

Region Metric: Um microserviço para o RegionALert e cidades inteligentes

Projeto final de programação

Gabriel Araujo - 2020948

Github do projeto: <https://github.com/gabrielaraujoc96/region-metric>

1. Introdução

O projeto final de programação, intitulado de “Region Metric: Um microserviço para o RegionALert e cidades inteligentes”, além de conter os diversos desafios necessários requisitados pelo escopo da matéria, tem como segundo intuito ser uma primeira prova de conceito para minha tese de mestrado “Os benefícios de uma arquitetura em microserviços para aplicações IoT”. O conceito por trás de aplicações da internet das coisas (IoT) envolve a conexão de diversos aparelhos (devices) ou "coisas" que se interagem para juntos, criarem uma aplicação que de valor ao usuário final. Através desse conceito clássico de IoT, a necessidade de uma arquitetura que consiga ser capaz de lidar com uma estrutura complexa de comunicação entre múltiplos sistemas, aparelhos, de forma independente e eficiente, não se encaixa no padrão monolítico. Por essa razão, para conseguirmos ter uma aplicação descentralizada, capaz de lidar com esses diferentes serviços, enorme quantidade de dados e complexa comunicação, a arquitetura orientada a serviços se encaixa perfeitamente nas necessidades de sistemas IoT. Ou seja, aplicações IoT e arquitetura orientada a serviços tem o mesmo objetivo: construir uma ou mais aplicações através de múltiplos serviços (como aparelhos e diferentes sistemas).

Apesar da semelhança de objetivos, a arquitetura orientada a microserviços tem sua maior utilização na quebra de um ou mais componentes (ou modelo) de um monolito, considerando as principais frentes que este componente utiliza: frontEnd, lógica de negócio, backend. No outro lado, para sistemas IoT, buscamos utilizar esses serviços, que podem ser diferentes tipos de aparelhos/devices, de forma a conseguir juntar estes sistemas, buscando uma interoperabilidade entre eles, para que a aplicação principal consiga atingir seu objetivo através desses múltiplos serviços. Ou seja, o uso comum de microserviços foca na quebra de um grande serviço em pequenos pedaços. Já no caso da utilização em sistemas IoT, buscamos utilizar vários pedaços que já estão supostamente separados e, através da arquitetura

orientada a microsserviços, criar um sistema que contenha todos os benefícios desta arquitetura, principalmente sua interoperabilidade.

Buscando então abordar esse tema de forma mais prática, o projeto Region Metric serve como exemplo onde utilizamos uma aplicação IoT já existente, o RegionALert. Seguindo a definição vista no artigo Supporting Multiple Smart-City Applications based on MUSANet, a Common IoMT Middleware [1] o RegionALert é uma aplicação que envia notificações de entidades externas como defesa civil, secretaria de turismo, entre outros, para usuários que utilizam a aplicação. Para isso é utilizado dois serviços webs para publicação de anúncios, um nó móvel gerador de dados do usuários e de dados de contexto, e uma aplicação mobile instalada no celular do usuário final que utiliza a infraestrutura provida pelo MUSANet [2].

Ao usar o RegionALert, usuários podem registrar áreas de interesse criadas pela administração da cidade, como áreas perto de sua casa, seu trabalho, ou rotas comuns do usuário. O celular do usuário mantém o sistema atualizado em relação a sua localização, utilizando a aplicação mobile instalada. Sempre que um alerta é criado para usuários em uma certa localidade, e o usuário está nessa área ou entra nessa área em algum momento no qual o alerta está ativo, a aplicação mobile gera um alerta através de texto, beep, vibração, dependendo da configuração escolhida pelo usuário. Um dos pontos principais do sistema criado é que, diferente da maioria dos diversos sistemas de alertas em cidades grandes, o RegionALert leva em consideração a localização do usuário.

Através do atual cenário existente do projeto, colocamos em prática um dos conceitos principais da utilização de microsserviços para aplicações IoT ao adicionar um novo serviço que agrega benefícios ao projeto atual do RegionALert, mantendo a interoperabilidade entre estes. O Region Metric então adiciona a possibilidade de coletar métricas dos alertas gerados e dados de confirmação por parte dos usuários de ponta, capacitando o projeto a reconhecer essas confirmações e a partir de então gerar métricas úteis sobre o atual uso da aplicação, além de ser capaz de prover a autoridades dados de quantas pessoas confirmaram estar ciente do alerta criado.

Esse documento aborda os pontos principais do desenvolvimento do projeto, como a criação da sua arquitetura, cenários de testes elaborados, descrição de seus componentes e boas práticas de código utilizadas.

2. Fundamentação Teórica

i. Monolitos

A arquitetura monolítica ainda é muito utilizada nos tempos atuais e tem sentido para muitas aplicações. Entre as vantagens de uma aplicação monolítica, ressalta-se a simplicidade da arquitetura, já que você precisa lidar apenas com um serviço e suas camadas internas, o seu rápido desenvolvimento por ser uma arquitetura mais simples e a unicidade da tecnologia usada, que acaba sendo necessário apenas conhecimento em uma única linguagem e tecnologia, facilitando a coesão dos times que o utilização. Entretanto, encontram-se desvantagens como a baixa escalabilidade e o preço para escalar uma aplicação, onde muitas vezes apenas um pedaço do serviço tem grande acesso, pela utilização do monolito, todo o sistema precisa ser escalado e isso tem um custo alto, uma base de código gigante para casos de sistemas grandes, já que todo o código e regras de negócio estão concentrados lá e também

a complexidade do entendimento devido a quantidade de informações centralizadas em um único local.

