**Propósito do documento**

Este documento descreve os processos de criação, arquitetura e testes realizados na aplicação “JavaDeveloper”.

**Hospedagem do projeto**

O projeto e o banco de dados desta aplicação se encontram disponíveis na internet, a aplicação pode ser acessada pela seguinte url:**XXXXXXXXXX**

No decorrer do documento descrever as chamadas e onde o banco de dados se encontra, caso queira adicionar informações ou efetuar testes online.

**Git do projeto:**

Este documento e o código fonte desenvolvido da aplicação se encontra na seguinte url: <https://github.com/rdmardegam/javaDeveloper>

**Definição de projeto**

Projeto constitui em efetuar consulta de CEP e a manutenção de endereços do cliente através de serviços disponibilizados por WebService Rest com comunicação através de json. O projeto também possui uma função para detectar qual a primeiro caráter de uma String que não se repete.

**Arquitetura do projeto**

Neste projeto ficou definido que existiria uma integração de sistema, pois o serviço de consulta CEP já existia e o serviço Rest iria consumi-lo. Neste caso, simulei que seria disponibilizado um EJB Remote em que os serviços Rest o consumiriam.

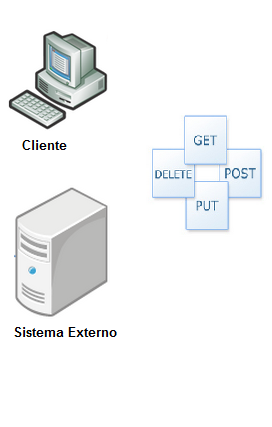
No projeto foi utilizado a seguinte arquitetura:

\* Um WebService Ret com a implementação “EasyRest” irá receber as solicitações através de “Post”, “Get”, “Put” ou “Delete”.

\* O Rest ira localizar o Ejb que o mesmo precisa invocar para que a solicitação seja atendida. Esse ejb poderia estar em outro local, mas no nosso exemplo está na mesma aplicação.

\* O ejb por sua vez solicitado, irá controlar a transação e executar as regras de negocio necessárias, chamando a camada de persistência DAO, onde foi utilizado o framework MyBatis para persistência de dados em um banco Mysql

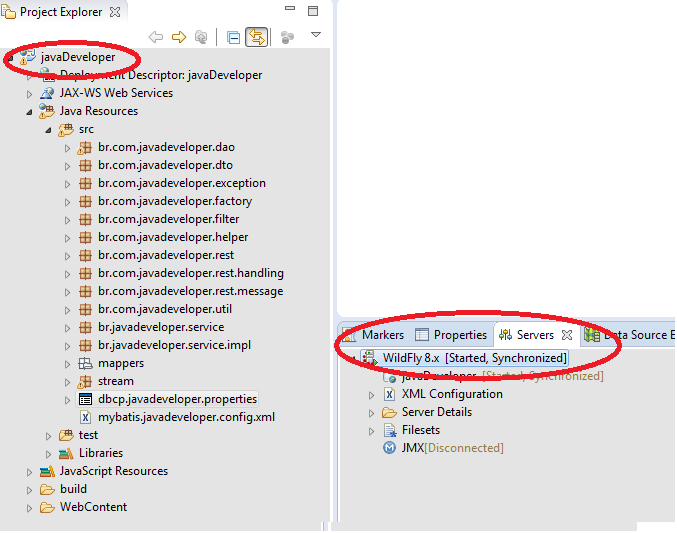
A arquitetura se resume a imagem abaixo:



**Funcionamento e direcional da aplicação:**

Foi desenvolvido 1 projeto web como na IDE Eclipse chamado javaDeveloper

Foi utilizado o servidor WildFly 8.2



**Banco de dados MySql:**

A aplicação utiliza um banco de dados Mysql, e sua configuração de conexao esta em um arquivo chamado “dbcp.javadeveloper.properties”, a configuração feita está apontando para um bando de dados que está na web, deixarei o mesmo liberado por um tempo para que se desejarem, possam conectar e manipular as informações, ou simplesmente utilizar a aplicação apontando para o mesmo, segue a configuração:

url/banco = 184.107.51.101:3306/koresy\_javaDeveloper

username=koresy\_javaDevel

password=123789

Se preferir criar o banco de dados localmente, alterar as configurações da aplicação para apontar para o banco de dados local no arquivo de configurações chamado “dbcp.javadeveloper.properties” e usar o seguinte script para criar o banco:

*create database javaDeveloper;*

*CREATE TABLE cep*

*( id INT(11) NOT NULL COMMENT 'id unico'*

*, TABLE1 VARCHAR(8) NOT NULL COMMENT 'cep'*

*, rua VARCHAR(255) NOT NULL COMMENT 'nome da rua, avenida, locacao'*

*, cidade VARCHAR(64) NOT NULL COMMENT 'cidade'*

*, estado VARCHAR(64) NOT NULL COMMENT 'estado'*

*, bairro VARCHAR(64) NULL COMMENT 'bairro'*

*, CONSTRAINT PRIMARY KEY ( id )*

*, CONSTRAINT CS\_UK UNIQUE KEY ( TABLE1 ) );*

*CREATE TABLE endereco*

*( id INT(11) NOT NULL COMMENT 'id unico'*

*, numero VARCHAR(8) NOT NULL COMMENT 'numero do local, podendo conter letra'*

*, complemento VARCHAR(512) NULL COMMENT 'complemento'*

*, cepId INT(11) NOT NULL COMMENT 'cepId fk'*

*, CONSTRAINT PRIMARY KEY ( id ) );*

*CREATE INDEX cepId1 ON cep ( cepId ASC);*

*Insert into cep (cep,rua,cidade,estado,bairro) values ('12345678','RUA DE TESTE','Sao Paulo','Sao Paulo','JD DO MAR');*

*Insert into cep (cep,rua,cidade,estado,bairro) values ('04158050','Doutor Rosalvo de Sales','Sao Paulo','Sao Paulo',null);*

*Insert into cep (cep,rua,cidade,estado,bairro) values ('00158000','Rua das Figueiras','Santo Andre','Sao Paulo',’Figueiras’);*

*Insert into cep (cep,rua,cidade,estado,bairro) values ('98765432','RUA DE TESTE','Rio de Janeiro','Rio de Janeiro','Barra');*

*Insert into cep (cep,rua,cidade,estado,bairro) values ('99900000','RUA TESTE 0','Rio de Janeiro','Rio de Janeiro','Ipanema');*

*Após ter o projeto publicado e apontando para o bando de dados correto, podemos iniciar a utilizar a aplicação, ou se preferir, pode utilizar a aplicação online disponível. (Obs: Verifique se o firewall não está bloqueando a mesma)*

**DOCUMENTANDO OS CENÁRIOS DESENVOLVIDO:**

Os cenários apresentados estão utilizando a aplicação disponível na web. Para realizar os testes foi utilizado um app do Chrome chamado Advanced Rest Client que permite a chamada e teste de Webservices Rest:

Para erros encontrados ou não previsto, o sistema gera um json padrão de erro:

{

sucess: false /\*AO OCORRER UM RESULTADO NÃO ESPERA, RETORNA FALSE\*/

status: 400 /\*RETORNADO O HTTP STATUS QUE AJUDA A IDENTIFICAR O PROBLEMA\*/

codigoErro:1 /\* CODIGO DE ERRO DA APLICAÇÃO, NO NOSSO EX FOI UTILIZADO UM FAKE\*/

mensagem: "A RUA DEVE SER INFORMADA CORRETAMENTE" /\*MENSAGEM DE ERRO POSSIVELMENTE A SER APRESENTADA\*/

link: "javaDeveloper/endereco/method/" /\*Poderia ser um link de ajuda ou referente ao metodo chamado\*/

mensagemDesenvolvedor: "O atributo rua é obrigatório e deve ser informado com até 128 caracteres." /\* MENSAGEM DE ERRO PARA AJUDAR O DESENVOLVEDOR A RESOLVER O PROBLEMA DA REQUISIÇÃO/

}

***1 - Testando alguns cenários da questão 1***:

Por convenção as pesquisas devem ser feitas por um método “GET” e como o GET não aceita corpo para receber um json, para que fosse possível passar um json, como a questão 1 estipula, solicitei que fosse passado um parâmetro para a chamada contendo um valor no formato json,

Method: **GET**

Exemplo Request:

[http://www.koresystems.com.br/javaDeveloper/rest/endereco/buscarCep/{"cep":"04158050"}](http://www.koresystems.com.br/javaDeveloper/rest/endereco/buscarCep/%7b%22cep%22:%2204158050%22%7d)

Exemplo Result Json:

{

sucess: true

data:

{

id: 2

cep: "04158050"

rua: "Doutor Rosalvo de Sales32"

bairro: null

cidade: "Sao Paulo"

