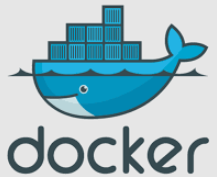
Desafio Técnico CWI

Solução BACK-END para Gerenciar Sessões de Votação em Assembleias

Autor: Eliseu Silveira Brito Data: 15/05/2020

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 3

Utilização pelo Usuário 3

Operação 4

 Pauta 4

 Cadastro 4

 Listagem 5

 Edição 6

 Exclusão 6

 Sessão de Votação 7

 Abertura Sessão 7

 Tempo de Votação 7

 Comportamento 8

Localhost 8

Heroku 8

 Votos 10

 Cadastro 10

 Listagem 11

 Associados 12

 Cadastro 12

 Listagem 13

 Edição 14

 Exclusão 15

 Resultado 15

 Mensageria e Filas 16

Programa Java 17

Tabela de Dados 17

Configuração 19

Classe de Configuração 19

Banco teste H2 20

Banco Postgre 20

 Banco Local e Heroku pelo Postgre: 20

 Dependências adicionadas: 20

 Obtendo script SQL a partir do PostgreSQL local 20

 327. Executando script SQL no servidor remoto - HEROKU 22

 329. Deploy do sistema no Heroku 24

Tratamento de Erros e exceções: 25

 Pesquisa por Id não existente da Classe Associado: 25

 Cadastrar associado sem informações no nome ou cpf: 25

 Adicionado mSG personalizada Ao cadastrar associado já existente 27

 Adicionado Mensagem QUANDO COLOCA LETRA ERRADA no Voto 27

 Adicionado Mensagem Quando Votação não ESTÁ aberta 27

 Mensagem quanto cadastra Pauta já existente 28

 Heroku 28

Aprendizados 28

Tabela de Dados 28

Pendências e Melhorias: 31

Conclusões 31

# 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.

O resultado da votação pode ser informado para o restante da plataforma, isto é feito através de mensageria. A solução possui um Producer que pode ser acessado via end point através de comando Get. Quando a sessão de votação fechar o usuário pode postar uma mensagem com o 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>

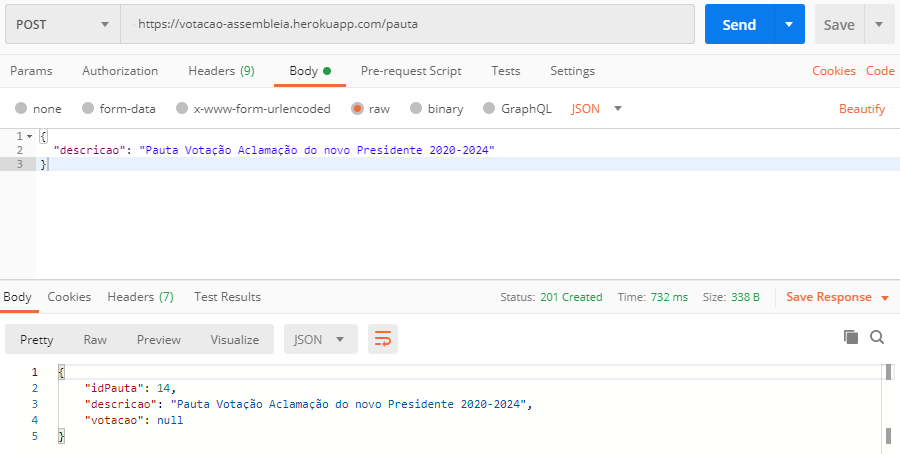
No Body deve-se usar o padrão Json conforme a seguir:

{

"descricao":"Título da Pauta"

}

Exemplo no Postman:

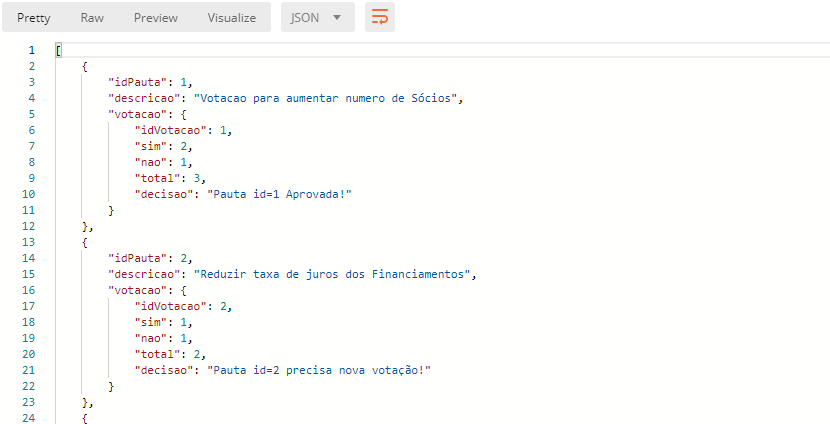


##### Listagem

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

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





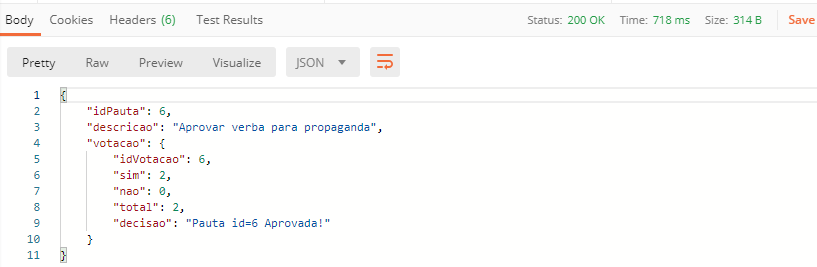
Para listar uma pauta especifica usa-se o comando GET / número da pauta desejada

[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

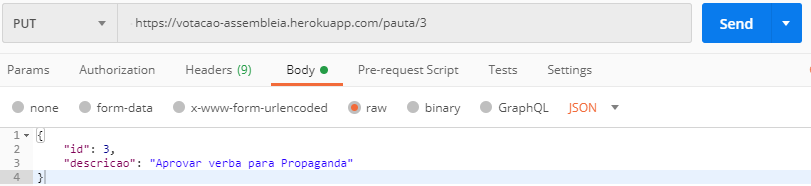
No Body deve-se usar o padrão Json conforme a seguir:

{

"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

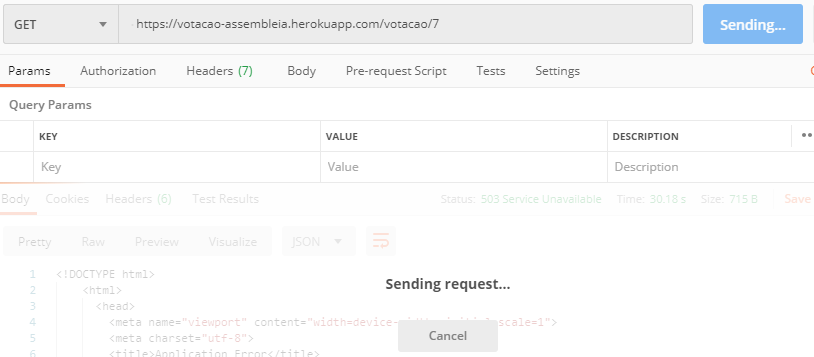
##### Abertura Sessão

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>

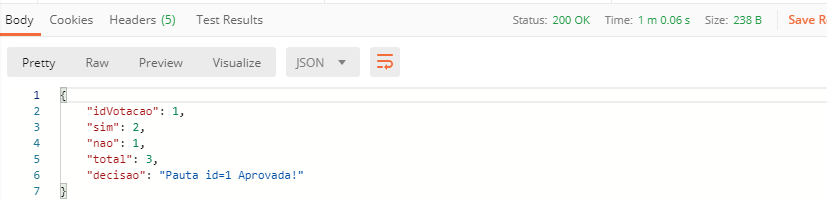
##### Comportamento

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

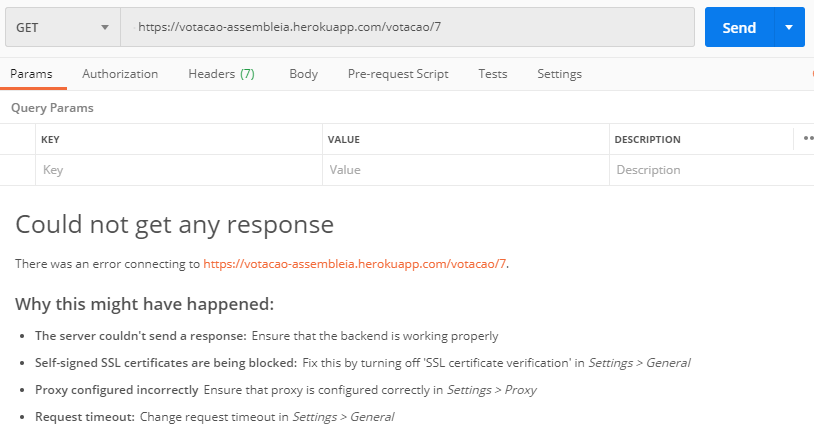


Imagem 1 - Could not get any response

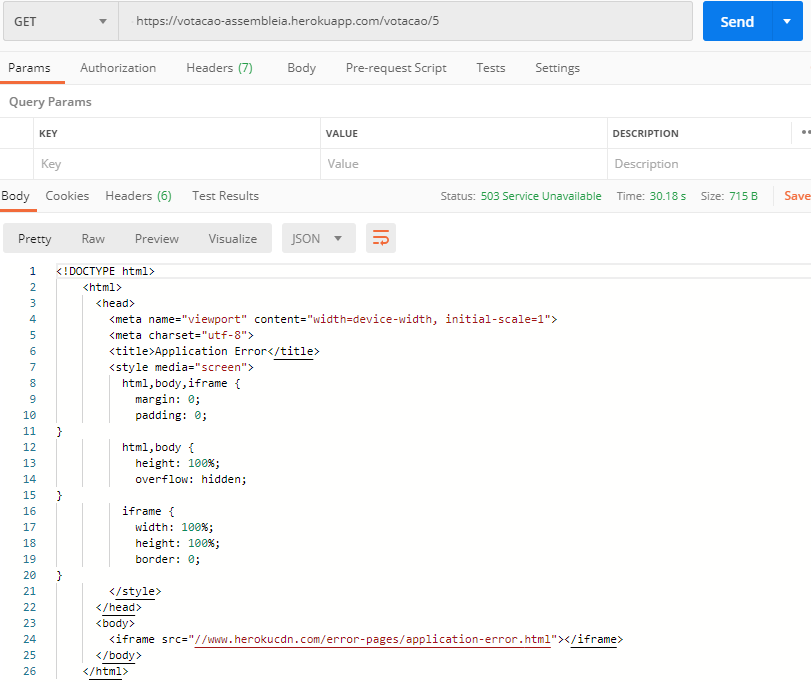
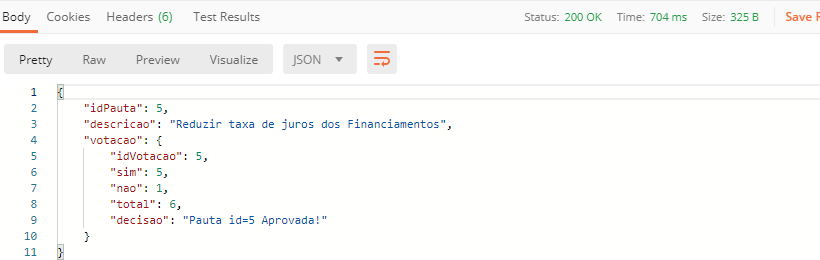


Imagem 2 - Status: 503 Service Unavailable

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)

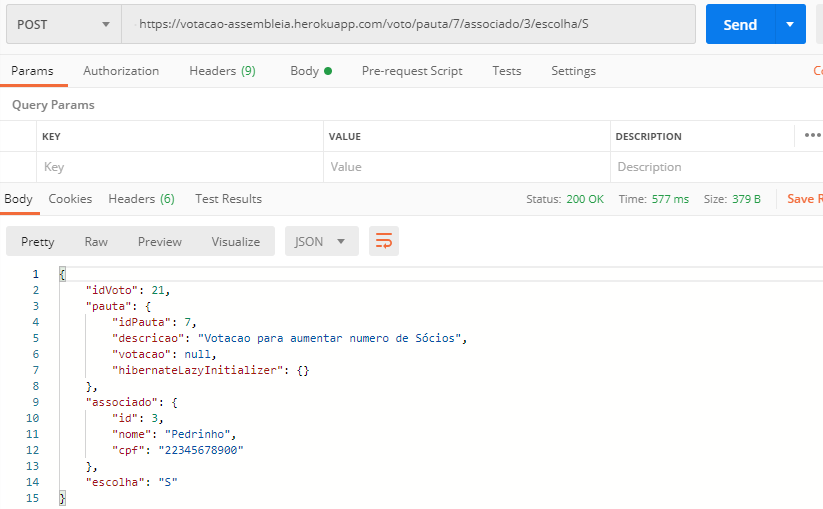
{idPauta} = Número da Pauta que se quer votar;