ii. Microserviços

Com o aumento da quantidade de sistemas que utilizam API e rotas para se comunicar, a palavra escalabilidade veio se tornando cada vez mais importante. Essa possibilidade de manter a disponibilidade do serviço mesmo com um aumento repentino de acessos, tem como nome escalabilidade horizontal, quando replicamos a mesma máquina ou serviço N vezes, até que o serviço aguarde a quantidade de requisições que está recebendo, dividindo entre esses serviços criados. Com esse novo conceito, as empresas começaram a adotar uma nova prática: separar a lógica de negócio em pequenos pedaços independentes que se completam. Com essa prática, criou-se o nome de arquitetura em microserviços a partir de um artigo publicado pelo Martin Fowler em seu site [3]. Em sua definição, Martin Fowler declarou essa arquitetura como "Uma abordagem que desenvolve um aplicativo único como uma suíte de pequenos serviços", ou seja, a junção de pequenos serviços que, juntos, formam uma aplicação. Entre as vantagens dessa arquitetura, a facilidade no desenvolvimento de apenas um pedaço da aplicação, agiliza o desenvolvimento e a entrega para um serviço específico do ecossistema da aplicação, os serviços são coesos e desacoplados, existe uma facilidade no deploy e na criação de testes para um serviço específico, além de ter a possibilidade de desenvolver as diferentes aplicações nas linguagens e tecnologias que façam mais sentido para tal.

Entretanto, entre seus problemas, a arquitetura orientada a serviços tende a ser mais complexa e com maiores complicações para documentar. Sam Newman listou em seu livro [4] um grupo de tecnologias básicas que são necessárias para a criação de uma boa arquitetura em microserviços:

- Desenvolvimento guiado a domínio - O foco no domínio das regras de negócio como centro do desenvolvimento de software, adicionando especialistas no negócio como Product Owners (dono do produto) mais perto do desenvolvimento.
- Entrega contínua - Considerar todo commit de código na branch principal do projeto como um commit que irá gerar uma nova entrega para produção. Para isso, é importante desenvolver partes como testes de performance e unitários, automação de pipeline, métricas sobre o software e alarmes.
- Utilização de diversas portas - Diferente da utilização comum em camadas de apresentação, controle e dados, microserviços tendem a ter diversas portas e adaptações para que seja possível interagir com suas várias camadas de contato, como database, cloud, front-end, integração, entre outros.
- Comunicação entre máquinas - É preciso que haja uma comunicação segura e garantida entre as máquinas existentes para o transporte de dados.

- Plataformas virtuais - Utilização de aplicações com containers como Docker, Kubernetes, entre outros.

Com essas tecnologias principais, a arquitetura voltada a microsserviços vem sendo a mais utilizada por gigantes da área, como Netflix, Amazon, Spotify, Twitter, entre outros. Focados na escalabilidade e manutenibilidade de suas plataformas, a migração para microsserviços e a criação de novos projetos já visando essa arquitetura já é considerado algo comum e a decisão correta a ser feita.

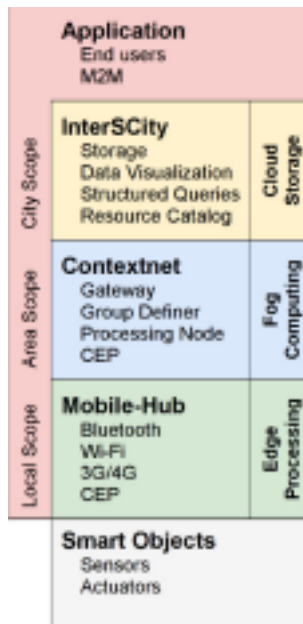
III. Aplicações IoT orientadas a serviço

O padrão de comunicação entre aplicações mais utilizado atualmente é utilizando API RESTful. Essa é uma escolha feita geralmente para serviços que precisam se comunicar, porém com uma troca de dados não tão constante, apenas para executar algum comando ou devolver algum valor. No caso de aplicações IoT, essa passagem de dados não se faz de forma simples. Os dados coletados por aparelhos nas pontas, como sensores, geralmente sofrem alterações constantes e devem estar sendo enviados para o serviço da aplicação a todo instante, com garantia de entrega e em uma velocidade rápida. A partir dessa necessidade, a utilização de uma arquitetura orientada a serviços em conjunto com protocolos de troca de mensagens como MQTT, se tornou uma escolha óbvia para esses tipos de aplicação.

Como exemplo dessa necessidade, temos cidades inteligentes que coletam dados de seus moradores. Esses dados devem ser compartilhados em diversos serviços que mantêm uma comunicação entre si, como por exemplo, um serviço de segurança que vê se a pessoa não tem uma ficha criminal em aberto, um serviço de dados para coletar a informação de possíveis trajetos que aquela pessoa faz como forma de entender fluxos na cidade, um serviço de transportes que pode avisar para um usuário a caminho de um ponto ônibus a distância a que distância o ônibus está, entre outras aplicações diversas possíveis com aplicações IoT. Neste cenários, microsserviços podem garantir resiliência e escalabilidade, garantindo baixo acoplamento na relação desses diversos serviços. Por esse motivo, aplicações IoT tem como escolha arquitetura orientada a serviços como uma escolha fácil, já que ela trás de benefício fatores chaves para que a aplicação consiga atingir o esperado.

IV. MUSANet [2]

MUSANet é uma aplicação multicamada distribuída com reconhecimento de contexto para capturar, armazenar e processar dados de sensores, receber e enviar informações através de protocolos publish-subscribe. A arquitetura criada foi dividida em 3 camadas seguindo a figura:



A parte superior é baseada em um ecossistema distribuído de armazenagem escalável na nuvem. Na sua implementação, faz-se o uso do InterSCity [6], uma aplicação open-source que prove blocos básicos de desenvolvimento para aplicações relacionadas a cidades inteligentes através de APIs REST.

A camada do meio implementada pelo ContextNet [7] na área de "*fog computing*", incluindo uma inter-conexão de gateways de serviços distribuídos através da cidade, possibilitando que aparelhos como celulares se conectem ao sistema. O ContextNet possibilita a comunicação com aparelhos de ponta através de mensagens em uma comunicação direto ou em grupo. Dessa forma, através das configurações criadas na camada do ContextNet a aplicação pode utilizar de nós de processamento ou grupos de nós de processamento e pontos de inter-conexão espalhados pela cidade para escolher a melhor ponte de contato entre o serviço e o nó móvel final.

