

**PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS**  
**PUC Minas Virtual**  
**Pós-graduação *Lato Sensu* em Arquitetura de *Software* Distribuído**

Projeto Integrado

Relatório Técnico

Sistema de Gestão de Ocorrência - SGO

Thiago Gutenberg Carvalho da Costa

Belo Horizonte  
Fevereiro 2022

## Projeto Integrado – Arquitetura de Software Distribuído

### *Sumário*

Projeto Integrado – Arquitetura de Software Distribuído	2
1. Introdução	3
2. Cronograma do Trabalho	5
3. Especificação Arquitetural da solução	7
3.1 Restrições Arquiteturais	7
3.2 Requisitos Funcionais	7
3.3 Requisitos Não-funcionais	9
3.4 Mecanismos Arquiteturais	9
4. Modelagem Arquitetural	11
4.1 Diagrama de Contexto	11
4.2 Diagrama de Container	11
4.3 Diagrama de Componentes	14
5. Prova de Conceito (PoC)	19
5.1 Integrações entre Componentes	19
5.2 Código da Aplicação	23
6. Avaliação da Arquitetura (ATAM)	25
6.1. Análise das abordagens arquiteturais	25
6.2. Cenários	25
6.3. Evidências da Avaliação	26
6.4. Resultados Obtidos	27
7. Avaliação Crítica dos Resultados	28
8. Conclusão	29
Referências	30

## **1. Introdução**

Segundo o Dicionário Escolar (2015, p. 284) a infraestrutura nada mais é que um conjunto de serviços básicos (de saneamento, de transporte, de energia e de telecomunicação) tornando-se uma área fundamental para o desenvolvimento socioeconômico de uma região. Nessa perspectiva, à medida em que centros urbanos se expandem, percebe-se a necessidade de manutenção dessas estruturas que compõem uma infraestrutura urbana, se tornando um ponto vital e de extrema importância para a não depreciação das mesmas e também para o bem estar da comunidade.

Neste contexto, o processo de solicitação de manutenção em estruturas dos serviços públicos (poste de luz, canos de distribuição de água, asfalto, etc.) é feito de forma geral unicamente via canal telefônico, onde além da restrição de horário de atendimento sendo somente em horário comercial, limita-se o registro de uma ocorrência distinta por vez para cada intervenção corretiva, mesmo sendo em uma mesma localização. Portanto, mediante este problema, indaga-se: diante de enorme incidência de reclamações em uma região, o processo de registro de ocorrência está sendo feito de forma desburocratizada e eficiente?

Então, afim de simplificar o processo de registro de incidentes pela população e também com o intuito de modernizar, ampliar e prover mais eficiência e transparência, há a necessidade de implementação de um Sistema de Gestão de Ocorrência onde seja possível a abertura de um chamado para resolução de vários problemas relacionados a um tipo de serviço em um determinado endereço.

Sendo assim, o objetivo geral deste trabalho é apresentar a descrição do projeto arquitetural de uma aplicação de gerenciamento de ocorrências de manutenção em estruturas públicas para uma administração regional.

Onde os objetivos específicos são:

- Implementar serviço de ocorrências, parte responsável por expor APIs Web em conformidade com o padrão arquitetural RESTful provendo interoperabilidade para realização do processamento negocial e de persistência de uma ocorrência;

## Sistema de Gestão de Ocorrência - SGO

- Implementar serviço de localização, que através da comunicação com um sistema externo será responsável por obter informações sobre um endereço a partir de um CEP;
- Implementar serviço de notificações, cuja responsabilidade será o processamento e disparo assíncronos de eventos de notificação relacionados a situação de uma ocorrência;
- Implementar serviço web, responsável por permitir que uma pessoa possa interagir com o sistema, a princípio através de um navegador web com uma interface gráfica responsiva;
- Implementar serviço de relatórios, responsável por disponibilizar relatórios com resumo e/ou estatísticas de atividades realizadas aos interessados.

## 2. Cronograma do Trabalho

A seguir é apresentado o cronograma proposto para as etapas deste trabalho.

Datas		Atividade / Tarefa	Produto / Resultado
De	Até		
01/01/2022	07/01/2022	1. Analisar ideias e selecionar uma para ser abordada e servir como a base para elaboração do relatório técnico.	Tema central do projeto integrado.
08/01/2022	15/01/2022	2. Apresentar informações gerais, referentes ao escopo do projeto, mais especificamente quanto ao contexto, a problemática e a motivação do mesmo.	Introdução, objetivo geral e objetivos específicos do projeto.
16/01/2022	22/01/2022	3. Definir os requisitos arquiteturais da solução.	Descrição dos requisitos que podem impactar diretamente ou indiretamente na macro arquitetura da solução.
23/01/2022	01/02/2022	4. Definir os requisitos funcionais da solução.	Descrição do fornecimento de serviços do sistema e como o mesmo deverá se comportar a partir de entradas específicas.
02/02/2022	06/02/2022	5. Definir os requisitos não-funcionais da solução.	Especificar restrições às funcionalidades do sistema.
07/02/2022	07/02/2022	6. Definição dos mecanismos arquiteturais.	Descrição de conceitos técnicos para padronizar aspectos do sistema.
13/02/2022	14/02/2022	7. Criar o diagrama de contexto.	Diagrama de alto nível que mostra de uma forma geral a macroarquitetura da solução no modelo C4.
13/02/2022	14/02/22	8. Vídeo de apresentação inicial.	Apresentação inicial do projeto, ideia completa sobre o projeto.
16/02/2022	19/02/2022	9. Criar o diagrama de container.	Diagrama que detalha o sistema, mostra os bancos de dados, serviços e descreve as tecnologias usadas.
20/02/2022	28/02/2022	10. Criar o diagrama de componentes.	Diagrama com detalhes de um serviço, mostra camadas de integração de componentes na base de código.
16/02/2022	16/05/2022	11. Iniciar o desenvolvimento do projeto.	Protótipo funcional contendo o em seu escopo os requisitos prioritários.
01/03/2022	08/03/2022	12. Criar o diagrama de código.	Diagrama de detalha a composição de classes definidas em um componente, refletindo diretamente o código.
16/05/2022	31/05/2022	13. Avaliar a arquitetura através do modelo ATAM ( <i>Architecture Tradeoff Analysis Method</i> ).	Análise da abordagem da arquitetura com os cenários e apresentação dos resultados.
01/06/2022	08/06/2022	14. Conclusão.	Lições aprendidas e pontos de melhorias na arquitetura.

