**Plano de aula 07 – WebHook (Python)**

Nesta aula falaremos sobre WebHooks com Python. Vamos seguir a arquitetura mostra na Figura 1.

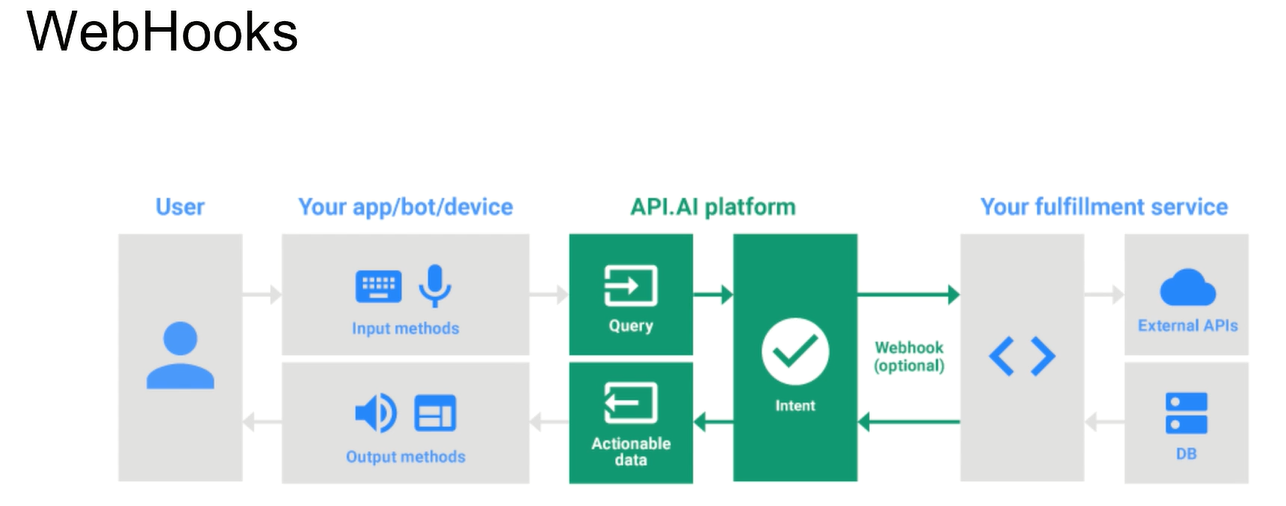


Figura 1

O usuário vai se comunicar com o bot através de um aplicativo, ou do **FaceBook Message**, por exemplo e, via uma entrada de texto ou voz e, então, a consulta vai ser direcionada para a plataforma **DialogFlow**, que pode através de **WebHook** (No nosso caso vamos utilizar Python), que pode acessar uma API externa para obter alguma informação que será retornada para o usuário, ou gravar alguma informação no banco de dados, ou recuperar alguma informação do banco de dados para retornar ao usuário.

**Criando uma Intenção para utilização de WebHook**

**Passo 1**

Uma integração via WebHook é uma forma de passarmos parâmetros de uma intente que recebeu match para um Web services e depois ler os resultados fornecidos por esse Web services para completar a resposta para o usuário.

* Então, vamos criar uma entidade curso para representar os cursos oferecidos. O nome da entidade é **curso** {Python, Machine Learning 🡪 ML, R e Hadoop}
* Depois vamos criar uma intente na qual o usuário pode perguntar o valor de um curso. O nome dessa intente é **curso.valor**
  + Qual o preço do curso de Machine Learning?
  + Quanto custa o curso de R?
  + Qual o valor do curso de Python?
  + @ Gostaria de saber o preço do curso @curso.curso
* Resposta padrão: O valor do curso $curso é de 100 reais.
* Antes de integrar ao backend salve e faças alguns testes.

**Passo 2**

**Criando uma conta no Heroku**

Para criar o backend nós precisamos de uma plataforma como serviço. Nós iremos fazer o deploy de nosso backend em uma plataforma de cloud computer. Poderia ser o AWS, poderia ser o Microsoft Azure ou até mesmo seu servidor dentro de sua empresa. Por questão de escolha, vamos fazer o deploy de nossa aplicação no Heroku, que é uma plataforma como serviço.

* Portanto, você deve criar uma conta no Heroku (<https://www.heroku.com>). Não esqueça de selecionar a linguagem primária como sendo Python.
* Dê uma olhada nos containers disponíveis e verifique as linguagens utilizadas por eles.
* Acesse <https://www.heroku.com/price> para vê os preços das assinaturas e suas limitações.