Na camada final, o Mobile-Hub [8] é um middleware que roda em um aparelho Android, como celulares, e consegue descobrir e conectar a sensores e atuadores, funcionando como um "*hub*" de dados e comandos vindo da camada central. Ele também suporta análise e processamento online de dados recebidos dos sensores conectados.

Esses três levels do MUSANet permite que nós de ponta que processem dados e eventos com latência perto de zero, tendo o ContextNet mais próximo processando eventos vindo de serviços na nuvem.

3. Análise e especificação de requisitos

Escopo e requisitos

Para ajudar na criação do projeto, entender a sua finalidade, características e funcionalidades, foi utilizado o levantamento de requisitos funcionais e não funcionais do projeto, listados a seguir:

Requisitos funcionais:

- **RF 01** - O sistema deve permitir ao usuário o acesso aos dados salvos em seu banco de dados.
- **RF 02** - O sistema deve permitir a visualização aos dados salvos em seu banco de dados através de uma plataforma simples, disponibilizada pelo sistema.
- **RF 03** - O sistema deve salvar um histórico de alertas e confirmação de alertas recebidos por este.
- **RF 04** - O sistema deve permitir ao usuário que acesse o sistema de visualização de dados sem fazer um cadastro.
- **RF 05** - O sistema deve permitir a adição de novos alertas.
- **RF 06** - O sistema deve permitir a adição de novas confirmações de recebimento de alerta.

Requisitos não funcionais:

- **RNF 01** - O sistema deve estar totalmente dockerizado.
- **RNF 02** - O sistema deve utilizar um banco de dados MongoDB para salvar os dados do projeto.
- **RNF 03** - O sistema deve utilizar da aplicação Metabase para gerar uma visualização de dados simples.
- **RNF 04** - O sistema deve ser capaz de gerar um relatório de testes unitários automaticamente através de um comando.
- **RNF 05** - O sistema não deve conter dependência de nenhuma outra aplicação para que seu funcionamento correto ocorra.
- **RNF 06** - O sistema deve gerar um login e senha para utilização da aplicação do Metabase automaticamente e prover ao usuário.
- **RNF 07** - O sistema deve ser capaz de validar as requisições feitas para tal, garantindo que o payload seja válido.
- **RNF 08** - O sistema deverá conter um script para que seja feito o "setup" inicial da aplicação de forma completa, para que não haja necessidade de alteração por parte do usuário.

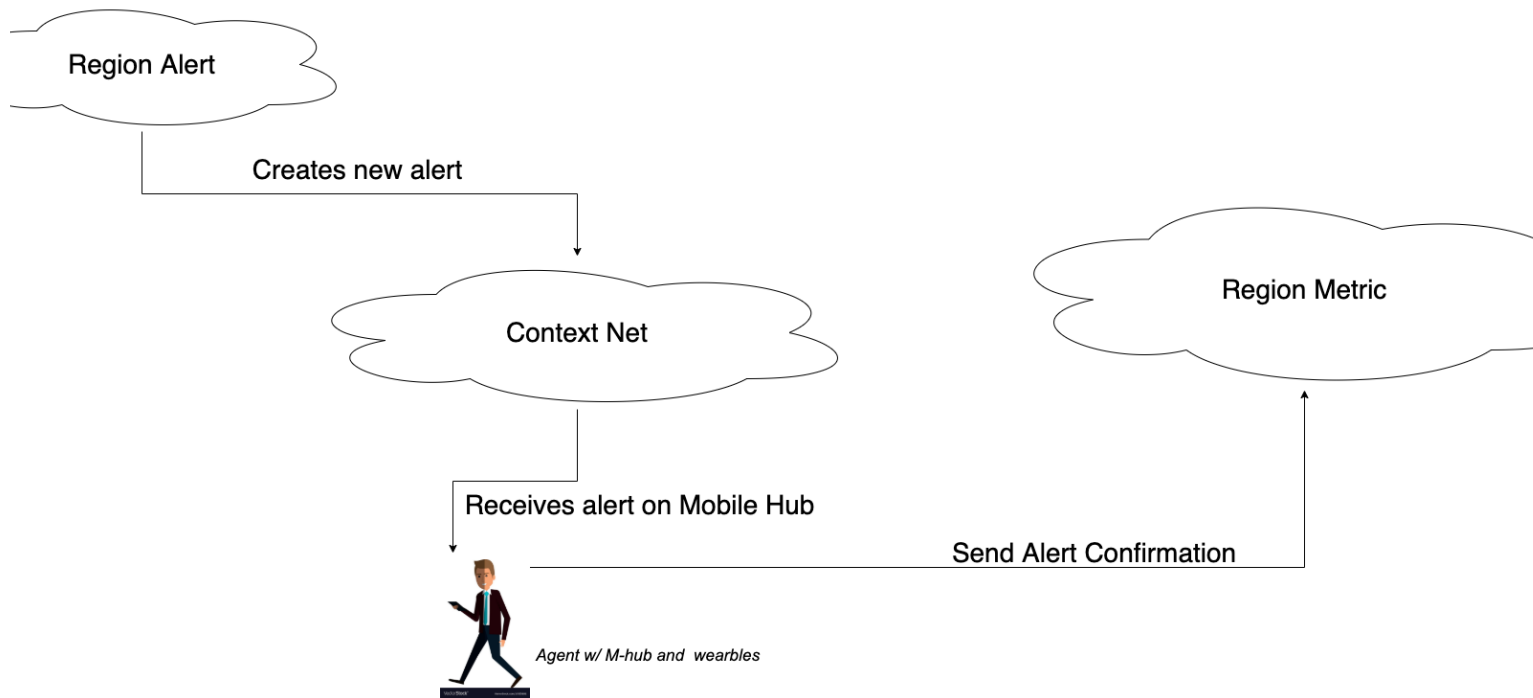
Após esse levantamento, o escopo do projeto pode ser bem definido e, dessa forma, tornou-se possível a divisão de tarefas para que o objetivo do projeto fosse atingido.