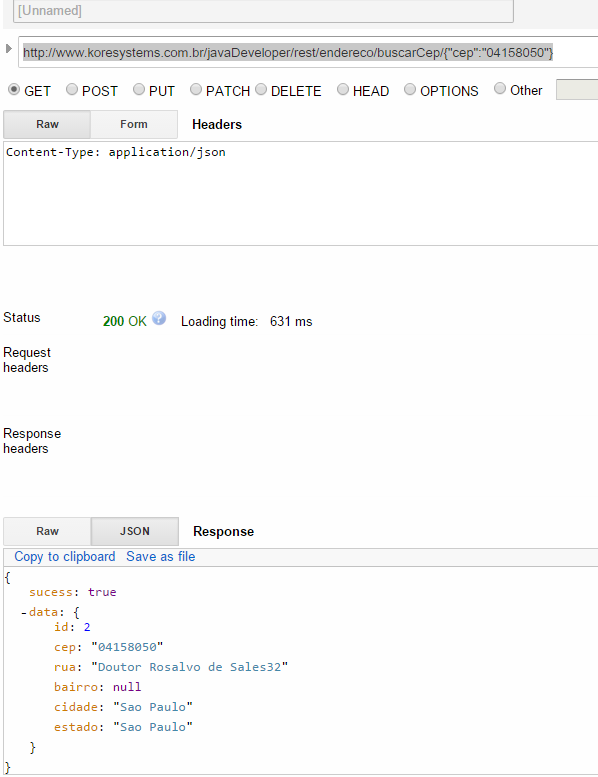
estado: "Sao Paulo"

}

}

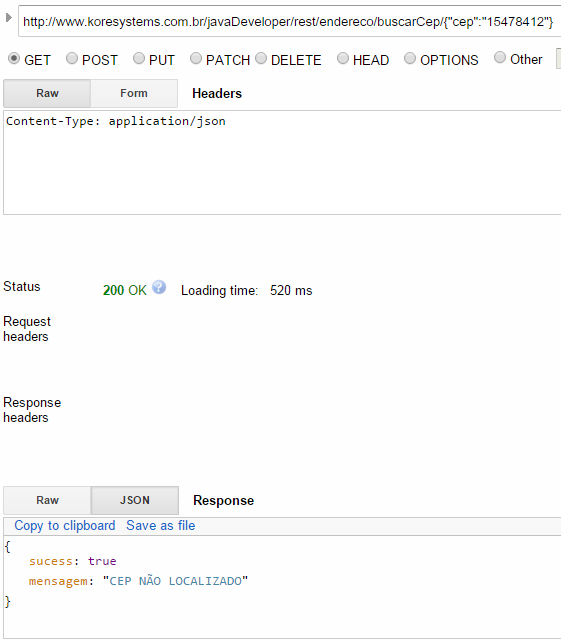
1.1 - Pesquisando um endereço correto que exista:

[http://www.koresystems.com.br/javaDeveloper/rest/endereco/buscarCep/{"cep":"04158050"}](http://www.koresystems.com.br/javaDeveloper/rest/endereco/buscarCep/%7b%22cep%22:%2204158050%22%7d)



1.2 - Pesquisando um endereço que não existe:

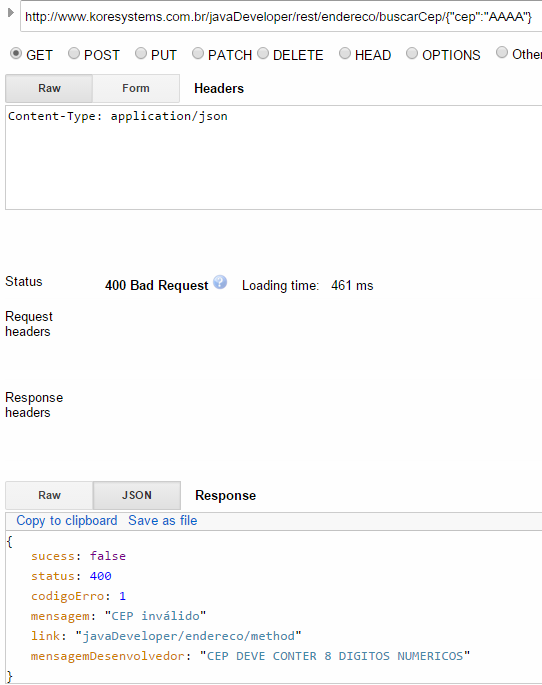
[http://www.koresystems.com.br/javaDeveloper/rest/endereco/buscarCep/{"cep":"15478412"}](http://www.koresystems.com.br/javaDeveloper/rest/endereco/buscarCep/%7b%22cep%22:%2215478412%22%7d)



\*Neste caso foi retornado um sucess true, pois o CEP foi pesquisado com sucesso, porem não existiu, dependendo do sistema podemos ter outra abordagem e retornar um status diferente de 200, como um NOT FOUND...