## Sistema de Gestão de Ocorrência - SGO

09/06/2022	11/06/2022	15. Vídeo de apresentação final.	Apresentação completa do projeto realizado, focando a arquitetura.
------------	------------	----------------------------------	--

### 3. Especificação Arquitetural da solução

#### 3.1 Restrições Arquiteturais

R1: Seguir o padrão arquitetural de microsserviços;

R2: As APIs devem estar no padrão RESTful preferivelmente seguindo o princípio HATEOAS (Hypermedia as the Engine of Application State) para prover melhor interoperabilidade;

R3: O back end deve ser implementado utilizando o conjunto de ferramentas para padrões comuns em sistemas distribuídos providos pelo projeto Spring Cloud;

R4: Os componentes que compõem a interface gráfica do sistema devem ter comportamento responsivo para se adequar melhor nas dimensões de tela dos dispositivos dos usuários;

R5: O sistema deve ser complacente à LGPD (Lei Geral de Proteção de Dados).

#### 3.2 Requisitos Funcionais

ID	Descrição Resumida	Dificuldade (B/M/A)*	Prioridade (B/M/A)*
RF01	O sistema deve permitir que um usuário externo se registre através de um formulário de cadastro indicando nome de usuário, e-mail, primeiro nome, último nome e senha.	M	A
RF02	O sistema deve permitir que o usuário se autentique através de um formulário de acesso (login) onde deverá ser informado um nome de usuário e senha previamente cadastrada.	A	A
RF03	O sistema deve permitir o registro de uma ocorrência por um usuário.	A	A
RF04	O sistema deve permitir a alteração de detalhes de uma ocorrência registrada, caso a mesma tenha sido devolvida pelo atendente para ajustes.	M	A
RF05	O sistema deve permitir que uma ocorrência registrada possa ser cancelada pelo usuário que a criou caso a mesma não esteja com situação já aprovada.	B	B
RF06	O sistema deve permitir que sejam vinculados vários incidentes de um mesmo tipo de serviço em uma ocorrência durante o processo de registro.	M	M

## Sistema de Gestão de Ocorrência - SGO

RF07	O sistema deve permitir anexar foto para cada incidente vinculado a uma ocorrência durante o processo de registro.	A	B
RF08	O sistema deve permitir o acompanhamento de uma ocorrência através de uma pesquisa a partir de um número do protocolo.	B	A
RF09	O sistema deve listar os protocolos de ocorrências registradas para o usuário logado.	B	M
RF10	O sistema deve permitir que um atendente analise e classifique priorizando uma ocorrência em uma escala de urgência de: Baixo, Moderado ou Alto.	B	A
RF11	O sistema deve permitir que um atendente altere a situação de uma ocorrência para: Aprovada, Cancelada, Devolvida.	B	A
RF12	O sistema deve permitir a buscar automaticamente um endereço a partir de um CEP informado pelo usuário durante cadastro de uma ocorrência.	M	A
RF13	O sistema deve permitir que o usuário selecione uma localidade a partir de um mapa durante cadastro de uma ocorrência.	A	B
RF14	O sistema deve permitir que o usuário possa desligar o recebimento de notificações via e-mail sobre a situação de uma ocorrência registrada.	M	A
RF15	O sistema deve enviar notificações via componente de interface (badge) sobre a situação da ocorrência para o usuário.	M	M
RF16	O sistema deve enviar notificações via e-mail para o usuário sobre a situação da ocorrência caso o usuário opte por receber esse tipo de notificação.	M	M
RF17	O sistema deve permitir gerar relatórios referente as localizações que mais tiveram ocorrências registradas.	M	A
RF18	O sistema deve permitir gerar relatórios sobre a quantidade de horas de participação de um atendente nas ocorrências - TMA (Tempo Médio de Atendimento)	M	M
RF19	O sistema deve permitir gerar relatórios sobre a quantidade de horas em que uma ocorrência ficou em aberto até ser atendida - TME (Tempo Médio de Espera)	M	M
RF20	O sistema deve notificar o sistema externo de ordem de serviço para gerar uma O.S (Ordem de Serviço) quando uma ocorrência mudar para situação aprovada.	M	A

\*B=Baixa, M=Média, A=Alta.

**Obs.:** acrescente quantas linhas forem necessárias.



### 3.3 Requisitos Não-funcionais

ID	Descrição	Prioridade B/M/A
RNF01	Desempenho — A operação de salvar uma ocorrência tem que ser rápida, não podendo exceder mais do que 3 segundos em média para ser executada.	A
RNF02	Usabilidade — A interface gráfica deve ser responsiva aplicando um design adaptativo para melhorar a disposição do conteúdo na tela do dispositivo melhorando a experiência do usuário.	M
RNF03	Compatibilidade — O sistema deverá funcionar corretamente nos navegadores web modernos mais usados (Google Chrome, Edge e Firefox).	B
RNF04	Segurança — O acesso a APIs privadas precisa ser seguro, exigindo a devida autenticação e autorização do usuário, caso contrário resultará respectivamente nos erros HTTP 401 - ( <i>Unauthorized</i> ) ou HTTP 403 ( <i>Forbidden</i> ).	A
RNF05	Padrão — O software deverá ser orientado a camadas possuindo baixo acoplamento e alta coesão seguindo o princípio de Responsabilidade Única para prover melhor manutenibilidade e evolução.	A
RNF06	Confiabilidade — Rastreamento distribuído para coleta e pesquisa de dados de tempo de requisições para detectar possíveis problemas de latência na arquitetura de serviços deverá ter um percentual de amostragem de 100%	A

\*B=Baixa, M=Média, A=Alta.

**Obs.:** acrescente quantas linhas forem necessárias.

### 3.4 Mecanismos Arquiteturais

Esta seção descreve os mecanismos que compõem a arquitetura do software fazendo uma ligação entre a definição conceitual da solução (análise) e sua implementação, passando pela visão de desenho.