4. Projeto do programa

Arquitetura e projeto

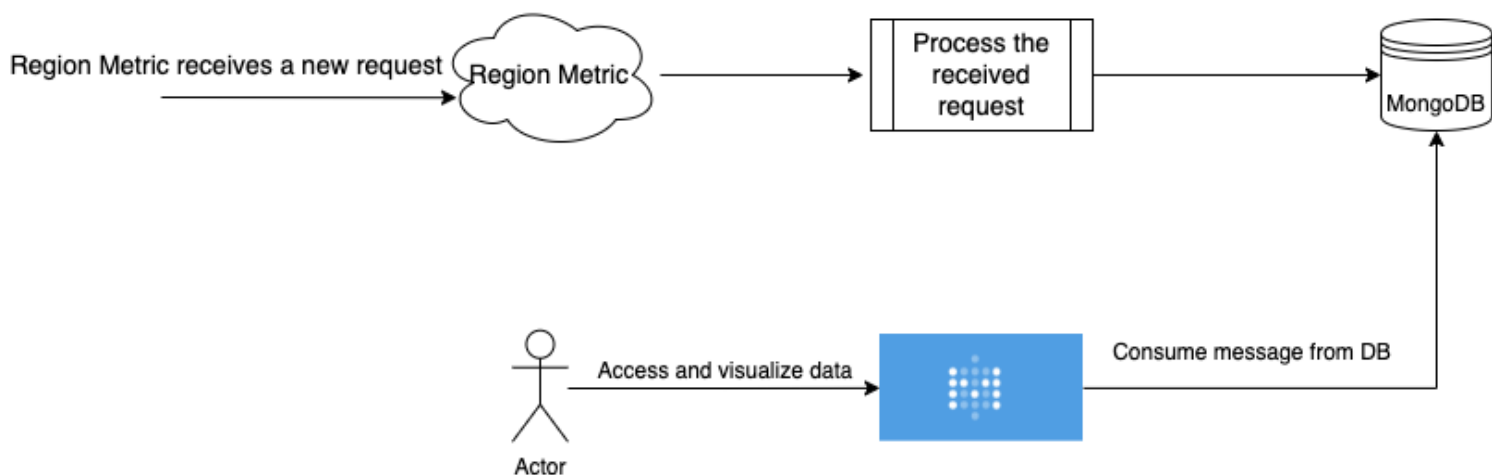
O projeto Region Metric se baseia no ecossistema do RegionALert e MUSANet, que utiliza também do ContextNet e do Mobile Hub. Como a intenção desse documento não é

dar uma explicação tão detalhada desses, a seguir há uma imagem que mostra como seria a comunicação desde o RegionAlert até chegar no Region Metric de forma geral, sem entrar em detalhes:

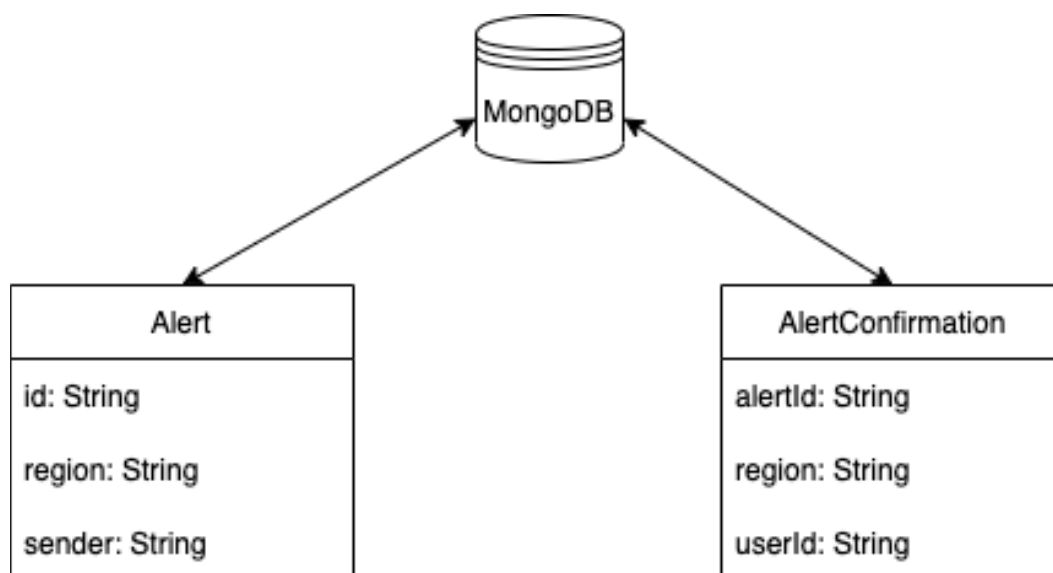


Comunicação RegionALert - Region Metric

Outra peculiaridade do sistema criado é que ele utiliza não apenas de um serviço capaz de receber requisições sobre alertas, confirmações, entre outros, mas também é responsável por criar e armazenar esses dados recebidos, e promover uma forma simples de exibição desses dados através do Metabase. A imagem a seguir mostra exatamente como funciona essa comunicação interna entre sistema - banco - aplicação, até o acesso do usuário.