{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>

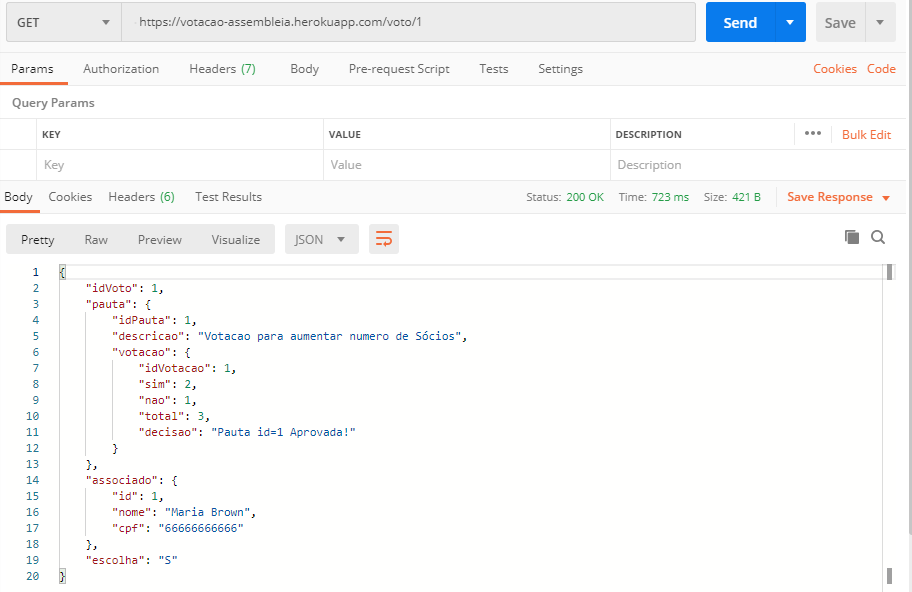
Para buscar um voto especifico:

<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>

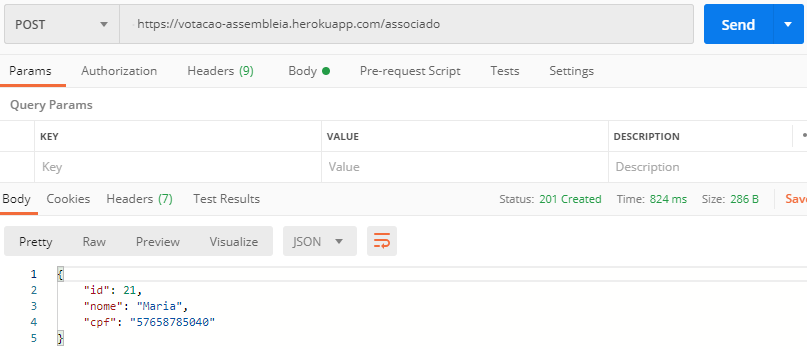
com o Body Json:

{

"nome":"Nome do Associado",

“cpf”:”xxxxxxxxxxx”

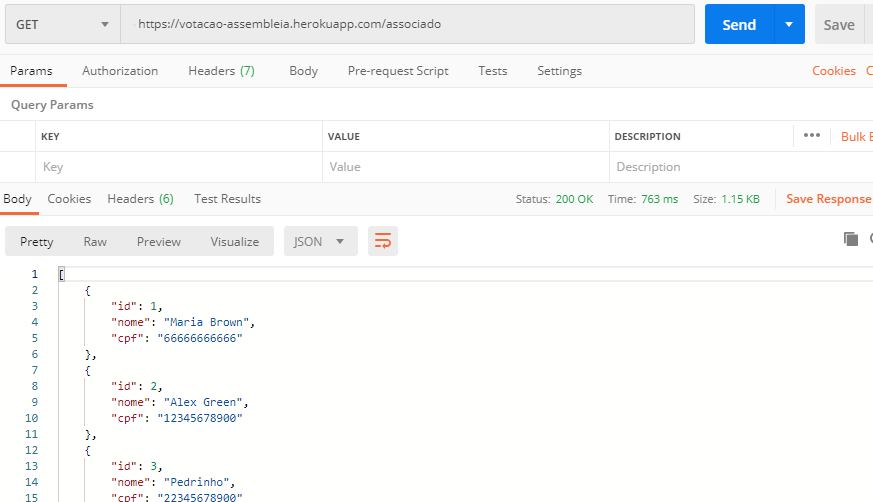
}



##### Listagem

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

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



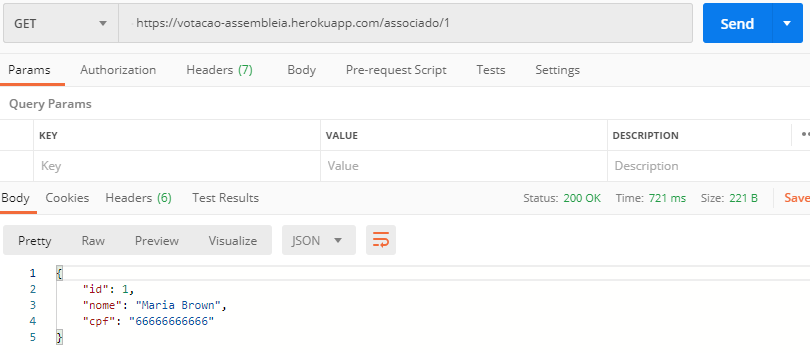
Para listar associado especifico:

[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:

{

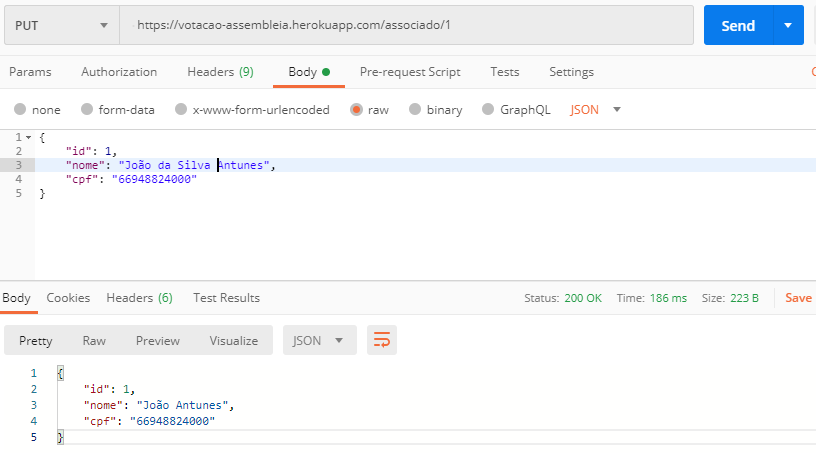
“id”:1,

"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

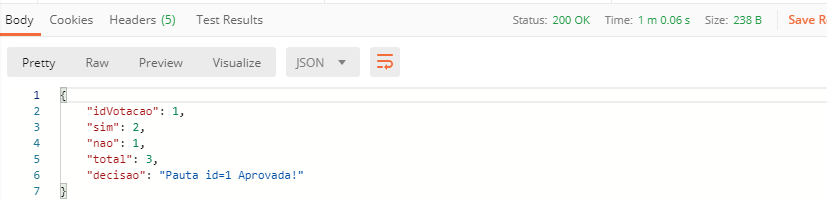
Exemplo:



#### Resultado

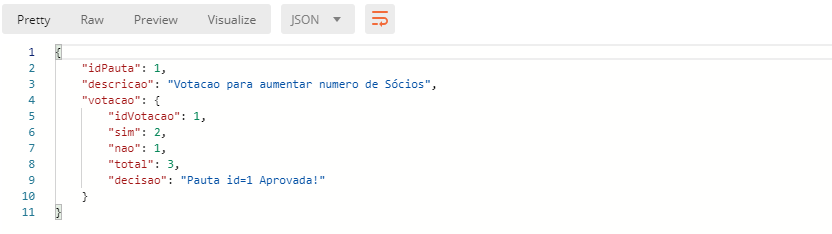
O resultado da votação é informado ao terminar o tempo de votação como resposta desta solicitação.





Também é possível verificar o resultado consultando a pauta desejada.





##### Mensageria e Filas

Nesta solução o resultado da votação pode ser enviado 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 disparada é via um endpoint via Postman utilizando o comando POST.

<http://localhost:8080/send>

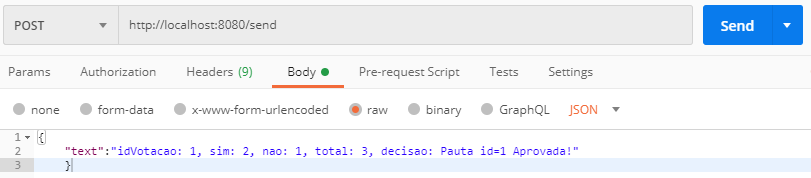
No Body deve-se usar o padrão Json conforme a seguir:

{

“text”:”Resultado da Votação”

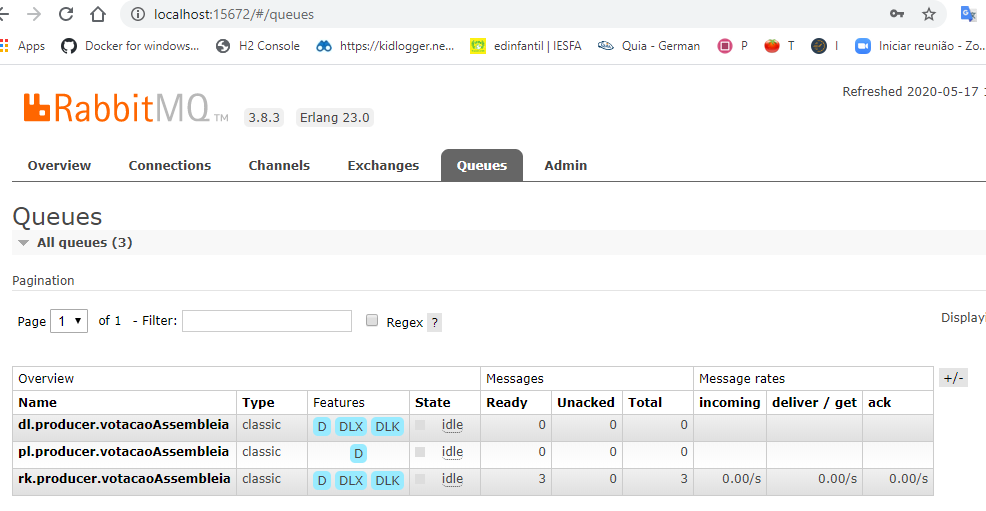
}

Exemplo:

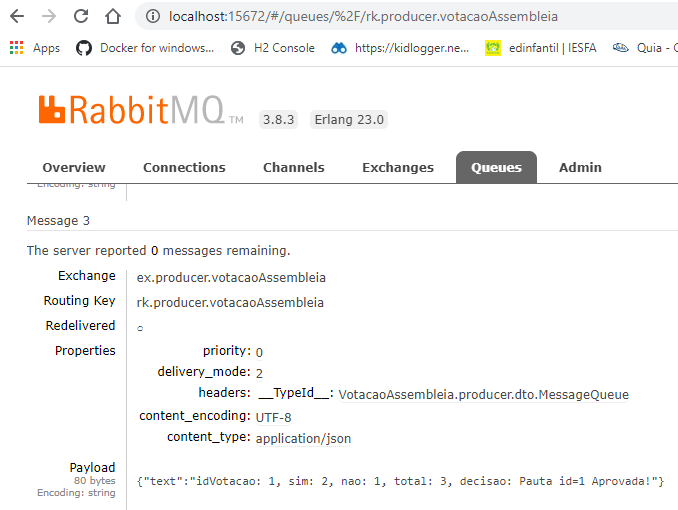




Esta mensagem fica armazenada na fila do RabbitMQ, conforme assinalada na imagem abaixo:



Pode-se verificar o conteúdo da mensagem conforme abaixo:



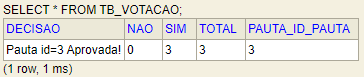
Uma opção de ativar o endpoint automaticamente ao termino de cada votação é possível e deverá estar disponível em versão futura desta solução.