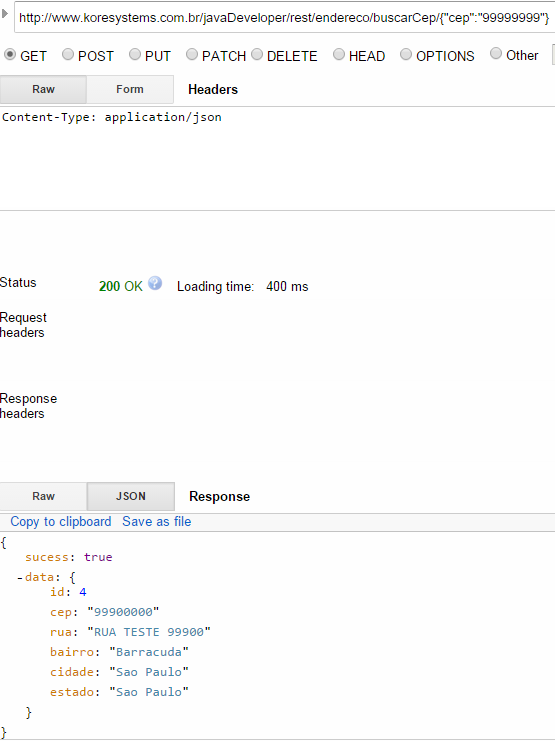
1.3 – Enviando um CEP com formato inválido:

[http://www.koresystems.com.br/javaDeveloper/rest/endereco/buscarCep/{"cep":"AAAA"}](http://www.koresystems.com.br/javaDeveloper/rest/endereco/buscarCep/%7b%22cep%22:%22AAAA%22%7d)



1.4 – Tentando localizar o CEP aproximado:

Foi pedido na questão que fosse adicionado zeros a direita ate que fosse encontrado o CEP, neste caso, existe no banco de dados o CEP de numero “99900000'', portanto, tentaremos pesquisar pelo CEP “99999999” e o sistema deve retornar a informação do CEP “''99900000''”:



***2 - Testando alguns cenários da questão 2***:

- A questão 2 se dividia varias chamadas ao nosso Rest para efetuar os CRUD.

1 - Inserir novo Endereço:

Como os endereços vão ser controlados por id, foi permitido que fosse cadastrado o mesmo endereço varias vezes, pensando no mundo real, varias pessoas que moram na mesma casa poderiam cadastrar o mesmo endereço. Nesse caso como o client sempre é obrigado a informar o estado, cidade, cep entre outros, No nosso exemplo fiz com que o cep fosse sempre atualizado com a informação que o usuário esta fornecendo ao cadastrar seu endereço, no mundo real isso não deveria acontecer, mas sim, considerar sempre o endereço da pesquisa de cep e não permitir que o nome da rua, estados entre outros mudasse. Sendo assim, todo mundo que está associado a aquele cep, terá seu sua informação de rua, estado entre ouros alterado.

Method: **POST**

Exemplo Request:

<http://www.koresystems.com.br/javaDeveloper/rest/endereco/cadastrar>

Exemplo de Envio de Json:

{

"numero": "25",

"complemento" : "Ap 443",

"cepInfo": {

"cep": "99900000", "rua": "Rua TESTE 99900",

"bairro": "Barracuda","cidade": "Sao Paulo",

"estado": "Sao Paulo"

}

}

Exemplo Result Json:

{

sucess: true

mensagem: "Endereço Cadastrado com Sucesso"

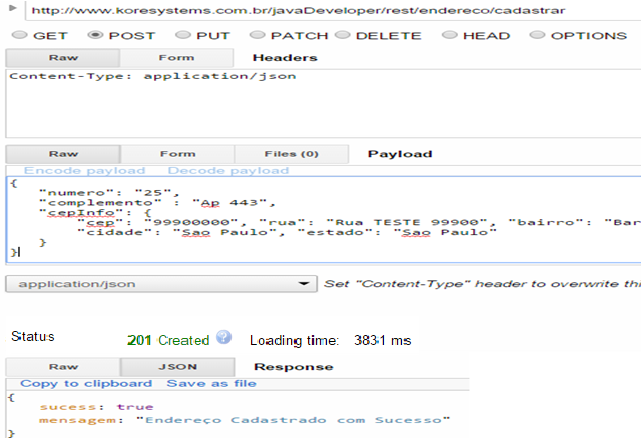
}

2.1.1 Inserindo um endereço com sucesso:

<http://www.koresystems.com.br/javaDeveloper/rest/endereco/cadastrar>

Json Enviado:

{ "numero": "25", "complemento" : "Ap 443", "cepInfo": { "cep": "99900000", "rua": "Rua TESTE 99900", "bairro": "Barracuda","cidade": "Sao Paulo", "estado": "Sao Paulo"}}