Um dos componentes do projeto é a utilização do MongoDB um banco não relacional que fornece uma liberdade bem grande em relação aos dados que estão sendo adicionados no banco. Sendo assim, os dados persistidos nele podem ser de qualquer tipo, sem restrição ao serem adicionados - apesar das requisições obrigarem campos mínimos necessários para a criação de alguns, como alertas. O projeto foi criado em seu primeiro momento com dois pontos iniciais, alertas e confirmação de alertas, que são visualizados através do Metabase, que faz o “parseamento” desses dados de forma automática na hora da visualização, sendo assim, não foi necessário a criação de classes de modelo para esse projeto, pelo menos não até o atual momento dele. De toda forma, para dar uma visualização básica desses dois pontos principais e seus campos obrigatórios segue um desenho de como seria esses modelos:



Estrutura de dados Alert e AlertConfirmation

A divisão de código do projeto foi utilizado um padrão comum na criação de microsserviços focados em backend, criando uma área para Routes, Controllers, Connectors e Schemas. Esses grupos tem responsabilidades únicas e bem claras, sendo elas:

- Routes - Responsável por todas as rotas de acesso externo (HTTP) ao sistemas.
- Controllers - Responsável pelas classes de controles, que tem as lógicas e regras de negócio da aplicação e são acessadas através das classes de rotas.

- Connectors - São as classes responsáveis por configurar, montar e executar todas as conexões internas e externas e a forma como essas são feitas, desde a conexão com o banco, até conexões HTTP e suas particularidades.
- Schemas - Todos os schemas .json do projeto são agrupados nessa pasta, como por exemplo, schemas de validação de requisições.

5. Código fonte

O código fonte foi disponibilizado através do Github <https://github.com/gabrielaraujoc96/region-metric>. O código contém comentários iniciais em todas as suas classes e segue as principais e melhores práticas de programação, além disso, também foi feito a utilização do Docker para manter o projeto utilizando containers e facilitando a sua execução tanto local, quanto na nuvem. A seguir, alguns exemplos do código fonte que mostram a utilização dessas boas práticas de código e arquitetura.

1. Comentários iniciais identificando o autor

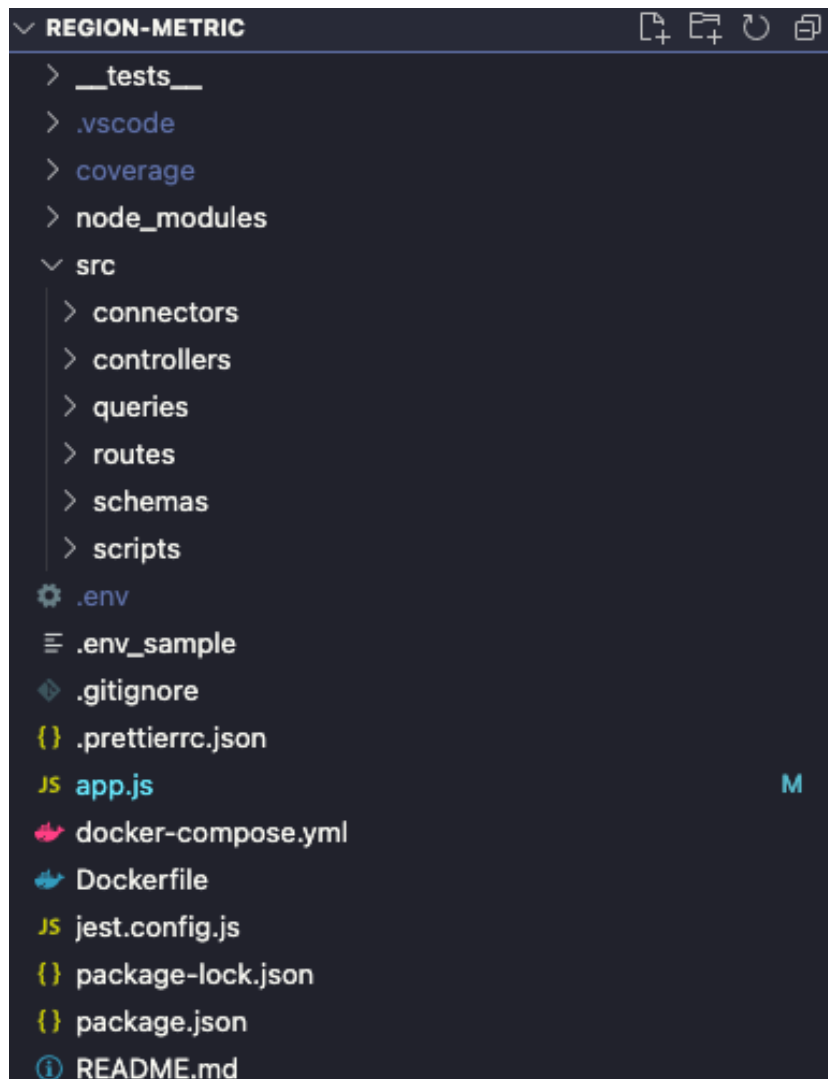
```
src > controllers > JS alertConfirmation.js > ...
1  /**
2  //
3  //  alertConfirmation.js
4  //  Region Metric
5  //
6  //  Created by Gabriel Araujo on 23/01/2022.
7  //
8  ****/
```

2. Cabeçalho de classes e módulos

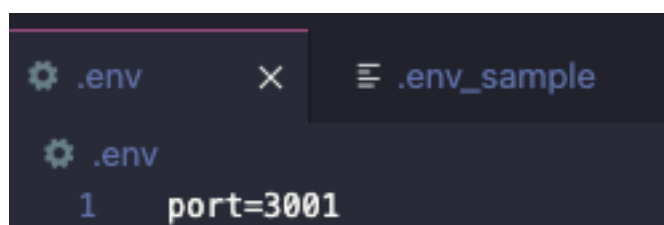
```
/* ****
**
**  MARK: Variables declaration
**
*****/
const dbQueries = require("../queries/transactionQueries");
const MongoConnector = require("../connectors/MongoConnector.js");
const mongoConnection = new MongoConnector();

/* ****
**
**  MARK: This class is responsible to do all the business logical withing the Alert Confirmation, like getting
**  an alert confirmation, creating and listing all alerts confirmations.
**
*****/
class AlertConfirmation {
```

3. Padrões de arquivos



4. Utilização de arquivo de variável de ambiente



5. Utilização de docker

```
🐳 docker-compose.yml
1  version: "3.7"
2
3  services:
4    region:
5      container_name: region-metric
6      build:
7        context: .
8        dockerfile: Dockerfile
9      ports:
10     - 3001:3001
```

6. Utilização de blocos try-catch

```
try {
  const controller = new AlertConfirmation();
  const response = await controller.get(alert_id);

  ctx.status = 200;
  ctx.body = response;
} catch (err) {
  console.log(err);
  ctx.status = 500;
  ctx.body = { response: 'Something went wrong, we are sorry' };
}
```

A criação de um código fonte limpo, seguindo boas práticas, é essencial para que seja possível manter uma manutenibilidade no projeto, uma fácil compreensão de seu escopo, além de outros benefícios, como é demonstrado no livro Clean Code[5].