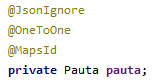
## Programa Java

A seguir, descreve-se pontos importantes durante o desenvolvimento

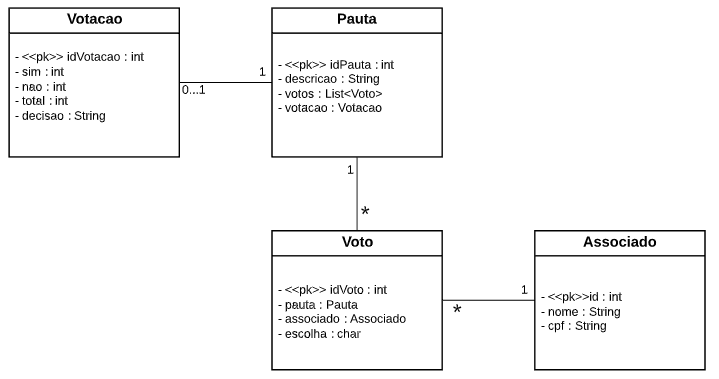
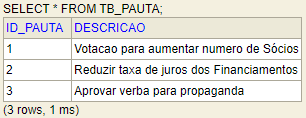
### Tabela de Dados

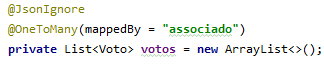
A solução tem a estrutura de tabelas conforme abaixo. A imagem mostra a relação de tabelas juntamente com a programação JPA-Hibernate e resultado no banco de dados do H2-console.

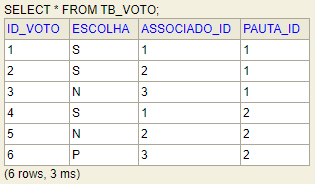
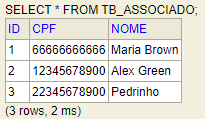










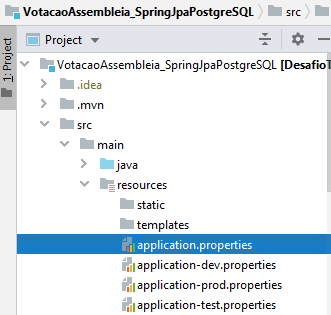


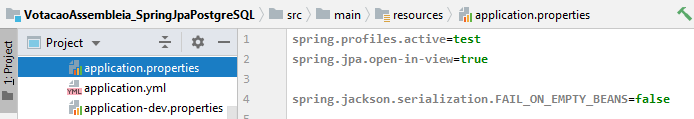
Desenhado em Lucidchart: <https://www.lucidchart.com/documents/edit/47b72a3f-6a36-4c11-bed0-0d635f7db075/0_0?beaconFlowId=48421A8D3D6A0B09#?folder_id=home&browser=icon>

### Configuração

O desenvolvedor tem três possibilidades de configuração de ambientes:

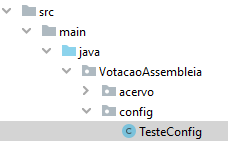
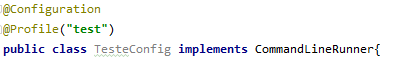
* application-test.properties: ambiente para testes utilizando o banco de dados H2;
* application-dev.properties: ambiente para desenvolvimento utilizando o bando de dados Postgres na máquina local;
* application-prod.properties: ambiente de produção usando bando de dados Postgres na nuvem utilizando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





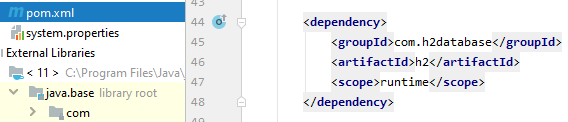
### Classe de Configuração

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>

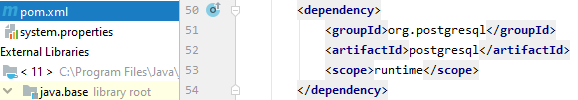
### Banco Postgre

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/>

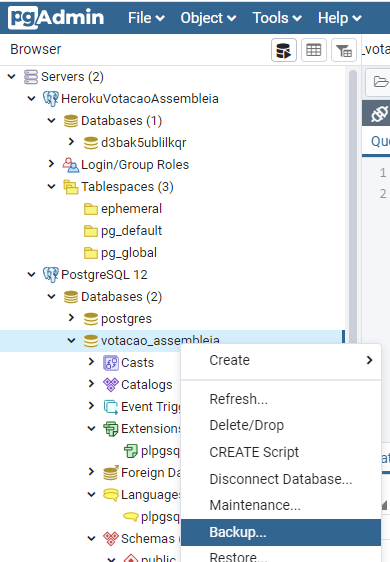
#### Dependências adicionadas:

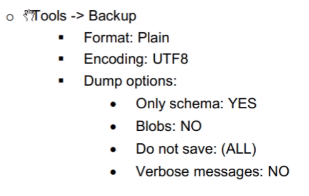


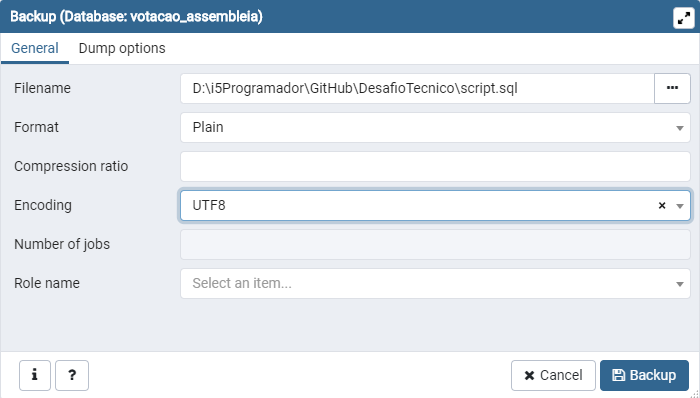
#### Obtendo script SQL a partir do PostgreSQL local

