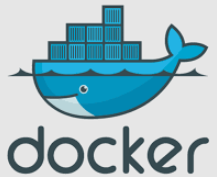
Desafio Técnico CWI

Solução BACK-END para Gerenciar Sessões de Votação em Assembleias

Autor: Eliseu Silveira Brito Data: 18/05/2020

GitHub: <https://github.com/eliseusbrito/DesafioTecnicoCwiVotacaoAssembleia_SpringJPAHibernatePostgreSQL>

App Heroku: https://votacao-assembleia.herokuapp.com

Sumário

Solução Back-end para Gerenciar Sessões de Votação em Assembleias 4

Utilização pelo Usuário 4

Operação 5

 Pauta 5

 Cadastro 5

 Listagem 7

 Edição 8

 Exclusão 8

 Sessão de Votação 9

 Abertura Sessão 9

 Tempo de Votação 9

 Comportamento 10

Localhost 10

Heroku 10

 Votos 12

 Cadastro 12

 Listagem 13

 Associados 14

 Cadastro 14

 Listagem 15

 Edição 16

 Exclusão 17

Resultado 17

 Mensageria e Filas 18

Programa Java 19

Tabela de Dados 19

Configuração 21

Classe de Configuração 21

Banco teste H2 22

Banco Postgre 22

 Banco Local e Heroku pelo Postgre: 22

 Dependências adicionadas: 22

 Obtendo script SQL a partir do PostgreSQL local 22

 Executando script SQL no servidor remoto - HEROKU 24

Deploy do sistema no Heroku 26

Tratamento de Erros e exceções: 27

 Pesquisa por Id não existente da Classe Associado: 27

 Cadastrar associado sem informações no nome ou cpf: 27

 Adicionado mSG personalizada Ao cadastrar associado já existente 29

 Adicionado Mensagem QUANDO COLOCA LETRA ERRADA no Voto 29

 Adicionado Mensagem Quando Votação não ESTÁ aberta 29

 Mensagem quanto cadastra Pauta já existente 30

 Heroku 30

Instalações 30

RabbitMQ 30

Aprendizados 30

Tabela de Dados 30

Conceitos: 32

Integração Package Mensageria com Votação 33

Pendências e Melhorias: 35

Conclusões 35

Anexos 36

Variação no api GET https://user-info.herokuapp.com/users/{cpf}: 36

# Solução Back-end para Gerenciar Sessões de Votação em Assembleias

No cooperativismo, cada associado possui um voto e as decisões são tomadas em assembleias, por votação. Esta solução back-end foi criada para gerenciar essas sessões de votação.

Sendo executada na nuvem, promove as seguintes funcionalidades através de uma API REST:

● Cadastrar uma nova pauta;

● Abrir uma sessão de votação em uma pauta (a sessão de votação deve ficar aberta por um tempo determinado na chamada de abertura ou 1 minuto por default);

● Receber votos dos associados em pautas (os votos são apenas 'Sim'/'Não'. Cada associado é identificado por um id único e pode votar apenas uma vez por pauta);

● Contabilizar os votos e dar o resultado da votação na pauta.

As pautas e os votos são persistidos e não são perdidos com o restart da aplicação.

Para fins de exercício, a segurança das interfaces foi abstraída e qualquer chamada para as interfaces é considerada como autorizada. A linguagem escolhida foi Java, e utiliza ferramentas como o Spring, Rest, JPA, Hibernate, H2, PostgreSQL, Heroku, Docker, RabbitMQ,Json e Postman.

A solução também foi integrada com um sistema externo que verifica a partir do CPF do associado, se ele pode votar.

O resultado da votação é informado para o restante da plataforma através de mensageria. A solução possui um Producer que ao sair o resultado da votação este dispara automaticamente uma mensagem para o RabbitMQ. O restante da plataforma pode acessar a fila e consumir esta mensagem. Mensagens também podem ser enviadas via endpoint através de comando POST. Quando a sessão de votação fechar o usuário pode postar uma mensagem complementar ao resultado da votação, por exemplo, via Postman.

## Utilização pelo Usuário

O responsável pela assembleia deve cadastrar os associados que ficarão aptos a votar.

Este também precisa cadastrar uma pauta de votação.

Após os dois pré-requisitos anteriores estarem atendidos, pode-se abrir uma votação com tempo de votação default de 1 minuto ou pode-se também definir este tempo.

Durante o período de votação, os associados poderão fazer seu voto com as opções de Sim ou Não.

Quando concluído o tempo de votação o sistema informará o resultado.

Caso seja aconteça um empate ou nenhum voto, uma nova pauta deverá ser cadastrada para esta nova votação.

### Operação

Todos os comandos foram testados e funcionam utilizando o Postman, tanto no ambiente Heroku ou localhost.

Demais alternativas de envio de requisições, como por exemplo o Navegador Chrome Desktop ou Mobile, foram testadas, tendo comportamento semelhante ao Postman.

No GitHub do projeto tem duas Coleções, uma para testes no Heroku e outra para teste em localhost.

#### Pauta

##### Cadastro

O utilizador pode cadastrar uma pauta utilizando o comando POST:

<https://votacao-assembleia.herokuapp.com/pauta> ou <http://localhost:8080/pauta>

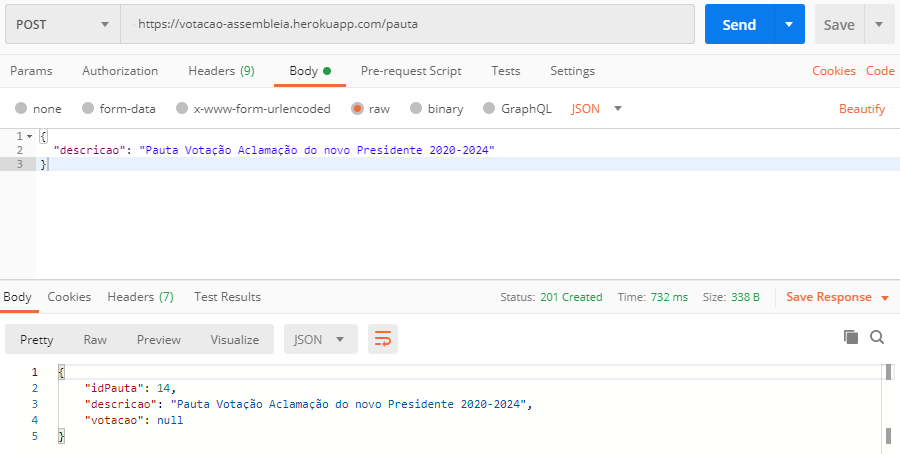
No Body deve-se usar o padrão Json conforme a seguir:

{

"descricao":"Título da Pauta"

}

Exemplo no Postman:



A solução foi integrada com um sistema externo que verifica, a partir do CPF do associado, se ele pode votar.

■ GET https://user-info.herokuapp.com/users/{cpf}

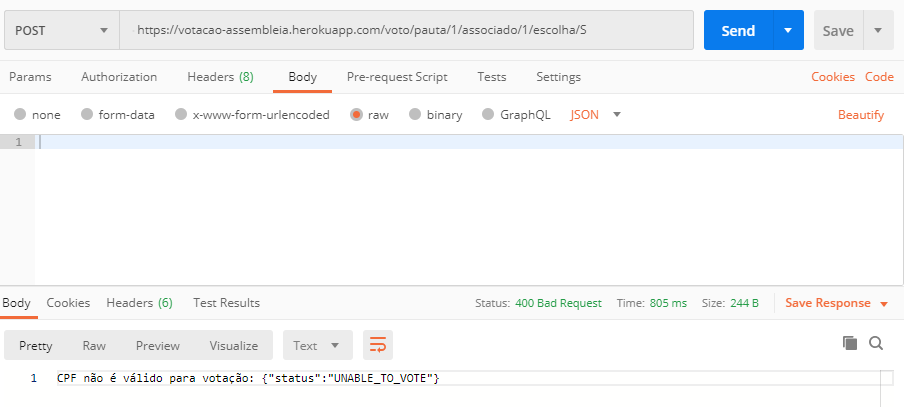
■ Caso o CPF seja inválido, a API retornará o HTTP Status 404 (Not found).

■ Caso o CPF seja válido, a API retornará se o usuário pode (ABLE\_TO\_VOTE) ou não pode (UNABLE\_TO\_VOTE) executar a operação

Estas requisições foram integradas e caso o CPF seja válido a votação se dá normalmente.