6. Roteiro de testes efetuado

A realização de testes do projeto se divide em dois grupos diferentes: testes unitários de código e testes de sistema. Esse capítulo irá ser dividido entre esses dois grupos, onde

iremos falar mais sobre seus processos, critérios e execuções, para **garantir o controle de qualidade do sistema**.

Testes unitários

O primeiro grupo de testes criados foram os testes unitários que garantem o bom funcionamento do código. Para isso, foi utilizado uma biblioteca de mock, que garante a possibilidade de testar apenas a classe específica que eu desejo testar, “mockando” e retirando qualquer dependência, criando assim um teste unitário.

Os testes unitários foram focados nas classes do grupo de Connectors, Routes e Connectors. Naturalmente, as classes de conexão e de queries dos bancos de dados não puderam ser testadas pois os testes rodam de forma paralela ao serviço e, dessa forma, não haveria como testar essas requisições e conexões com o banco, já que esse não estaria disponível. Os critérios de testes para as classes foi que os testes cobrissem todas as funções dessas classes (se possível serem testadas) e que os testes deveriam garantir critérios fundamentais do código. Um exemplo é a validação de dados de entrada de uma requisição, ou seja, se uma requisição é feita com um payload inválido, o código deve verificar isso e validar. Esse é um ótimo cenário de exemplo de caso de teste unitário que podemos utilizar, para garantir que o código age de forma como esperamos.

Sendo assim foram criados 4 suits de testes e 10 testes para a versão inicial do projeto, que podem ser executados de forma automatizada executando o comando **“npm test”**, que executa um script que testa todos os arquivos e, além disso, gera um output verificando para os arquivos disponíveis, qual a cobertura de testes atuais. No momento do fim da escrita dessa seção esse era o resultado desse output:

```
PASS __tests__/controllers/alert.spec.js
PASS __tests__/controllers/alertConfirmation.spec.js
PASS __tests__/routes/alertConfirmation.spec.js
PASS __tests__/routes/alert.spec.js
```

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	56.7	36.66	55	58.86	
connectors	54.54	50	50	54.54	
MongoConnector.js	54.54	50	50	54.54	27,34-40
controllers	84.61	50	100	84.61	
alert.js	84.61	50	100	84.61	35-38,54,70
alertConfirmation.js	84.61	50	100	84.61	38,55,69-72
queries	4	0	0	5.26	
transactionQueries.js	4	0	0	5.26	18-76
routes	55.26	100	33.33	55.26	
alertConfirmation.js	53.84	100	33.33	53.84	26-37,42-49,74-76
alerts.js	56.75	100	33.33	56.75	26-35,40-47,73-75

Testes de sistema

Os testes de sistema se resumem em testes manuais do sistema, para garantir que todas as etapas do sistema estejam funcionando de forma completa. Para isso, foi gerado uma lista de testes de casos que deveriam ser visitados para garantir o perfeito funcionamento do sistema, baseando nos requisitos funcionais que foram ditos anteriormente, e como aplicar esses testes. Além disso, nesse documento irei adicionar o resultado desses testes. Todos os testes assumem que o projeto está sendo executado e já foi configurado utilizando o script de setup. Os critérios para motivação dos testes foram:

1. Funcionalidade
2. Conectividade
3. Confiabilidade
4. Usabilidade
5. Eficiência

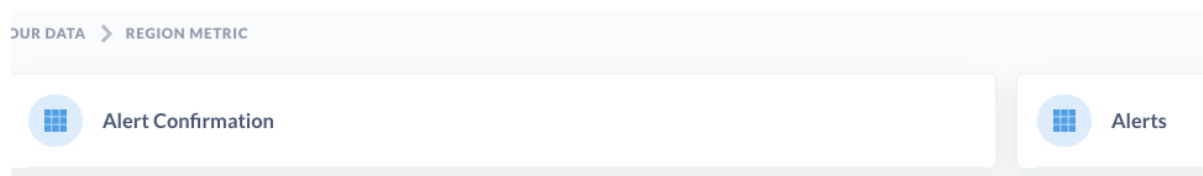
Os testes a seguir foram os testes escolhidos para garantir esses critérios de funcionamento do sistema:

5.1.O sistema permite acesso a informações do banco de dados de forma simples

5.1.1.O usuário deverá acessar a aplicação do Metabase.

5.1.2.A aplicação deve exibir uma tabela com os Alertas e outra tabela com as Confirmações de alertas criadas.

Resultado: OK



5.2.O sistema permite visualizar todos os alertas já criados

5.2.1.O usuário deverá acessar a aplicação do Metabase.

5.2.2.O usuário deverá acessar a área de Our Data e clicar na tabela “Alerts”.

