

# Region Metric: Um microserviço para o RegionALert e cidades inteligentes

Projeto final de programação

Gabriel Araujo - 2020948

## 1. Introdução

Para o projeto final de programação, através de conversas com meu orientador Markus Endler, chegamos a conclusão que poderíamos criar um projeto que envolvesse os desafios necessários requisitados pelo escopo da matéria, mas que também pudesse ser uma primeira prova de conceito para a tese que eu pretendo defender. A ideia por trás da minha tese é demonstrar os benefícios da utilização de microserviços para aplicações IoT, sem querer me estender muito então do foco principal deste documento, o projeto aqui criado serve também como prova de conceito para a minha tese.

Utilizando como base o ecossistema do projeto criado na tese de doutorado do Prof. Alexandre Meslin, Region Alert - <https://github.com/meslin8752/RegionAlert>, a criação do projeto Region Metric vem para mostrar como podemos criar microserviços independentes que, ainda assim, estão dentro de um contexto igual e geram valor para o ecossistema criado.

O projeto Region Metric então viabiliza a criação de diversos endpoints que servem de coleta de dados gerados a partir do Mobile Hub (ponta de recebimento de alertas por parte do Region Alert), e, através de um projeto totalmente dockerizado, gera um banco de dados em MongoDB que guarda todos esses dados recebidos e ainda trás junto a criação de uma aplicação Metabase para que seja possível fazer a análise de todos esses dados de forma simples, além de todas as features entregues pelo Metabase. Acho que um ponto importante a ressaltar é a complexidade existente em criar um projeto do zero que contenha um banco de dados funcionando e uma aplicação para consumir esse banco, além de ter tudo automatizado e dockerizado, pronto para ser usado apenas inserindo uma linha de comando. Essas "features" sem dúvidas foram pontos chaves e desafios intensos para o projeto, mas que, na minha opinião, adicionam um valor enorme para o projeto.

Acredito que o projeto serve tanto para o objetivo principal da disciplina de averiguar se o aluno sabe empregar técnicas eficazes para especificar, projetar, desenvolver, controlar a qualidade e documentar programas, criando um projeto com um nível de complexidade e

que ainda gera valor para um projeto real, além de ser um artefato útil para minha atual pesquisa.

Por último, acredito que possa se encontrar muito valor na criação desse projeto, até para melhorias futuras após disciplina, como continuar com sua utilização no estudo da minha tese. De toda forma, acredito que através dele pude ser capaz de apresentar os aspectos de criação de um projeto demonstrando conhecimento nos aspectos esperados.