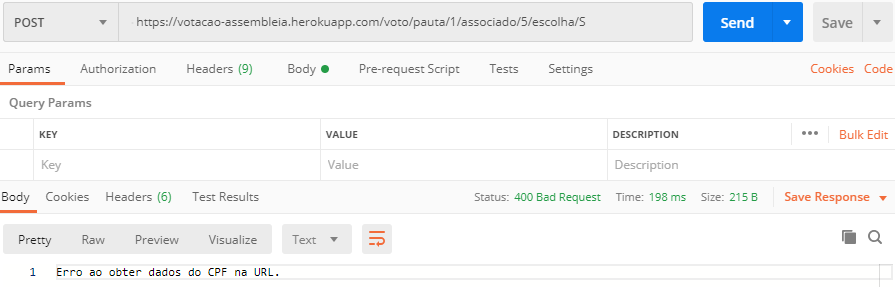
Caso o CPF não seja válido o sistema retorna mensagem:

“CPF não é válido para votação: {"status":"UNABLE\_TO\_VOTE"}”



Caso o número do cpf cadastrado do Associado tenha alguma divergência com a base de dados pesquisada a mensagem é conforme a seguir:

“Erro ao obter dados do CPF na URL”

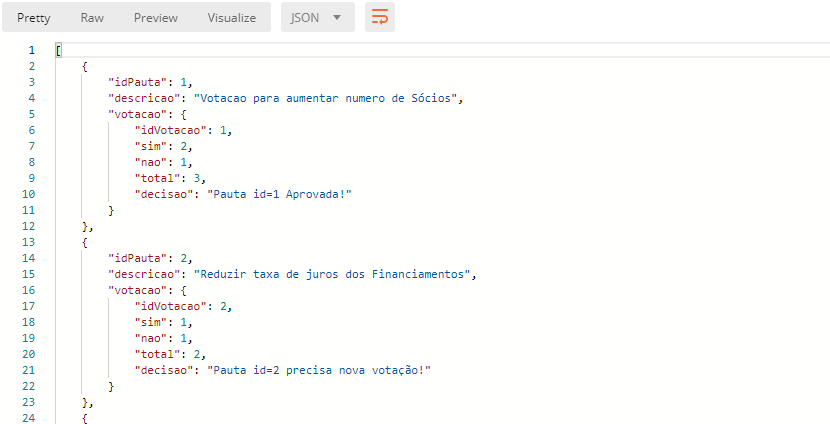


##### Listagem

Para pesquisar todas as pautas cadastradas usa-se o comando GET

<https://votacao-assembleia.herokuapp.com/pauta> ou <http://localhost:8080/pauta>





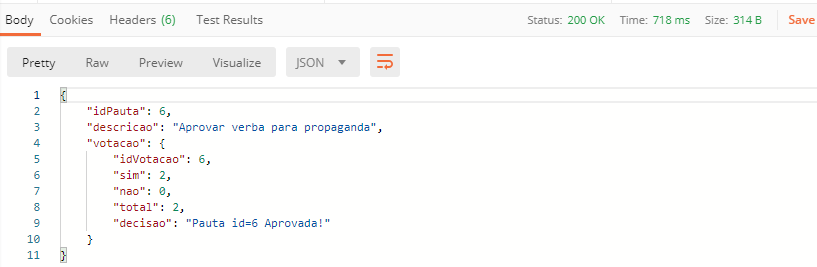
Para listar uma pauta especifica usa-se o comando GET / número da pauta desejada

[https://votacao-assembleia.herokuapp.com/pauta/{id}](https://votacao-assembleia.herokuapp.com/pauta/%7bid%7d) ou [http://localhost:8080/pauta/{id}](http://localhost:8080/pauta/%7bid%7d)

{id} = número da pauta que se quer listar

Ex.: <https://votacao-assembleia.herokuapp.com/pauta/6>





##### Edição

Para editar a pauta desejada usa-se o comando PUT:

[https://votacao-assembleia.herokuapp.com/pauta/{id}](https://votacao-assembleia.herokuapp.com/pauta/%7bid%7d) ou [http://localhost:8080/pauta/{id}](http://localhost:8080/pauta/%7bid%7d)

{id} = número da pauta que se quer editar

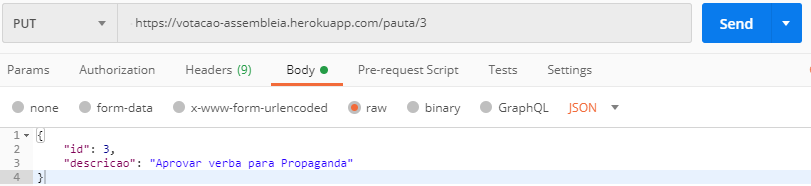
No Body deve-se usar o padrão Json conforme a seguir:

{

"descricao":"Alteração do Título da Pauta"

}

Exemplo:



##### Exclusão

Para excluir a pauta desejada usa-se o comando DELETE:

[https://votacao-assembleia.herokuapp.com/pauta/{id}](https://votacao-assembleia.herokuapp.com/pauta/%7bid%7d) ou [http://localhost:8080/pauta/{id}](http://localhost:8080/pauta/%7bid%7d)

{id} = número da pauta que se quer excluir

Exemplo:



#### Sessão de Votação

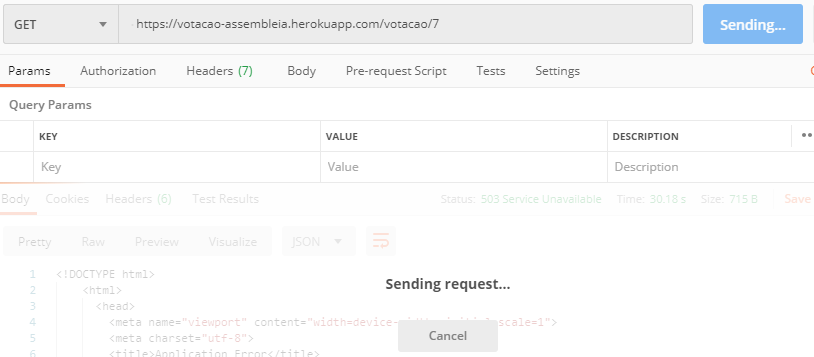
##### Abertura Sessão

Para abrir uma sessão de votação, o utilizador usa o comando GET

https://votacao-assembleia.herokuapp.com/votacao/{id} ou http://localhost:8080/votacao/{id}

{id} = Número da Pauta que se quer abrir a sessão de votação.

Exemplo: <https://votacao-assembleia.herokuapp.com/votacao/7>



Somente pode ser aberta uma vez a sessão de votação para cada pauta cadastrada.

Caso seja necessária uma nova votação para uma pauta, deve ser cadastrada uma nova pauta.

Exemplo:

Pauta 36 teve o resultado empatado 🡪 Deve se cadastrar uma nova pauta

O cadastro desta nova pauta poderia ser como a seguir:

Nova votação da Pauta 36 – Aumentar o valor de contribuição por associado 🡪 Esta nova pauta ganhará uma nova numeração.

##### Tempo de Votação

O tempo de votação por padrão é de um minuto.

Para definir tempo de votação diferente do padrão usa-se o comando GET a seguir:

[https://votacao-assembleia.herokuapp.com/votacao/{id}/tempo/{t}](https://votacao-assembleia.herokuapp.com/votacao/%7bid%7d/tempo/%7bt%7d) ou

http://localhost:8080/votacao/{id}/tempo/{t}

{id} = número da pauta que se quer abrir a sessão de votação.

{t} = tempo que a sessão de votação ficará aberta em minutos.

Exemplo: <https://votacao-assembleia.herokuapp.com/votacao/1/tempo/2>

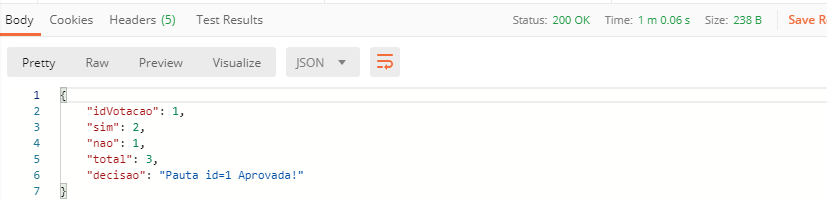
##### Comportamento

A resposta da requisição get, com tempo de votação dentro do Java, tem comportamentos diferentes dentro do localhost e do heroku. No local host responde normalmente. No Heroku responde com mensagens de erro mas os dados são gravados normalmente.

###### Localhost

Responde conforme imagem abaixo:





Testes mostram que tanto no ambiente de teste (Banco de dados H2) como no desenvolvimento (Banco de Dados Postdegre), tem o mesmo comportamento.

