2.4.5 REST

REST é um acrônimo para *Representation State Transfer*, que, nas palavras de Richardson e Ruby (2007), trata-se um grupo de critérios de projetos a serem aplicados no desenvolvimento de aplicações que se caracterizam como *webservice*. Inclusive, é uma alternativa muito mais simples que os WS-\* *standards* (SOMMERVILLE, 2016).

Bass, Clements e Kazman (2021) declaram que o REST consiste em seis restrições impostas na interação entre os componentes do sistema. A primeira restrição implica que o sistema deve aplicar a arquitetura cliente-servidor, que, por consequência, implica na separação de conceitos discutida no item 2.4.2. A restrição seguinte, conforme Massé (2012), exige que a aplicação cliente armazene toda a informação contextual considerada relevante em cada interação com o servidor. Para Fielding (2000) essa informação contextual denomina-se *session state* e por isso o servidor deve ser *stateless*. Ainda, designar a complexidade de gerenciamento do estado de sessão para o cliente potencializa a utilização de recursos do servidor, tornando-o capaz de atender uma quantidade maior de clientes.

A terceira restrição se aplica no conteúdo anexado às mensagens de resposta do servidor para que seja identificado como “cacheável” ou “não-cacheável” (a explicação sobre o termo “*cache*” encontra-se no item 2.3). De acordo com Fielding (2000), “cachear” os dados de uma resposta do servidor pode contribuir para uma melhora de desempenho perceptível ao usuário e maior eficiência.

Massé (2012) explica que a quarta restrição estabelece que haja uniformidade entre as interfaces do servidor, do cliente e das demais entidades intermediárias (*proxies* e *gateways*) no processo de troca de mensagens. A uniformidade é atingida quando quatro limitações são respeitadas, sendo elas: [1] Os recursos devem ser endereçáveis por identificadores únicos, como, por exemplo, o URL (item 2.3.1); [2] Um único recurso pode ser representado em diversos formatos de dados através de diferentes maneiras para atender às necessidades de qualquer tipo de cliente web; [3] As mensagens devem ser auto descritivas, contendo metadados para dar suporte ao encaminhamento de mensagens realizado pelas entidades intermediárias; [4] Sumariamente, o estado da aplicação deve ser representando como um conjunto de endereços relacionados a ele, remetendo ao *Hypermedia As The Engine Of Application State* (HATEOAS).

A penúltima restrição impõe que o sistema aplique a arquitetura *multi-tier* (item 2.4.2),na qual o servidor, o cliente e as entidades intermediárias atuam em suas respectivas camadas, impossibilitando cada camada de “enxergar” além do seu próprio escopo (FIELDING, 2000).Já a última restrição é opcional e permite que o cliente baixe recursos executáveis no próprio ambiente da aplicação, como *scripts*. Assim, apesar do HTTP apresentar alguns de seus termos utilizados pelas restrições REST e ser o protocolo de comunicação mais adotado por esse conjunto de regras, não há relação de dependência entre REST e HTTP (BASS; CLEMENTS; KAZMAN, 2021).

### 2.4.6 Representações do recurso

Fielding (2000) enuncia que a representação de um recurso é a sequência de bytes (um byte equivale a um agrupamento de oito algarismos binários) junto dos metadados que descrevem esses bytes.A sequência de bytes é o conteúdo que se expressa como documento ou arquivo, enquanto os metadados do conteúdo são o nome do arquivo, tamanho, tipo do arquivo (*mime-type* ou *media-type*), data de criação, entre outros.

Recursos que não se adéquam aos formatos de arquivo padronizados da computação por se tratarem de dados personalizados, de acordo com a lógica de negócio de cada sistema, costumam ser formatados em XML ou JSON.XML quer dizer *Extensible Markup Language*. Enquanto isso, JSON significa *JavaScript Object Notation*, ou notação de objeto javascript, e é utilizada na representação e transporte de estruturas de dados genéricas e complexas (RICHARDSON; RUBY, 2007).

Em relação ao formato XML, o JSON é mais compacto e legível, além de ser compatível com aplicações-cliente escritas na linguagem javascript. Sommerville (2021) recomenda o uso de JSON como forma de representação de dados em detrimento do XML.

### 2.4.7 REST API

Por intermédio da explicação de Massé (2012), entende-se que, no contexto de arquiteturas web, API, ou *Application Programming Interface*, é um componente integrado ao *webservice* responsável por interagir com os clientes utilizando um protocolo de comunicação (HTTP por exemplo). O autor ilustra uma API web como o “rosto” de um *webservice*.

De acordo com sua nomenclatura, um REST API é uma API que atende às restrições impostas pelo REST e consiste em um conjunto de elementos interligados. Sendo assim, o REST API é um exemplo de interface pontuada durante a caracterização de Sommerville (2016) sobre *webservices* (item 2.4.4).

### 2.4.8 Arquitetura MVC

Explicações nos parágrafos a seguir fazem uso do termo “estado”, que se traduz como os valores de um conjunto de atributos que caracterizam uma entidade. Tal entidade pode ser uma estrutura de dados, um subsistema ou o próprio sistema. Portanto, palavras como “estado da aplicação”, “estado da estrutura de dados” e “estado do componente” fazem referência a esse termo.

Retornando ao tópico deste item, conforme a descrição de Ingeno (2018), o padrão MVC, Model-View-Controller, é aplicável em sistemas *frontend* e *backend* que apresentam interação direta com o usuário, separando a aplicação em três camadas lógicas (*multi-layer*): *Model*; *View* e *Controller*.A camada *Model* engloba um conjunto de componentes que formatam a estrutura dos dados usados na lógica do negócio; armazenam o estado dessa estrutura durante a execução; e fornecem funcionalidades para alterar seu estado. Essas funcionalidades são acionadas por componentes externos à estrutura em questão.

Já a camada *View* agrega componentes responsáveis pela interface, assim, todos os detalhes que o usuário enxerga são descritos pelos componentes desta camada. Qualquer manipulação do usuário com a interface, como entrada de dados ou cliques em botões, deve disparar ações a serem gerenciadas pela camada *Controller*.Por sua vez, a camada *Controller* contém componentes cuja função é basicamente interligar as camadas *Model* e *View*, certificando que as ações do usuário se reflitam nos estados dos componentes do tipo *Model* e que os componentes *View* reajam de acordo com as interações do usuário.

### 2.4.9 Arquitetura MVVM

Variante do modelo MVC, a MVVM, Model-View-ViewModel, é aplicável a aplicações *frontend* e recomendada em casos que exijam interfaces gráficas com reações instantâneas às ações do usuário, como os aplicativos de telefone, aplicações web, etc. Suas camadas lógicas são: *Model*; *View* e *ViewModel*.A camada *Model* funciona de forma semelhante à sua contraparte do MVC, porém, desta vez, não necessita fornecer funcionalidades para alterar seu estado, pois, conforme Raj, Ramam e Subramanian (2017), o estado de um componente *Model* pode ser alterado diretamente através da camada *View*.

A camada *View* também se assemelha a sua contraparte MVC, entretanto apresenta maior interação com as camadas *Model* e *ViewModel* através do vínculo de dados, ou *data-biding*, responsável pelas atualizações de estado das informações em tempo real. Outro fato é que os elementos desta camada passam a gerenciar parte dos eventos de interface e se comportam dinamicamente, como a validação de dados instantânea. Através da declaração de Raj, Ramam e Subramanian (2017), observa-se que, apesar do acréscimo de funcionalidades, a camada *View* não se responsabiliza em persistir os dados durante uma atualização de página

A última camada, batizada de *ViewModel*, atua de forma semelhante à *Controller*. Além de ser responsável por gerenciar o estado dos dados; gerenciar a navegação; conter códigos de chamadas ao *backend*; e auxiliar no disparo de eventos da camada *View*.

## 2.5 Segurança

Através da análise de Spilçã (2020), segurança é um aspecto não-funcional essencial que pode impactar a lucratividade e confiabilidade de um sistema de softwarecaso não levada em consideração com antecedência.Diante da vastidão e complexidade do assunto, mecanismos de autenticação e autorização são elementos fundamentais que diferem um sistema minimamente seguro de um sistema totalmente inseguro.

### 2.5.1 Autenticação e autorização

Conforme as afirmações de Sommerville (2021), autenticação é um processo que assegura que o usuário do sistema é quem ele declara ser. O autor classifica três formas de prover a autenticação em que as informações que cada abordagem exige são fornecidas durante o cadastro.A primeira é baseada em conhecimento do usuário, na qual, ao acessar o sistema, o usuário deve fornecer as informações como senha ou perguntas pessoais. Por exemplo, “qual o nome do animal de estimação?”, entre outras.

A segunda é baseada em objetos que o usuário possui e que possam ser interligados no sistema de autenticação. Um exemplo é fornecer o número do telefone em posse do usuário para confirmar sua identidade. Por fim, a terceira é baseada nos atributos do usuário. Nesta categoria se enquadram métodos que podem se configurar em biometria de digitais, reconhecimento facial, dentre outros.

No entanto, Sommerville (2021) comenta que, apesar da eficácia de cada um dos formatos de autenticação, há desvantagens de uso em particular. Uma forma de mitigá-los, e até aumentar o nível de segurança, pode ser a exigência de mais de uma abordagem durante a autenticação. Assim, o nível de autenticação depende das necessidades do software.

Com relação à autorização, baseado em Spilçã (2020), o termo significa um processo que permite o acesso a funcionalidades ou dados específicos através da verificação das permissões de um usuário autenticado. Para Sommerville (2021), as permissões do usuário são estabelecidas obedecendo à política de controle de acesso submetida à lógica do negócio.

### 2.5.2 JWT

Baseado em Spilçã (2020), no ramo de segurança de sistemas,*token* é um termo que representa uma cadeia de caracteres utilizada como identificador de acesso. Por sua vez, JWT, ou *JSON Web Token*, é um *token* intercambiável entre sistemas web distribuídos. Segundo o autor, a estrutura do JWT é composta por três partes, separadas pelo caractere ponto (.).

A parte inicial é cabeçalho, ou *header*, encarregado de conter dados relativos às configurações do *token*, como tipo e algoritmo de assinatura.A parte intermediária é chamada de *body*, *payload* ou corpo do JWT, onde se armazena os dados necessários para o processo de autorização. Basicamente, esses dados são denominados credenciais. Já a parte final é opcional, utilizada para verificar a assinatura expedida pelo sistema que emitiu o *token*. A assinatura garante a autenticidade e inviolabilidade do *token*, já que se baseia na cifragem de todas as partes do JWT, gerando um *token* único. Qualquer adulteração no *token* gera uma assinatura completamente diferente.

O sistema emissor pode utilizar algoritmos de criptografia assimétrica, como o HS256, RS256, ES256 e diversas variantes,os quais embaralham o conteúdo de forma que seja matematicamente inviável (ou impossível) retornar para a mensagem original sem a posse da chave secreta, ou “chave-privada”, armazenada no sistema emissor. Desse modo, para gerar um JWT, o sistema deve:

* Definir o *header* e o *payload* em formato JSON (debatido no item 2.4.2);
* Logo em seguida, transformar o *header* e o *payload* em respectivas cadeias de caracteres utilizando o algoritmo de codificação *radix*64;
* Gerar a parte final aplicando o algoritmo de chave assimétrica, utilizando como entrada ambas as cadeias obtidas na etapa anterior e uma “chave-pública”, separadas pelo caractere ponto (.). Nesta etapa a “chave-privada” é opcionalmente convertida no formato base 64;
* Por fim, unir os campos resultantes da segunda e terceira etapas em um *token* interligados pelo caractere ponto (.).

De acordo com Stallings (2008), a codificação *radix*64, ou codificação de transferência base 64, é um formato de dados que protege o conteúdo de ser corrompido durante os processos de envio e recebimento.

### 2.5.3 Protocolo *OpenAuthorization*

Conhecido como *OAuth*, trata-se de uma série de especificações para permitir que um recurso protegido pelo servidor seja acessado por aplicações desenvolvidas por terceiros com o consentimento do proprietário do recurso.Essas especificações são publicadas em documentos pela Força Tarefa de Engenharia da Internet (IETF), em inglês, *Internet Engineering Task Force*, uma comunidade internacional de projetistas, pesquisadores e empresas que contribuem para o desenvolvimento da internet.

Dentre todos os demais relacionados ao protocolo *OAuth*, o documento central é referente ao framework *OAuth*, cuja última versão estável é a 2.0, apesar da versão 2.1 se encontrar em fase de rascunho e funcional para algumas necessidades. Por esse motivo, este trabalho se baseia na versão 2.1, que pouco se difere nos conteúdos principais, todavia se encontra atualizada em relação à versão anterior.