## 2. 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:

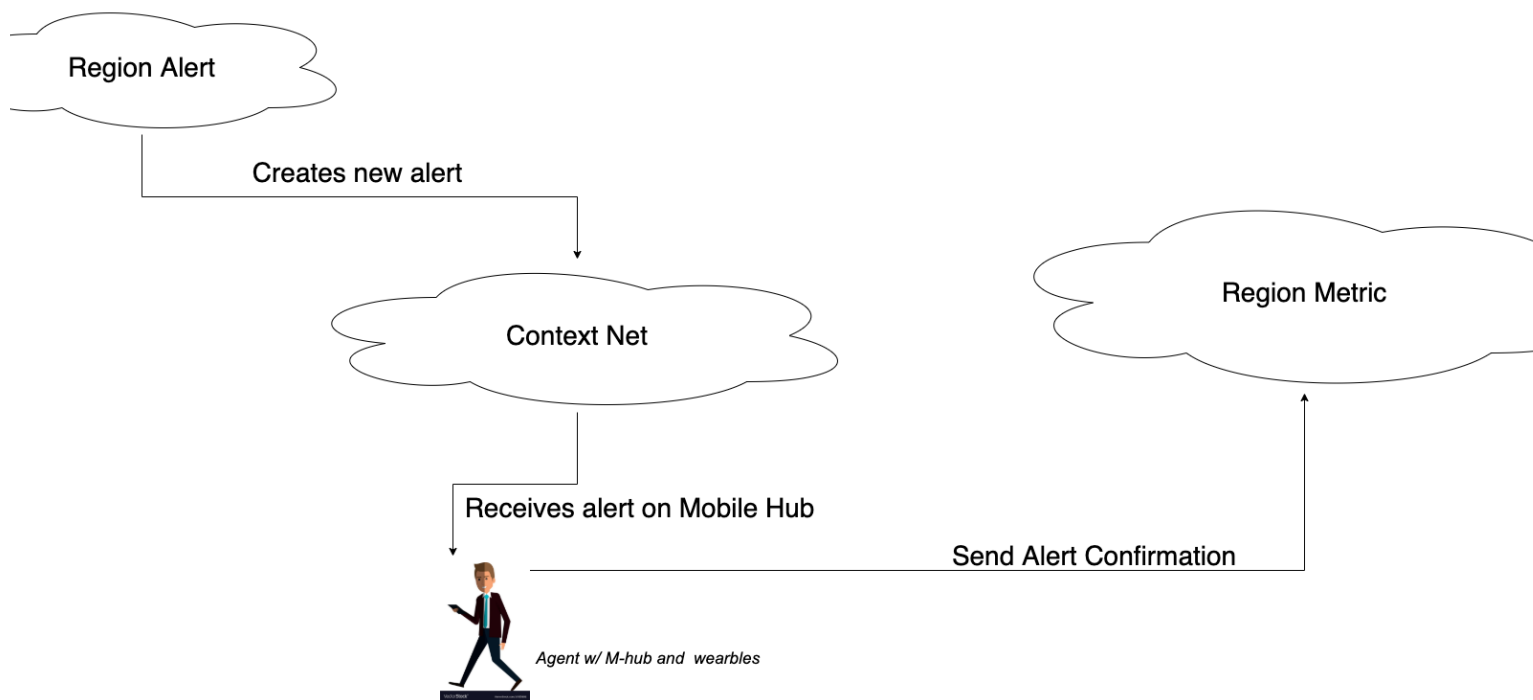
- **RNF 01** - O sistema deve estar totalmente dockerizado.
- **RNF 02** - O sistema deve utilizar um banco de dados MongoDB para salvar os dados do projeto.
- **RNF 03** - O sistema deve utilizar da aplicação Metabase para gerar uma visualização de dados simples.
- **RNF 04** - O sistema deve ser capaz 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, acredito que o escopo do projeto ficou mais fácil de ser visualizado e, dessa forma, foi possível criar diversas tarefas separadas para que fosse possível atingir os objetivos do projeto. Para a criação dessas tarefas eu utilizei de uma ferramenta chamada Trello, que me ajuda no levantamento de tarefas e na organização dessas.