**Passo 2**

**Criando uma aplicação manualmente no Heroku**

* Se conecte ao Heroku com seu login e senha recém criada, e você será direcionado para o dashboard o Heroku, como mostra a Figura 2.

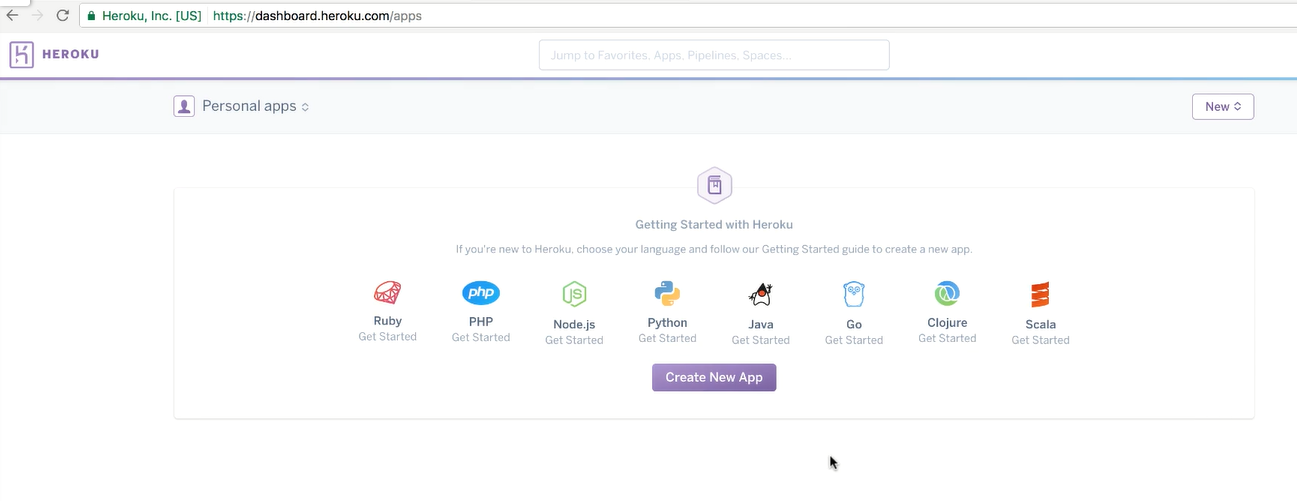
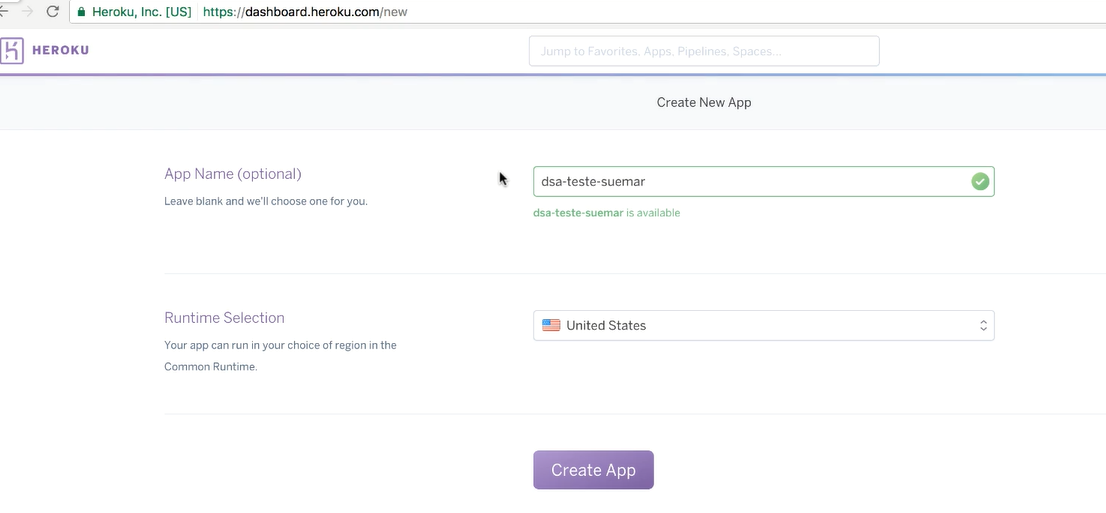


Figura 2 Dashboard o Heroku

* Clique no botão Create New App, você será direcionado para a tela da Figura 3, onde você pode informar o nome do bot que você deseja criar e a região onde será depositado esse bot.