###### Heroku

Responde com erro conforme imagens abaixo: Could not get any response ou Status: 503 Service Unavailable

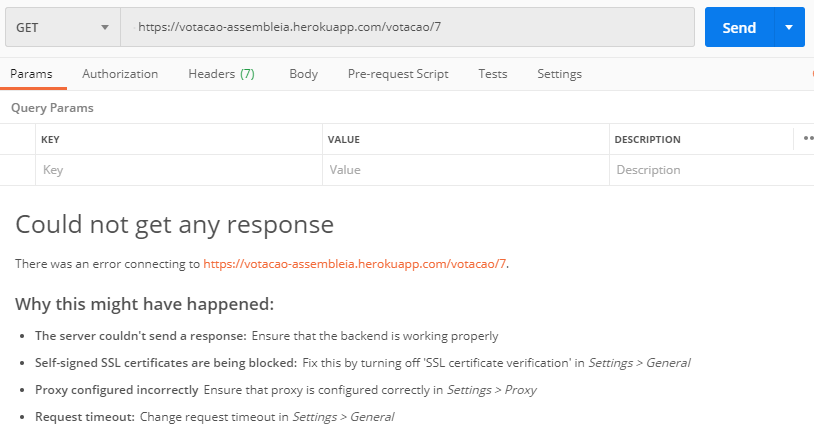


Imagem 1 - Could not get any response

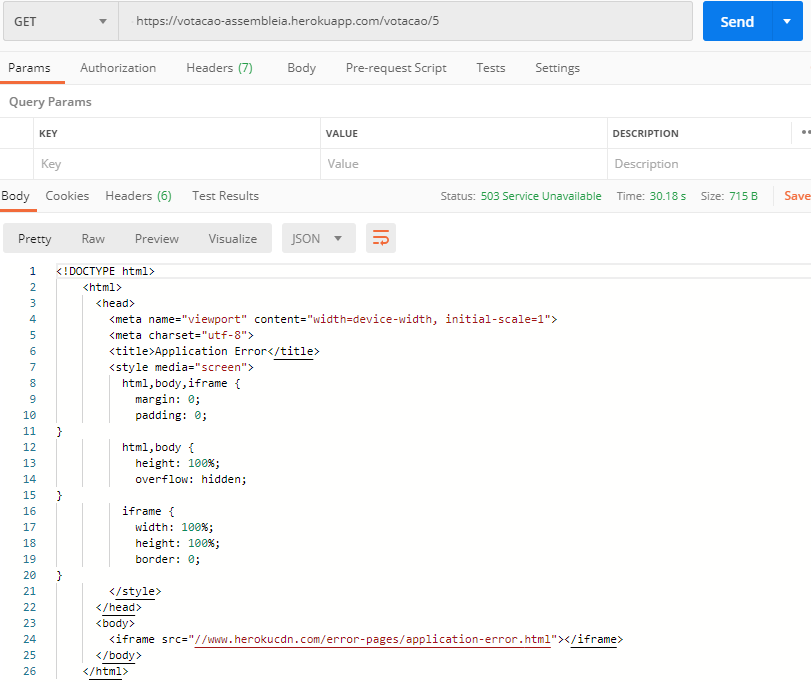
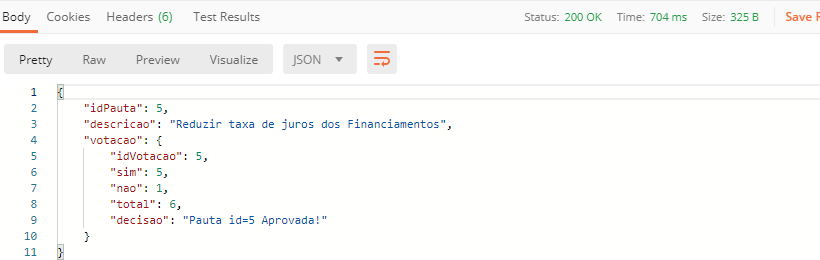


Imagem 2 - Status: 503 Service Unavailable

No entanto, os dados são gravados normalmente e pode-se confirmar listando a pauta que teve a votação.

Exemplo:





Solução para este problema, que acontece somente no ambiente de produção, virá nas próximas atualizações em novos deploy para o heroku.

#### Votos

##### Cadastro

Para votar o utilizador usa o comando POST:

https://votacao-assembleia.herokuapp.com[/voto/pauta/{idPauta}/associado/{idAssociado}/escolha/{escolha}](http://localhost:8080/voto/pauta/%7bidPauta%7d/associado/%7bidAssociado%7d/escolha/%7bescolha%7d)

ou http://localhost:8080[/voto/pauta/{idPauta}/associado/{idAssociado}/escolha/{escolha}](http://localhost:8080/voto/pauta/%7bidPauta%7d/associado/%7bidAssociado%7d/escolha/%7bescolha%7d)

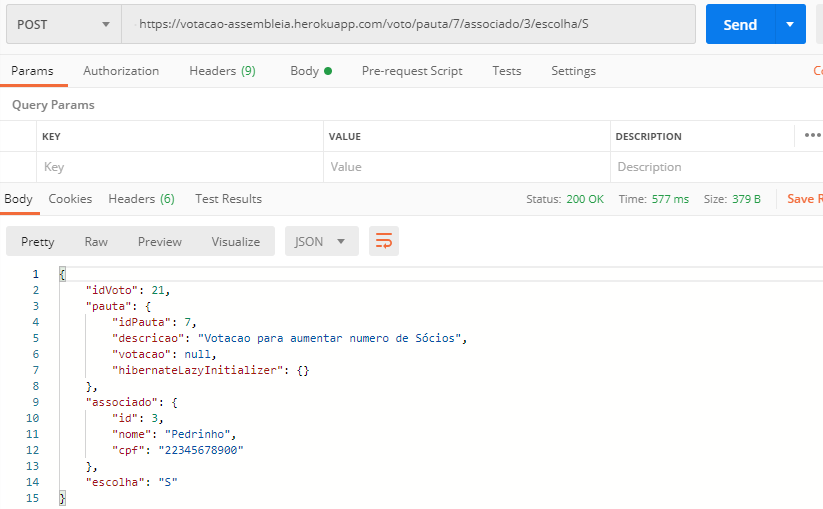
{idPauta} = Número da Pauta que se quer votar;

{idAssociado} = identificação do associado que esta votando.

{escolha} = para votar sim usa-se o S e para votar não se usa o N.

Exemplo:

<https://votacao-assembleia.herokuapp.com/voto/pauta/6/associado/1/escolha/S>



##### Listagem

Para pesquisar deve-se usar o comando GET:

Busca todos os votos

<https://votacao-assembleia.herokuapp.com/voto> ou <http://localhost:8080/voto>

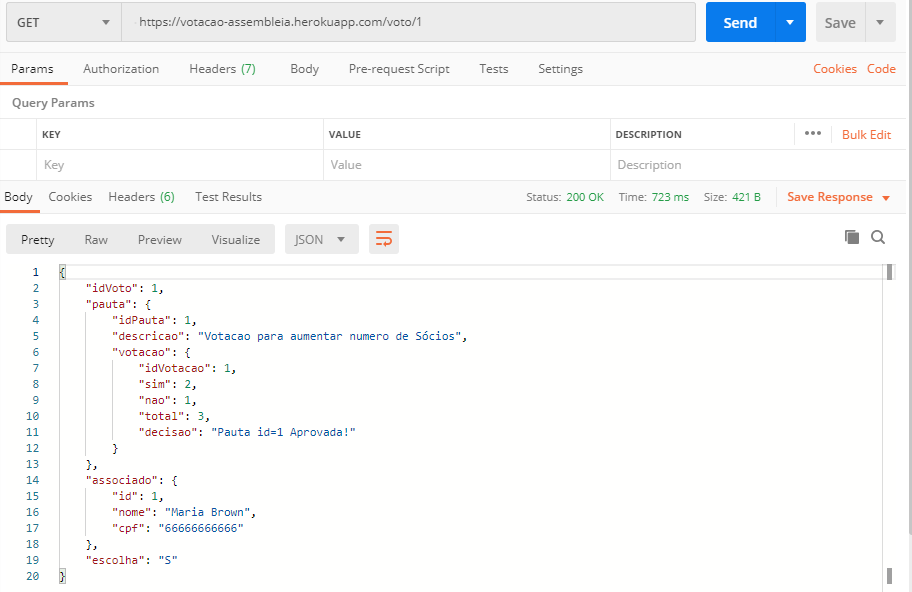
Para buscar um voto especifico:

<https://votacao-assembleia.herokuapp.com/voto>{id} ou [http://localhost:8080/voto/{id}](http://localhost:8080/voto/%7bid%7d)

{id} = Número do voto que se quer pesquisar

Exemplo:

https://votacao-assembleia.herokuapp.com/voto/1



#### Associados

##### Cadastro

Para cadastrar usuários usa-se o POST

<https://votacao-assembleia.herokuapp.com/associado> ou <http://localhost:8080/associado>