### 3. Projeto do programa

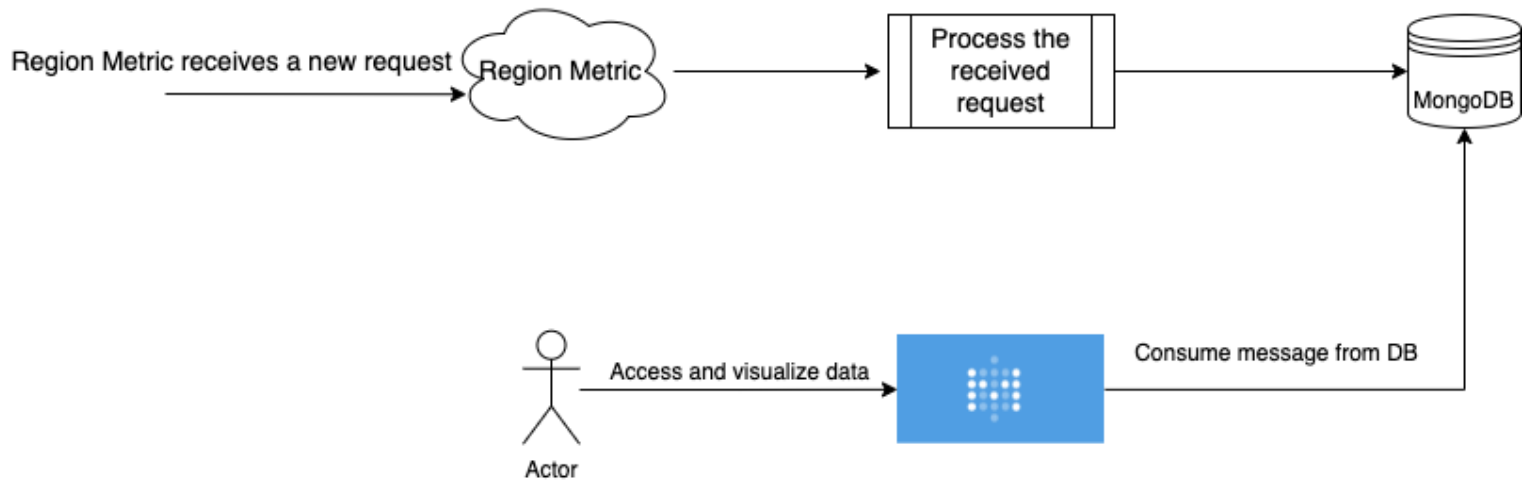
#### Arquitetura e projeto

Como dito anteriormente, esse projeto se baseia no ecossistema do Region Alert, 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 Region Alert até chegar no Region Metric de forma geral, sem entrar em detalhes:



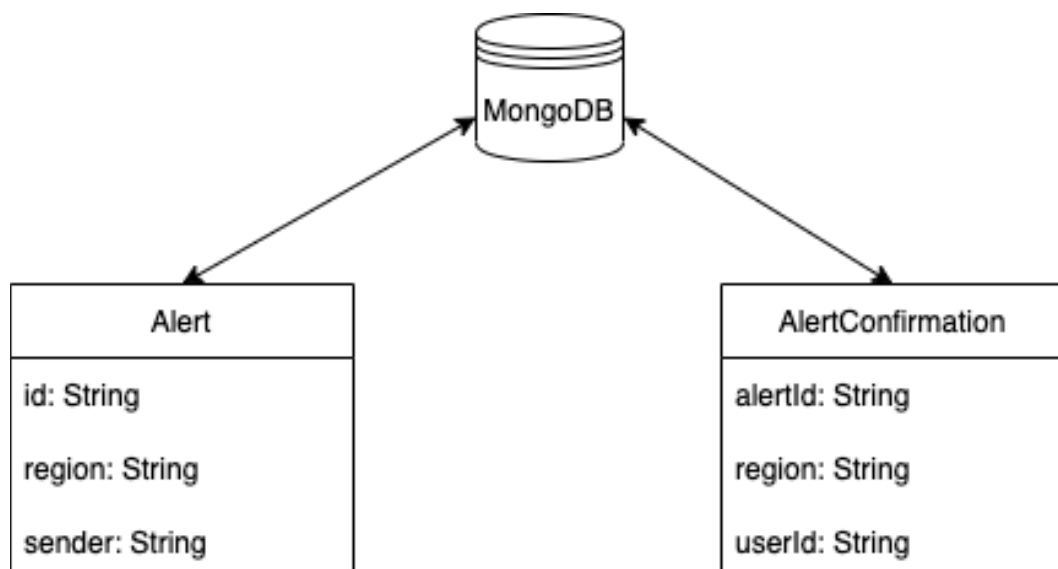
Comunicação Region alert - Region Metric

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 acima mostra exatamente como funciona essa comunicação interna entre sistema - banco - aplicação, até o acesso do usuário.



### Comunicação interna entre serviço-banco-aplicação

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:



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 responsabilidade 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.

## 4. 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.

Acredito que essa área do projeto possa conter alguns exemplos do código e de suas boas práticas, porém para uma melhor visualização recomendo abrir o projeto e que o leitor faça sua busca e entenda melhor sob. A seguir então vou adicionar pontos que acho que são cruciais para a garantia da qualidade de código.