**Figura 3**

* Se você clicar no botão Create App, você será direcionado para a tela da Figura 4, onde você pode verificar várias informações sobre o bot recém criado.

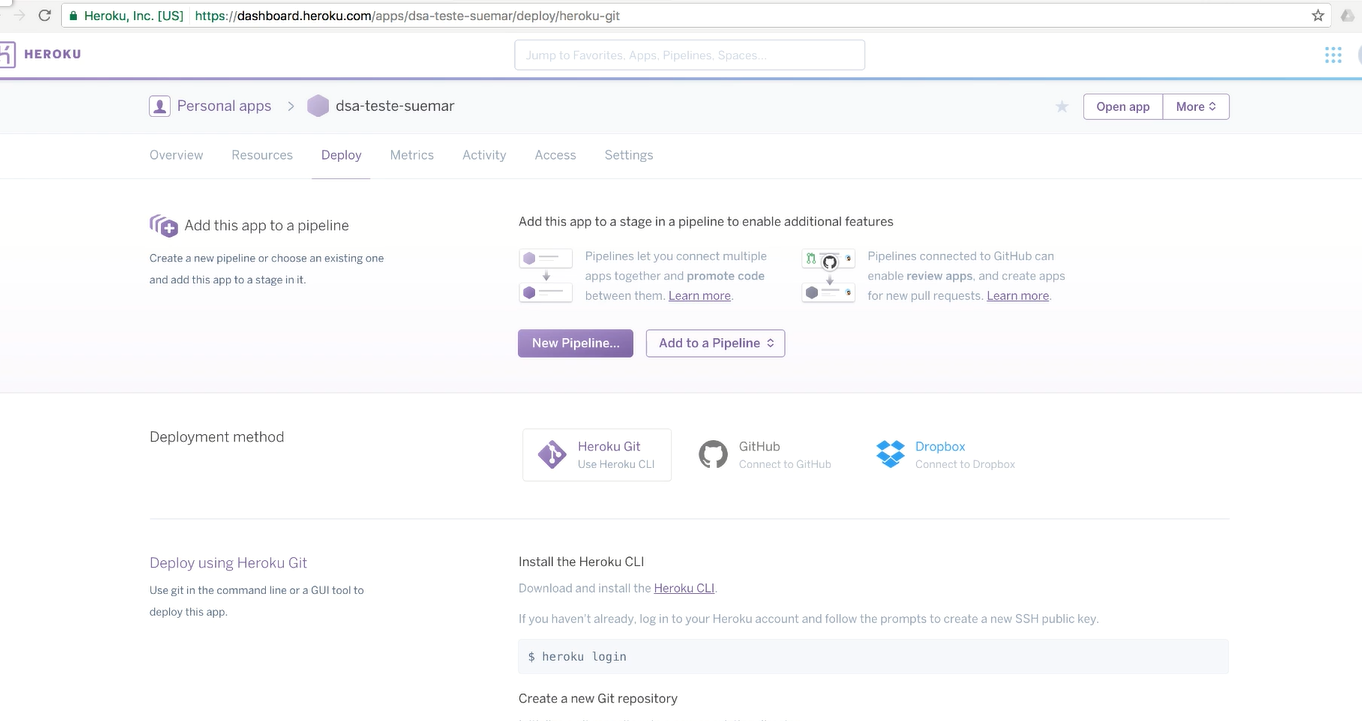


Figura 4

Verifique os diversos métodos de fazer deploy (Heroku CLI, GitHub, DropBox)

**Passo 4**

**Criando uma aplicação no Heroku via GitHub**

Vamos criar uma aplicação no Heroku a partir de um código fonte já disponível no GitHub. Para isto, vá até o link <https://github.com/svet4/shipping-costs-sample> .

Na seção inferior da página tem um botão **Deploy no Heroku**. Então clique nele para fazer o deploy.

Especifique o nome do aplicativo, por exemplo **ifrn-bot-teste**, e clique no botão **Deploy**.

Nós fizemos um deploy no código fonte que trata de frete de acordo com a região informada como parâmetro. Não isso que queremos, então, vamos precisar fazer algumas alterações no código fonte para atender nossos requisitos, que é informar sobre preços de cursos.

Para isso, vamos precisar clonar o código fonte para nossa máquina. Feita a clonagem, então, faremos as mudanças no código e, então, vamos subir essas mudanças para o Heroku fazendo um novo deploy, agora por linha de comando. Antes, teremos que instalar o Heroku CLI em nossa máquina, o faremos em seguida.