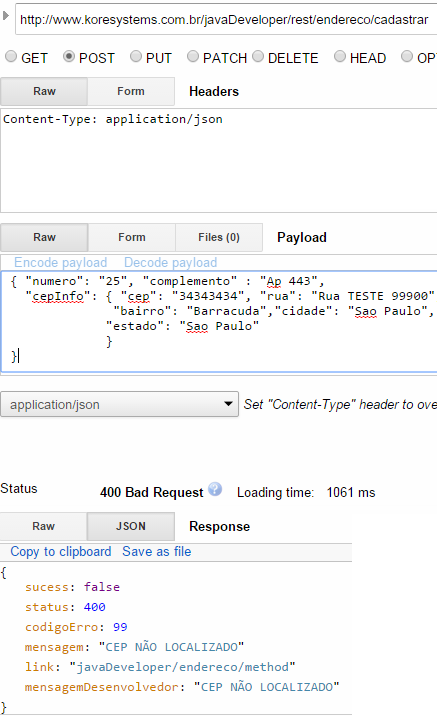
\*\*OBS: Aqui poderíamos ter um status “OK 200” dependendo do padrão da aplicação

2.1.2 Tentando inserir um endereço com um CEP não localizado

<http://www.koresystems.com.br/javaDeveloper/rest/endereco/cadastrar>

Json:

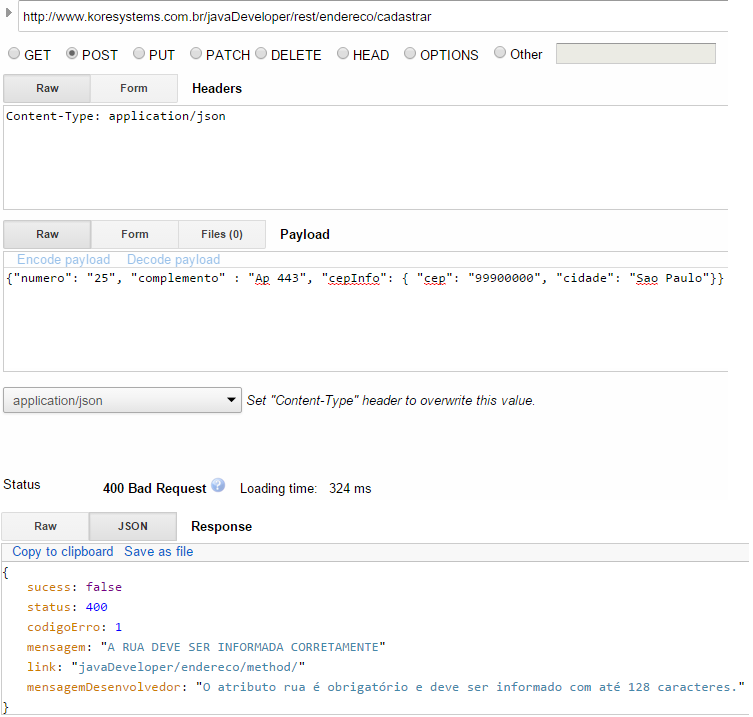
{ "numero": "25", "complemento" : "Ap 443", "cepInfo": { "cep": "34343434", "rua": "Rua TESTE 99900", "bairro": "Barracuda","cidade": "Sao Paulo", "estado": "Sao Paulo"}}



2.1.3 Tentando inserir um endereço não informando algum campo obrigatório: <http://www.koresystems.com.br/javaDeveloper/rest/endereco/cadastrar>

Json Enviado:

{"numero": "25", "complemento" : "Ap 443", "cepInfo": { "cep": "99900000", "cidade": "Sao Paulo"}}



2 - Alterar endereço existente:

Para que o endereço seja alterado, o mesmo deve existir previamente cadastrado, portanto, temos que enviar seu id para alteração.

Method: **PUT**

Exemplo Request:

<http://www.koresystems.com.br/javaDeveloper/rest/endereco/atualizar>

Exemplo de Envio de Json:

{

"id" : 33,

"numero": "356",

"complemento" : "Prox ao Posto XYZ",

"cepInfo": {

"cep": "99900000",

"rua": "RUA TESTE 99900",

"bairro": "Barracuda 2",

"cidade": "Sao Paulo",

"estado": "Sao Paulo"

}

}

Exemplo Result Json:

{

sucess: true

mensagem: "Endereço Alterado com Sucesso"