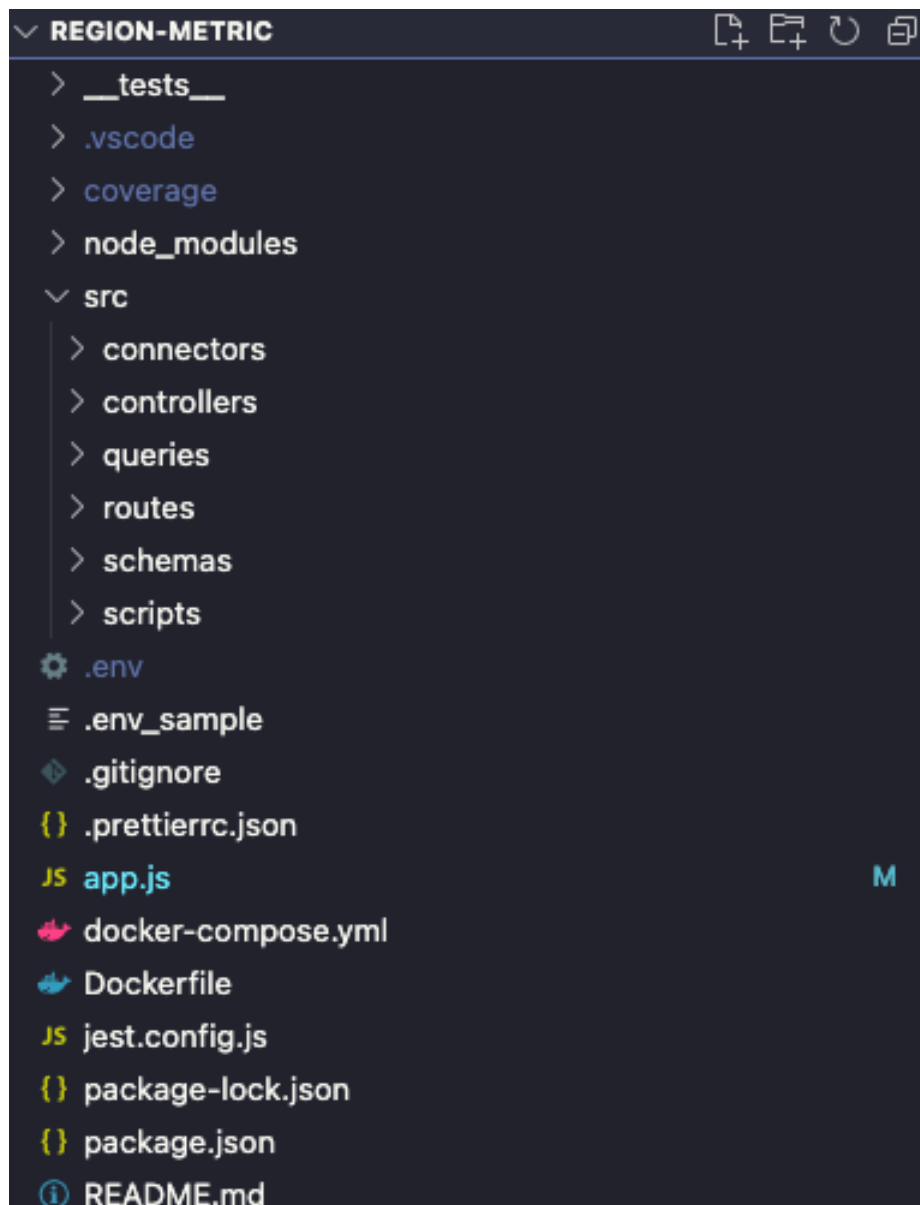
### 1. Comentários iniciais identificando o autor

```
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  *****/
```

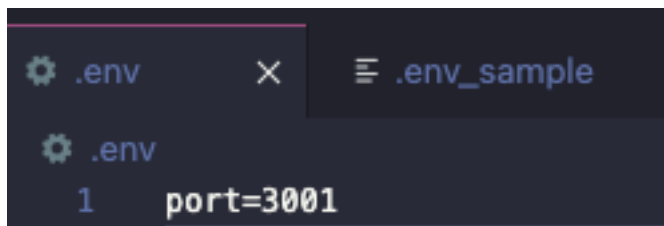
## 2. Cabeçalho de classes e módulos

```
/* *****  
**  
** 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 {
```