**Clonando o repositório do GitHub.**

Clique no Botão **MapApop** no dashboard do aplicativo <https://dashboard.heroku.com/apps/ifrn-bot-teste/deploy/heroku-git> , clique na opção Deploy. Nessa página são exibidas as opções de containers disponíveis, para esse estudo de caso, estaremos usando o Heroku CLI. Nessa mesma página tem uma opção **install Heroku CLI**, então, clique em **Heroku CLI**. Veja a Figura 5

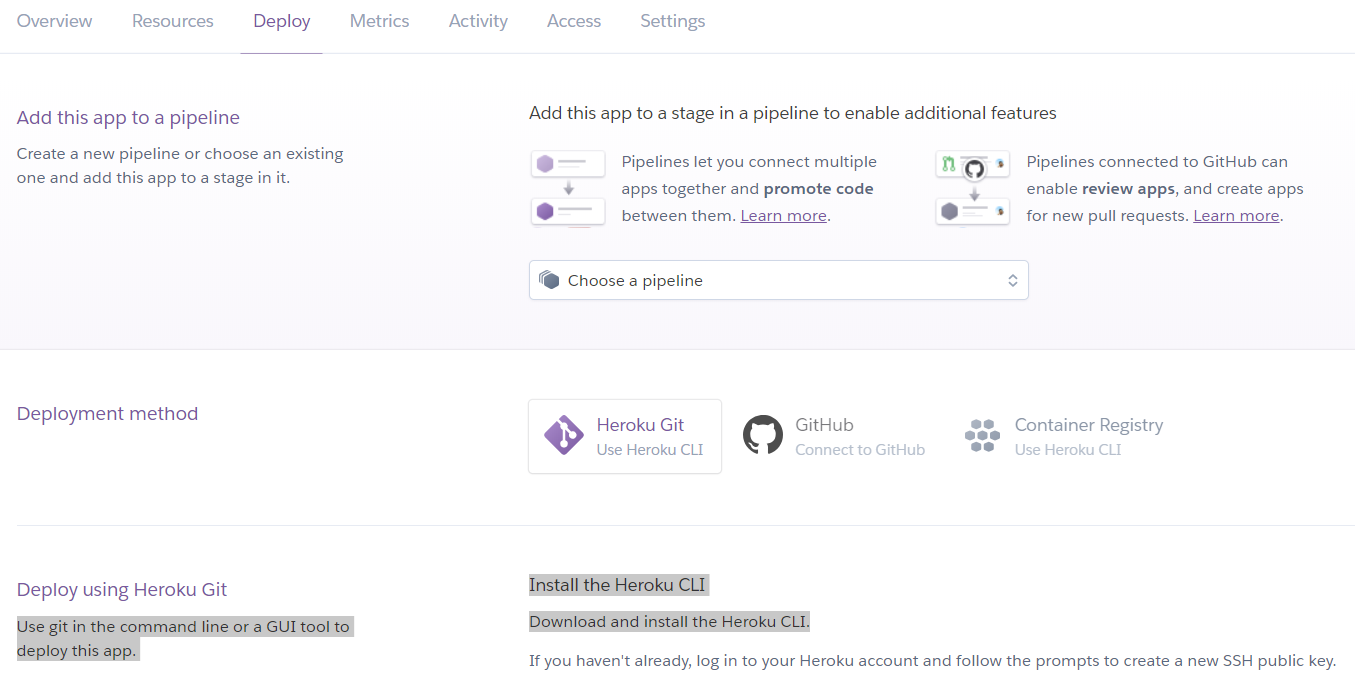
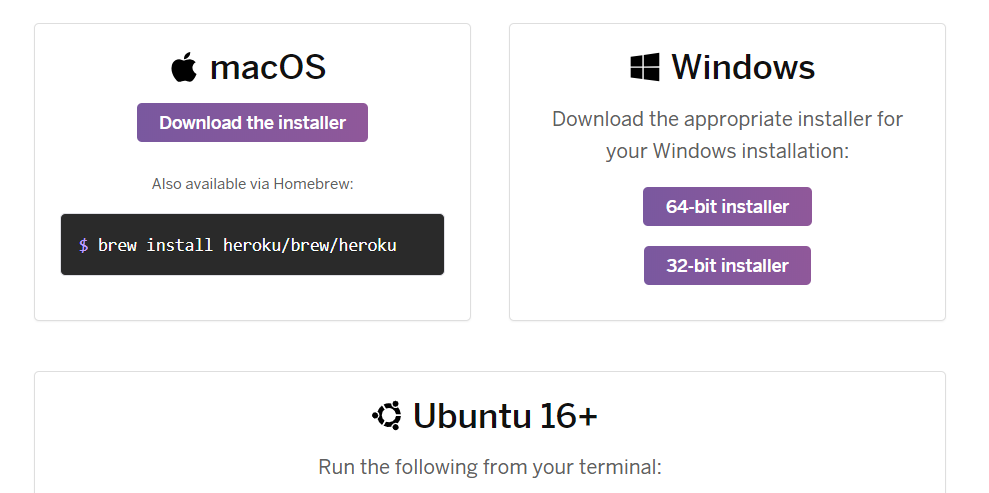