Condizente com o documento principal, o framework *OAuth*2.1 determina quatro papéis, que são:

* ***Resource owner*(RO)**: Geralmente é o usuário final;
* **Cliente**: É a aplicação *frontend*;
* ***Authorization server*(AS)**: Servidor de identificação, responsável por verificar as credenciais do usuário. Caso as informações estejam corretas, o servidor emite um identificador de acesso ao cliente, geralmente em formato JWT (item 2.5.2). Por motivos de segurança, o servidor de identificação deve previamente conter informações sobre todas as aplicações que solicitam seus serviços;
* ***Resource server*(RS)**: Servidor que processa a lógica de negócio, responsável por verificar se o cliente tem permissão de acesso a determinada funcionalidade ou recurso de dados protegido através do identificador de acesso emitido pelo AS. Se o identificador for válido, o cliente é autorizado, caso contrário o pedido é rejeitado. Obviamente o acesso a um recurso também depende das permissões do usuário, entretanto o mecanismo que estabelece permissões está fora do escopo do protocolo e deve ser definido pela lógica do negócio.

Também são especificados pelo documento, levando fortemente em consideração a segurança contra invasores, detalhes como os que envolvem tempo de validade do identificador de acesso emitido pelo AS; tipos de concessão de acesso, conhecidos como *grant-types*, e seus fluxos, dentre outros.A consistência do protocolo *OAuth* viabiliza o desenvolvimento de aplicações que deleguem a tarefa de autenticação a sistemas especializados de forma segura, reduzindo a complexidade do RS, e permite que diversos clientes em diferentes formatos (página web, aplicativo móvel) acessem o RS em segurança.

## 2.6 Ferramentas

Seguem abaixo as ferramentas tecnológicas escolhidas que forneceram suporte ao projeto e desenvolvimento do sistema:

* **StarUML**: Utilizada em sua versão gratuita, permite a edição de diversos diagramas UML, como o diagrama de classes. Dentre as ferramentas gratuitas e instaláveis em computadores, foi escolhida pela quantidade satisfatória de funcionalidades;
* **Trello**: Plataforma web focada em organização de projetos em um quadro de tarefas, ou *task board*. De acordo com Sbrocco e Macedo (2012), o quadro de tarefas torna a observação do andamento do projeto acessível, de maneira intuitiva e objetiva, utilizando o formato *post-it*;
* **Figma**: Aplicação web para design colaborativo de componentes gráficos e prototipagem de interfaces gráficas para aplicações *frontend* ou aplicativos de dispositivos embarcados.
* **Java**: Linguagem de programação orientada a objetos, popular em aplicações de diversos segmentos: desktop, web, móvel e outros dispositivos embarcados. É uma das ferramentas mais populares em aplicações empresariais de pequeno a grande porte, além de ser uma linguagem madura que apresenta um sólido suporte;
* **Lombok**: Biblioteca Java utilizada para lidar com a verbosidade da linguagem, como os métodos *getters* e *setters*;
* **Gradle**: Aplicação responsável por gerenciar as dependências (sejam bibliotecas de código ou módulos genéricos de programas) para a construção de um softwareweb. Ao baixar as dependências como pacotes, suas funcionalidades podem se tornar disponíveis para utilização no código do sistema em desenvolvimento. Sua configuração é feita pelo arquivo “*build.gradle*” e a gestão de versões do *gradle* é mais eficiente que o *maven* (aplicação concorrente);
* **Spring**: Existem diversos aspectos envolvidos na construção de software, como segurança; armazenamento de dados; desenvolvimento *mobile*; desenvolvimento web; processamento orientado a eventos (reatividade); dentre outros aspectos voltados ao desenvolvimento *backend*. Os *frameworks* disponibilizam uma estrutura de programas funcional e configurável, servindo como base para aplicações construídas sobre ela. Dessa forma, o desenvolvedor não necessita desperdiçar recursos construindo um sistema do início e se dedica no desenvolvimento de programas que atendam às regras de negócio. Spring refere-se como um ecossistema de *frameworks* de aplicação que podem ser usados em conjunto de acordo com as necessidades da aplicação. Cada *framework* Spring é especializado em um dos aspectos de desenvolvimento mencionados. Além disso, eles são independentes entre si, o que permite aplicá-los em conjunto com outros *frameworks* que não fazem parte do ecossistema Spring. Os *frameworks* selecionados são: *Spring Framework*, servindo como base das funcionalidades das aplicações *backend*; *Spring Data*, que provê funcionalidades especializadas em manipulação de banco de dados; *Spring Security*, responsável por estruturar os níveis de acesso, criptografar os dados sensíveis e configurar o *resourceserver* (debatido no item 2.5.3); *Spring Security OAuth2 Authorization Server*, utilizado para configurar o servidor de credenciais; e*Spring Boot*, encarregado de orquestrar os demais frameworks e abstrair (vide o termo “abstração” no item 2.4.1) a configuração dessas ferramentas;
* **H2**: Sistema de gerenciamento de banco de dados usado em aplicações em fase de desenvolvimento e teste. A escolha em utilizar esta ferramenta foi induzida pela economia de recursos no ambiente de desenvolvimento. É necessário mencionar que o H2 não é recomendável para ambientes de produção;
* **Typescript**: Linguagem de programação multi-paradigma. Mais precisamente, é uma extensão da linguagem Javascript, oferecendo suporte a características que auxiliam na construção de aplicações empresariais, como tipagem, característica que evita falhas indetectáveis em tempo de execução;
* **Angular**: *Framework* de aplicações *frontend*, baseia-se na arquitetura MVVM (item 2.4.9). Popular em aplicações empresariais, o Angular é construído usando a linguagem Typescript. Esse framework fornece um padrão estrutural e funcionalidades, como o *data binding*; validação de dados; renderização; roteamento; compartilhamento de dados entre componentes, dentre outras;
* **Ionic**: Conjunto de recursos visuais reutilizáveis e responsáveis que se comportam uniformemente entre dispositivos diferentes, como: celulares, computadores e tablets. Além de prover recursos visuais, o Ionic engloba recursos que possibilitam tornar uma aplicação web em uma aplicação nativa no formato de aplicativo para celulares ou desktops utilizando a mesma base de código;
* **JSZIP**: Biblioteca Javascript que fornece funcionalidades e estruturas de dados que manipulam arquivos compactados sem precisar salvá-los no sistema de arquivos do sistema operacional que hospeda a aplicação;
* **Ng2Charts**: Biblioteca que contém funcionalidades para a representação gráfica de dados estatísticos.

# 3 APRESENTAÇÃO DO PROBLEMA DE PESQUISA

O presente capítulo visa ampliar o entendimento na contextualização do problema, descrevendo a motivação do projeto. Para que fosse possível compreender a motivação, houve necessidade de coletar informações através de sucessivas conversas informais e um breve convívio com o Inova Macaé, junto do acesso a informações dos projetos da equipe e a gravação de uma reunião com os colaboradores responsáveis pelo atendimento. Isto posto, resultou em uma análise informal da motivação, descrita a seguir.

## 3.1 Motivação

De acordo com Sant’ana (2009), a utilização das TICs como ferramentas de suporte aproxima a administração pública da população, pois a prestação de serviços públicos demanda comunicação. Ademais, hoje as ferramentas computacionais superam os processos de atendimento convencionais, seja em acessibilidade, volume de informações ou custo e tempo de resposta.

Conforme o contexto apresentado, sobre a iniciativa do município de Macaé na inauguração Laboratório de Inovação em Gestão Pública (INOVA MACAÉ), que culminou na construção do aplicativo Macaé App, diversos segmentos de serviço foram portados para o meio digital, seja pela interligação com sistemas próprios da prefeitura ou com sistemas de terceiros. Dessa forma, o Macaé Fiscaliza, também construído pelo laboratório de Inovação da cidade, é um exemplo de sistema interligado ao Macaé App, tendo como propósito fornecer um meio de comunicação interativo entre os agentes públicos (colaboradores) e cidadãos.

Ainda assim, os representantes do Inova Macaé não se satisfazem com o sistema porque o Macaé Fiscaliza consiste em uma mescla de ferramentas proprietárias, que são o Fabapp (Fábrica de Aplicativos S/A) e a plataforma de serviços conhecida como Jotform. Quanto à estrutura do sistema, há um ofuscamento provocado por ambas as ferramentas, as quais obrigam uma adaptação do fluxo de trabalho das equipes e removem a autonomia da prefeitura sobre o sistema. Mais um motivo é que a base do Macaé Fiscaliza depende de softwares não gratuitos.

Em vista do apresentado, faz-se imprescindível desenvolver uma alternativa utilizando ferramentas de desenvolvimento de software por meio de código para garantir que o sistema seja de total propriedade da prefeitura. Bem como permitir total liberdade na manipulação da estrutura do sistema para que se adapte às regras de negócio do município.

É relevante a reclamação dos funcionários pela falta de integração do atendimento com o processo de encaminhamento de feedback para a equipe Inova Macaé. Atualmente o sistema encaminha a solicitação do cidadão para a caixa de e-mail da secretaria responsável cadastrada na plataforma Jotform e todo o fluxo do atendimento prossegue através de e-mails:resposta ao usuário, encaminhamento para outras secretarias, entre outros. Após o atendimento, o funcionário necessita reportar os dados da ocorrência para o Laboratório de Inovação. Este, por sua vez, repassa o feedback em formato de dados estatísticos para a secretaria de planejamento do município.

Torna-se nítida a necessidade de automatização no encaminhamento de dados para o Inova Macaé após o atendimento. Essa deficiência pode ser uma das causas de existir uma discrepância entre a quantidade de ocorrências e a frequência de atendimentos, já que em seus primeiros três meses de lançamento, dentre um total de 175 ocorrências, menos de 50 foram atendidas. Esse cenário implica dificuldades em acompanhamento das ocorrências e na rastreabilidade dos funcionários que respondem as ocorrências.

Dessa forma, o sistema deve apresentar interfaces intuitivas aos usuários, viabilizando melhorias em seu fluxo de trabalho. Além de cumprir com seu propósito, que é atuar na entrega de solicitações de serviço através dos cidadãos; no recebimento de solicitações pelos colaboradores; no registro das solicitações e suas tratativas; no fornecimento de indicadores para os gestores; dentre as demais funcionalidades, as quais serão especificadas posteriormente. Assim, um benefício de adquirir um sistema próprio é a possibilidade de reaproveitá-lo em outras áreas que se relacionarem com o processo de zeladoria urbana, poupando recursos na produção de outro sistema ou artifício que necessite fornecer as mesmas funcionalidades.

# 4 PROCESSO DE CONSTRUÇÃO DO SISTEMA DE ZELADORIA URBANA

Antes de iniciar o desenvolvimento, foi necessário elaborar um planejamento dos requisitos com o auxílio dos conceitos de *personas*, cenários e *user stories*; também foi necessário projetar as arquiteturas que basearam o sistema. É importante ressaltar que os conceitos de metodologia ágil discutidos a seguir foram utilizados como método de resolução do problema.

## 4.1 Definição das *personas*

Com base no problema e motivação apresentados no capítulo anterior, o primeiro passo necessário consiste em identificar as *personas* (discutidas no item 2.1):

* Cidadão: indivíduo que mora, trabalha ou estuda no município de Macaé. Pode ser jovem, adulto ou idoso; possui acesso à internet e não se restringe a gênero ou classe social. Basicamente, o cidadão tem o poder de fiscalizar os serviços, infraestrutura e demais ocorrências que podem ser intermediadas pela esfera municipal. É inevitável um cidadão usufruir de um serviço público, assim, o sistema proposto será um meio de certificar a qualidade dos serviços da prefeitura;
* Colaborador: funcionário do município resignado a uma secretaria; que não apresenta restrição de idade ou gênero; possui conhecimento técnico no setor que atua. É pré-disposto a utilizar um sistema que melhora a sua qualidade de trabalho;
* Administrador: integrantes do Inova Macaé ou indivíduos que exercem a *persona* de colaborador, responsáveis por gerenciar a secretaria que atuam. Sua motivação para utilizar o sistema é a mesma do colaborador.