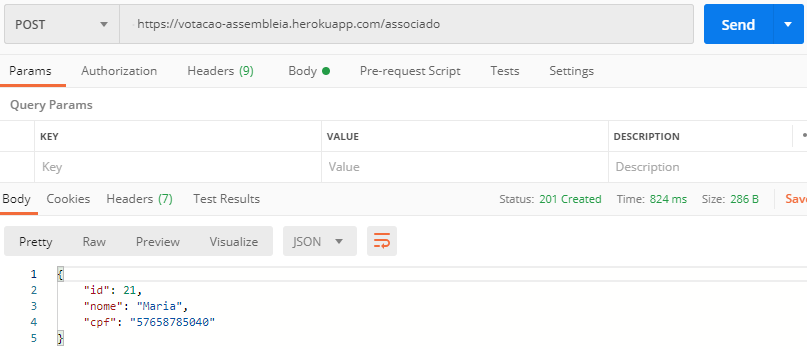
com o Body Json:

{

"nome":"Nome do Associado",

“cpf”:”xxxxxxxxxxx”

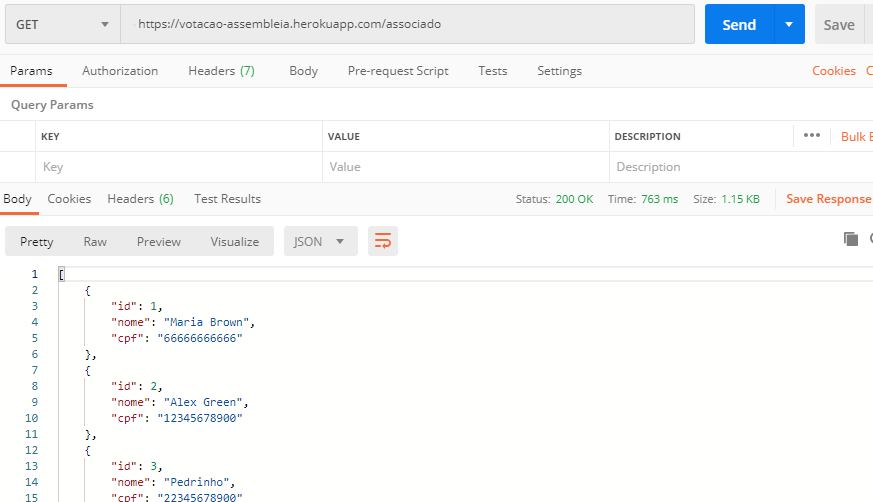
}



##### Listagem

Para pesquisar todos os usuários cadastrados usa-se o comando GET

<https://votacao-assembleia.herokuapp.com/associado> ou <http://localhost:8080/associado>



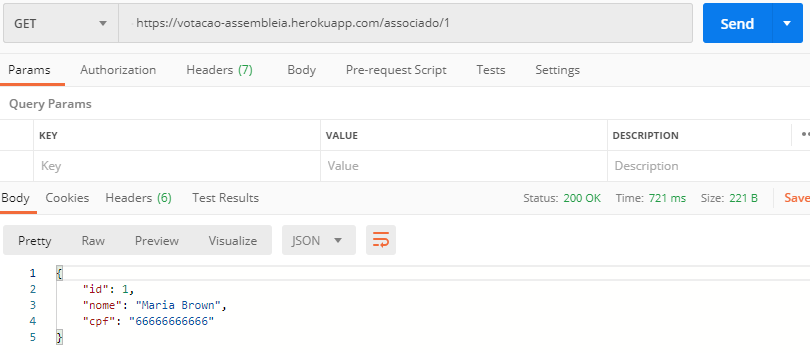
Para listar associado especifico:

[https://votacao-assembleia.herokuapp.com/associado/{id}](https://votacao-assembleia.herokuapp.com/associado/%7bid%7d)

ou [http://localhost:8080/associado/{id}](http://localhost:8080/associado/%7bid%7d)

{id} = número da identificação do associado

Exemplo: <http://localhost:8080/associado/1>



##### Edição

Para editar o associado desejada usa-se o comando PUT:

[https://votacao-assembleia.herokuapp.com/associado/{id}](https://votacao-assembleia.herokuapp.com/associado/%7bid%7d)

ou [http://localhost:8080/associado/{id}](http://localhost:8080/associado/%7bid%7d)

{id} = número da pauta que se quer editar

No Body deve-se usar o padrão Json conforme a seguir:

{

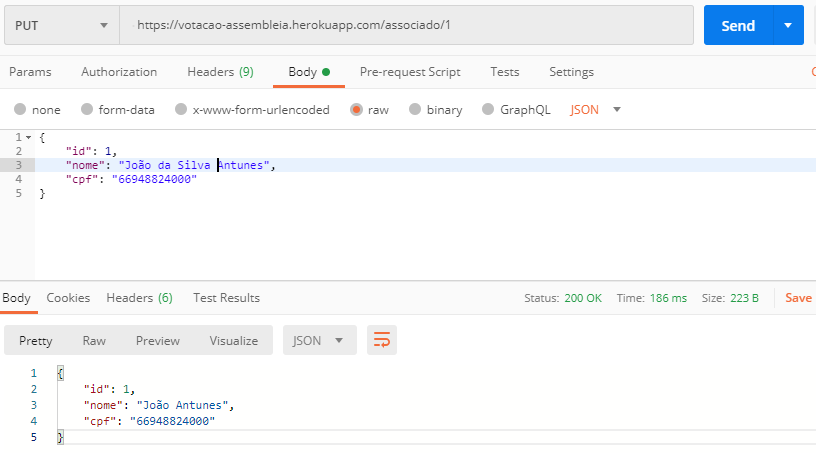
“id”:1,

"nome":"Nome do Associado",

“cpf”:”xxxxxxxxxxx”

}

Exemplo:



##### Exclusão

Para excluir o associado desejado usa-se o comando DELETE:

[https://votacao-assembleia.herokuapp.com/associado/{id}](https://votacao-assembleia.herokuapp.com/associado/%7bid%7d)

ou [http://localhost:8080/associado/{id}](http://localhost:8080/associado/%7bid%7d)

{id} = número da pauta que se quer excluir

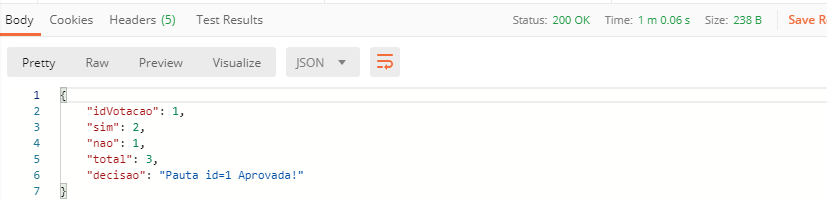
Exemplo:



### Resultado

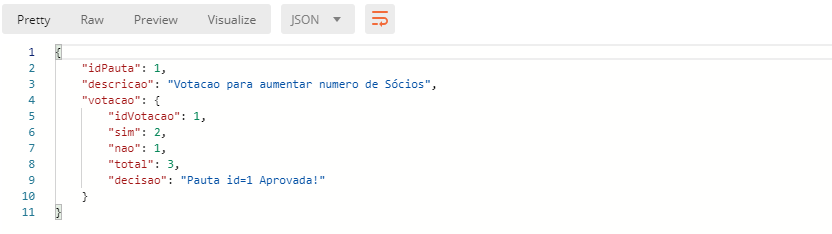
O resultado da votação é informado ao terminar o tempo de votação como resposta desta solicitação.





Também é possível verificar o resultado consultando a pauta desejada.