Análise	Design	Implementação
Segurança - Gestão de Identidade e Acesso	Método de acesso - Single-Sign On (SSO)	Keycloak
Segurança - Autenticação e Autorização	Protocolo de autenticação interoperável	OpenID Connect – OIDC
Persistência – Abstração	Spring Data JPA (Java Persistence API)	Hibernate
Persistência – Relacional	Sistema de gerenciamento de banco	MySQL

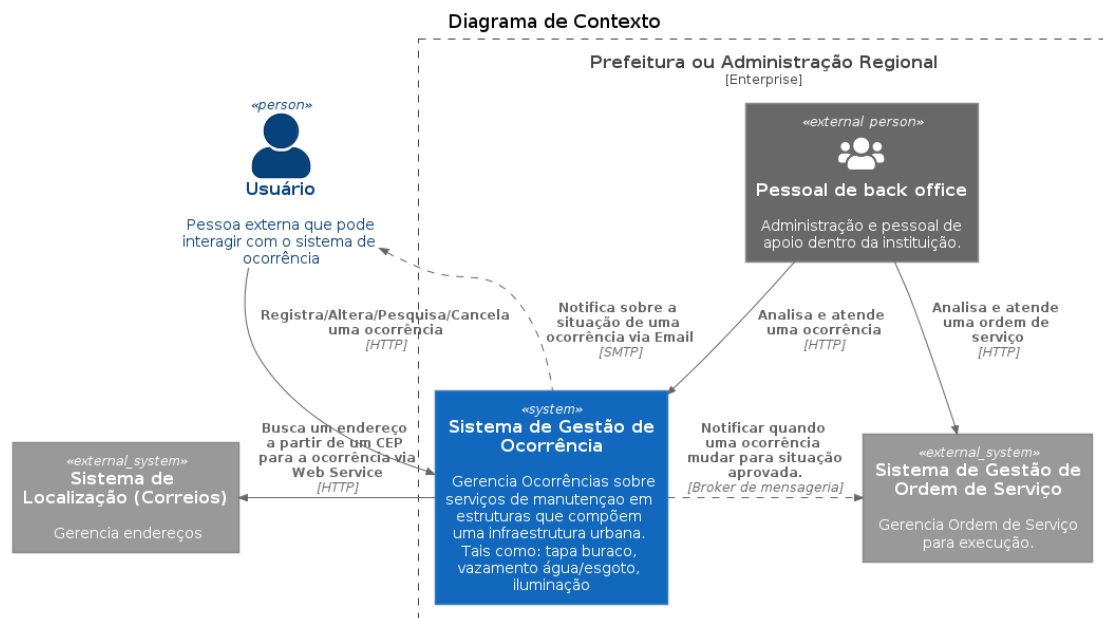
## Sistema de Gestão de Ocorrência - SGO

	de dados (SGBD)	
Persistência – Não-relacional	Banco de dados distribuído e baseado em documentos	MongoDB
Versionamento – Ferramenta	Sistema de controle de versões distribuído	Git
Versionamento – Sistema online	Sistema online de controle para desenvolvimento de software	GitHub
Front end – Estratégia de implementação web	Single Page Application	Angular
Front end – Componentes	Biblioteca de Componentes de Interface do Usuário	Angular Material
Back end - Framework	Framework de aplicação	Spring
Back end - Ferramentas	Padrões comuns para sistemas distribuídos (Serviço de registro e descoberta, rastreamento distribuído)	Spring Cloud
Integração – Assíncrona	Broker de mensageria	RabbitMQ
Integração – Síncrona	Web Service - REST API	Comunicação de requisição/resposta via protocolo <i>Hypertext Transfer Protocol</i> (HTTP)
Integração - Rastreo de eventos distribuídos	Coletar dados de tempo necessários para solucionar problemas de latência em arquiteturas de serviço	Zipkin
Log do sistema	Simple Logging Facade for Java (SLF4J) API	Logback
Teste de Software	Framework para teste unitário	JUnit 5
Teste de Software – Mock	Framework de mocking para teste unitário	Mockito
Deploy – Distribuição	Unidade padrão de software que empacota o código e todas as suas dependências de forma autocontida.	Docker Container
Deploy – Orquestração	Orquestrar ambientes distribuídos baseados em containers	Kubernetes

## 4. Modelagem Arquitetural

Esta seção apresenta a modelagem arquitetural através de diagramas hierárquicos de forma a permitir completo entendimento em diferentes níveis de abstração da solução proposta visando à implementação da prova de conceito (seção 5).

### 4.1 Diagrama de Contexto



**Figura 1 – Diagrama de Contexto (Visão Geral da Solução).** Fonte: [SGO C4 Context](#) (Alta resolução).

A figura 1 descreve a macroarquitetura do Sistema de Gestão de Ocorrência através do diagrama de contexto, mostrando como o sistema se encaixa no mundo em termos das pessoas que o utilizam e dos outros sistemas de *software* com os quais ele interage. Em azul está o Sistema de Gestão de Ocorrência que será construído, usuários externos e internos podem interagir com ele afim de registrar ou atender uma ocorrência. Em cinza, são os sistemas de software externos que já existem no qual o Sistema de Gestão de Ocorrência irá interagir com eles para buscar uma localização ou para notificar sobre a situação de uma ocorrência para que seja gerada uma ordem de serviço.