## 4.2 Descrição dos cenários

É necessário descrever os cenários (definidos no item 2.1) para cada *persona*. Sommerville (2021) sugere que os cenários podem deixar de ter descrições breves para descrições longas quando utilizadas nas fases de criação do projeto, portanto, a seguir serão descritos cenários para cada *persona*:

* Cenário do cidadão: devido ao seu direito de fiscalizar, o cidadão pode acessar o sistema através de um aplicativo de celular e, em seguida, selecionar a categoria de serviço que julga ser referente à ocorrência. Após, preenche o formulário com os dados da ocorrência, sendo possível enviar fotos do incidente. O endereço da ocorrência pode ser preenchido no formulário manualmente ou o cidadão pode compartilhar sua posição caso esteja no mesmo local da situação. O cidadão pode solicitar um serviço criando uma conta no sistema ou de forma anônima, porém clientes anônimos não são capazes de acompanhar o andamento de seus chamados. Também é possível que ele acesse o sistema através de um computador;
* Cenário do colaborador: precisa se cadastrar para acessar o sistema. Ao ser autenticado é apresentada uma lista de ocorrências abertas em seu setor. Ainda, ao visualizar uma ocorrência, ele pode responder o formulário que gera um atendimento; em caso de trote, o colaborador pode rejeitar; em caso de falha na interpretação da ocorrência, pode-se encaminhá-la para o setor correto. Ao visualizar uma ocorrência, é possível verificar seu histórico de atendimentos caso um serviço seja realizado por mais de um setor ou de uma ocorrência encaminhada. O colaborador é capaz de navegar entre listas de ocorrências abertas ou atendidas, além das listas de atendimentos realizados pelo setor.
* Cenário do Administrador: responsável pelo gerenciamento de contas dos colaboradores, também necessita de cadastro para acessar o sistema. É capaz de visualizar relatórios, ocorrências e gerenciar contas de outros colaboradores. Seu escopo é restrito apenas ao setor quando vinculado a um. Caso o administrador seja integrante da equipe Inova Macaé, não possui restrição de escopo.

## 4.3 Levantamento de requisitos

O próximo passo é elaborar os requisitos, os quais, neste caso, foram estruturados em *user stories*. Complementando as definições dadas no item 2.1, Sommerville (2021) explica que uma *user story* deve ser estruturada na forma de solicitação que uma *persona* faria caso fosse questionada, conforme a lista a seguir:

* + “Como cidadão, gostaria de ser capaz de criar ocorrências de forma anônima, sem necessidade de criar ou vincular contas”;
  + “Como colaborador, preciso ter uma lista de ocorrências designadas ao meu setor”;
  + “Como colaborador, preciso que a ocorrência tenha descrição, data, local, status, possíveis imagens, o histórico de atendimentos realizados sobre tal ocorrência e o *feedback* do cidadão quando autenticado”;
  + “Como cidadão, gostaria de poder criar conta no sistema para poder acompanhar o andamento de minhas ocorrências”;
  + “Como colaborador, devo prover respostas às ocorrências criadas pelo cidadão”;
  + “Como cidadão, gostaria de poder enviar imagens da ocorrência para auxiliar em seu detalhamento”;
  + “Como cidadão autenticado, gostaria de poder acompanhar o andamento de minhas ocorrências”;
  + “Como cidadão, gostaria de acessar o sistema em dispositivos móveis”;
  + “Como cidadão, quero ter a possibilidade de compartilhar minha localização usando o GPS caso me encontre presente no local da ocorrência”;
  + “Como cidadão, gostaria de poder criar conta através de meus e-mailsou redes sociais em casos que necessitem cadastro para poder acompanhar minhas solicitações”;
  + “Como cidadão autenticado, desejo ver detalhes de cada item em meu histórico de ocorrências”;
  + “Como cidadão autenticado, preciso responder aos atendimentos se minha ocorrência foi resolvida ou não”;
  + “Como colaborador, preciso encaminhar ocorrências para seus respectivos setores quando mal interpretadas pelo cidadão”;
  + “Como colaborador, preciso ter um histórico de atendimentos pelo meu setor”;
  + “Como colaborador, preciso que as ocorrências sejam separadas por listas (abertas, avaliadas, não avaliadas e indeferidas)”;
  + “Como colaborador, preciso ser capaz de fechar ocorrências indevidas ou trotes”;
  + “Como colaborador, ao acessar o atendimento, preciso de informações, como a descrição, data, o responsável e seu devido setor, além do feedback do cidadão, se houver”;
  + “Como colaborador, preciso ser capaz de alterar a senha da minha conta”;
  + “Como administrador, preciso visualizar relatórios de ocorrências atendidas e não atendidas por setor em determinado intervalo de tempo”;
  + “Como administrador, preciso cadastrar secretarias”;
  + “Como administrador, preciso cadastrar colaboradores”;
  + “Como administrador, preciso alterar a secretaria do colaborador em caso de remanejamento”;
  + “Como administrador, preciso recuperar senhas de colaboradores”;
  + “Como administrador, preciso ativar/desativar as contas dos colaboradores”.

A lista acima foi ordenada de acordo com a prioridade de cada requisito para que esteja em conformidade com o *product backlog* e não seja necessário repeti-la.

## 4.4 Definição da arquitetura

Devido à complexidade e à exigência de interfaces específicas para os usuários, o projeto aplica as arquiteturas cliente-servidor (item 2.4.3) e *multi-tier* (item 2.4.2), segregando o sistema em duas aplicações *frontend* e duas aplicações *backend*. O protocolo de comunicação será o HTTP e o protocolo *OAuth*2.1 será adotado.

O ideal é que cada aplicação seja executada em um computador diferente, todavia todas foram testadas em um único computador, o qual foi o ambiente de desenvolvimento. Ainda, devido a sua capacidade de execução paralela das aplicações, o sistema não pôde ser repartido em uma quantidade maior de aplicações.

Dentre as aplicações *frontend*, uma é designada aos colaboradores e outra, aos cidadãos. Essa divisão foi necessária visto que a segunda aplicação pode ser instalada no dispositivo do cidadão e, por consequência, é possível obter acesso à sua lógica, por isso foi necessário isolar as respectivas lógicas de cada aplicação-cliente.

Levando em consideração que para garantir acessibilidade para o cidadão a interface do Inova Macaédeve ser acessível tanto em desktops quanto em aplicativos móveis, foi escolhida a ferramenta Ionic na construção do *frontend* respectivo ao cidadão. Para que não houvesse confusões por trocas de ambiente de desenvolvimento, o Ionic também foi usado no *frontend* dos colaboradores devido ao desenvolvimento em paralelo de ambas as aplicações de interface.

Quanto ao *backend*, uma das aplicações é dedicada ao processo de autenticação (*authorization server*) e a outra centraliza a lógica do negócio(*resource server*). Por causa das restrições no ambiente de desenvolvimento, as aplicações *backend* compartilham do mesmo banco de dados e, além da lógica de negócio, o *resource server* também é responsável pela realização do cadastro de cidadãos e colaboradores.

O servidor de autenticação é construído utilizando o*spring boot* como base e o *spring data* para as consultas ao banco de dados. Já o *spring security oauth*2 *authorization server*é usado para configurá-lo como emissor de *tokens* de acesso. Esta aplicação se comporta como um *webservice*, o qual pode ser aproveitado em futuros projetos web como meio de autenticação tanto de colaboradores quanto de cidadãos.

É importante evidenciar que a autenticação de cidadãos e colaboradores foi centralizada em uma única aplicação por restrições de desenvolvimento,uma vez que o ideal seria designar uma aplicação para cada tipo de autenticação. Manter ambos os tipos de autenticação na mesma aplicação gera um custo menor, porém uma escalabilidade menor em comparação com a decisão de segregá-las.

Por sua vez, as ferramentas de construção do *resource server* são o *spring boot* como base, o *spring data* para gerenciamento do banco de dados e o *spring security oauth*2 *resource server* para o controle de acesso aos recursos. Na possibilidade de reutilização por aplicações futuras, o *resource server* deve apresentar comportamentos baseados em REST API.

## 4.5 Definição do fluxo de desenvolvimento

A respeito do ciclo de desenvolvimento de software, não houve rigor em adotar um processo específico devido à escassez de recursos humanos, que consistiu apenas de um único individuo responsável por uma quantidade elevada de atividades, como: levantamento de dados; estruturação do problema; levantamento de requisitos; elaboração da arquitetura; esboço de interfaces; modelagem dos dados; e a construção do software (*frontend* e *backend*).

Nesse sentido, o propósito de entregar o máximo de funcionalidades possíveis se sobrepôs ao propósito de seguir algum processo de ciclo de desenvolvimento de software, como o *SCRUM* ou *RUP*. Para tanto, foi aplicado um ciclo de desenvolvimento tão simples quanto possível com aspectos interativos (caracterizado pelo acompanhamento ativo dos *stakeholders*) e incrementais (caracterizado como um processo evolutivo em que a cada entrega um conjunto de funcionalidades é adicionado).

Contudo, mesmo que o processo não se submeta a formalidades, foram adotados alguns aspectos da metodologia *SCRUM*, que são: *sprint*; *product backlog* e *sprint backlog*. Primeiramente o *SCRUM* estabelece um fluxo de desenvolvimento cíclico e interativo da mesma forma que o processo adotado pelo presente projeto.

No *SCRUM*,cada ciclo do fluxo de desenvolvimento é denominado como *sprint*. Segundo Sbrocco e Macedo (2012), o *product backlog* define-se como uma lista de requisitos ordenados com base na prioridade de construção, enquanto isso, o *spring backlog* “representa todas as tarefas que devem ser desenvolvidas durante um sprint ou iteração” (p. 169).

A lista de funcionalidades exigidas está de acordo com o item 4.3, em formato de um *product backlog*. Ademais, seus elementos foram construídos ao longo de quatro incrementos (*sprints*), os quais serão explorados a seguir.

### 4.5.1 *Sprint* 1

Por motivos de aprendizagem, neste *sprint* primeiramente as bases das aplicações *backend* foram construídas. Quanto aos requisitos selecionados para este *sprint*, a escolha se deu por conveniência em desenvolvimento, apesar de não estarem no topo do *product backlog*.Afinal, o manifesto ágil em sua essência prioriza resultados sobre conformidade com regras. Sendo assim, foi selecionada uma *userstory* para a entrega, sendo esta: “Como cidadão, gostaria de poder criar conta no sistema para poder acompanhar o andamento de minhas ocorrências”, que se reparte nas atividades a seguir:

* + Criar aplicação *backend* da lógica de negócio para que sirva de base para as próximas funcionalidades (figura 3);
  + Criar aplicação *backend* de autenticação para limitar o acesso aos recursos protegidos (figura 2);
  + Criar a estrutura de dados que representa o cidadão (modelar cidadão);
  + Configurar a representação de dados (DAO – *Data Access Object*) do cidadão para seu compartilhamento entre aplicações;
  + Criar lógica de cadastro do cidadão;
  + Criar tela de cadastro do cidadão;
  + Criar lógica de autenticação do cidadão;
  + Criar tela de autenticação do cidadão.

No aspecto da lógica de negócio, o trabalho concluído reflete-se no modelo de classes, ilustrado pela figura 1.

Figura Modelo de classes



Fonte: Autoria própria (2022)

Na figura acima, “Conta” pode ser explicada como uma representação genérica cujo atributo “autoridades” é obrigatório para qualquer tipo de conta, portanto, “ContaCidadao” apresenta atributos de e-mail, senha e autoridades. O diagrama também expõe a representação de “Cidadão”, que é composta por uma única conta “ContaCidadao” e o atributo “nome”. Em caso de exclusão da classe “Cidadao” do sistema, “ContaCidadao” também é excluída.

Nessa *sprint*primeiramente foi construída a estrutura da aplicação referente ao servidor de credenciais. Para todo e qualquer nome de arquivo ou trecho de código apresentado foi adotada a língua inglesa como convenção. Possíveis omissões ou abreviações podem ocorrer nos códigos apresentados em qualquer *sprint*, com o intuito de evitar perdas de foco.

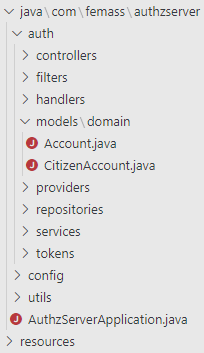
A figura 2 contempla a organização das pastas criadas com os códigos configurados para o fornecimento de credenciais. Assim, vale destacar que havia o desejo de obter maior controle nos processos de autenticação e por esse motivo, além da intenção didática, foi necessário personalizar uma quantidade considerável de arquivos.

A aplicação foi nomeada como *AuthzServer* (uma espécie de acrônimo de *Authorization Server*), que apresenta 3 pastas principais: “*auth*”, contém toda a lógica de autenticação; “config”, armazena as configurações; e “utils”, possui códigos utilitários.Abaixo da hierarquia da pasta “*auth*” localizam-se as pastas:

* “*controllers*”, cujo processamento é acionado quando sua respectiva rota é interceptada;
* “*filters*” envolve programas personalizados que filtram a interceptação das rotas e assim certificam que os *controllers* sejam acionados em segurança, pois a “espinha dorsal” do *Spring Security* é uma cadeia de filtros configurável;
* “*handlers*”, por sua vez, é local para códigos de manipulação, os quais, no caso do projeto, manipulam o direcionamento da página de *login* de acordo com o tipo de usuário;
* “*models*”, que possui a subpasta “*domain*”, contendo os arquivos “*Account*.java”, relativo à classe “Conta”, e “*CitizenAccount*.java”, referente à classe “ContaCidadao”, conforme a figura 1;
* “*providers*”, que abriga a lógica responsável por conferir as credenciais e decidir se deve ou não autenticar o usuário;
* “*repositories*”, possui códigos relacionados especificamente à busca no banco de dados de acordo com o critério de pesquisa, quando existir;
* “*services*”, cujo conteúdo serve de interface para que outras camadas não acessem diretamente os arquivos da pasta “*repositories*”;
* “*tokens*” é a pasta relativa às estruturas de dados que armazenam as credenciais para serem transportadas pela cadeia de filtros do *Spring Security*.