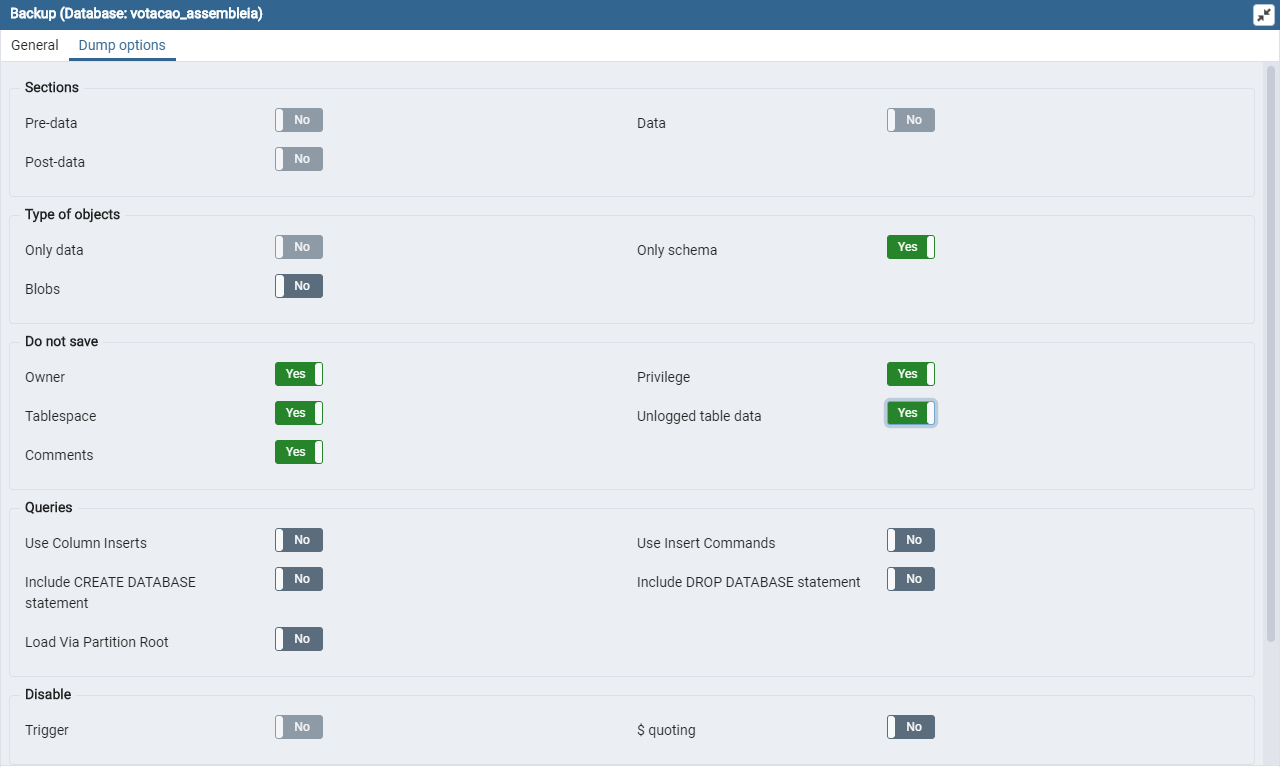
Fonte: Udemy

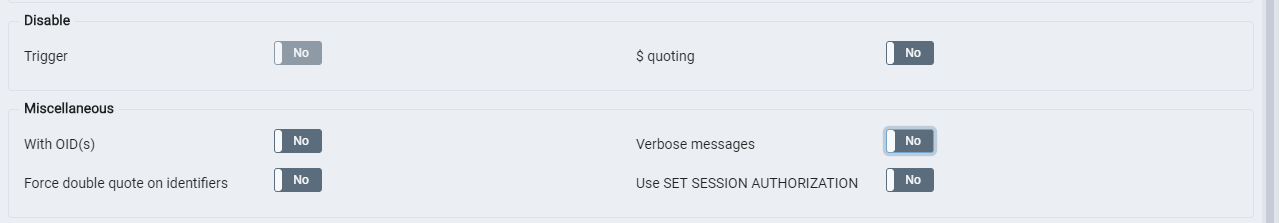
🡪 326. <https://www.udemy.com/course/java-curso-completo/learn/lecture/17057402#bookmarks>