### 4.2 Diagrama de Container

## Sistema de Gestão de Ocorrência - SGO

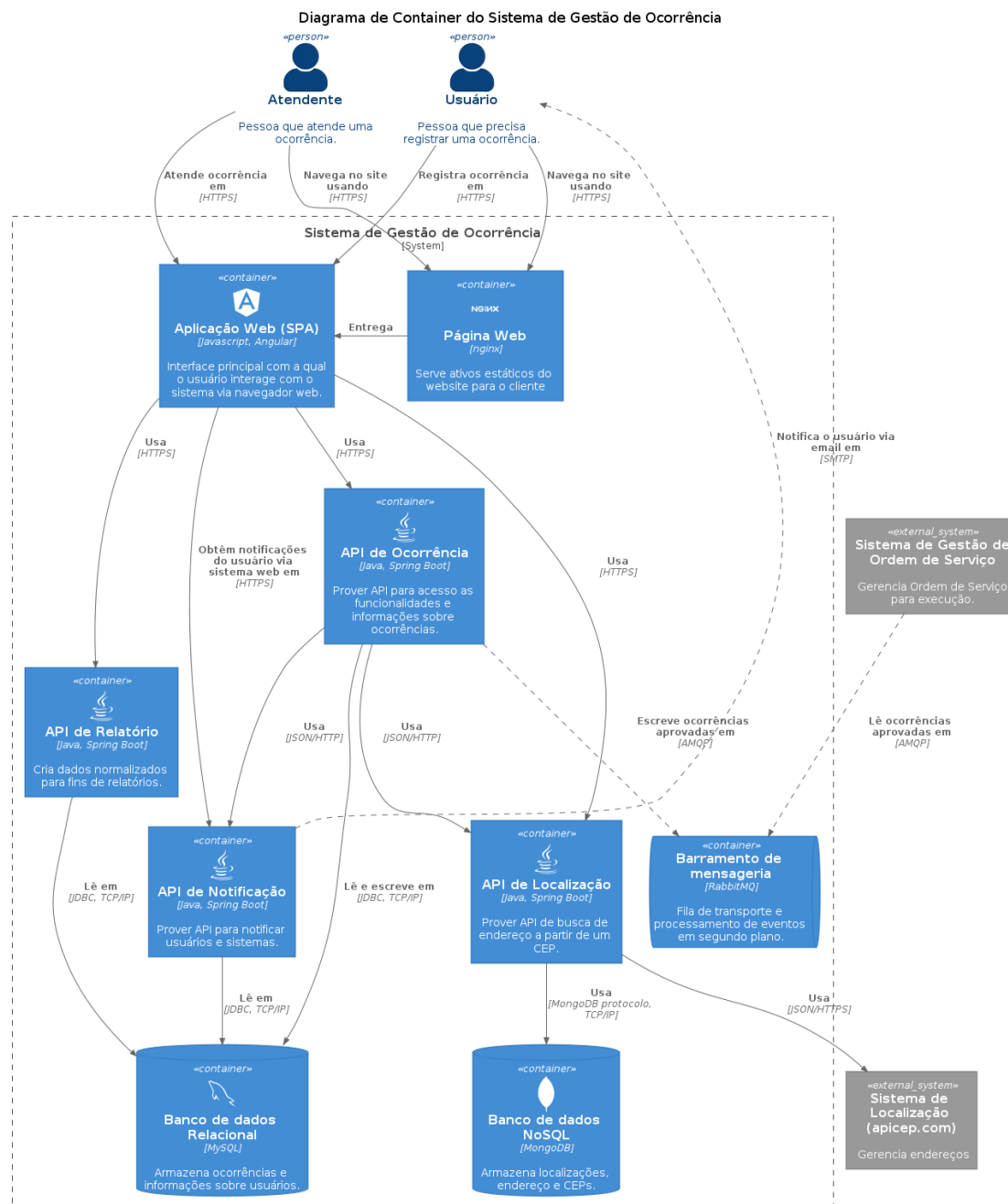


Figura 2 – Diagrama de Container. Fonte: [SGO C4 Container](#) (Alta resolução).

A figura 2 detalha através do diagrama de *container* mostrando de forma amplificada os *containers* (em azul) que compõem o Sistema de Gestão de Ocorrência, onde em cada *container* é evidenciado sua natureza, bem como a tecnologia que é usada e também os protocolos de integração com os quais eles interagem entre si. A página web é onde são servidos os conteúdos estáticos (HTML, CSS e JavaScript) fornecido pelo NGINX que compõem a aplicação web feita em Angular que por sua vez será renderizada aos usuários via navegador web. Os *containers* cujo nome começa com “API de” são microserviços feito em Java/Spring

Boot que expõem APIs *RESTful* provendo funcionalidades específicas e coesas para determinados assuntos e podem utilizar o banco de dados relacional para manter dados. O microserviço de localização se comunica com um outro sistema externo, para busca de informações sobre endereços relacionados a um determinado Código de Endereçamento Postal – CEP e utiliza o banco de dados NoSQL para persistir e ler dados. O microserviço de ocorrência mantém ocorrências criadas pelo usuário e se integra com um sistema externo de atendimento e gestão de ordem de serviços via barramento de mensageria enviando para uma fila RabbitMQ as ocorrências em situação aprovada. O microserviço de notificação mantém notificações relacionadas às situações das ocorrências e pode avisar o usuário via e-mail. O microserviço de relatório usa as informações mantidas em banco de dados, agregando-as para geração de relatórios.