É curioso perceber que, ao descrever a estrutura de subpastas do diretório “*auth*”, trata-se de uma aplicação *multi-layer* (item 2.4.2). Ainda, no mesmo nível de hierarquia da pasta *authzserver*,existe a pasta *resources*, gerada pelo *Spring* por padrão, que guarda conteúdos estáticos, como páginas HTML (*HyperText Markup Language*), código CSS (*Cascade Style Sheet*), imagens e alguns dados constantes que não precisam ser explícitos no código fonte. Todavia, há um receio de que, ao se aprofundar sobre o conteúdo dessas pastas do servidor de autenticação, o escopo do tema em questão seja extrapolado, por isso, este não será explorado.

Figura Estrutura de pastas do servidor de credenciais



Fonte: Autoria própria (2022)

O arquivo “*AuthzServerApplication*.java”, localizado no mesmo nível de hierarquia de “*auth*”, “config” e “utils”, é responsável por iniciar a aplicação. A figura 2 retrata o estado do A*uthzServer*no primeiro *sprint*, porém a imagem oculta seus arquivos por não parecer necessário explorá-los no contexto desta monografia.

Na sequência foi iniciada a aplicação responsável pela lógica do negócio, batizada de *ResourceServer*, e suas estrutura é exibida na figura 3, que apresenta uma estrutura semelhante ao *AuthzServer*. Ainda assim, o *ResourceServer* se distingue por apresentar a pasta “init”, responsável por povoar o banco de dados quando vazio. Os elementos desta pasta são primariamente usados para teste. Apenas em casos em que se deseja povoar com dados que raramente são alterados e em grande quantidade, como a situação dos bairros, a sua aplicação é fundamental. Outra pasta importante é a “dto”, a qual, sumariamente, contém lógica para a conversão de dados do *backend* para o *frontend* e vice-versa. Vale pontuar aqui que DTO é a sigla para *Data Transfer Protocol*.

Outra diferença é que o *AuthServer*utiliza as classes de repositório e domínio apenas para a consulta de dados, delegando a responsabilidade de inserção, atualização e remoção ao *ResourceServer*. Dentre os dados de domínio, apenas as classes “Conta” e suas especializações são preocupações do servidor de autorização.

O ideal seria destinar uma nova aplicação dedicada ao gerenciamento de conta, como cadastro e alterações, de modo a remover tais responsabilidades do *ResourceServer*. Contudo, a máquina usada no desenvolvimento não suporta a adição de uma terceira aplicação *backend* executando em paralelo com uma aplicação *frontend*. Consonante a esse fato, a complexidade do gerenciamento de aplicações pode aumentar ligeiramente.

Figura Estrutura de pastas do servidor de recursos



Fonte: Autoria própria (2022)

O termo “*CitizenAccount*” refere-se à classe “ContaCidadao”, enquanto “Citizen” diz respeito apenas a “Cidadao”. Adiante, é exibido o resultado do código da classe de configuração do servidor de recursos (figura 5). A classe “*ResourceServerConfig*” apresenta o atributo “*corsAllowedOrigins*”, que funciona como uma espécie de “lista branca”, que contém os endereços das aplicações que possuem permissão para requisitar um recurso. Na linha 35 da figura 5 a anotação “@*Value*” aponta os verdadeiros dados rotulados pelo indicador “*cors.allowed-origins*” (linha 2 da figura 5) em um arquivo estático da pasta “*resources*”. O arquivo que armazena dados evitando que estes sejam expostos no código fonte é chamado de “*application*.yml” (figura 4).

Figura Dados estáticos no arquivo *application.yml*



Fonte: Autoria própria (2022)

Outro atributo “*jwkSetUri*” (figura 5, linha 39) armazena o URL das credenciais do *AuthzServer*(originalmente armazenado na linha 12 da figura 4), para que seja possível verificar se o *token* utilizado pelo cliente foi assinado pelo servidor de credenciais. Já o método “securityFilterChain” configura as regras de acesso às rotas. Caminhos iniciados por “/*registration*”, “/*anonymous/*” e “/h2” são livres para todos interceptarem, enquanto apenas aqueles com credenciais “USER” podem interceptar rotas iniciadas com “/user/”.

De acordo com a figura 5, qualquer rota não iniciada pelos padrões descritos é interceptada apenas por usuários autenticados independente da posse de credenciais. Então, a partir do intervalo entre a linha 54 até a linha 60, fica explícito quequalquer caso de exceção (erro de execução) será retornado com o status Http 401 “não autorizado”. Logo abaixo, seguem as configurações do *ResourceServer* e JWT.

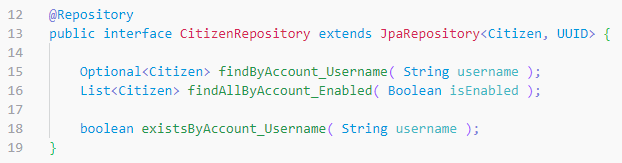
Figura Código do arquivo *ResourceServerConfig*.java



Fonte: Autoria própria (2022)

Alusivo ao conteúdo das classes “Cidadao” (*Citizen*) e “ContaCidadao” (*AccountCitizen*), ocorreu uma reestruturação drástica em *sprints* posteriores que não merece ser mencionada. Não obstante, há quatro arquivos que merecem menção: “*CitizenRepository*.java” (figura 6); “*CitizenService*.java” (figura 7); “*CitizenDTO*.java” (figura 8) e “*CitizenController*.java” (figura 9).

Figura Conteúdo do arquivo *CitizenRepository*.java



Fonte: Autoria própria (2022)

*CitizenRepository* é uma interface que herda funcionalidades da classe *JpaRepository*,a qual pertence ao *Spring Data*. A linha 12 sinaliza ao *framework* que o conteúdo é um componente do tipo *Repository*gerenciável pelo *Spring*.

A nomenclatura dos métodos é transformada pelo *Spring Data*em comandos inteligíveis pelo banco de dados configurado. Termos como “*findBy*” e “*existsBy*” são traduzidos em comandos como “*select*... *from* ... *where*...” e “*select*... *from* ... *whereexists*” por exemplo. Ainda, o uso do caractere “\_” (sublinhado) denota a hierarquia do parâmetro de pesquisa. Na linha 15 da figura 6 o parâmetro de pesquisa é o atributo “*username*” da estrutura “*Account*”.

Figura Conteúdo do arquivo *CitizenService*.java



Fonte: Autoria própria (2022)

Na linha 16 da figura 7, a anotação “@*Service*” aponta que a classe “*CitizenService*” é um componente do tipo *service* gerenciável pelo *framework.* A anotação “@*Autowired*” designa ao *Spring*a tarefa de instanciar o dado que a mesma anotação precede. Dentre os métodos de *CitizenService*, podemos citar:

* Na linha 27 da figura 7, o método “*createOrUpdate*” cria ou edita uma entidade do tipo “*Citizen*”(classe “Cidadao”), retornando um valor verdadeiro em caso de sucesso e falso quando houver falhas;
* Na linha 40 da figura 7, o método “*createMultiple*”, que pode criar ou editar uma lista de entidades do tipo “*Citizen*”, retornando um valor booleano que sinaliza a ocorrência de falhas ou sucesso;
* A seguir, o método “*delete*”, que remove a entidade “*Citizen*” e retorna um *RuntimeException* em caso de falhas;
* Os demais métodos da figura 7, como “*findByUsername*”, “*existsByUsername*” e “*countUsers*”, utilizam das funcionalidades dos atributos “*citizenRepository*” e “*accountRepository*” para evitar que componentes de outras camadas acessem os repositórios diretamente.

Figura Conteúdo do arquivo *CitizenDTO*.java



Fonte: Autoria própria (2022)

Na figura 8, as anotações da linha 22 são artifícios da biblioteca *Lombok* para evitar a verbosidade dos métodos *getters* e *setters*. *CitizenDTO* é uma estrutura utilizada para transportar dados do cidadão entre o servidor e a aplicação cliente. Dentre os atributos da classe, apenas o “*email*” é obrigatório e não pode ser nulo.

Na linha 25 da figura 8, atributos vazios são ignorados, permitindo flexibilidade sobre quais dados não obrigatórios transportar. Enquanto na linha 28 ordena-se que os atributos “id”, “*password*” e “*active*” sejam ocultados quando enviados do servidor para o cliente. Os métodos “*serialize*”, que converte uma estrutura de dados interna ao *backend* em uma estrutura de dados externa, e “*deserialize*” são públicos e estáticos, possibilitando que outros módulos os utilizem quando for necessário transportar dados do cidadão.

Figura Conteúdo do arquivo *CitizenController*.java



Fonte: Autoria própria (2022)

Na figura 9, as linhas 29 a 32 indicam o *mime-type* (conferir item 2.4.6) como JSON, ou seja, tanto na entrada quanto na saída os dados são representados em notação de objeto *Javascript*. Dentre os atributos instanciáveis pelo *framework* estão: “*citizenService*” e “*passwordEncoder*”, usado para criptografar senhas, além de verificar paridade entre senhas armazenadas e as fornecidas pelo usuário. Na linha 41 a rota “/*registration-user*” foi exposta para o cadastro de cidadãos. Essa rota obedece ao padrão estabelecido pela linha 44 da figura 5, em que se localiza a configuração do servidor de recursos. Ainda, como pode ser observado, rotas iniciadas em “/*registration*” podem ser interceptadas por qualquer usuário, seja autenticado ou anônimo.

É importante informar que a lógica aplicada aos arquivos referentes à estrutura *Citizen* é padronizada para as demais estruturas do *backend*, respeitando a camada lógica. Ou seja, arquivos da camada *repository* apresentam o mesmo padrão, bem como os arquivos da camada *controller*, e assim sucessivamente. Quanto às interfaces entregues neste ciclo, as telas de cadastro e *login* serão apresentadas a seguir:

Figura Protótipo da tela de cadastro utilizando o Figma



Fonte: Autoria própria (2022)

O protótipo acima é uma interface de acesso rápido para o cidadão. Ao lado esquerdo são apresentadas as formas de solicitação de serviço responsáveis por direcionar para suas respectivas telas. Por sua vez, ao lado direito localiza-se o formulário de cadastro. A tela de acesso do cidadão é exibida a seguir, através da figura 11, e é autoexplicativa.

Figura Tela de acesso do cidadão



Fonte: Autoria própria (2022)

### 4.5.2 *Sprint* 2

Foram selecionados alguns *user stories* e épicos para serem entregues neste *sprint*. O primeiro é o seguinte: “Como cidadão, gostaria de ser capaz de criar ocorrências de forma anônima, sem necessidade de criar ou vincular contas”. Os passos para completar esta funcionalidade foram:

* Modelar ocorrência;
* Modelar endereço da ocorrência;
* Modelar secretaria destinada;
* Modelar categoria de serviço;
* Criar função para publicar ocorrências;
* Criar função para listar as secretarias disponíveis;
* Criar lista de seleção de serviços;
* Criar formulário de ocorrências.

O segundo *user story* é: “Como colaborador, preciso que a ocorrência tenha descrição, data, local, status, possíveis imagens, o histórico de atendimentos realizados sobre tal ocorrência e o *feedback* do cidadão quando autenticado”. Para cumprir com este épico foi preciso:

* + - Modelar o *feedback*;
    - Criar função para o carregamento de ocorrências referentes à secretaria do colaborador;
    - Criar tela para a listagem de ocorrências;
    - Criar tela de detalhes da ocorrência.

Outro épico é: “Como colaborador, preciso ter uma lista de ocorrências designadas ao meu setor”. Esta funcionalidade foi construída através de:

* + - Criar método de autenticação para o colaborador acessar o sistema;
    - Criar tela de autenticação do colaborador.

O quarto *user story* apresenta: “Como colaborador, devo prover respostas às ocorrências criadas pelo cidadão”. Este foi resolvido através de:

* + - Modelar a estrutura do atendimento;
    - Criar função para gerenciar o armazenamento do atendimento;
    - Criar o formulário de atendimento.