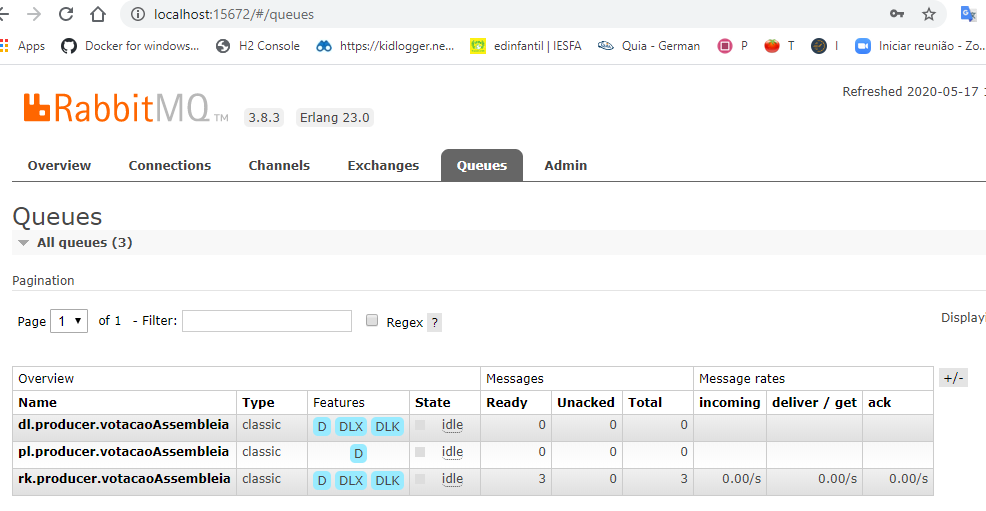




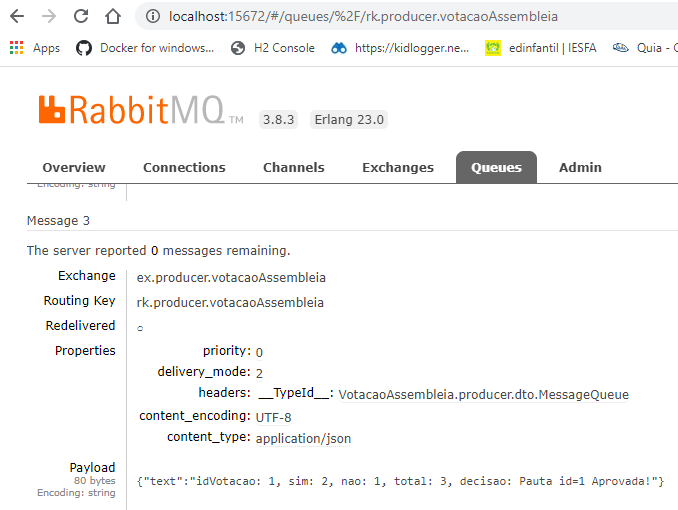
#### Mensageria e Filas

Nesta solução o resultado da votação é enviado automaticamente para um serviço de mensageria. Foi escrito um pacote no Java chamado Producer que produz esta mensagem e envia para o RabbitMQ, ficando na fila para ser consumida conforme disponibilidade de um Consumer.

Esta mensagem fica armazenada na fila do RabbitMQ, conforme assinalada na imagem abaixo:



Pode-se verificar o conteúdo da mensagem conforme abaixo:



O utilizador também pode enviar uma mensagem complementar ao resultado da votação, disparando via comando POST.

<https://votacao-assembleia.herokuapp.com/send> ou <http://localhost:8080/send>

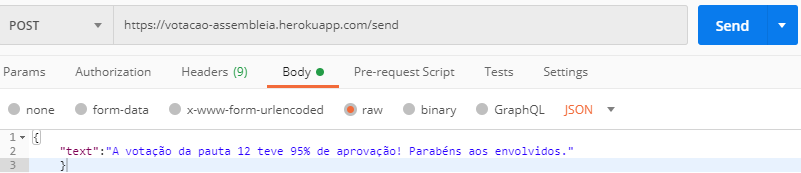
No Body deve-se usar o padrão Json conforme a seguir:

{

“text”:”Escreva sua mensagem complementar aqui”

}

Exemplo:





Referencia:

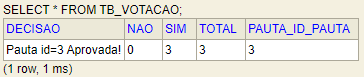
<https://www.udemy.com/course/rabbitmq-com-springboot-e-docker/learn/lecture/17405674#overview>

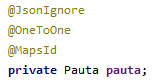
## Programa Java

A seguir, descreve-se pontos importantes durante o desenvolvimento

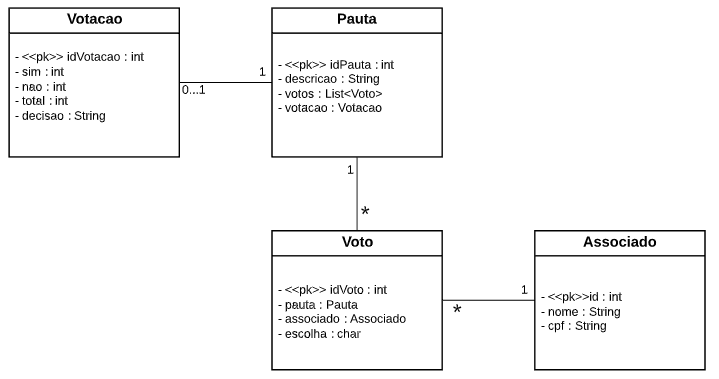
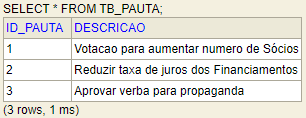
### Tabela de Dados

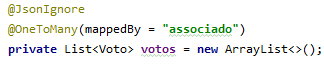
A solução tem a estrutura de tabelas conforme abaixo. A imagem mostra a relação de tabelas juntamente com a programação JPA-Hibernate e resultado no banco de dados do H2-console.

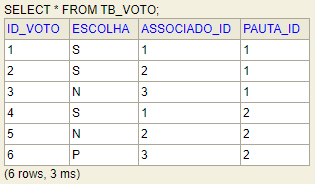
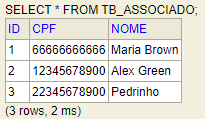










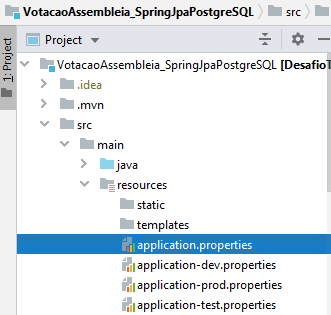


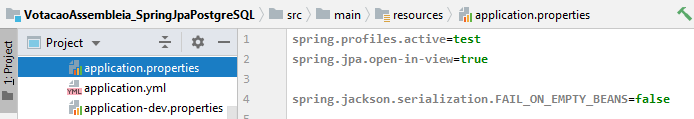
Desenhado em Lucidchart: <https://www.lucidchart.com/documents/edit/47b72a3f-6a36-4c11-bed0-0d635f7db075/0_0?beaconFlowId=48421A8D3D6A0B09#?folder_id=home&browser=icon>

### Configuração

O desenvolvedor tem três possibilidades de configuração de ambientes:

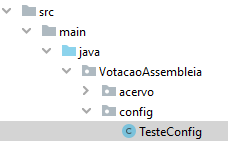
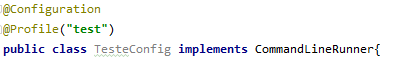
* application-test.properties: ambiente para testes utilizando o banco de dados H2;
* application-dev.properties: ambiente para desenvolvimento utilizando o bando de dados Postgres na máquina local;
* application-prod.properties: ambiente de produção usando bando de dados Postgres na nuvem utilizando o Heroku.





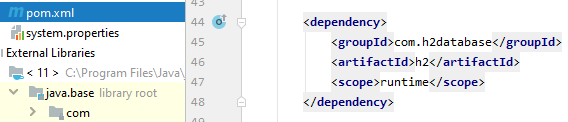
### Classe de Configuração

Classe auxiliar para salvar dados iniciais que carregam no inicio do programa no bando de teste.

### Banco teste H2

Um banco teste H2 console foi configurado para testes. A cada vez que o programa reinicia o banco zera todos os seus dados. É necessário acrescentar no arquivo pom.xml as dependências para este banco funcionar.



<http://localhost:8080/h2-console>

<http://localhost:8080/h2-console/login.do?jsessionid=2046bb4921a409a56a1848b8e008774c>

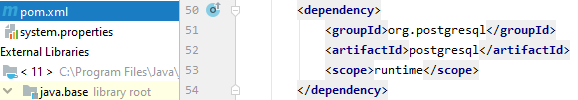
### Banco Postgre

O Postgre foi escolhido para armazenar os dados para o ambiente de desenvolvimento e para o de Produção.