### 4.3 Diagrama de Componentes

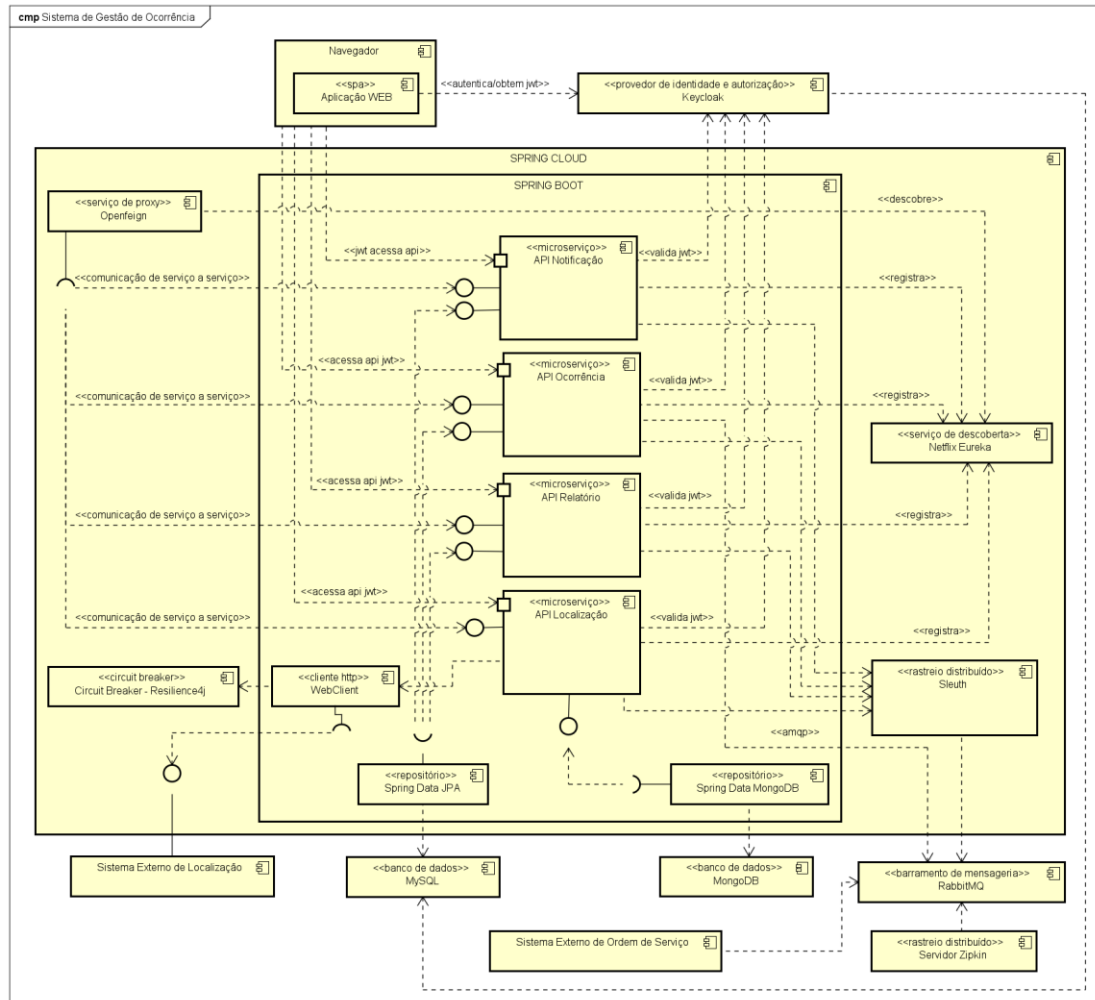


Figura 2 – Diagrama de Componentes. Fonte: [SGO UML Component](#) (Alta resolução).

A figura 3 mostra a estrutura do Sistema de Gestão de Ocorrência, descrevendo suas dependências e suas interfaces reutilizáveis e substituíveis através do diagrama de componentes da aplicação baseado na UML.

Conforme diagrama apresentado na figura 3, o sistema segue padrão/estilo arquitetural orientado a microserviços, respeitada a definição de Restrição Arquitetural, onde a aplicação é dividida em uma coleção de serviços (módulos) separados, possuindo características de baixo acoplamento, independência de implantação, alta manutenibilidade e testabilidade e separação de capacidades negociais, permitindo então uma entrega rápida, frequente e confiável de funcionalidades. Além disso, proporciona também oportunidade de evolução dos

módulos no que tange à utilização de outras tecnologias e/ou linguagens de programação.

Padrões comuns para sistemas distribuídos, definidos nos Mecanismos Arquiteturais, são seguidos nessa arquitetura e estão representados no diagrama da figura 3, sendo eles:

- Padrão de Registro de Serviços, viabilizado através dos componentes Netflix Eureka, mantém um cadastro das instâncias e localidades dos serviços facilitando a descoberta e solicitação sem a necessidade de que um cliente e/ou API *gateway* saiba exatamente o IP e porta de um determinado microserviço para eventuais solicitações. As instâncias de serviço são registradas durante inicialização e desregistradas no encerramento.
- Padrão de Rastreamento Distribuído, agrega o conceito de observabilidade, onde através do componente Sleuth são atribuídos identificadores únicos para as requisições. Essa identificação é mantida por toda cadeia de processamento em todos os microserviços acionados e métricas de tempo transcorrido sobre um percentual de amostragem das solicitações e operações realizadas são enviadas à uma fila (Componente RabbitMQ) e consumidas por um serviço centralizado (Componente Zipkin) afim de identificar possíveis problemas de latência em arquiteturas de serviços distribuídos.
- Padrão de *Circuit Breaker*, através do componente Resilience4J proporciona confiabilidade durante integrações com APIs remotas, rejeitando novas invocações por um certo período depois que falhas consecutivas são observadas, evitando assim onerar ainda mais um serviço que já está sobrecarregado, não gerando um efeito dominó, ou seja, derrubando serviços de forma consecutiva. Respostas alternativas (*fallbacks*) podem ser retornadas para o serviço invocador como por exemplo uma resposta cacheada.

Os elementos do modelo que representam partes independentes e intercambiáveis da solução são:

- Componente API de Ocorrência – Microserviço desenvolvido utilizando o framework SpringBoot, expõe *endpoints* RESTful acessíveis a partir de solicitações HTTP para um IP e porta específico juntamente com um JSON

Web Token no cabeçalho de autenticação para possibilitar a manipulação de dados que compõem uma ocorrência e através do processamento negocial manter as informações no banco de dados MySQL. Esse serviço depende de um *Identity Provider* (IdP) representado pelo componente Keycloak para verificar a identidade do usuário que invoca a API.

- Componente API de Localização – Microserviço desenvolvido utilizando o framework SpringBoot, expõe *endpoints* RESTful acessíveis a partir de solicitações HTTP para um IP e porta específico juntamente com um JSON Web Token no cabeçalho de autenticação para possibilitar busca de informações de endereço relacionadas à um CEP. Esse serviço, através de um cliente HTTP, se integra com um sistema externo para realizar solicitações de busca, obtendo como retorno o endereço em formato JSON que por sua vez é cacheado no banco de dados MongoDB para propiciar certa confiabilidade de resposta caso a integração externa tenha algum problema. Esse serviço depende de um *Identity Provider* (IdP) representado pelo componente Keycloak para verificar a identidade do usuário que invoca a API.
- Componente API de Notificação – Microserviço desenvolvido utilizando o framework SpringBoot, expõe *endpoints* RESTful acessíveis a partir de solicitações HTTP para um IP e porta específico juntamente com um JSON Web Token no cabeçalho de autenticação para inserção e busca de mensagens informativas relacionadas a uma ocorrência no banco de dados MySQL. Esse serviço depende de um *Identity Provider* (IdP) representado pelo componente Keycloak para verificar a identidade do usuário que invoca a API.
- Componente API de Relatório – Microserviço desenvolvido utilizando o framework SpringBoot, expõe *endpoints* RESTful acessíveis a partir de solicitações HTTP para um IP e porta específico juntamente com um JSON Web Token no cabeçalho de autenticação para gerar relatórios em PDF ou Excel e/ou obter informações agregadas no banco de dados MySQL para servir de insumo para dashboards para auxiliar em tomadas de decisões. Esse serviço depende de um *Identity Provider* (IdP) representado pelo componente Keycloak para verificar a identidade do usuário que invoca a API.