Por último, destaca-se: “Como colaborador, preciso encaminhar ocorrências para seu respectivo setor quando mal interpretadas pelo cidadão”. Além do que foi visto, cabe pontuar que todas essas funcionalidades impactaram em alterações no diagrama de classes a seguir:

Figura Alterações no diagrama de classes



Fonte: Autoria própria (2022)

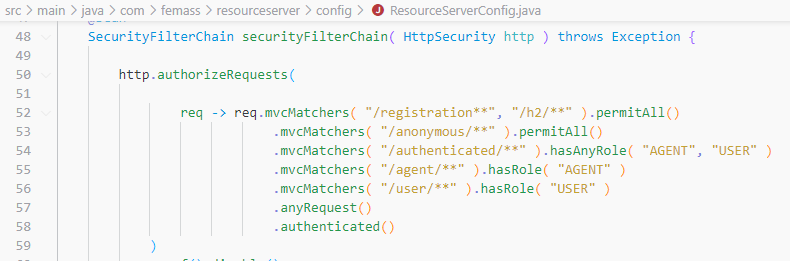
Como é possível observar no diagrama acima, foi adicionado o atributo “habilitada” à classe “Conta”, útil para administradores ao gerenciar contas de colaboradores desligados e para cidadãos que desejarem desativar sua conta. Além disso, para qualquer tipo de conta torna-se obrigatório definir métodos de acesso às credenciais e o *login*, os quais são ilustrados pelos métodos explicitados na estrutura da “Conta”.

Houve a adição da “ContaColaborador”, constituída pelos atributos matrícula e pelas credenciais, que são o “cpf” e a “senha”. No caso da “ContaCidadao”, o atributo “senha” foi renomeado para “credenciais”. Com o acréscimo do “Colaborador”, composto pela “secretaria” e “conta”, ocorreu a generalização dos atributos comuns entre “Cidadao” e “Colaborador”, que passaram a fazer parte de uma estrutura genérica denominada “Usuario”. O atributo “conta” presente em “Usuario” faz referência à estrutura genérica “Conta”, enquanto o mesmo atributo em cada tipo de usuário faz referência ao seu respectivo tipo de estrutura da conta.

Quanto à estrutura da “Ocorrencia”, se tem os atributos: “protocolo”, “endereço”, “descricao”, “imagens”, “autor” (Cidadao), “dataPostagem”, “status”, “atendimentos”, “servico” e “destinatario”. Um cidadão pode criar várias ocorrências e cada uma pode ser composta por vários atendimentos. Por sua vez, o atendimento é descrito pela “dataExecucao”, “descricao”, “ocorrência”, “feedbacks” e “protocolo”. O atendimento é basicamente uma resposta que o colaborador provê à solicitação do cidadão expressa em forma de ocorrência.

Já o “Feedback*”* é uma resposta do cidadão para o Atendimento, discriminando a descrição, o cidadão e o atendimento. O cidadão pode prover vários feedbacks para o mesmo atendimento. Ademais, ocorreu a necessidade de modelar o endereço da ocorrência, cujos atributos são: “latitude”, “longitude”, “cep”, “logradouro”, “bairro” e “referencia”. Os dados referentes à latitude e longitude são fornecidos quando o usuário aceita informar o endereço através do compartilhamento da localização.

Figura Alterações em *ResourceServerConfig*.java



Fonte: Autoria própria (2022)

Neste *sprint* o arquivo de configuração do servidor de recursos (figura 13) teve acréscimo nas linhas 54 e 55, as quais adicionaram regras para rotas iniciadas com “/*authenticated*” e “/*agent*”. A primeira regra prescreve que apenas usuários autenticados (colaboradores e cidadãos) podem interceptá-las, enquanto a segunda pode ser acionada apenas por colaboradores. O termo “*agent*” faz alusão ao agente público ou colaborador.

Figura Conteúdo de *DistrictTableSeeder*.java



Fonte: Autoria própria (2022)

Para que seja possível criar uma ocorrência é necessário indicar a localização. Um de seus itens é o bairro e, devido ao software se aplicar apenas ao município de Macaé, seus dados não estão sujeitos a constantes alterações e podem ser preenchidos automaticamente quando o banco de dados estiver vazio. Ainda, a figura 14 ilustra o código responsável por popular uma lista de bairros no banco de dados. A classe foi batizada de *DistrictTableSeeder* porque “*District*” é uma alusão a bairro.

Figura Conteúdo da classe de domínio *Call*.java



Fonte: Autoria própria (2022)

A figura 15 apresenta a classe “*Call*”, referente à estrutura “Ocorrencia”. As anotações “@*Entity*” e “@*Column*” configuram os nomes da tabela e das colunas respectivamente. No que diz respeito aos atributos, estes se configuram em: “id”; “*protcol*” (“protocolo”); “status”; “*description*” (“descricao”); “*address*” (“endereço”); “*images*” (“imagens”); “author” (“autor”); “*postingDate*” ( “dataPostagem” ) e “*attendances*” (“atendimentos”).

A anotação “@*Embedded*” embute listas de valores em uma única coluna enquanto “@*ManyToOne*” e “@*OneToMany*” tratam das cardinalidades das relações de “Ocorrencia” com as demais estruturas. Assim, ao criar uma ocorrência, o cidadão deve escolher dentre um acervo de serviços, os quais são organizados em categorias para facilitar seu acesso, como pode ser ilustrado pelas figuras 16 e 17. O serviço é organizado em: “descricao” (nome do serviço), “secretaria” e “grupo” (categoria do serviço).

Figura Lista de categorias de serviço



Fonte: Autoria própria (2022)

Ao selecionar uma das categorias conforme a figura 16, a aplicação carrega a lista de serviços respectiva, representada a seguir pela figura 17.

Figura Lista de serviços



Fonte: Autoria própria (2022)

Quando selecionar o serviço, o formulário da ocorrência (figura 18) é exibido:

Figura Formulário da ocorrência



Fonte: Autoria própria (2022)

As anteriores desse *sprint* são referentes à interface do cidadão utilizando o navegador web. Por isso, o formulário de ocorrência não apresenta botões para o compartilhamento de localização.

Além do desenvolvimento da aplicação de interface do cidadão, funcionalidades da aplicação dos colabores também foram entregues. Como é o caso da tela de *login* (figura 19), que apresenta um campo de entrada adicional (campo de CPF) em comparação com o *login* do cidadão.

Figura Acesso do colaborador



Fonte: Autoria própria (2022)

Também foi entregue a lista de ocorrências logo que o colaborador é autenticado, conforme a ilustração a seguir. Dentre os campos exibidos estão o nome do serviço, status, bairro, data de postagem, hora e o nome do autor (quando cadastrado). Pontua-se que os valores na lista são meramente ilustrativos e para propósitos de teste.

Figura Lista de ocorrências



Fonte: Autoria própria (2022)

Caso selecionado um item da lista, o colaborador é direcionado ao formulário da ocorrência, ilustrado em sequência.

Figura Formulário de ocorrência



Fonte: Autoria própria (2022)

Através deste formulário o colaborador é capaz de corrigir o tipo de serviço e o setor destinatário. Assim, ao clicar no botão adicionar resposta, é exibido o formulário de atendimento, vide a figura 22, cujo campo de descrição do atendimento deve ser preenchido antes de finalizar o atendimento, seja em casos de redirecionamento ou de conclusão do serviço.

Figura Formulário de atendimento



Fonte: Autoria própria (2022)

Também é através do formulário de atendimento que o colaborador aponta se a ocorrência é um trote. Este é automaticamente excluído caso confirmado pelo colaborador.

### 4.5.3 *Sprint* 3

Devido a uma falha de interpretação no *sprint* 2 foi necessário reestruturar o projeto ao perceber que não havia necessidade de o cidadão fornecer mais de um feedback por atendimento. Desse modo efetuou-se a eliminação da estrutura “Feedback”; remoção do atributo “feedbacks” na classe “Atendimento”; acréscimo do atributo “comentarioCidadao”, propriedade que passou a assumir a responsabilidade do feedback e basicamente armazena o texto que descreve a avaliação do cidadão.

Também ocorreu uma preocupação em identificar ocorrências atendidas que não foram devidamente resolvidas, assim foi adicionado o status “não resolvida” na classe “Status”. Aproveitando a reestruturação do sistema, outras alterações na estrutura Atendimento foram efetuadas para facilitar a geração de relatórios, sendo estas:

* A criação do atributo “tipoAtendimento”, que distingue um atendimento direcionado ao cidadão com notificações sobre a resolução do problema de um atendimento interno que redireciona a solicitação de uma secretaria para outra. O segundo caso pode acontecer seja por falha de interpretação do criador da ocorrência ou pela necessidade de repassar o atendimento a outro setor para finalizar o serviço;
* Adição do atributo “avaliação”, que retrata a avaliação do cidadão sobre o atendimento. Quando um atendimento é imediatamente criado, sua avaliação assume o valor “não avaliado”. Após o cidadão autenticado receber o atendimento, a avaliação pode ser “positiva” ou “negativa”;
* Inclusão do atributo “setor” em “Atendimento”, relacionando o atendimento diretamente à secretaria. Antes era referenciado através do atributo “destinatário” da classe “Ocorrencia”, mas a mudança da secretaria destinatária da ocorrência provocava efeitos indesejáveis no registro de atendimentos.

A figura a seguir ilustra as alterações aplicadas no diagrama de classes. Pode-se perceber o contraste quando comparada à figura 12.

Figura Diagrama de classes corrigido



Fonte: Autoria própria (2022)

Em paralelo à correção, a seguinte *user story* foi concluída neste *sprint*: “Como cidadão, gostaria de poder enviar imagens da ocorrência para auxiliar em seu detalhamento”. A atividade necessária para cumprir com esta funcionalidade consistiu em criar um serviço destinado à manipulação de arquivos (resultado na figura 24).

Figura Estrutura do arquivo *FileStorageService.*java



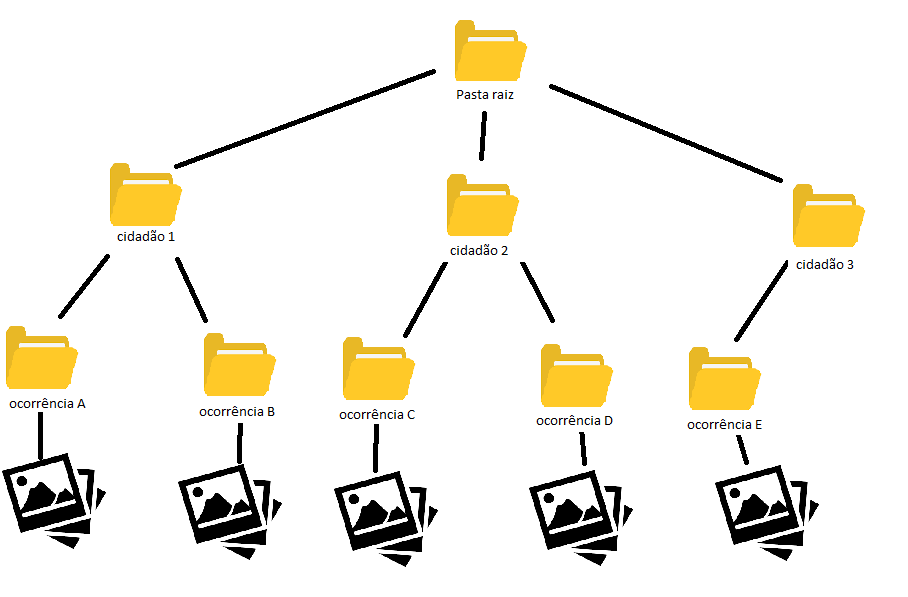
Fonte: Autoria própria (2022)

A figura 24, referente ao *FileStorageService*,é responsável por gerenciar uma pasta de imagens no sistema de arquivos do *ResourceServer*. Para iniciar o serviço precisa do caminho da pasta raiz em que se deseja criar a pasta de imagens.

Os métodos criados são: “*store*”, que necessita do identificador da entidade “Cidadao” e da “Ocorrência”, além do conteúdo do arquivo; “*loadAsResource*” busca um arquivo pelo caminho relativo, contendo o identificador do usuário e o da ocorrência e o nome do arquivo; “*loadAllFilesFromDirectory*” procura todos os arquivos do diretório de imagens da ocorrência após receber o caminho; “delete” remove o arquivo ou pasta informando seu caminho relativo; e “createDir” cria uma nova pasta.