 +

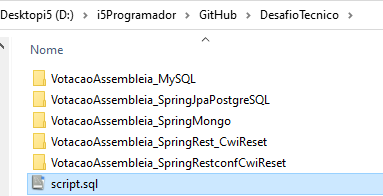
+

 +

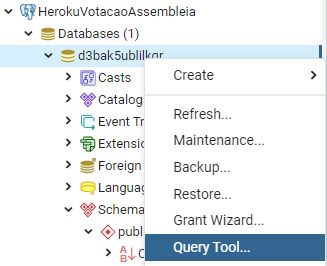


 +

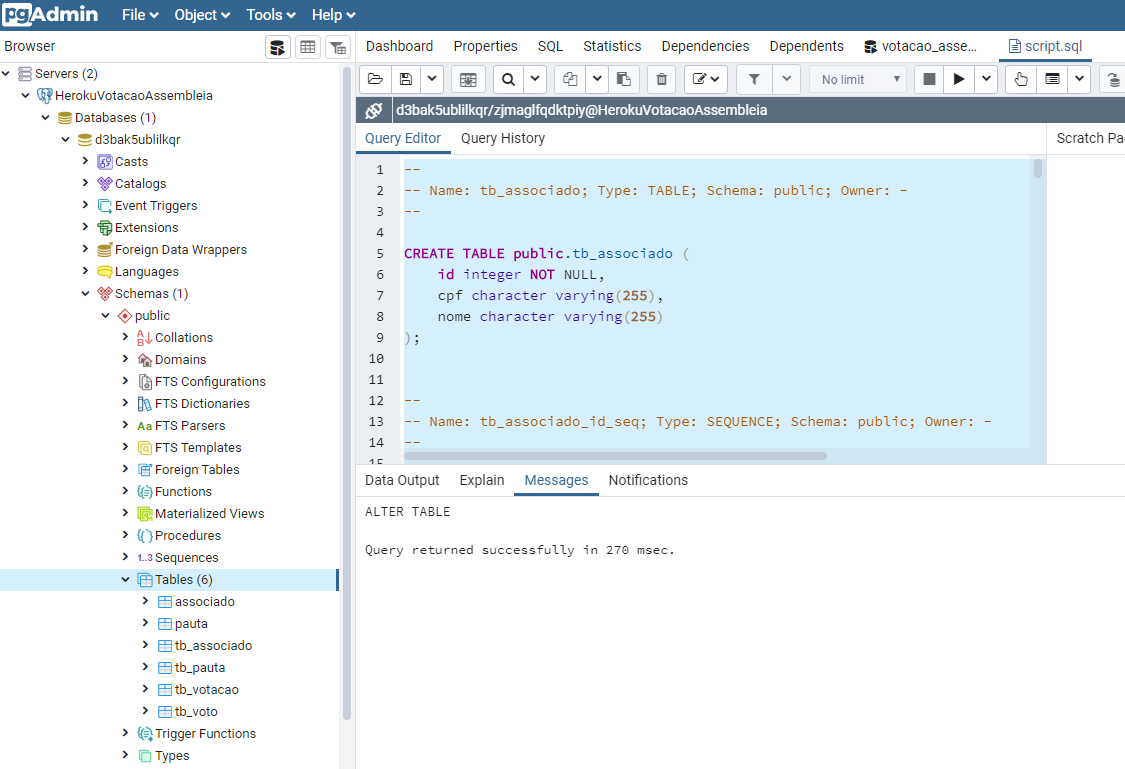
 +

 + Apagar as linhas do início do arquivo

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

 + + 

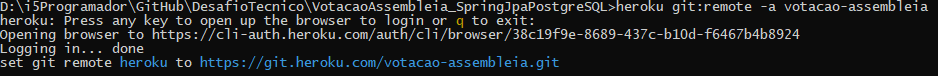
+ Ctrl A +



+ 

#### 329. Deploy do sistema no Heroku

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

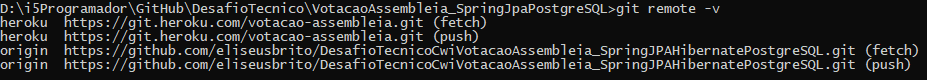
heroku git:remote -a votacao-assembleia

Gerou o endereço no heroku

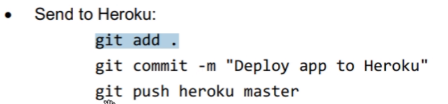
<https://git.heroku.com/votacao-assembleia.git>

Para verificar ser ficou ok:

git remote -v



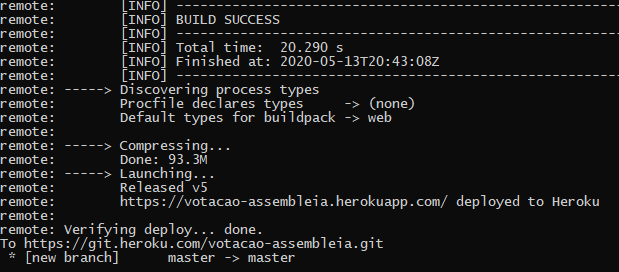
Enviar para o Heroku



git add .

git commit -m “Deploy app to Heroku”

git push heroku master



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

### Tratamento de Erros e exceções:

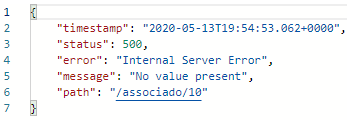
#### Pesquisa por Id não existente da Classe Associado:

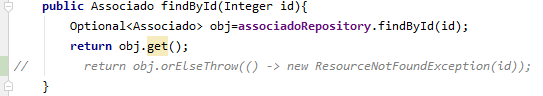
Mensagens foram personalizadas para a aplicação nos casos onde o usuário pesquisa por um id não existente:

Erro era Status 500 e agora 404

Mensagem padrão:

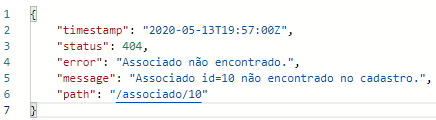


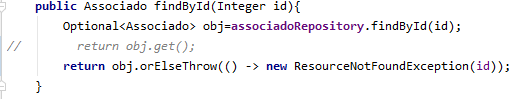




Mensagem Customizada:





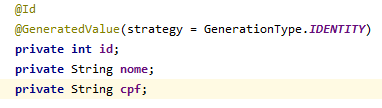


#### Cadastrar associado sem informações no nome ou cpf:

Situação Inicial: Erro e código





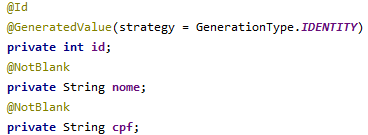
 Classe Entities(Camada de Domínio)

 Classe Rest (Camada de Controller):

Corrigido mensagem



Código adicionado Annotation JPA @NotBlank e @Valid



Código adicionado @Valid



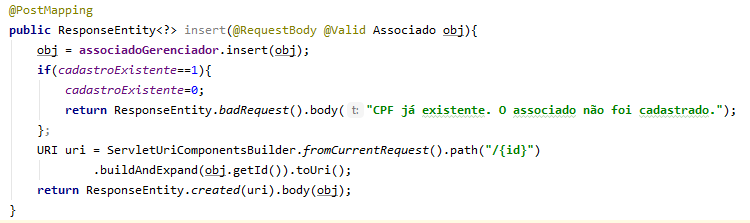
 

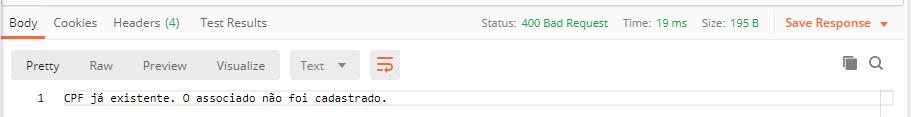
Referencia: <https://www.udemy.com/course/fundamentos-de-programacao-com-java/learn/lecture/18619834#overview>

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

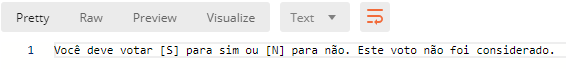
Mensagem original : Retornava o usuário já cadastrado

Mensagem Atualizada: Mensagem Status 400 Bad Request personalizada



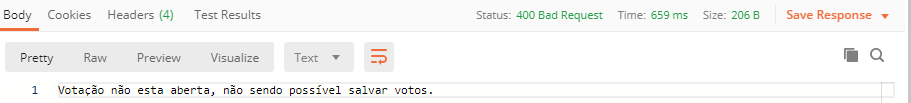


#### Adicionado Mensagem QUANDO COLOCA LETRA ERRADA no Voto

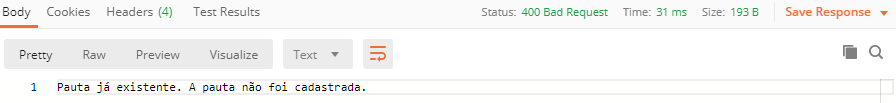


#### Adicionado Mensagem Quando Votação não ESTÁ aberta

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.