- Componente Aplicação Web – Portal Web desenvolvido com Angular e Angular Material, permiti que uma pessoa possa interagir com o sistema, através de um navegador web com uma interface gráfica responsiva e simples, possui integração com um provedor de identidade através do OpenID Connect, camada de identificação baseada no protocolo OAuth 2, para prover acesso às funcionalidades privadas que exigem autenticação.
- Componente Keycloak – Servidor de autenticação para gerenciamento de acesso e identificação de usuários e sistemas. Os serviços dependem desse *Identity Provider* (IdP) para verificação da identidade do usuário que invoca uma API.
- Componente Spring Cloud OpenFeign – Cliente declarativo para serviço web que fornece funcionalidade de chamada HTTP balanceada e integração com serviços de descoberta como o Eureka.
- Componente Spring Cloud Netflix Eureka – Serviço de registro e descoberta de serviços, como um catálogo para proporcionar alta disponibilidade de comunicação entre microserviços em uma arquitetura distribuída.
- Componente Spring Cloud Sleuth – Componente que proporciona rastreamento distribuído através da adição de identificadores ao *Slf4J Mapped Diagnostic Context* (MDC) e integração com serviços agregadores de log como o Zipkin.
- Componente Spring Cloud Circuit Breaker – Componente que tem função semelhante à um disjuntor para prover tolerância a falhas durante integração com outros serviços.
- Componente Spring Data JPA – Componente que facilita e abstrai a complexidade de implementação de repositórios (camada de acesso a dados) baseados na especificação *Java Persistence API* (JPA) para reduzir o esforço de acesso à um banco de dados.
- Componente Spring Data MongoDB – Componente que fornece integração com banco de dados orientados a documentos através de um modelo centralizado em POJO para interagir com coleções facilitando a implementação de repositórios para acesso à camada de dados. Propiciando acesso aos endereços das ocorrências.

- Componente Spring WebClient – Cliente Web de alta performance para fazer chamadas HTTP utilizando uma abordagem de Reatividade através do projeto Spring Reactor. Proporcionando características de baixa latência e alta produtividade, além de não bloqueio durante solicitações HTTP de integração com outros serviços.
- Componente MySQL – Banco de dados relacional escalável e de alto desempenho que é bastante popular. Usado principalmente pelo sistema para manter as informações oriundas do processo de gestão de ocorrências e de regras de perfil de acesso de usuários.
- Componente MongoDB – Banco de dados orientado a documento. Usado para cadastro e busca de informações relacionadas à localização da ocorrência, como informações de endereço e CEP. E no futuro poderá auxiliar em funcionalidade de Geo Localização.
- Componente Zipkin – Sistema de rastreamento distribuído onde através da coleta de dados resume a porcentagem de tempo gasto em um serviço e se as operações falharam ou não, ajudando a solucionar problemas de latência em arquiteturas orientada a serviços.
- Componente RabbitMQ – Barramento de mensageria extremamente performático, bastante usado e conhecido. Sua principal função no sistema é prover uma comunicação assíncrona através de tipo de mensagens enviadas à uma determinada fila para processamento futuro. Sendo um componente que permite uma resiliência de integração entre serviços.
- Componente Sistema externo de Localização – Sistema externo que auxilia na busca de informações sobre endereços a partir de um CEP através de um catálogo de endereços atualizados.
- Componente Sistema externo de Ordem de Serviço – Representa um sistema fictício que é responsável por processar ocorrências que após um atendimento inicial tiveram a situação aprovada. Iniciando então o ciclo de conserto e resolução do problema que foi reportado através do registro de ocorrência.

## 5. Prova de Conceito (PoC)

Esta seção apresenta a prova de conceito arquitetural, detalhando e avaliando importantes aspectos das funcionalidades em relação ao requisito funcionais e não-funcionais da arquitetura.

### 5.1 Integrações entre Componentes

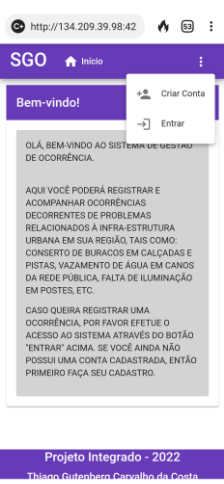
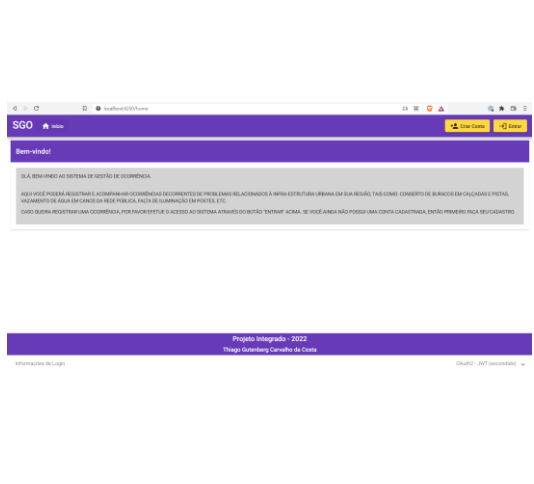
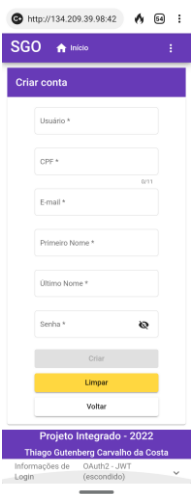
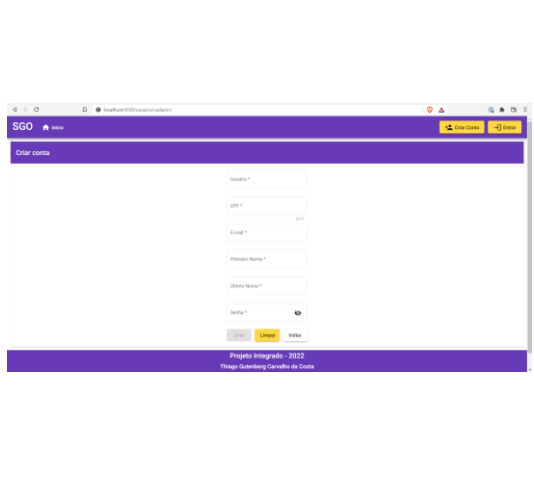
A PoC foi desenvolvida a partir de requisitos prioritários coerentes a especificação proposta e implementada de maneira eficaz afim de permitir a validação das integrações entre os componentes.