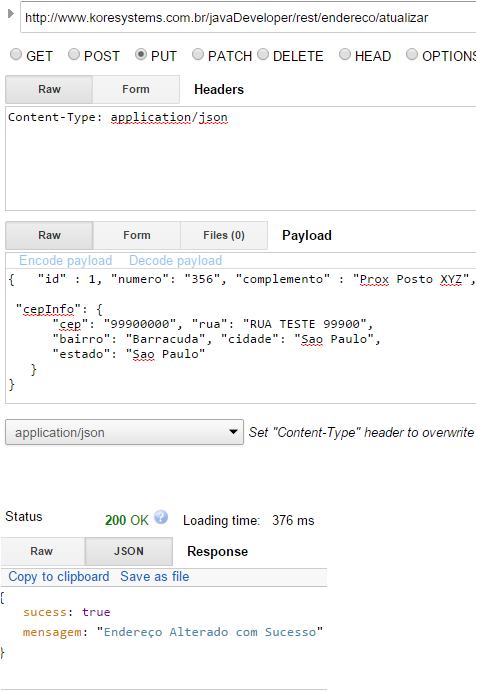
}

2.2.1 Alterando um endereço com sucesso:

<http://www.koresystems.com.br/javaDeveloper/rest/endereco/cadastrar>

Json:

{ "id" : 1, "numero": "356", "complemento" : "Prox Posto XYZ", "cepInfo": {"cep": "99900000", "rua": "RUA TESTE 99900", "bairro": "Barracuda", "cidade": "Sao Paulo","estado": "Sao Paulo"}}

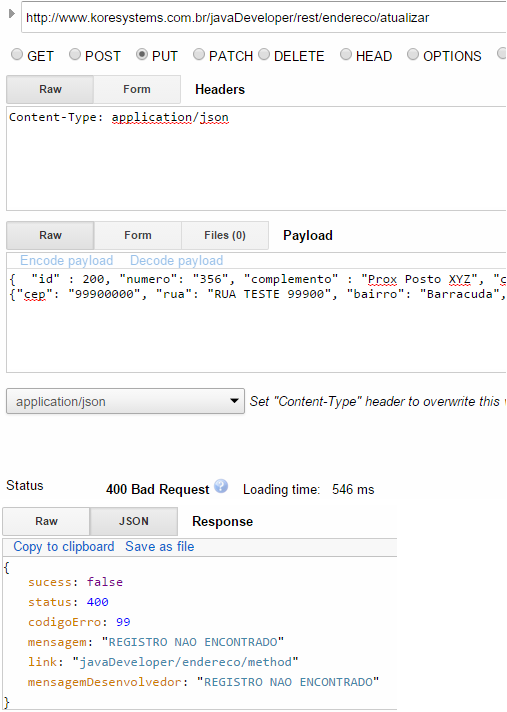


2.2.2 Alterando um endereço que ainda não existe, para isso, se passa um id que ainda não existe:

<http://www.koresystems.com.br/javaDeveloper/rest/endereco/cadastrar>

Json:

{ "id" : 200, "numero": "356", "complemento" : "Prox Posto XYZ", "cepInfo": {"cep": "99900000", "rua": "RUA TESTE 99900", "bairro": "Barracuda", "cidade": "Sao Paulo","estado": "Sao Paulo"}}



3 – Pesquisando endereço existente:

Outra vez utilizaremos um serviço de método GET, mas neste caso não irei esperar um json como no pesquisa cep e sim um parâmetro na própria url simplesmente com o id do endereço desejado.

Method: **GET**

Exemplo Request:

<http://www.koresystems.com.br/javaDeveloper/rest/endereco/buscarEndereco/1>

Exemplo Result Json:

{

sucess: true

data:

{

id: 1

numero: "356"

complemento: "Prox Posto XYZ"

cepInfo:

{

id: null

cep: "99900000"

rua: "RUA TESTE 99900"

bairro: "Barracuda"

cidade: "Sao Paulo"

estado: "Sao Paulo"