Assim, quando a pasta raiz de imagens existe, cria-se uma pasta para o cidadão, a qual é nomeada pelo respectivo identificador. Dentro da pasta do cidadão são gerados os diretórios das ocorrências, cada uma nomeada pelo respectivo indicador. Por fim, cada pasta de ocorrências armazena suas imagens. A figura 25 retrata a organização dos diretórios de imagem da ocorrência. Vale ressaltar que a nomenclatura nela apresentada é meramente ilustrativa:

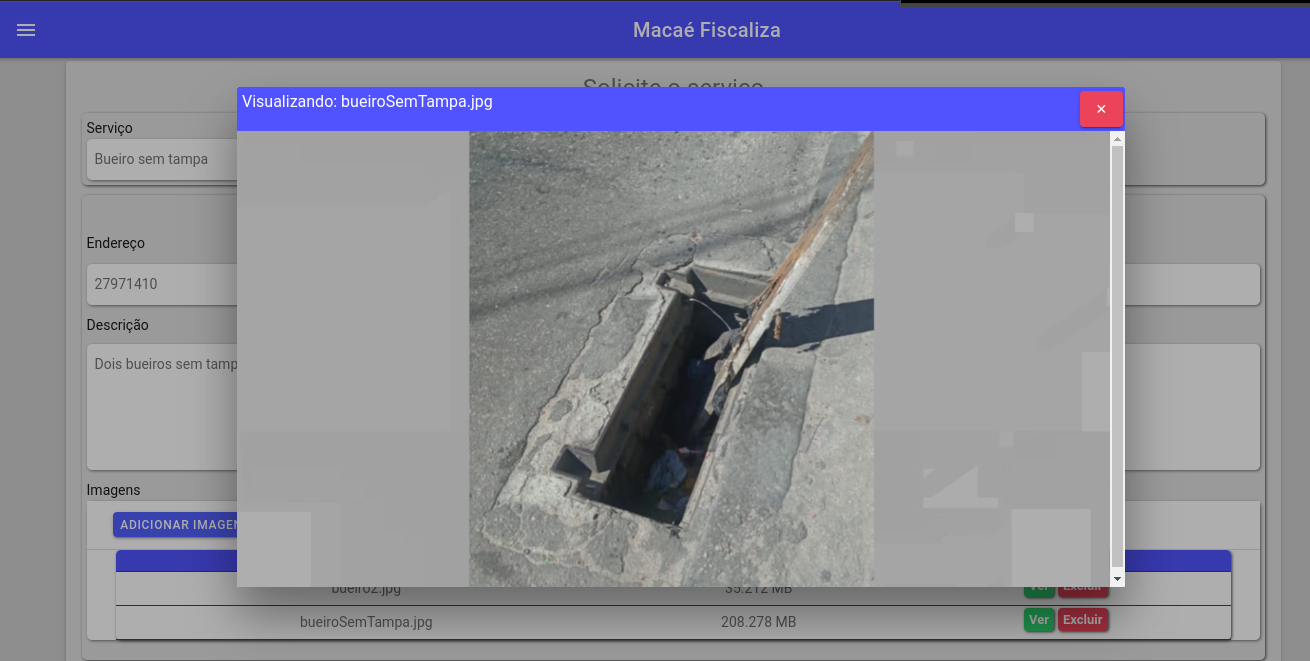
Figura Organização de imagens no sistema de arquivos



Fonte: Autoria própria (2022)

A figura 26, a seguir, exibe o resultado do visualizador de imagens, que cumpre com a exibição de fotos. O cidadão pode enviar até duas imagens e o limite de tamanho para cada uma foi estipulado em até 270 *megabytes*.

Figura Visualizador de imagens construído para formulário de ocorrências



Fonte: Autoria própria (2022)

Para economizar na quantidade de chamadas realizadas ao servidor, o *ResourceServer* envia suas imagens em um arquivo compactado no formato ZIP. Para que as aplicações *frontend* sejam capazes de gerenciar as imagens compactadas, foi necessário adicionar à biblioteca *JSZip* o trecho de código da aplicação *frontend*do cidadão. A figura 27 exibe a utilização desta biblioteca.

Figura Trecho de código *frontend* que gerencia imagens compactadas



Fonte: Autoria própria (2022)

Na linha 47 da figura 27 é realizada uma chamada ao servidor de recursos para obter as imagens da ocorrência usada como parâmetro (“*this*.*call*”). Quando não ocorrem falhas na solicitação ou no processamento do servidor, o bloco de código entre as linhas 49 a 52 é executado. Caso contrário, o bloco da linha 53 é acionado, exibindo uma mensagem de falha ao usuário.

Em seguida, na linha 51 da figura 27 acontece um chamado à função responsável por descompactar as imagens para que possam ser visualizadas no sistema. A partir da linha 60 utiliza-se uma funcionalidade da biblioteca *JSZip*,que efetua a leitura do conteúdo em formato binário (“*blob*”) do arquivo compactado e transforma em uma estrutura de dados inteligível ao código em *Typescript* (objeto denominado “*File*”). Por último, na linha 69, o objeto é adicionado na lista de imagens da ocorrência. Assim, destaca-se a *user story*: “Como cidadão autenticado, gostaria de poder acompanhar o andamento de minhas ocorrências”. O resultado destaé apresentado pela figura 28.

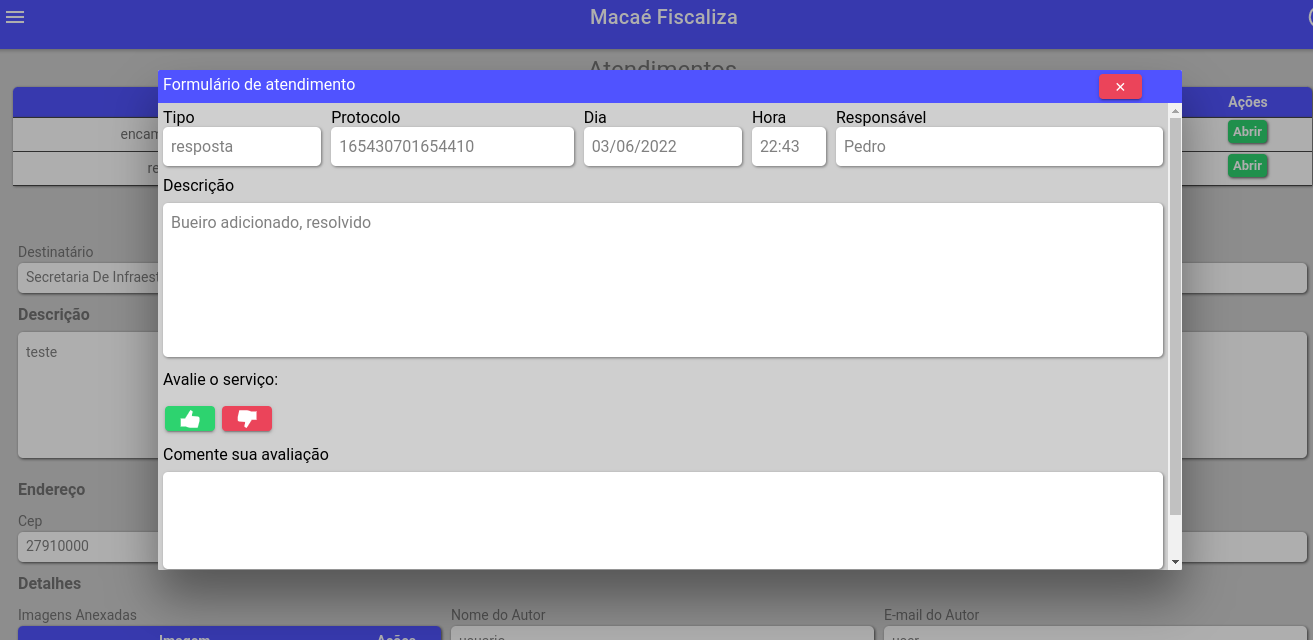
Figura Lista de atendimentos da ocorrência



Fonte: Autoria própria (2022)

Outra *user story* é: “Como cidadão autenticado, desejo ver detalhes de cada item em meu histórico de ocorrências”.O resultado desta é apresentado pela figura 29. Também destaca-se: “Como cidadão autenticado, preciso responder aos atendimentos se minha ocorrência foi resolvida ou não”.O resultado destatambém é apresentado pela figura 29.

Figura Diagrama de classes corrigido



Fonte: Autoria própria (2022)

Outra *user story* é: “Como colaborador, preciso que as ocorrências sejam separadas por listas (abertas, avaliadas, não avaliadas e indeferidas)”. A figura 30 demonstra o código elaborado para enviar a lista de ocorrências conforme os parâmetros da pesquisa:

Figura Método da classe *CallController*.java



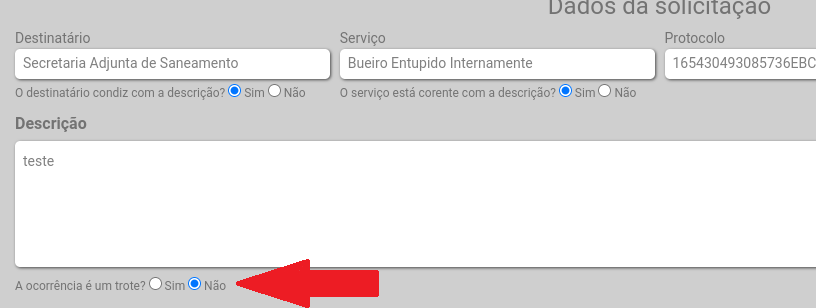
Fonte: Autoria própria (2022)

Na linha 162 da figura 30 é exposta a rota “/*agent*/*calls*/*all*”, que o colaborador acessa através de sua respectiva aplicação *frontend* para obter a listagem de ocorrências. A lista de ocorrências salvas no sistema pode ser filtrada pelo parâmetro “status”, declarado na linha 164, e ordenada de acordo com o atributo especificado pelo parâmetro “*order*” da linha 165.

Na linha 168 da figura 30 a função “extractLoginFromContext” extrai a identidade do colaborador pelo *token*enviado no corpo da requisição. Em seguida, em caso de sucesso entre as linhas 170 e 176, a lista de ocorrências é carregada na variável “*calls*”. Por fim, na linha 178 a função “*prepareResponse*” verifica o conteúdo “*calls*” e, em caso de falha, envia a mensagem da linha 179.

Como é possível observar na figura 30, os parâmetros das linhas 164 e 165 não são obrigatórios e, quando não informados, a lista de todas as ocorrências, ordenada pela data de criação, é repassada ao cliente web. Assim sendo, apresenta-se a *user story*: “Como colaborador, preciso ser capaz de fechar ocorrências indevidas ou trotes”. A resolução desta encontra-se na figura 31.

Figura Recurso para indicação de trote



Fonte: Autoria própria (2022)

A seta em vermelho na figura 31 mostra um indicador em que, ao selecionar a opção “sim”, a ocorrência será rotulada como trote e redirecionada para a listagem de ocorrências indeferidas. Já outra *user story* expõe: “Como colaborador, ao acessar o atendimento, preciso de informações, como a descrição, data, o responsável e seu devido setor, além do feedback do cidadão, se houver”. Esta funcionalidade foi elaborada em forma de formulário de atendimento, conforme sua representação na figura 32.

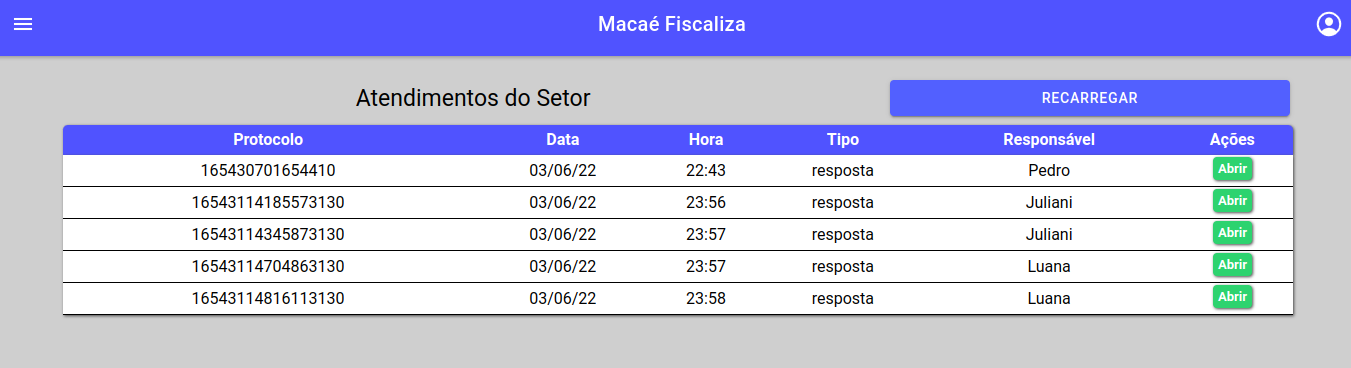
Figura Formulário de atendimento



Fonte: Autoria própria (2022)

Outra *user story* é: “Como colaborador, preciso ter um histórico de atendimentos pelo meu setor”. A resolução desta funcionalidade é exibida na figura 33.