#### Banco Local e Heroku pelo Postgre:

 <http://127.0.0.1:65024/browser/>

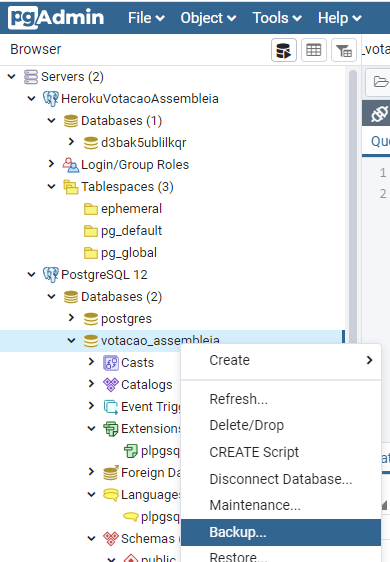
#### Dependências adicionadas:

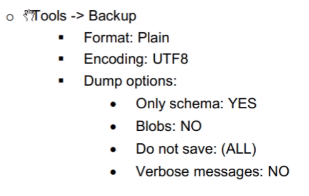


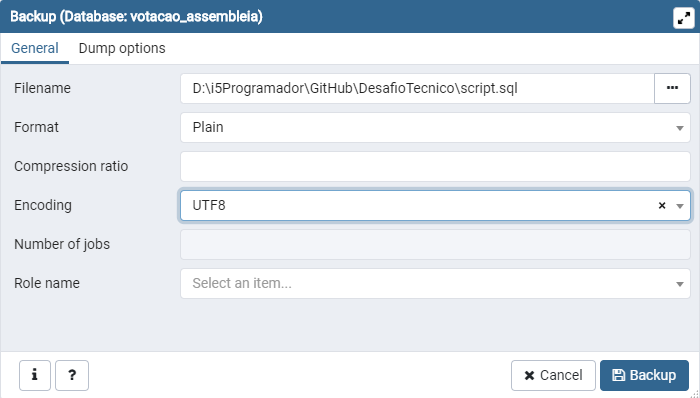
#### Obtendo script SQL a partir do PostgreSQL local

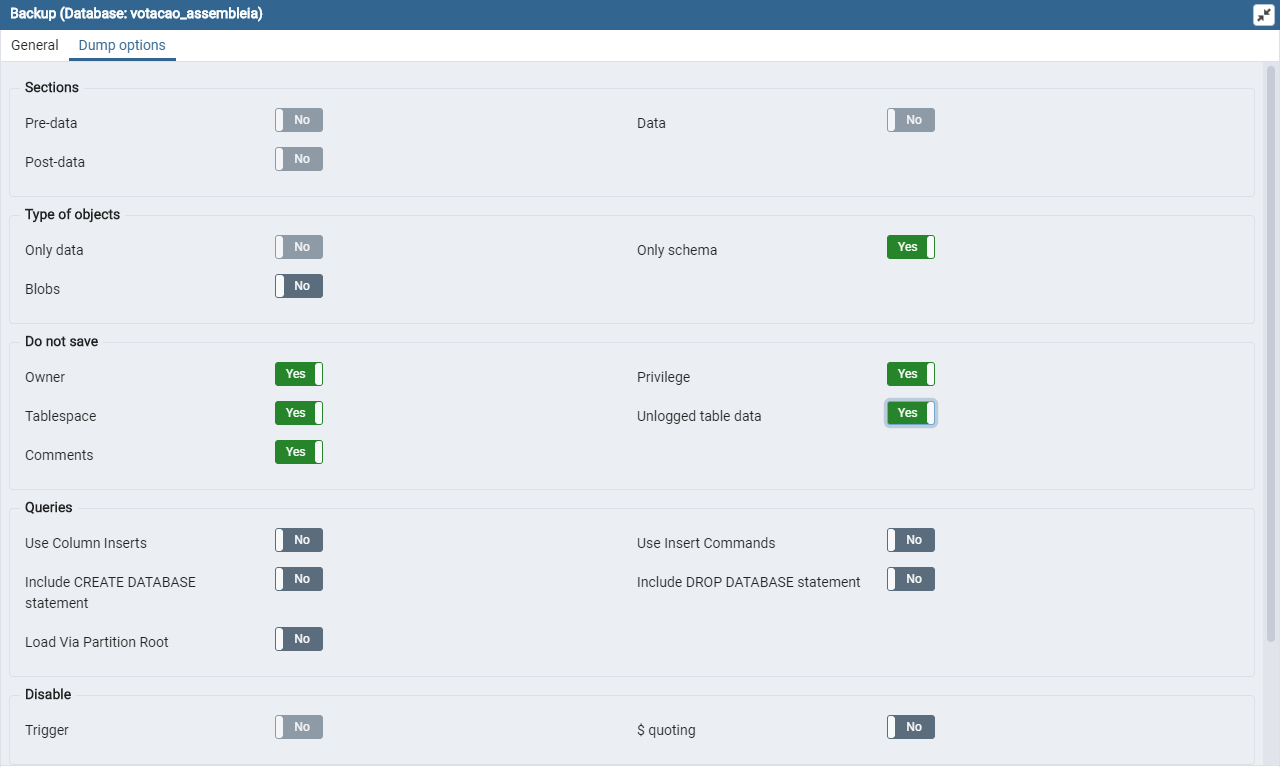
Fonte: Udemy

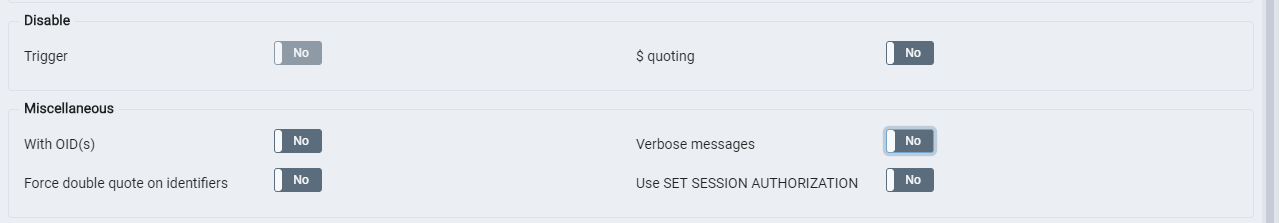
🡪 326. <https://www.udemy.com/course/java-curso-completo/learn/lecture/17057402#bookmarks>

 +

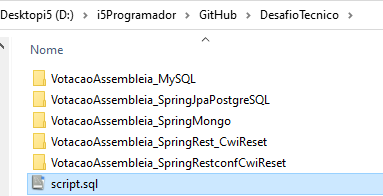
+

 +

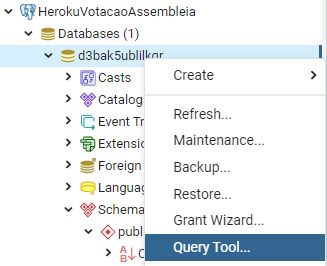


 +

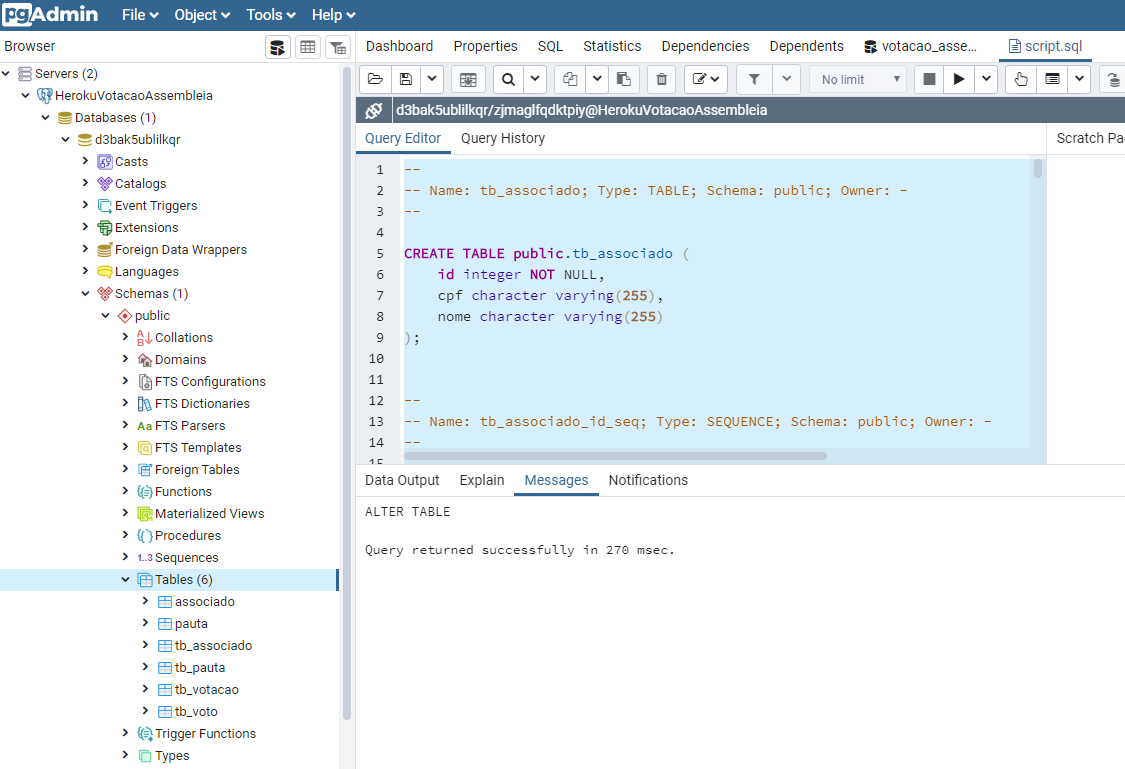
 +

 + Apagar as linhas do início do arquivo

#### Executando script SQL no servidor remoto - HEROKU

 + + 

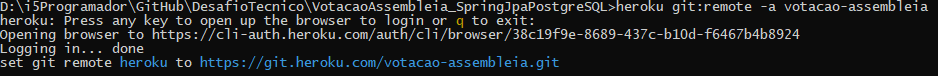
+ Ctrl A +



+ 

### Deploy do sistema no Heroku

1º)Faz este procedimento na primeira vez para sincronizar pasta local com app criado no heroku