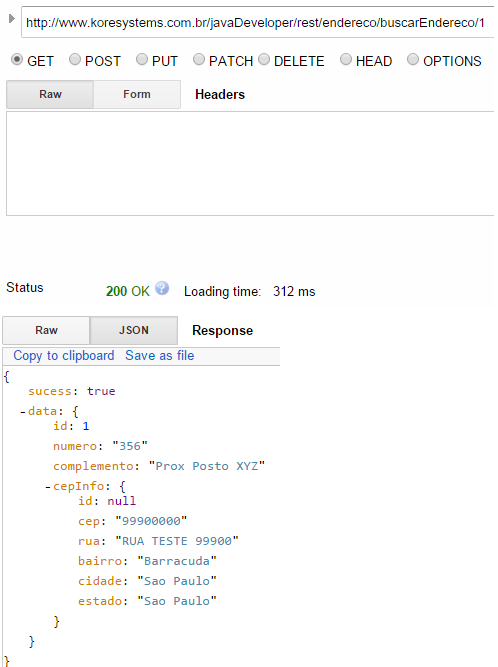
}

}

}

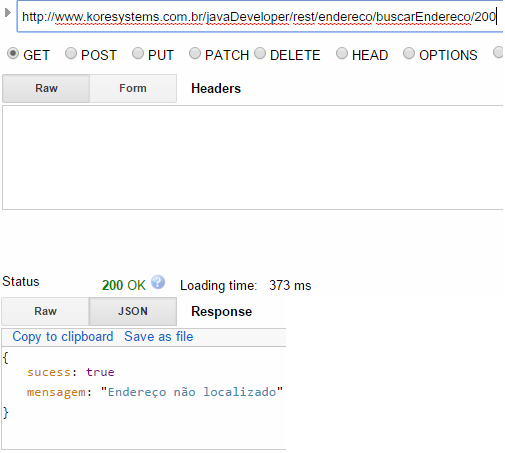
2.3.1 Pesquisando um endereço com sucesso:

<http://www.koresystems.com.br/javaDeveloper/rest/endereco/buscarEndereco/1>



2.3.2 Pesquisando um endereço que não existe

<http://www.koresystems.com.br/javaDeveloper/rest/endereco/buscarEndereco/200>



\*\*Obs, dependendo do padrão da aplicação poderia ser status diferente de 200 para informar que não encontrou o item pesquisado. Neste exemplo a pesquisa foi realizada com sucesso e simplesmente não existem dados para retornar.

4 – Deletar endereço existente

Devemos informar o id do endereço que gostaríamos de apagar. Mesmo que o endereço seja apago, a informação de cep permanecem no sistema.

Method: **DELETE**

Exemplo Request:

<http://www.koresystems.com.br/javaDeveloper/rest/endereco/deletar>

Exemplo de Envio de Json:

{

"id":8

}

Exemplo Result Json:

{

sucess: true

mensagem: "Endereço Deletado com Sucesso"

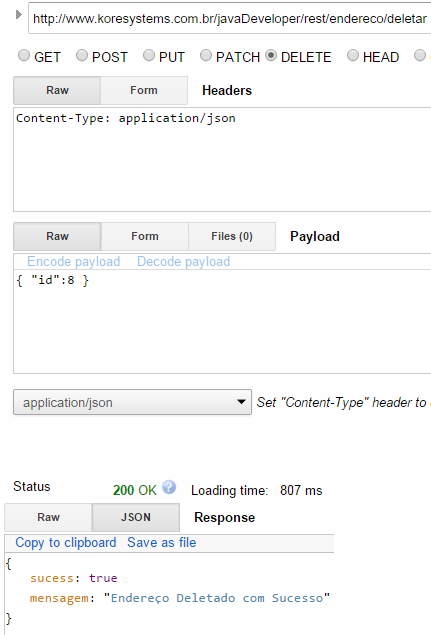
}

2.4.1 Deletando um endereço com sucesso:

<http://www.koresystems.com.br/javaDeveloper/rest/endereco/cadastrar>

Json:

{ "id":8 }

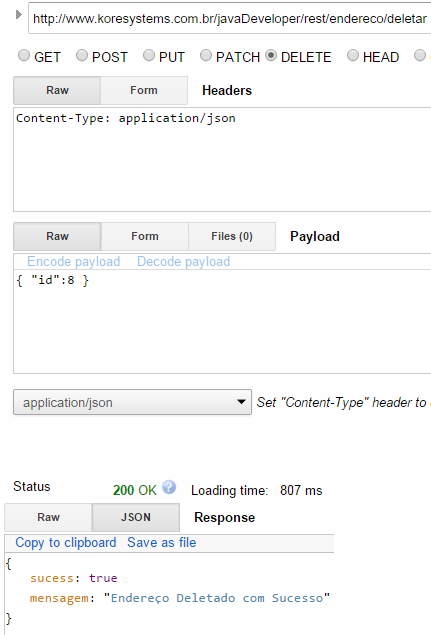


2.4.1 Deletando um endereço com sucesso:

<http://www.koresystems.com.br/javaDeveloper/rest/endereco/cadastrar>

Json:

{ "id":8 }

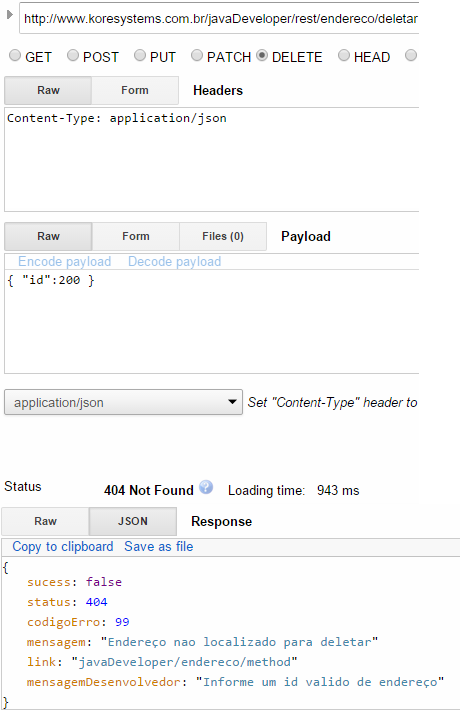


2.4.2 Tentando deletando um endereço que não existe:

<http://www.koresystems.com.br/javaDeveloper/rest/endereco/cadastrar>

Json:

{ "id":200 }

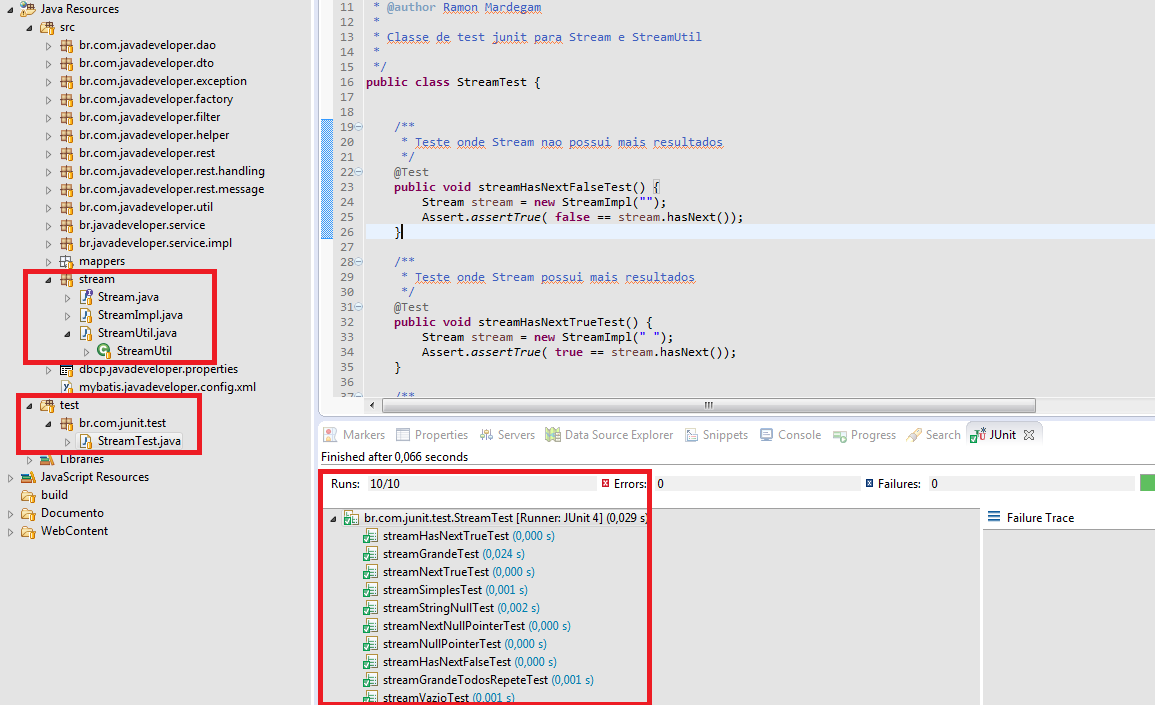


***3 - Testando alguns cenários da questão 3:***

- A questão 3 se dividia criar uma implementação para a interface Stream para leitura de uma string caracter por caracter e tentar localizar a primeira palavra que não se repete da melhor maneira possível.

No projeto disponibilizado temos a interface “Stream”, sua implementação “StremUtil” e também a classe “StreamUtil” que contem o método **public** **static** **char** fistChart(Stream stream) responsável por efetuar a regra de localização da primeira palavra que não se repete.

Efetuei alguns testes com difersos cenários através do junit e os disponibilizei jutamente com o codigo, o mesmo está localizado em test.br.com.junit.test.StreamTest.java



Também criei um WebService Rest que pode ser chamado para utilizar esse método, para chamar:

Method: **GET**

[http://www.koresystems.com.br/javaDeveloper/rest/stream/firstChar/ABDJYUYUASDBKJJDAHSDJCU\*(@#B](http://www.koresystems.com.br/javaDeveloper/rest/stream/firstChar/ABDJYUYUASDBKJJDAHSDJCU*(@#B)