Figura Histórico de atendimentos pelo setor do colaborador



Fonte: Autoria própria (2022)

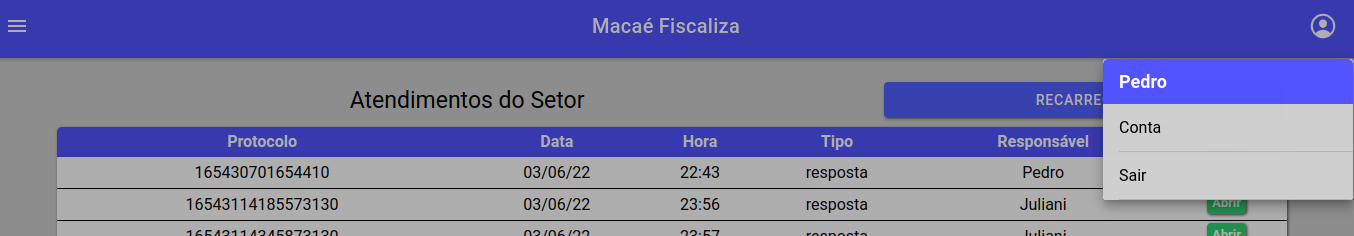
A figura 33 ilustra a tabela de atendimentos do setor. Por causa do acesso simultâneo de vários colaboradores foi necessário acrescentar o botão “recarregar” para atualizar a listagem.

### 4.5.4 *Sprint* 4

Nesta última *sprint* foram construídas funcionalidades referentes à administração do sistema de interface do colaborador. Não houve alterações na estruturação da regra de negócio, por isso o diagrama de classes manteve-se no mesmo estado da quarta entrega. Assim sendo, para completar a funcionalidade “Como colaborador, preciso ser capaz de alterar a senha da minha conta” é necessário:

* + - Criar menu para gerenciamento de conta (resultado na figura 34);
    - Criar formulário de dados da conta do colaborador (resultado na figura 35);
    - Criar formulário de alteração de senha do colaborador (resultado na figura 36).

Figura Opções da conta na aplicação de interface para o colaborador

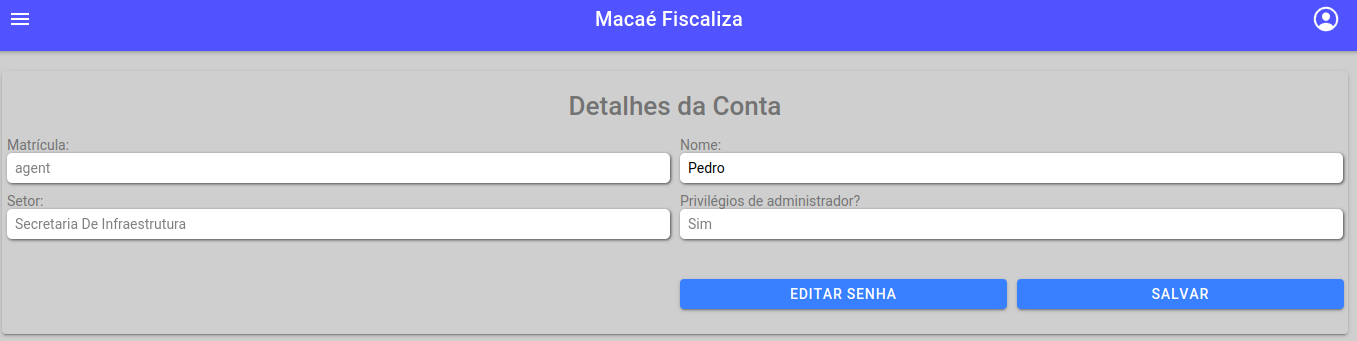


Fonte: Autoria própria (2022)

Como solução, na figura 34, foi criado um menu que é exibido ao clicar no ícone da extrema direita. O menu revela o nome do colaborar que estiver acessando o sistema, a opção “Conta”, onde é capaz de gerenciar seus dados, e a opção “Sair”, para encerrar sua sessão.

Ao selecionar a opção “Conta”, o usuário é direcionado a um formulário conforme a figura 34, em que são descritos a matrícula, o setor, o nome e se o colaborador possui permissão administrativa. Neste formulário o colaborador é capaz apenas de editar seu nome e senha.

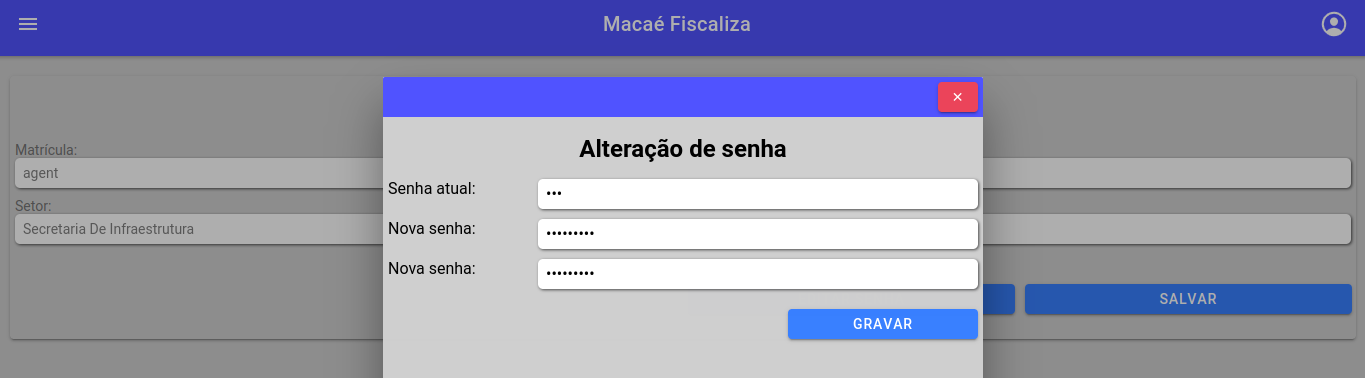
Figura Formulário de dados pessoais do colaborador



Fonte: Autoria própria (2022)

Quando o colaborador clica no botão “Editar Senha” é disparado um formulário flutuante com três campos. O primeiro contém a senha atual, já o segundo e o terceiro possuem a nova senha.

Figura Formulário de alteração de senhas



Fonte: Autoria própria (2022)

Outra *user story* é a seguinte: “Como administrador, preciso visualizar relatórios de ocorrências atendidas e não atendidas por setor em determinado intervalo de tempo”. Essa funcionalidade tem seu resultado apresentado na figura 37.

Figura Relatório gráfico: ocorrências x atendimentos por secretaria



Fonte: Autoria própria (2022)

Para o fornecimento de gráficos foi necessário utilizar a biblioteca *Ng2Charts*, que fornece estruturas de dados para gráficos em formatos distintos, como o “gráfico de pizza” ou o “gráfico de barras”. Além do mais, através da figura 37 percebe-se a presença dos campos de data inicial e data limite para requisitar ao servidor de recursos todas as ocorrências e atendimentos realizados no intervalo de tempo definido. Também é possível selecionar o tipo de relatório ao procurar por outros indicadores.

As estruturas de dados de ocorrência e atendimento armazenam diferentes tipos de informação que viabilizam a montagem de indicadores por bairro, serviço ou tipo de cidadão (autenticados e anônimos). Assim, outra *user story* é: “Como administrador, preciso cadastrar secretarias”, a qual se desmembra nas atividades seguintes:

* + - Criar lista de secretarias (completado conforme a figura 38);
    - Construir formulário de criação de secretaria (feito conforme a figura 39).

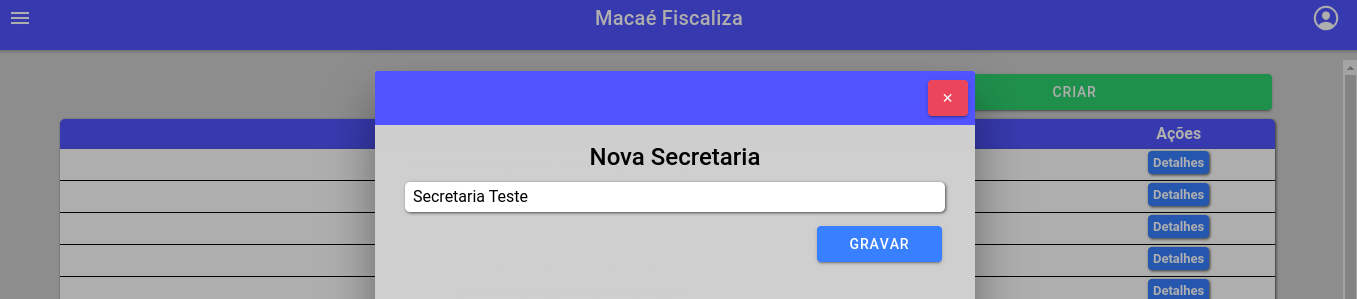
Figura Lista de secretarias



Fonte: Autoria própria (2022)

Pela figura 38 é possível observar uma lista de secretarias com o botão de adição de secretarias acima. Ao dispará-lo, é exibido um formulário simples (figura 39) contendo um único campo designado para informar o nome da nova secretaria.

Figura Formulário de criação de secretaria

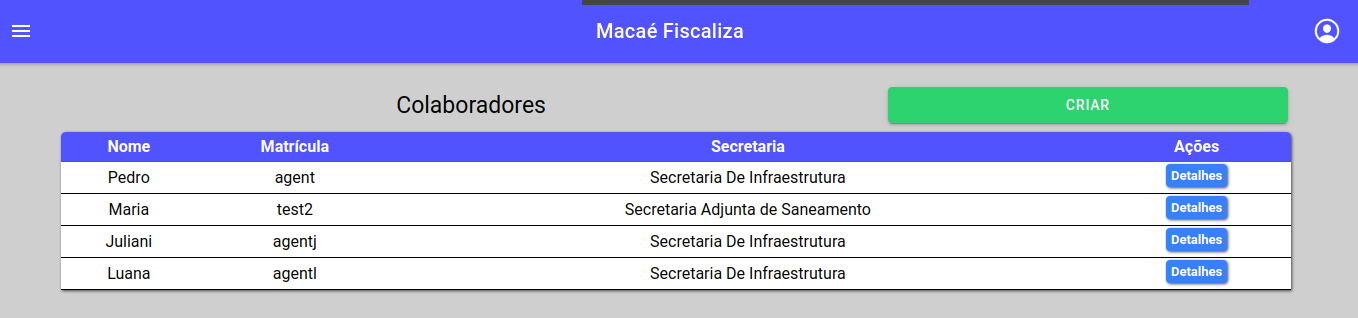


Fonte: Autoria própria (2022)

O formulário de edição de secretaria apresenta a mesma estrutura que o formulário de criação. Seguindo adiante, outra *user story* é: “Como administrador, preciso cadastrar colaboradores”. Esta é atendida ao completar as atividades a seguir:

* + - Criar lista de colaboradores (solucionado conforme a figura 40);
    - Construir formulário de cadastro de colaboradores (solucionado conforme a figura 41).

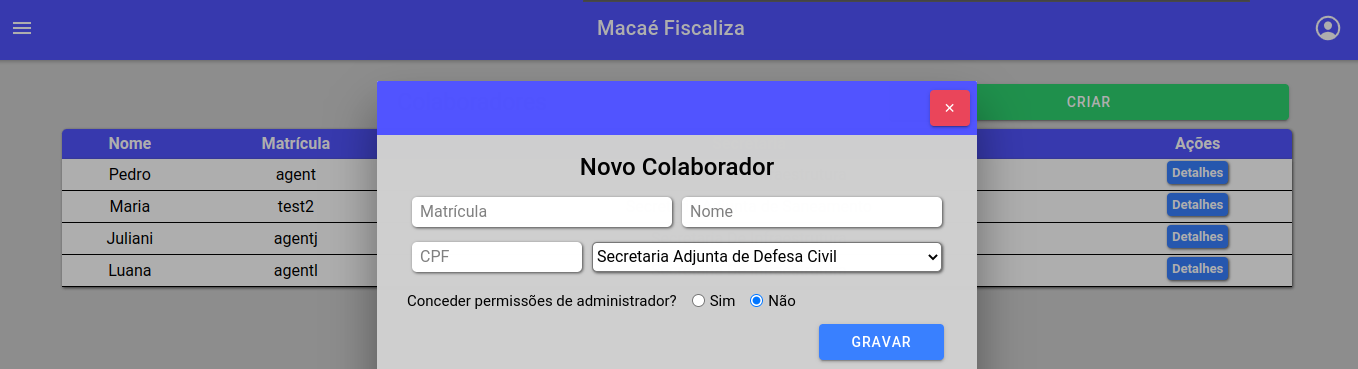
Figura Lista de colaboradores



Fonte: Autoria própria (2022)

A lista de colaboradores seguiu o mesmo esquema da lista de secretarias, porém a listagem de colaboradores fornece uma quantidade maior de colunas, informando o nome do colaborador, a matrícula e a secretaria a qual ele é designado. Pelo fato da aplicação se encontrar em fase de protótipo, o valor de cada matrícula na figura 40 não representa o padrão de matrícula a ser usado quando o sistema estiver em fase de utilização. Ainda, ao clicar no botão “Criar”, surge o formulário de cadastro do colaborador no sistema, exibido na figura 41.