Figura 5

Quando você clica no link Heroku CLI, você é direcionado para uma tela onde temos diversas formas de baixar e instalar o Heroku CLI, dependendo do seu sistema operacional. Veja a Figura 6



**Figura 6**

Fazendo a instalação do Heroku CLI, abra um terminal e execute o comando:

**install heroku**

após instalação, conecte-se ao heroku

**$ heroku login**

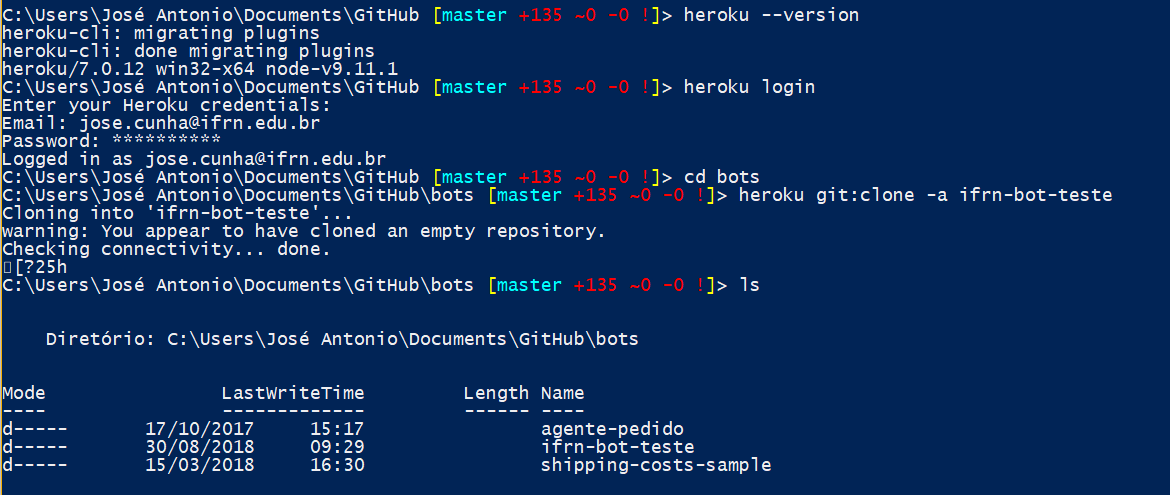


Clonando o repositório do GitHub

No meu caso, eu tenho criado uma pasta chamada bots, então vou abrir essa pasta e clonar o bot nela, para isso, vou rodar o comando

**$ heroku git:clone -a ifrn-bot-teste**

Não se esqueça que o ifrn-bot-teste é o nome da aplicação que você fez o deploy no heroku.

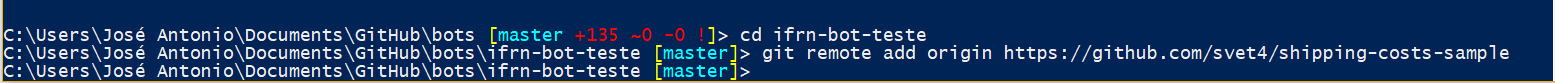


Você clonou o repositório, porém ele está vazio. Se você olhar a mensagem acima, você vai ver a mensagem “You appear to have cloned na empty respository.”