Resultado: OK

5.3.O sistema permite visualizar todos as confirmações de alertas já criados

Region Metric • Alerts

ID ▾	ID ▾	Region ▾	Alerts ▾	Sender ▾	Alert ID ▾	User ID ▾
61f0b70e58e224001280eb49	abc-123	zona-sul	100	user-id		
61f0b79a70c7850012e1549a	abc-123	zona-sul	100	user-id		
61f0be959490f7001327155f		zona-sul			abc-123	user-id
61f0c0214aa37e0013ab3d4a		zona-sul			abc-123	user-id
61f1a051a0d5a300132129b4	abc-test	zona-sul			abc-test	user-id
61f20eda2f9f340012b8b627	abc-test	zona-sul			abc-test	user-id

- 5.3.1.O usuário deverá acessar a aplicação do Metabase.
- 5.3.2.O usuário deverá acessar a área de Our Data e clicar na tabela “Alerts Confirmation”.

Resultado: OK

Region Metric • Alert Confirmation

ID ▾	Alert ID ▾	Region ▾	User ID ▾
61f0c060200bdf00135ad045	abc-123	zona-sul	user-id
61f0dffe9bc3219001216ec95	abc-1234	zona-sul	user-id

- 5.4.O sistema permite visualizar todas as confirmações para um alerta específico
 - 5.4.1.O usuário deverá acessar a aplicação do Metabase.
 - 5.4.2.O usuário deverá acessar a área de Our Data e clicar na tabela “Alerts Confirmation”.
 - 5.4.3.O usuário deve clicar na coluna “Alert ID”e buscar por um id específico (nesse exemplo id = abc-123).

Resultado: OK

Region Metric • Alert Confirmation

 Alert ID is abc-123 ✕

ID ▾	Alert ID ▾	Region ▾	User ID ▾
61f0c060200bdf00135ad045	abc-123	zona-sul	user-id

- 5.5.O sistema mantém salvo alertas e confirmações de alertas previamente recebidos
- 5.5.1.O usuário deverá acessar a aplicação do Metabase.
 - 5.5.2.O usuário deverá acessar a área de Our Data e clicar na tabela “Alerts Confirmation”.
 - 5.5.3.O usuário deve clicar na coluna “Alert ID” e buscar por um id específico (nesse exemplo id = abc-123) e verificar a existência de uma confirmação de alerta para esse alerta específico.
 - 5.5.4.O usuário deve fechar a aplicação, cancelar o processo que está rodando o projeto através do comando `ctrl + c`.
 - 5.5.5.O usuário deve rodar novamente o processo da aplicação e subir do zero suas configurações através do comando `"docker-compose up --build"`.
 - 5.5.6.O usuário deve repetir os passos 5.5.1 há 5.5.3.

Resultado: OK*

*Para esse caso não há como promover válida evidência.

- 5.6.O sistema garante um cadastro único gerado para acesso livre
- 5.6.1.O usuário deverá subir o sistema utilizando o comando `"docker-compose up"`.
 - 5.6.2.O usuário, através de um terminal, deverá executar o script `setupMetabase`, como descrito no ReadMe.
 - 5.6.3.O usuário deverá acessar o Metabase e, ao ser requisitado um login e senha, deverá colocar os dados providos pelo script.

Resultado: OK

```
> npm run setupMetabase

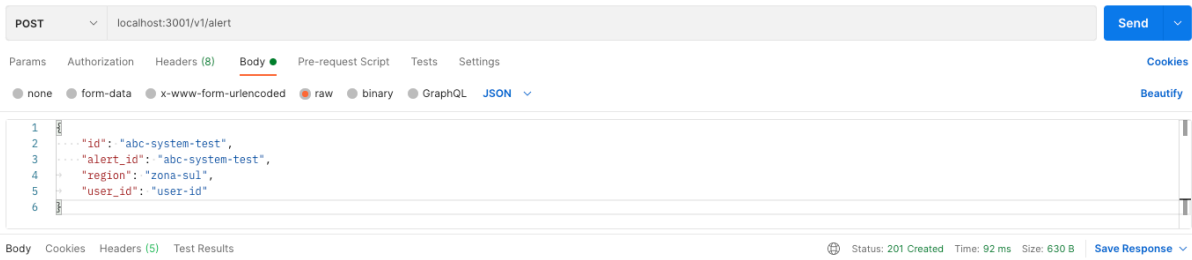
> region-metric@1.0.0 setupMetabase
> node src/scripts/setupMetabase.js

Setting up metabase, please wait, this may take a few seconds
Setup finish. Access Metabase via localhost:3000
To authenticate, use email: region@metric.com, password: 123456
Have fun ☺:
```

- 5.7.O sistema permite adição de novos alertas
- 5.7.1.Através de um serviço de requisição como Postman ou CURL via terminal, o usuário deve fazer uma requisição para o endpoint de alerta, utilizando um payload válido.
 - 5.7.2.O usuário deverá acessar a aplicação do Metabase.

5.7.3.O usuário deverá acessar a tabela de “Alerts” e verificar se seu alerta está lá.

Resultado: OK



POST localhost:3001/v1/alert

Body

```
1 {
2   "id": "abc-system-test",
3   "alert_id": "abc-system-test",
4   "region": "zona-sul",
5   "user_id": "user-id"
6 }
```

Status: 201 Created Time: 92 ms Size: 630 B Save Response

Region Metric • Alerts

Alert ID is abc-system-test x

ID	ID	Region	Alerts	Sender	Alert ID	User ID
61f34c1e6783c300126958fc	abc-system-test	zona-sul			abc-system-test	user-id

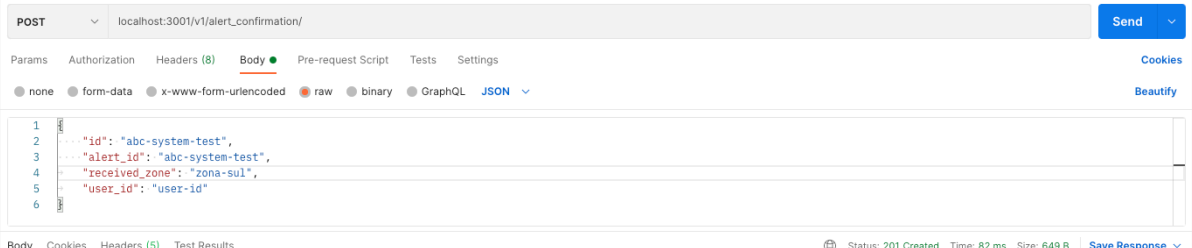
5.8.O sistema permite adição de novas confirmações de alertas recebidos.