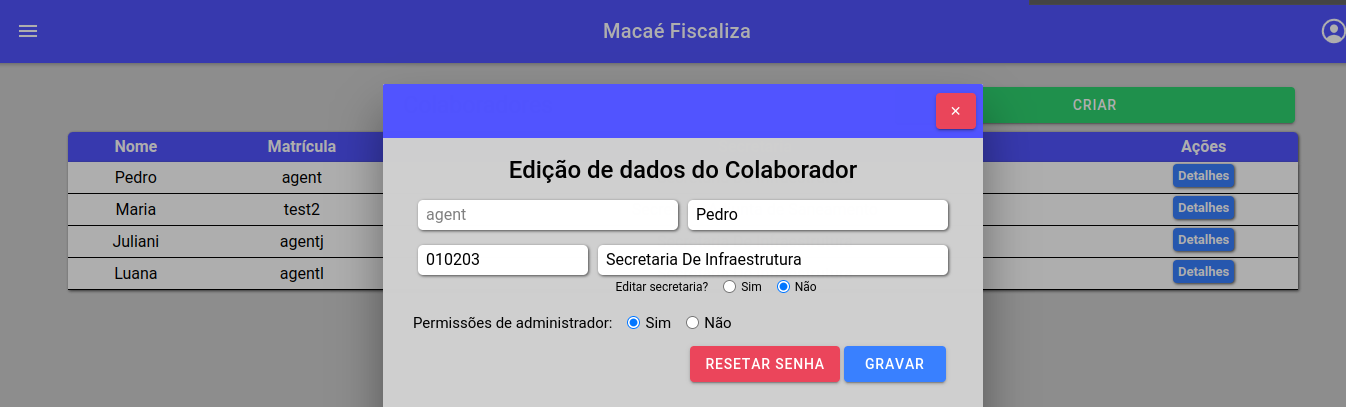
Figura Formulário de cadastro do colaborador



Fonte: Autoria própria (2022)

Dentre os dados a serem fornecidos estão: a matrícula, o nome, o CPF e a concessão de privilégios (“administrador” ou “não administrador”). Apenas os usuários da secretaria Inova Macaé conseguem cadastrar usuários de secretarias diferentes, enquanto administradores das demais secretarias cadastram e editam usuários de seus respectivos setores. Sendo assim, destaca-se outro *user story*: “Como administrador, preciso alterar a secretaria do colaborador em caso de remanejamento”. Resultado apresentado na figura 42.

Figura Formulário de edição do colaborador



Fonte: Autoria própria (2022)

A estética e o funcionamento do formulário de edição são similares aos do formulário de criação. Ademais, dentre os valores disponíveis, apenas a matrícula não é alterável. Também se destaca como diferencial o botão “Resetar Senha”, esta deve ser trocada após o colaborador se autenticar novamente.

# CONSIDERAÇÕES FINAIS

Em suma, o sistema de zeladoria desenvolvido foi estruturando levando em consideração uma potencial interação com as aplicações do Macaé App. Apesar do conjunto de técnicas referenciadas, como visto no capítulo 2, e aplicadas na construção do sistema distribuído, conforme o capítulo 4, é necessário um maior refinamento para que o sistema possa ser disponibilizado ao atendimento ao público.

Referente aos objetivos específicos, o diagrama de classes produzido no capítulo 4 é um artefato que descreve estruturalmente o projeto. Assim, as *user stories* elaboradas ao longo do projeto qualificam as funcionalidades do sistema no ponto de vista dos *stakeholders*. Por sua vez, as metodologias ágeis foram seguidas focando principalmente nos resultados. Já as funcionalidades foram listadas em um *product* backlog e o processo de desenvolvimento foi estruturado em *sprints*, confirmando a utilização da metodologia *SCRUM*.

Foi constatado que as aplicações *backend* interagem entre si, o que pode caracterizá-las como *webservices*. Ainda, o servidor de credencias (“*authzserver*”) centraliza a autenticação de usuários (colaboradores e cidadãos) e, ao utilizar o *Spring Security OAuth2 Authorization Server* como base à aplicação, segue o protocolo *OAuth*2.1. Por outro lado, o servidor de recursos (“*resourceserver*”) fornece dados e realiza o processamento de acordo com a lógica de negócio. Também vale destacar que duas aplicações *frontend* foram construídas separadamente, uma é a interface para os colaboradores e a outra é a interface para os cidadãos. Todavia, ambas foram construídas com a mesma aparência, conforme exibido em figuras do capítulo 4.

Dentre as *user stories* requisitadas, quatro não foram construídas, sendo estas:

* + “Como cidadão,gostaria de acessar o sistema em dispositivos móveis”;
  + “Como cidadão, quero ter a possibilidade de compartilhar minha localização usando o GPS caso me encontre presente no local da ocorrência”;
  + “Como cidadão, gostaria de poder criar conta através de meus e-mails ou redes sociais em casos que necessitem cadastro para poder acompanhar minhas solicitações”;
  + “Como administrador, preciso ativar/desativar as contas dos colaboradores”.

Entretanto, no aspecto de atender às necessidades centrais dos *stakeholders*,o protótipo construído cumpre com seu propósito, mas não cumpre com todas as funcionalidades secundárias. Portanto, a questão de o software produzido cumprir com seu objetivo geral é relativa, ainda que seja construído no ecossistema *Spring* e utilize as ferramentas de segurança propostas para garantir meios de autenticação e controle de acesso.

Como sugestão para trabalhos futuros, sugere-se que haja a continuidade no desenvolvimento do protótipo desenvolvido para torná-lo apto para uso pleno dos colaboradores e cidadãos do município de Macaé. Além de completar as funcionalidades pendentes e promover melhorias. Também se faz necessário o desenvolvimento de um sistema de zeladoria urbana com uma abordagem voltada ao formato de publicações utilizado por aplicativos de redes sociais. Por fim, considera-se importante que seja realizado um estudo de integração do Macaé App com o protótipo resultante desta pesquisa.

**REFERÊNCIAS**

‌BASS, L.; CLEMENTS, P.; KAZMAN, R. **Software Architecture In Practice**. 4. ed. [S.l.]: Addison-Wesley, 2021.

‌BEZERRA, E. **Princípios de análise e projeto de sistemas com UML**. 3. ed. Rio de Janeiro, Brasil: Elsevier, 2015.

BRASIL. Constituição (1988). **Constituição da República Federativa do Brasil**: texto constitucional promulgado em 5 de outubro de 1988, com as alterações adotadas com as emendas constitucionais n. 1/92 a 48/2005 e pelas emendas constitucionais de revisão n.1 a 6/94, pelas Emendas Constitucionais n. 1/92 a 73/2013 e pelo Decreto Legislativo n. 168/2008. Brasília: Senado Federal, Secretaria Especial de Editoração e Publicação, Subsecretaria de Edições Técnicas, 2013. Disponível em: <http://www2.senado.gov.br/bdsf/it em/id/70316>. Acesso em: 28 mar. 2022.

CÓDIGOS de status de respostas HTTP.**MDN Web Docs**, 16 ago. 2021. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Status>. Acesso em: 16 mar. 2022.

FÁBRICA de Aplicativos S/A. **Fabapp**, 2013. Disponível em: <https://fabricadeaplicativos. com.br/>. Acessoem: 15 mar. 2021.

FIELDING, R. T. **Architectural Styles and the Design of Network-based Software Architectures**. Dissertação (Doutorado em Filosofia em Informação e Ciência da Computação) – University of California. Irvine, 2000.

‌

FORMULÁRIOS poderosos facilitam vidas. **Jotform:**Criador de formulários online, 2022. Disponível em: <https://www.jotform.com/pt/>. Acesso em: 15 mar. 2021.

GOURLEY, D.; TOTTY, B. **HTTP**: The Definitive Guide. [S.l.]: O’Reilly Media, 2002.

HARDT, D.; PARECKI, A.; LODDERSTEDT, T. The OAuth 2.1 Authorization Framework. **Data tracker**, jul. 2020. Disponível em: <https://datatracker.ietf.org/doc/html/draft-parecki-oauth-v2-1>. Acesso em: 25 fev. 2022.

HOME. **Fiscaliza VR:** Prefeitura de Volta Redonda, 2022. Disponível em: <https://volta-redonda.fiscalizaja.com.br/home>. Acesso em: 21 abr. 2022.

INGENO, J. **Softward architect’s handbook**: become a successful software architect by implementing effective architecture concepts. Birmingham, Reino Unido: Packt Publishing Ltd, 2018.

MACAÉ. **Decreto Municipal Nº 97, de 17 de junho de 2019**. Inauguração do Laboratório de Inovação em Gestão Pública, doravante denominado INOVA MACAÉ. Macaé, 2019. Disponível em: <https://sistemas.macae.rj.gov.br:84/sim/midia/anexolegislacao/1566458364. pdf>. Acesso em: 10 mar .2022.

**MACAÉ APP**: Software Governamental. 14.0. [S.l.], março/2020. Disponível em: <https://play.google.com/store/apps/details?id=br.com.app.gpu2120664.gpu94dfdedcdf66d8d1b9be787aae3c999d&hl=pt\_BR&gl=US>. Acesso em: 05 jun. 2021.

MASSÉ, M. **REST API design rulebook**. [S.l.]: O’Reilly Media, 2012.

PROJECT Management Institute. **Agile practice guide**. Newton Square, Pennsylvania, EUA: Project Management Institute, Inc., 2017.

PRODANOV, C. C.; FREITAS, E. C. **Metodologia do Trabalho Científico**. 2. Ed. Rio Grande do Sul, Brasil: Universidade FEEVALE, 2013.

PRESSMAN, R. **Engenharia de Software**: Uma Abordagem Profissional. 7. ed. Porto Alegre, Brasil: AMGH, 2011.

RAJ, P.; RAMAN, A.; SUBRAMANIAN, H. **Architectural patterns**: uncover essential patterns in the most indispensable realm of enterprise architecture. Birmingham, Reino Unido: Packt Publishing, 2017.

RICHARDS, M.; FORD, N. **Fundamentals of software architecture**: an engineering approach. Cambridge, Reino Unido: O’Reilly Media, 2020.

RICHARDSON, L.; RUBY, Sam. **RESTful web services**. [S.l.]: O’Reilly Media, 2007.

RUMPE, Bernhard. **Agile modeling with UML**: code generation, testing, refactoring. Cham, Suíça: Springer, 2017.

SANT'ANA, R. **Tecnologia e Gestão Pública Municipal**: Mensuração da Interação com a Sociedade. São Paulo: Cultura Acadêmica, 2009.

SBROCCO, J. H.; Macedo, P. C.. **Metodologias Ágeis**: Engenharia de Software sob Medida. São Paulo, Brasil: Érica, 2012.

‌SHVETS, A. **Dive Into Design Patterns**. [S.l.]: [S.n.], 2021. E-book.

SOMMERVILLE, I. **Engineering software products**: an introduction to modern software engineering. Harlow, Inglaterra: Pearson, 2021.

\_\_\_\_\_\_. **Software engineering**. 8. ed. Harlow, Reino Unido: Addison-Wesley Publishers Ltda, 2006.

\_\_\_\_\_\_.**Software engineering**. 10. ed. [*S.l.*]: Pearson, 2016.

SPILÇÃ, L. **Spring security in action**. Nova Iorque, EUA: Manning Publications, 2020.

‌STALLINGS, W. **Criptografia e segurança de redes**. 4. ed. São Paulo, Brasil: Pearson Prentice Hall, 2008.

TANENBAUM, A.; WETHERALL, D. **Computer Networks**. 5. ed. [*S.l.*]: Prentice Hall, 2010.

UMA visão geral do HTTP. **MDN Web Docs**, 13 jan. 2022. Disponível em: <https://developer. mozilla.org/pt-BR/docs/Web/HTTP/Overview#componentes\_de\_sistemas\_baseados\_em\_ http>. Acesso em: 16 mar. 2022.

**ANEXO A – Declaração de correção gramatical**

Eu, Louise Branquinho Goveia, portadora da carteira de identidade 27709701-0 (DETRAN/RJ), formada em Bacharelado em Letras Português-Inglês pela Universidade Federal do Rio de Janeiro, realizei a correção gramatical do Trabalho de Conclusão de Curso do aluno Gustavo Alberto de Souza Lemos, curso de Graduação em Sistemas de Informação, orientado pelo professor Alan Carvalho Galante.

Texto, Carta

Descrição gerada automaticamente

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Louise Branquinho Goveia

Revisora de textos

Data: 08/07/2022.