Agora vamos linkar esse repositório a um outro repositório que possui o código fonte e baixa-lo para nosso repositório, portanto, execute o comando

**cd ifrn-bot-teste** //para entrar para a pasta

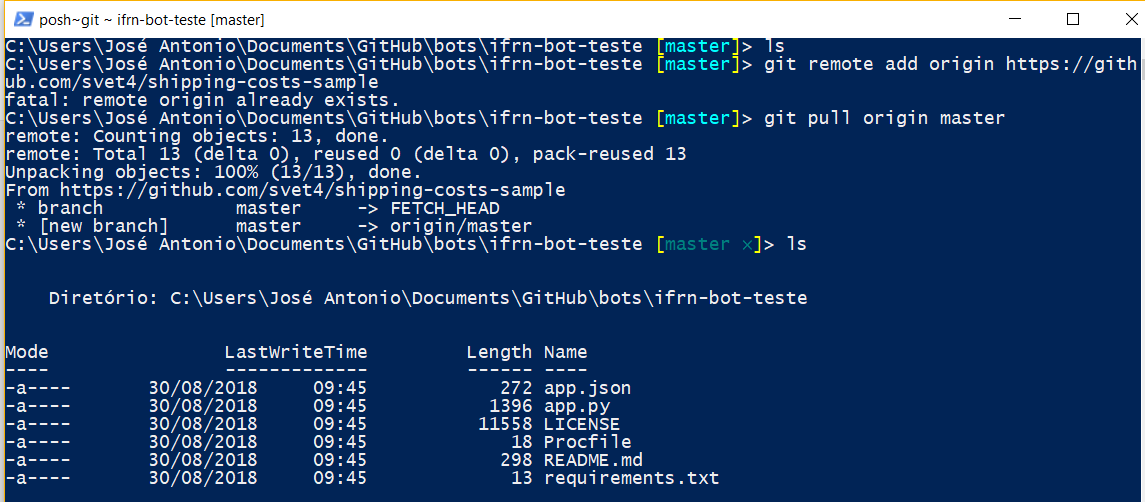
**git remote add origin** [**https://github.com/svet4/shipping-costs-sample**](https://github.com/svet4/shipping-costs-sample)



Agora execute o comando a seguir para baixar os arquivos.

**git pull origin master**

Agora temos 6 arquivos, como mostra o código a seguir



Então, nós clonamos o repositório do GitHub, baixamos o código fonte e o próximo passo será fazermos as devidas alterações e fazer novo deploy, agora via linha de comando. É o que veremos a seguir.

**Entendo o código app.py**

O código fonte que baixamos retorna o preço de um frete de uma determinada região. Nós precisamos alterar o código para retornar o preço de um curso. Então, vamos abrir o arquivo app.py em um editor de código para entender como ele funciona.

Trecho do código fonte



**Integrando o Dialogflow como o Heroku**

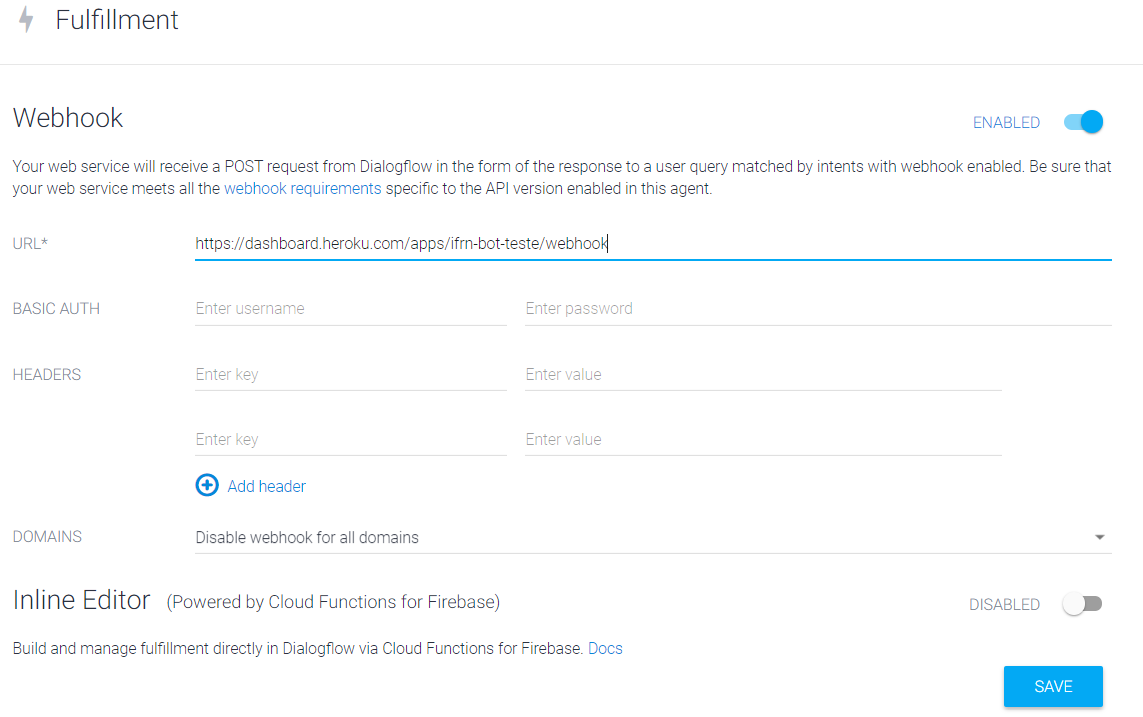
No código anterior, podemos ver que é obtido um json via webhook e no corpo desse json um objeto “action”, que no código está comparando o valor “shipping.cost”. Então, de onde vem esse valor?