5.8.1.Através de um serviço de requisição como Postman ou CURL via terminal, o usuário deve fazer uma requisição para o endpoint de confirmação de alertas, utilizando um payload válido.

5.8.2.O usuário deverá acessar a aplicação do Metabase.

5.8.3.O usuário deverá acessar a tabela de “Alert Confirmations” e verificar se sua confirmação de alerta está lá.

Resultado: OK



POST localhost:3001/v1/alert_confirmation/

Body

```
1 {
2   "id": "abc-system-test",
3   "alert_id": "abc-system-test",
4   "received_zone": "zona-sul",
5   "user_id": "user-id"
6 }
```

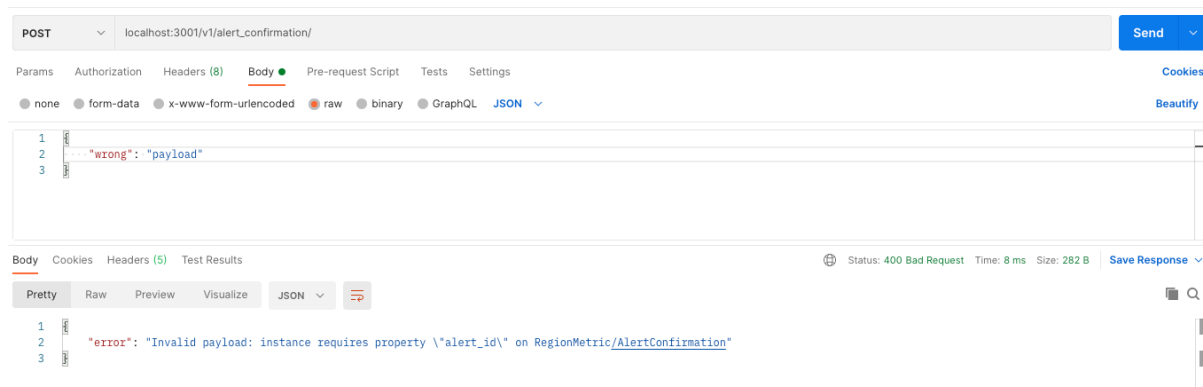
Status: 201 Created Time: 82 ms Size: 649 B Save Response

5.9.O sistema recusa requisições com payload inválidos

5.9.1.Através de um serviço de requisição como Postman ou CURL via terminal, o usuário deve fazer uma requisição para o endpoint de confirmação de alertas, utilizando um payload inválido.

5.9.2.O usuário deverá receber da aplicação uma informação de erro como resposta pela requisição.

Resultado: OK



A cobertura de testes apresentadas consegue garantir que as funcionalidades principais do sistemas estão em funcionamento de forma correta, tanto por meio dos testes de sistema quanto por meio dos testes unitários.

7. Documentação para o usuário

A documentação de utilização do projeto se encontra disponível na página ReadMe do projeto no Github ([README.md](#)). Por lá, existe uma explicação detalhada do intuito e de como utilizar o projeto. Além disso, para facilitar o uso do projeto devido a complexidade do setup do MongoDB e do Metabase, foi criado um script chamado setupMetabase que ao ser executado configura toda a aplicação e garante um login e senha de acesso a aplicação do Metabase. Também está na pagina de documentação do projeto como executar esse script.

Assim como dito anteriormente, os testes unitários do projeto rodam através de um comando **“npm test”**. NPM é a principal biblioteca de pacotes do NodeJS, e é possível obter acesso a mais informações através da sua documentação [9]

8. Referências bibliográficas

[1] A. Meslin, N. Rodriguez and M. Endler, Supporting Multiple Smart-City Applications based on MUSANet, a Common IoMT Middleware, Workshop em Clouds e Aplicações (WCGA), Jan 2020.

[2] A. Meslin, N. Rodriguez and M. Endler, Scalable Mobile Sensing for Smart Cities: The MUSANet Experience, IEEE Internet of Things Journal, Feb 2020.

[3] Fowler, M. Microservices: a definition of this new architectural term. <https://martinfowler.com/articles/microservices.html>, March 2014.

[4] Newman, S. Building Microservices. O'Reilly Media, 2015.

[5] Martin, Robert C. Clean Code: A Handbook of Agile Software **Craftsmanship**. Upper Saddle River, NJ: Prentice Hall, 2009.

[6] D. M. Batista, A. Goldman, R. Hirata, F. Kon, F. M. Costa, and M. Endler, "InterSCity: Addressing future Internet research challenges for smart cities," in Proc. 7th Int. Conf. Netw. Future (NOF), Buzios, Brazil, 2016, pp. 1–6. [Online]. Available: <http://ieeexplore.ieee.org/abstract/document/7810114/>

[7] M. Endler et al., "ContextNet: Context reasoning and sharing middleware for large-scale pervasive collaboration and social networking," in Proc. Workshop Posters Demos Track, Lisbon, Portugal, 2011, pp. 1–2.

[8] L. E. Talavera, M. Endler, I. Vasconcelos, R. Vasconcelos, M. Cunha, and F. J. D. S. e Silva, "The mobile hub concept: Enabling applications for the Internet of Mobile Things," in Proc. IEEE Int. Conf. Pervasive Comput. Commun. Workshops (PerCom Workshops), St. Louis, MO, USA, 2015, pp. 123–128. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/7134005>

[9] NPM Docs, npm Docs, 2022, Downloading and installing Node.js and npm. Disponível em: <https://docs.npmjs.com/downloading-and-installing-node-js-and-npm#using-a-node-version-manager-to-install-node-js-and-npm>. Acesso em: 30 de Jan. de 2022.