Requisitos Funcionais (prioritários)	Componentes e <i>middlewares</i> envolvidos	Integrações e protocolos de comunicação	Requisitos Não-Funcionais
Cadastro de usuários.	Portal Web	Comunicação com a API <i>RESTful</i> de usuários (HAL+JSON).	Segurança, Usabilidade, Compatibilidade
	API Ocorrência	Comunicação TCP/JDBC com MySQL e via REST (JSON) com a API do Keycloak.	Segurança, Padrão, Desempenho, Confiabilidade.
	Keycloak	Comunicação com MySQL via TCP/JDBC.	Segurança
Acesso (login) de usuários.	Portal Web	Comunicação via OAuth2/ <i>OpenID Connect</i> com o Keycloak.	Segurança, Usabilidade, Compatibilidade.
	Keycloak	Comunicação com MySQL via TCP/JDBC.	Segurança
Registro de ocorrência.	Portal Web	Comunicação com a API de ocorrências e endereços via <i>RESTful</i> (HAL+JSON).	Segurança, Usabilidade, Compatibilidade
	API Ocorrência	Comunicação com banco de dados MySQL via TCP/JDBC.	Segurança, Padrão, Desempenho, Confiabilidade.
	API Localização	Comunicação com API de externa de CEP via REST (JSON).	Padrão.

## Sistema de Gestão de Ocorrência - SGO

Lista de ocorrências.	Portal Web	Comunicação com a API de ocorrências (HAL+JSON).	Segurança, Padrão, Desempenho, Compatibilidade.
	API Ocorrência	Comunicação com banco de dados MySQL via TCP/JDBC.	Padrão.

As telas do sistema foram desenhadas para proporcionar uma boa usabilidade tanto em ambiente *mobile* quanto *desktop*, utilizando componentes Angular Material e Angular Flexbox. São elas:

Mobile	Desktop
Tela Inicial	
	
Tela Cadastro	
	
Tela Login	

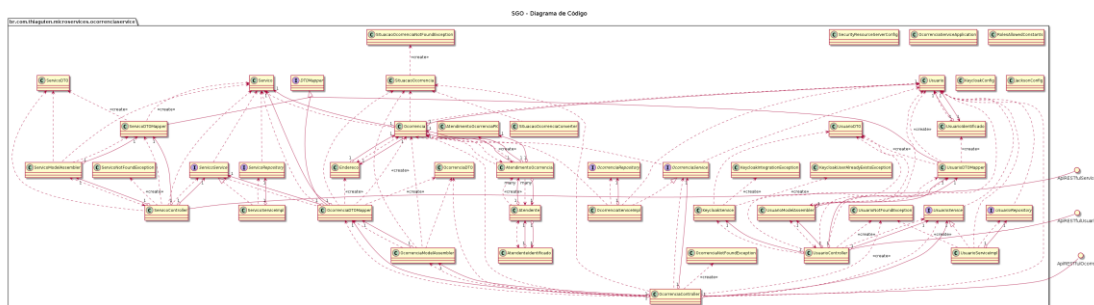
# Sistema de Gestão de Ocorrência - SGO



Figuras das telas do sistema – Fonte em (Alta resolução): <https://bit.ly/37OpUCY>

## 5.2 Código da Aplicação

Nesta seção é apresentado o diagrama de classes de componentes que foram pensados para o Sistema de Gestão de Ocorrência - SGO.



**Figura 4 – Estrutura de código da Módulo de Ocorrências. Fonte: [Módulo-Ocorrência C4-Code](#) (Alta resolução)**

A estrutura da aplicação mostrada na Figura 4 apresenta os componentes de código e suas funções no software implementado:

- Conjunto de classes para configurações das características da aplicação e de integração, tais como configurações de segurança e interoperabilidade, além de classes de exceções, de integração com serviços de gestão de identidade, classes de suporte, etc.
- Conjunto de classes que compõem API *RESTful* de usuários, classes de modelo de dados relacional, classes da camada de validação negocial e classes para persistência e busca de informações no banco de dados.
- Conjunto de classes que compõem API *RESTful* de serviços, classes de modelo de dados relacional, classes da camada de validação negocial e classes para persistência e busca de informações no banco de dados.
- Conjunto de classes que compõem API *RESTful* de ocorrências, classes de modelo de dados relacional, classes da camada de validação negocial e classes para persistência e busca de informações no banco de dados.



**Figura 5 – Estrutura de código da Módulo de Localização. Fonte: [Módulo-Localização C4-Code](#) (Alta resolução)**

A estrutura da aplicação mostrada na Figura 5 apresenta os componentes de código e suas funções no software implementado:

- Conjunto de classes para configurações das características da aplicação e de integração, tais como configurações de segurança e interoperabilidade, além de classes de exceções, de integração com serviços de consulta de CEP, etc.
- Conjunto de classes que compõem API *RESTful* de endereços, classes de modelo de dados relacional, classes da camada de validação negocial e classes para persistência e busca de informações no banco de dados.

O *link* do vídeo de apresentação da PoC e do repositório onde encontra-se o código-fonte, diagramas, roteiros, etc., sobre o protótipo funcional está descrito no apêndice deste documento.



## 6. Avaliação da Arquitetura (ATAM)

**Conforme o cronograma de trabalho, essa parte será feita futuramente na Etapa**

**3.** A avaliação da arquitetura desenvolvida neste trabalho é abordada nesta seção visando avaliar se ela atende ao que foi solicitado pelo cliente, segundo o método ATAM.

### 6.1. Análise das abordagens arquiteturais

**Conforme o cronograma de trabalho, essa parte será feita futuramente na Etapa**

**3.** Apresente aqui um breve resumo das principais características da proposta arquitetural. Para isto, utilize o método Architecture Tradeoff Analysis Method (ATAM), no qual são utilizados cenários para fazer essa análise.

Exemplo:

Atributos de Qualidade	Cenários	Importância	Complexidade
Interoperabilidade	Cenário 1: O sistema deve se comunicar com sistemas de outras tecnologias.	A	M
Usabilidade	Cenário 2: O sistema deve prover boa usabilidade.	M	B
Manutenibilidade	Cenário 3: O sistema deve ter a manutenção facilitada.	M	M

### 6.2. Cenários

**Conforme o cronograma de trabalho, essa parte será feita futuramente na Etapa**

**3.** Mostre os cenários utilizados na realização dos testes da sua aplicação. Escolha cenários de testes que demonstrem os requisitos não funcionais (atributos de qualidade) sendo satisfeitos. Priorize os cenários para a avaliação segundo critérios quantitativos ou qualitativos.