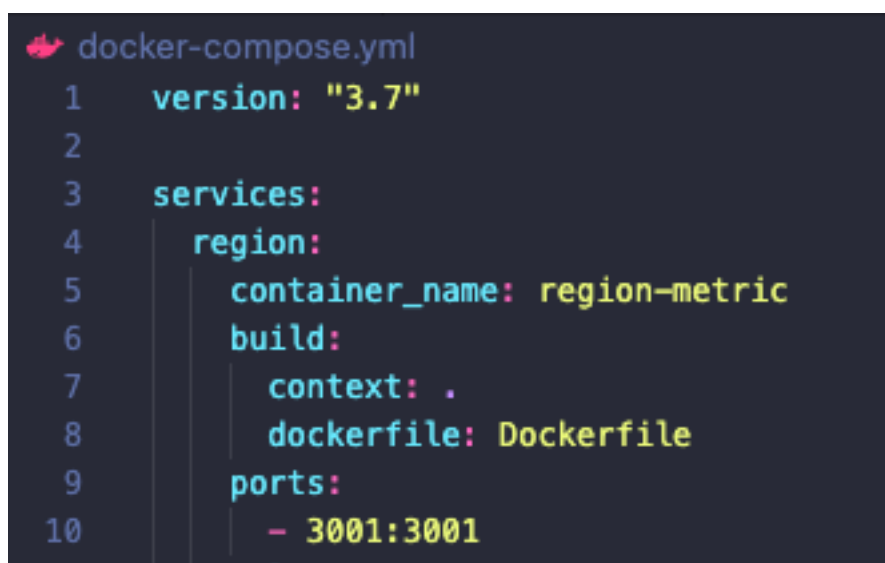
## 3. Padrões de arquivos



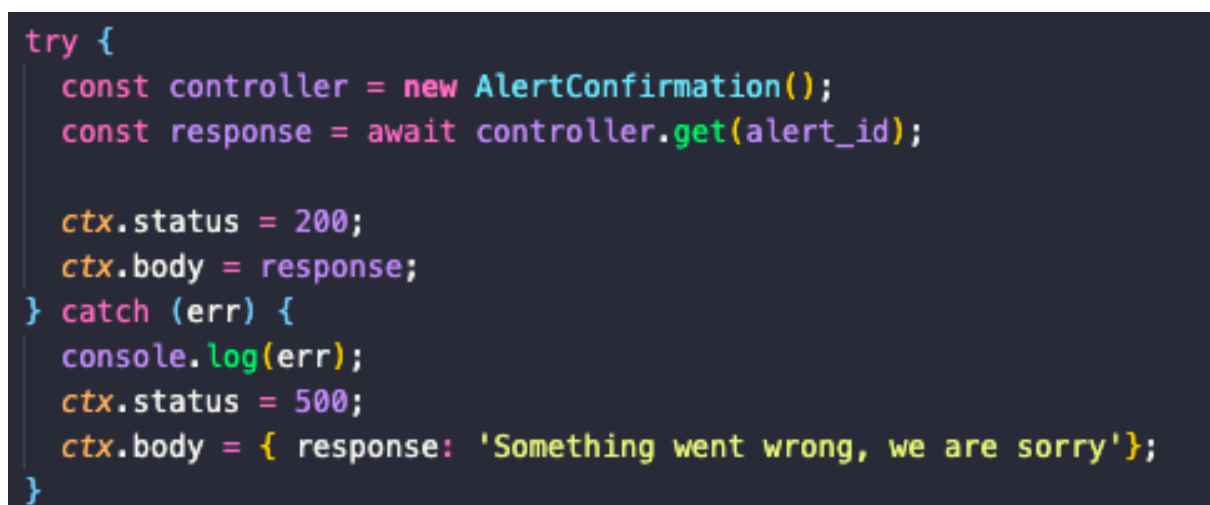
4. Utilização de arquivo de variável de ambiente



5. Utilização de docker



6. Utilização de blocos try-catch



Esses são apenas grupos de exemplos de boa utilização do código fonte, seus módulos e boas práticas. Acredito que essa é uma parte essencial de qualquer projeto e deve ser sempre buscada por qualquer desenvolvedor, independente do projeto e seu estado, além da utilização de testes para verificação do seu código, que será comentado no próximo capítulo.