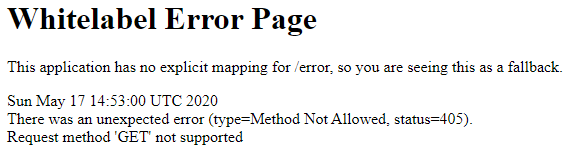
#### Mensagem quanto cadastra Pauta já existente



#### Heroku

Exceções vinda do Heroku ainda não foram tratadas nesta solução.





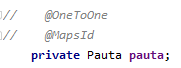
## Aprendizados

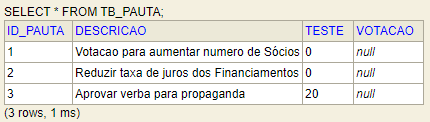
### Tabela de Dados

Testes para verifica 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

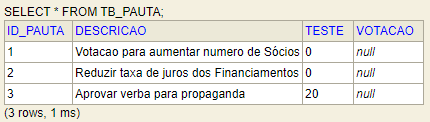
Estudo 1:



Estudo 2:





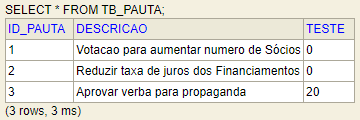
Estudo 3:





Estudo 4:

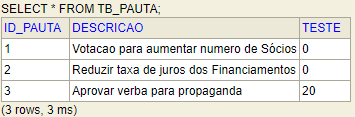






Estudo 5:





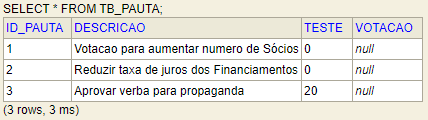
Estudo 6:

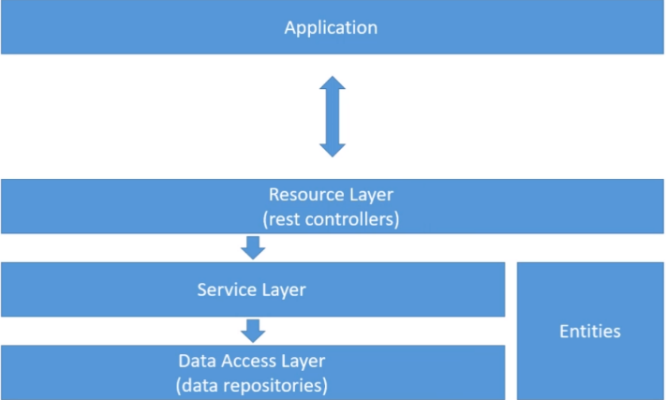


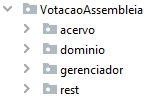


Estudo 7:

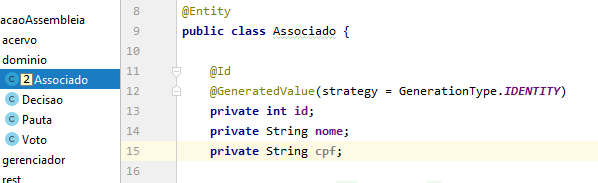




Conceitos:

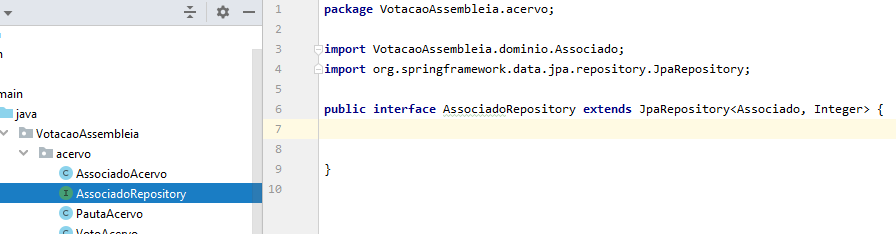


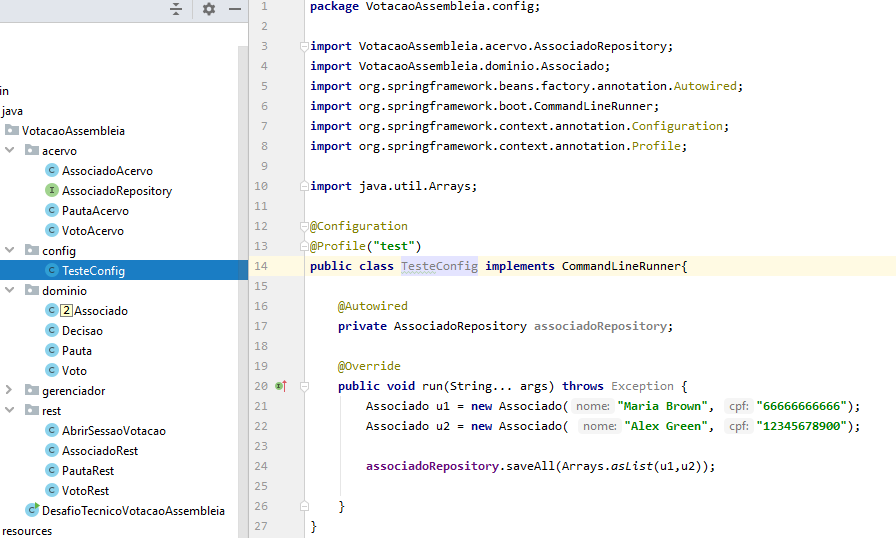
1)Classe domínio



2)Classe Repository(começa por baixo)

Criado uma interface

+ Teste Config



## Pendências e Melhorias:

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

## 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.