Esse valor vem do objeto json que é passado para o webhook. Basta olhar o json gerado pelo Dialogflow quando fazemos uma consulta ao bot. Veja a Figura 7.



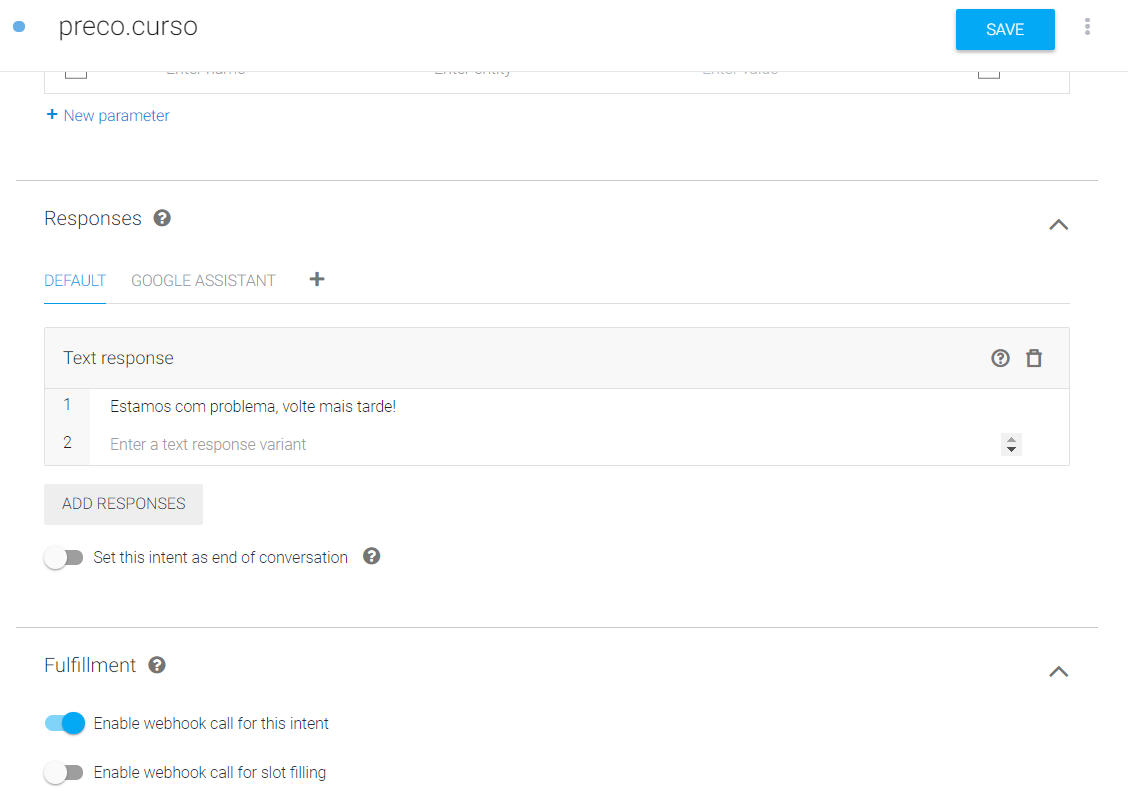
**Figura 7**

Precisamos ir até a aba de fulfilment e habilitar o WebHook e informar a URL de nossa aplicação, como mostra a Figura 8.



**Figura 8**

Ainda precisamos habilitar o webhook para a intente que queremos utilizá-lo. Nem toda intente usa webhook. Veja a Figura 9.