## 5. Roteiro de testes efetuado

A realização de testes do projeto se divide em dois grupos diferentes: testes unitários de código e testes de sistema. Acredito que esses casos possam garantir para um projeto de escopo razoavelmente reduzido em seu primeiro momento, garantias necessárias de seu bom funcionamento. Sendo assim, 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 utilizada 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

Acredito que ainda há espaço para melhorias no desenvolvimento dos testes e estou em busca de uma biblioteca para que seja possível também “mockar” o banco de dados de forma completa, para que seja possível fazer testes de ponta a ponta também.

## 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, eu gerei 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**

OUR DATA > REGION METRIC	
 Alert Confirmation	 Alerts

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**

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

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
61f0dffeabc3219001216ec95	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

The screenshot shows a Postman API client interface. The request is a POST to `localhost:3001/v1/alert` with a JSON body: `{ "id": "abc-system-test", "alert_id": "abc-system-test", "region": "zona-sul", "user_id": "user-id" }`. The response status is 201 Created. Below this, the Metabase application is shown with the 'Alerts' table. The table has columns: ID, ID, Region, Alerts, Sender, Alert ID, and User ID. A single row is displayed with values: `61f34c1e6783c300126958fc`, `abc-system-test`, `zona-sul`, `abc-system-test`, and `user-id`.

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

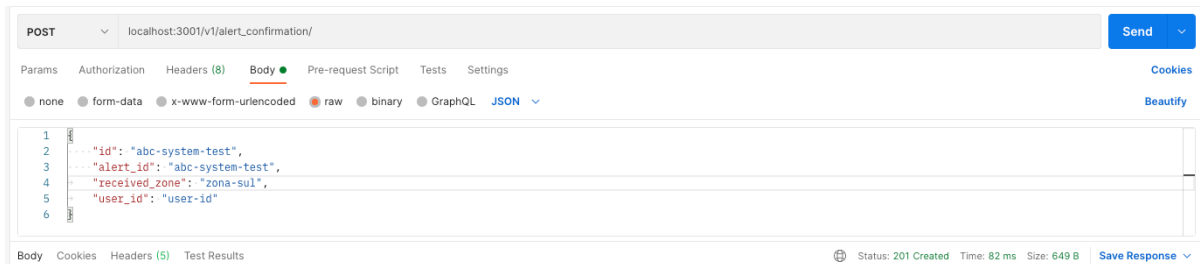
### 5.8.O sistema permite adição de novas confirmações de alertas recebidos.

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

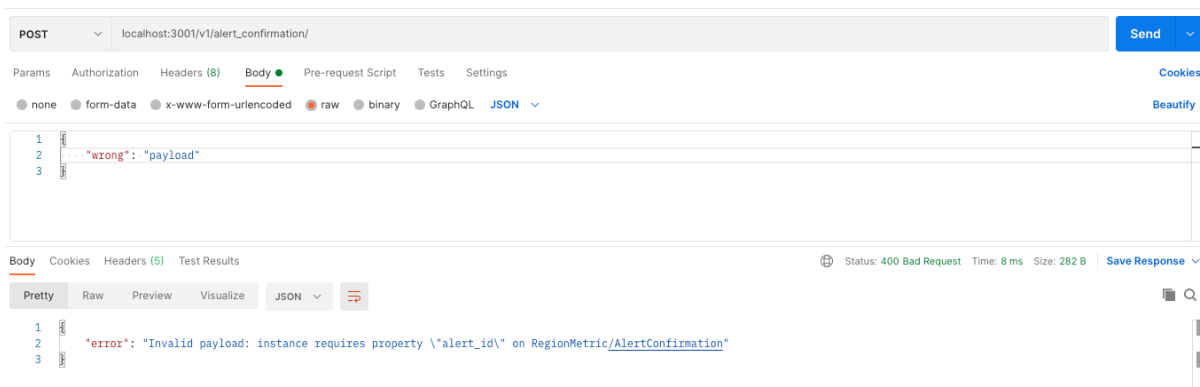


## 5.9.O sistema recusa requisições com payload inválidos

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.

## 6. 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 página de documentação do projeto como executar esse script.