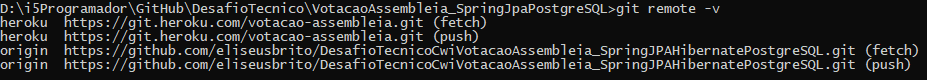
heroku git:remote -a votacao-assembleia

Gerou o endereço no heroku

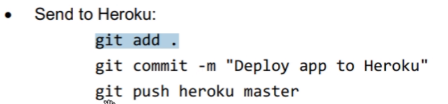
<https://git.heroku.com/votacao-assembleia.git>

Para verificar ser ficou ok:

git remote -v



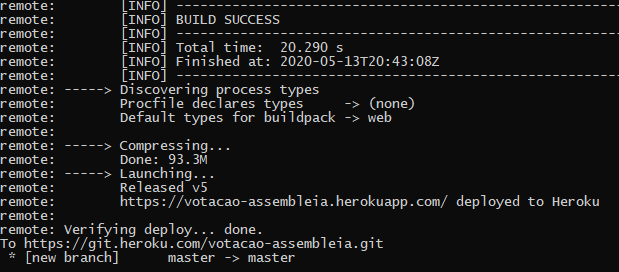
Enviar para o Heroku



git add .

git commit -m “Deploy app to Heroku”

git push heroku master



<https://votacao-assembleia.herokuapp.com>

### Tratamento de Erros e exceções:

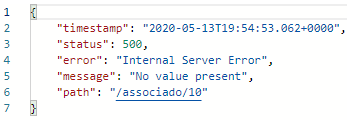
#### Pesquisa por Id não existente da Classe Associado:

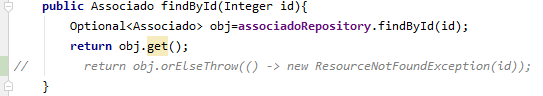
Mensagens foram personalizadas para a aplicação nos casos onde o usuário pesquisa por um id não existente:

Erro era Status 500 e agora 404

Mensagem padrão:

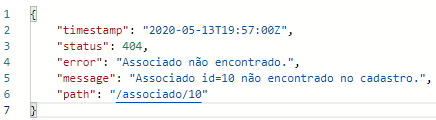


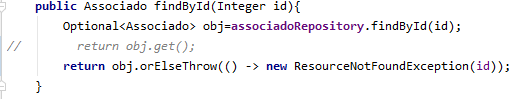




Mensagem Customizada:





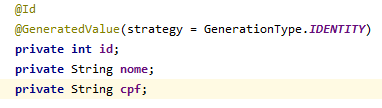


#### Cadastrar associado sem informações no nome ou cpf:

Situação Inicial: Erro e código





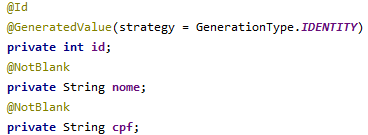
 Classe Entities(Camada de Domínio)

 Classe Rest (Camada de Controller):

Corrigido mensagem



Código adicionado Annotation JPA @NotBlank e @Valid



Código adicionado @Valid



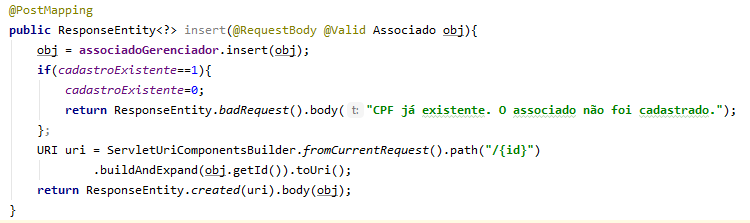
 

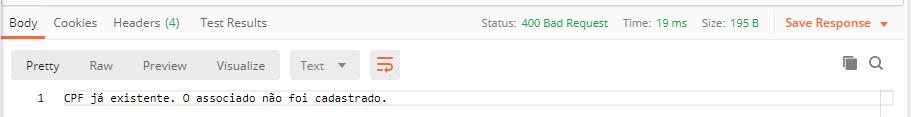
Referencia: <https://www.udemy.com/course/fundamentos-de-programacao-com-java/learn/lecture/18619834#overview>

#### Adicionado mSG personalizada Ao cadastrar associado já existente

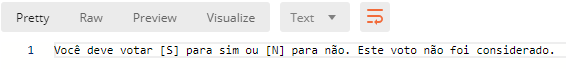
Mensagem original : Retornava o usuário já cadastrado

Mensagem Atualizada: Mensagem Status 400 Bad Request personalizada



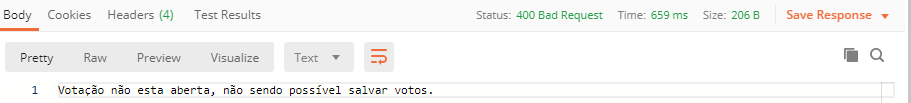


#### Adicionado Mensagem QUANDO COLOCA LETRA ERRADA no Voto

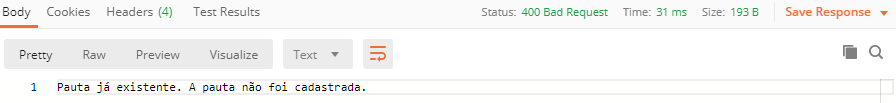


#### Adicionado Mensagem Quando Votação não ESTÁ aberta

Se uma votação não esta aberta (ainda não foi realizada ou já foi realizada) o programa não deve permitir votos. Também deve dar mensagem que não é possível votar antes de contar o tempo default ou definido pelo usuário para votação.



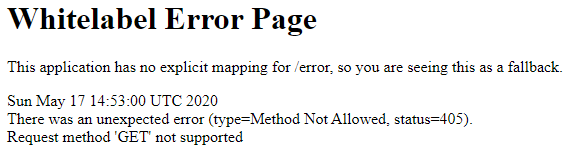
#### Mensagem quanto cadastra Pauta já existente



#### Heroku

Exceções vinda do Heroku ainda não foram tratadas nesta solução.





## Instalações

### RabbitMQ

Referência:

<https://dev.senior.com.br/documentacao/instalacao-simplificada/2-instalando-o-rabbitmq/>

<https://www.youtube.com/watch?v=3sEPqKrFQf8>

<https://www.erlang.org/downloads/23.0>

<https://www.rabbitmq.com/install-windows.html>

O RabbitMQ pode ser utilizado via Docker ou diretamente na máquina local.

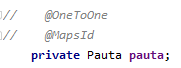
## Aprendizados

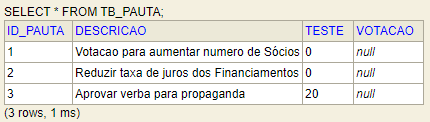
### Tabela de Dados

Testes para verificar comportamento dos comandos JPA para definição de relacionamento entre as Classes Pauta e Votação. Para cada Estudo tem uma configuração e um resultado de comportamento das tabelas de dados.

Classe Pauta: Classe Votação

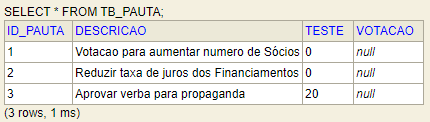
Estudo 1:



Estudo 2:





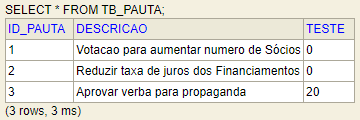
Estudo 3:





Estudo 4:

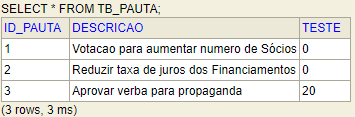






Estudo 5:





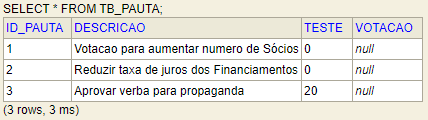
Estudo 6:



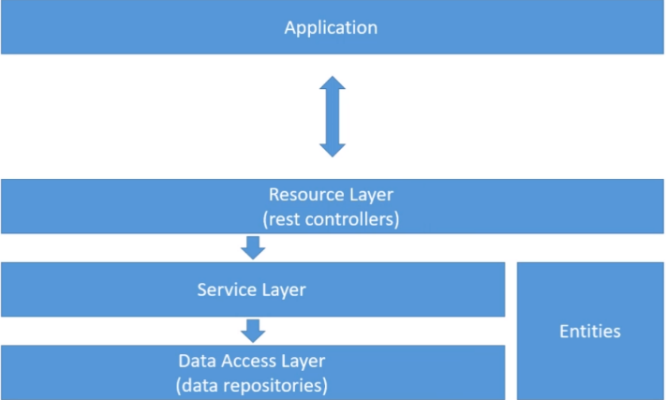


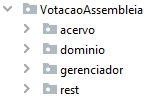
Estudo 7:



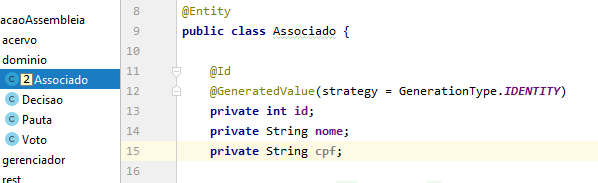


### Conceitos:



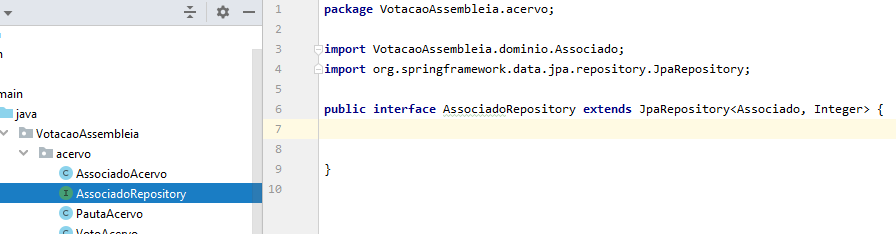


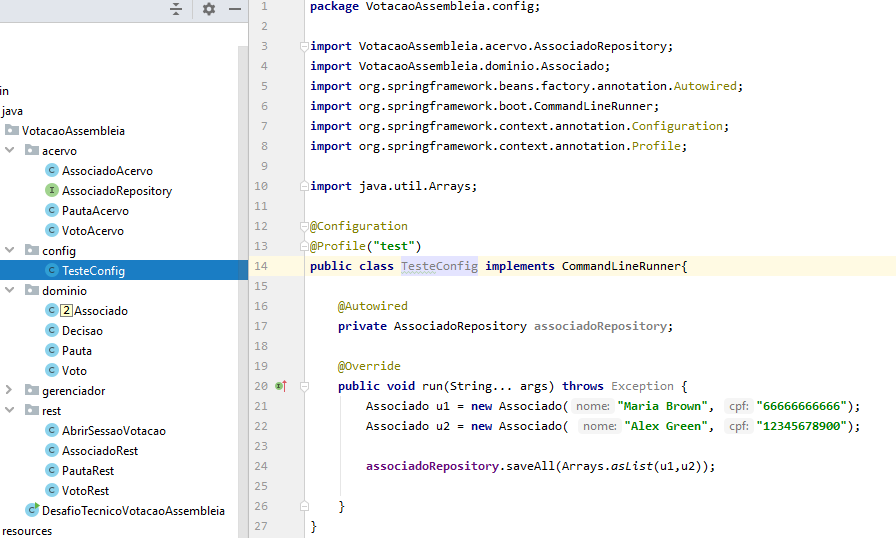
1)Classe domínio



2)Classe Repository(começa por baixo)

Criado uma interface

+ Teste Config



### Integração Package Mensageria com Votação

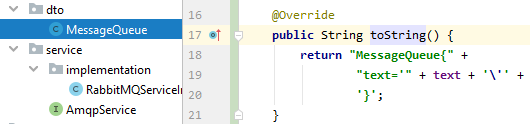
No Postman enviei:

"text":"idVotacao: 1, sim: 2, nao: 1, total: 3, decisao: Pauta id=1 Aprovada3!"

Em java recebi:

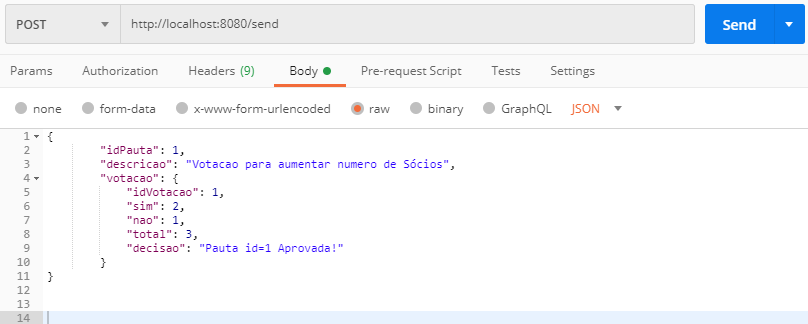
message dentro de postMapping: VotacaoAssembleia.producer.dto.MessageQueue@26bb0b39

Após colocar toString na classe recebi o texto convertido para string. Ao invés de imprimir o endereço de memória passou a imprimir o conteúdo:



message dentro de postMapping: MessageQueue{text='idVotacao: 1, sim: 2, nao: 1, total: 3, decisao: Pauta id=1 Aprovada3!'}

Ao enviar a mensagem no Postman:



Ele recebe no java:

message dentro de postMapping: MessageQueue{text='null'}

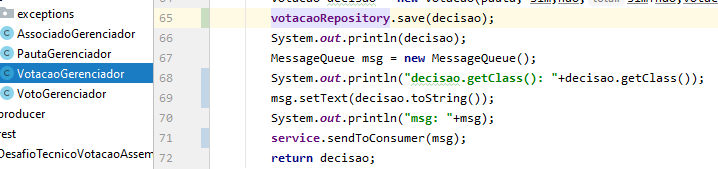
Usando o java para enviar a mensagem Classe VotacaoGerenciador:

msg.setText(**"teste"**);

Ele recebe em 

message dentro de postMapping: MessageQueue{text='teste'}

Alterado o código conforme abaixo foi possível receber a mensagem corretamente:





msg: MessageQueue{text='Votacao{idVotacao=1, sim=2, nao=0, total=2, decisao='Pauta id=1 Aprovada!'}'}

## Pendências e Melhorias:

Criar explicit mapping for /error para exceções vinda do Heroku;

Talvez seja possível tirar equals e hashCode em alguns dos códigos;

Verificar necessidade de Colocar @NotBlank e @Valid nas outras classes;

Como usar o navegador para dar Comandos POST, PUT e DELETE;

Alterar programação para não utilizar variáveis de controle para mensagens REST de erros.

## Conclusões

Este desafio foi de extrema importância para colocar em prática os conhecimentos adquiridos no CWI Reset e pesquisar sobre novos conceitos e conhecimentos. Ter a capacidade de pesquisar e adequar ao seu projeto tecnologias ainda não conhecidas, acredito que pode diferenciar um futuro profissional de TI.

Neste desafio foi necessário pesquisar sobre Banco de Dados, converter dados entre bancos relacionais e linguagem orientada a objeto através de JPA(**Java Persistence API)** e Hibernate. Também foi necessário aprender sobre como colocar os dados e o aplicativo na nuvem. A solução encontrada para esta aplicação foi utilizar o Heroku e o PostgreSQL.

Já em relação a mensageria a melhor alternativa encontrada nas pesquisas foi utilizar o RabbitMQ, em função do material disponível e por ser um dos softwares mais utilizados para esta função.

Estes assuntos todos são novidades para mim, mas me sinto feliz e realizado por entregar o trabalho no nível que consegui. Sei que tenho muito a melhorar no desafio realizado e tarefas futuras, mas com certeza, já evoluí muito.

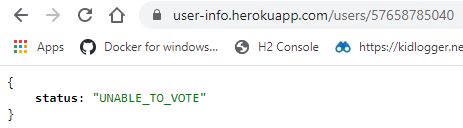
## Anexos

### Variação no api GET [https://user-info.herokuapp.com/users/{cpf}](https://user-info.herokuapp.com/users/%7bcpf%7d):

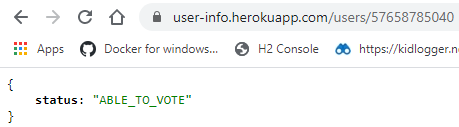
Conforme o momento da pesquisa um mesmo CPF pode ter seu Status alterado:

<https://user-info.herokuapp.com/users/57658785040>

CPF 57658785040 Inválido

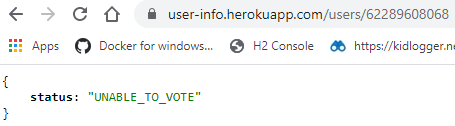


CPF 57658785040 Válido



<https://user-info.herokuapp.com/users/62289608068>

CPF 62289608068 Inválido



CPF 62289608068 Válido