*Exemplos de cenários:*

*Cenário 1 - Interoperabilidade: Ao acessar a URL do serviço de informações gerenciais via HTTP GET, o mesmo deve retornar as informações no formato JSON.*

*Cenário 2 - Usabilidade: Ao navegar na tela, o sistema deve apresentar boa usabilidade. A navegação deve apresentar facilidade e o acesso as funcionalidades deve ser*

*bem objetivo para a função que precisar ser realizada, o usuário deve ser capaz de efetuar uma compra em no máximo 5 minutos, assim garantindo a agilidade e a usabilidade para ficar de acordo com um dos requisitos não funcionais.*

*Cenário 3 - Manutenibilidade: Havendo a necessidade de alterar o gateway de pagamento somente será necessário fazer alteração no broker da funiconalidade de pagamento, facilitando a manutenção e os testes.*

### 6.3. Evidências da Avaliação

**Conforme o cronograma de trabalho, essa parte será feita futuramente na Etapa**

**3.** Apresente as medidas registradas na coleta de dados. Para o que não for possível quantificar apresente uma justificativa baseada em evidências qualitativas que suportem o atendimento ao requisito não-funcional.

Atributo de Qualidade:	Interoperabilidade
Requisito de Qualidade:	O sistema deve se comunicar com outras tecnologias.
Preocupação:	
O sistema deve ter como resposta a uma requisição uma saída de fácil leitura por outro componente.	
Cenário(s):	
Cenário 1	
Ambiente:	
Sistema em operação normal	
Estímulo:	
O sistema de monitoramento envia uma requisição para o serviço REST do módulo de informações gerenciais.	
Mecanismo:	
Criar um serviço REST para atender às requisições do sistema de monitoramento	
Medida de resposta:	
Retornar os dados requisitados no formato JSON	
Considerações sobre a arquitetura:	
Riscos:	Alguma instabilidade na rede pode deixar a conexão lenta ou mesmo a perda de pacotes.
Pontos de Sensibilidade:	Não há
Tradeoff:	Não há

*Acrescente imagens e descreva os testes realizados, de tal forma que se comprove a realização da avaliação.*

*Faça isto para todos os cenários apresentados no tópico 6.1.*

## **6.4. Resultados Obtidos**

**Conforme o cronograma de trabalho, essa parte será feita futuramente na Etapa**

**3.** Apresente os resultados da arquitetura produzida, indicando seus pontos fortes e suas limitações. A título de sugestão construa uma tabela apresentando esses resultados, como no exemplo que segue:

<b>Requisitos Não Funcionais</b>	<b>Teste</b>	<b>Homologação</b>
RNF01: O sistema deve ...	OK	OK
RNF02: O sistema deve ...	OK	N.A.
RNF03: ...	OK	N.A.

*Obs: N.A.: não se aplica.*

## 7. Avaliação Crítica dos Resultados

**Conforme o cronograma de trabalho, essa parte será feita futuramente na Etapa**

**3.** Apresente aqui, de forma resumida, os principais pontos positivos e negativos da arquitetura proposta. Adote uma postura crítica que permita entender as limitações arquitetais, incluindo os prós e contras das tecnologias. Você pode utilizar o formato textual ou produzir um quadro resumo.

Ex. de quadro resumo:

Ponto avaliado	Descrição
XXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXX

## 8. Conclusão

**Conforme o cronograma de trabalho, essa parte será feita futuramente na Etapa**

**3.** Descreva, de forma sucinta, quais foram as lições aprendidas na execução do seu projeto arquitetural. Procure apresentá-las de tal forma que fiquem configurados os *trade-offs* da arquitetura produzida, como por exemplo, Segurança X Desempenho, Granularidade X Manutenibilidade, etc.

Aqui deve ser apresentado também tudo que se aprendeu com esse projeto, de modo a servir como ajuda para outros profissionais.

Também se faz necessário evidenciar as possibilidades de melhoria do projeto, caso se deseje dar continuidade a ele. Nesse sentido, indique possíveis ajustes ou melhorias arquiteturais, que possam vir a ser realizados futuramente.

Lições aprendidas (ex.):

1. xxxxxxxxxxxxxxxxxxxx
2. xxxxxxxxxxxxxxxxxxxx
3. xxxxxxxxxxxxxxxxxxxx

## ***Referências***

BROWN, Simon. **O modelo C4 de documentação para Arquitetura de Software**. Tradução de: Marcelo Costa. InfoQ, São Paulo, 01, ago. de 2018. Arquitetura. Disponível em: <https://www.infoq.com/br/articles/C4-architecture-model>. Acesso em: 13, fev. 2022.

PLANTUML. **Drawing UML with PlantUML**: PlantUML Language Reference Guide. Brest: PlantUML, 2021 p. 390. Disponível em: <https://plantuml.com/guide>. Acesso em: 14, fev. 2022.

MIHALCEA, Vlad. **High-Performance Java Persistence**. Victoria: Leanpub, 2021.

RAIBLE, Matt. **The Angular Mini-Book**. Toronto, Canadá: C4Media, 2022. *E-book*. Disponível em: <https://www.infoq.com/minibooks/angular-mini-book-v2/>. Acesso em: 10, abr. 2022.

SPRING, Guides. **Building REST services with Spring**. Disponível em: <https://spring.io/guides/tutorials/rest/>. Acesso em: 20 fev. 2022.

BOUCHERON, Brian. **Initial Server Setup with Ubuntu 20.04**. New York, NY: DigitalOcean, 23, abr. de 2020. Disponível em: <https://www.digitalocean.com/community/tutorials/initial-server-setup-with-ubuntu-20-04>. Acesso em: 13, abr. 2022.

## ***Apêndice***

URL do vídeo de apresentação da Etapa 1: <https://youtu.be/GZdCEQZyvSQ>.

URL do repositório do protótipo funcional: [Github - Sistema de Gestão de Ocorrência](#).

URL do vídeo de apresentação da Etapa 2: <https://youtu.be/yafyIPz1Nc4>.

URL do protótipo funcional disponível na Nuvem: <http://134.209.39.98:4200/>.  
Usuário: pucminas. Senha: 123456