**Figura 9**

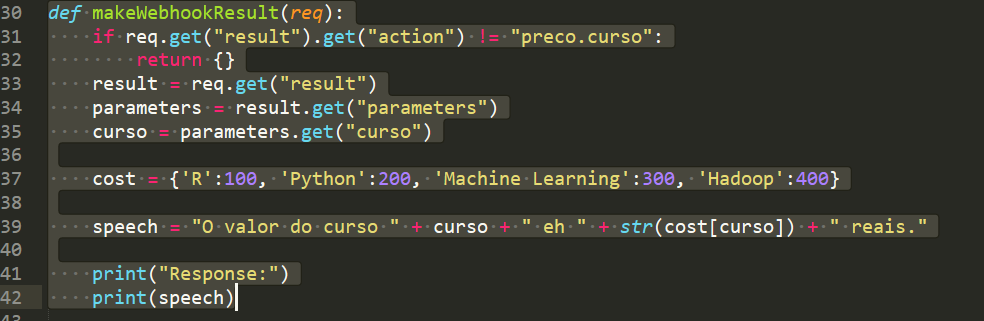
Salve e faça um teste. Observe o json a seguir



Veja que, apesar do webhook está ativo, ele não funcionou corretamente. Por que?

**Alterando o Código Fonte do BackEnd**

Vamos fazer a seguinte alteração no código fonte



O próximo passo é fazer o deploy e testar essas mudanças.

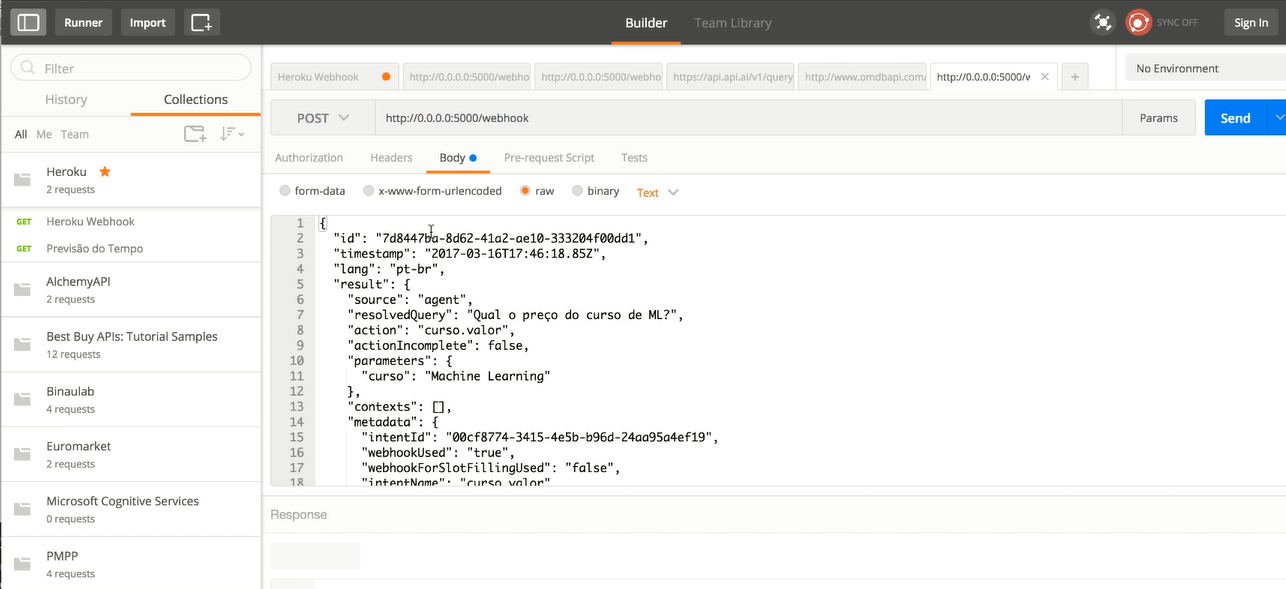
**Testando as alterações no código fonte antes de fazer o deploy**

Abra o seu editor de código com o código fonte, no meu caso, usei o Sublime text. Veja a Figura



Na opção Tools selecione Build. Se não tiver erro, o sistema exibe que a aplicação está rodando em <http://0.0.0.0:5000>

A partir desse momento, podemos disparar uma requisição POST. Vamos fazer isso usando o **Postman**.



**Figura 10**. Postman

Veja que foi selecionado o verbo POST e foi fornecida a URL: <http://127.0.0.1:5000/webhook>

E em raw foi colado o json fornecido pelo Dialogflow, quando fazemos um teste.

**Fazendo o Deploy de uma nova versão**

Na seção anterior fizemos um teste de nossa aplicação, agora estamos prontos para fazermos o deploy para o servidor. Portanto, abra um terminal e digite os comandos

Vá para a pasta onde estão seus arquivos, no meu caso,

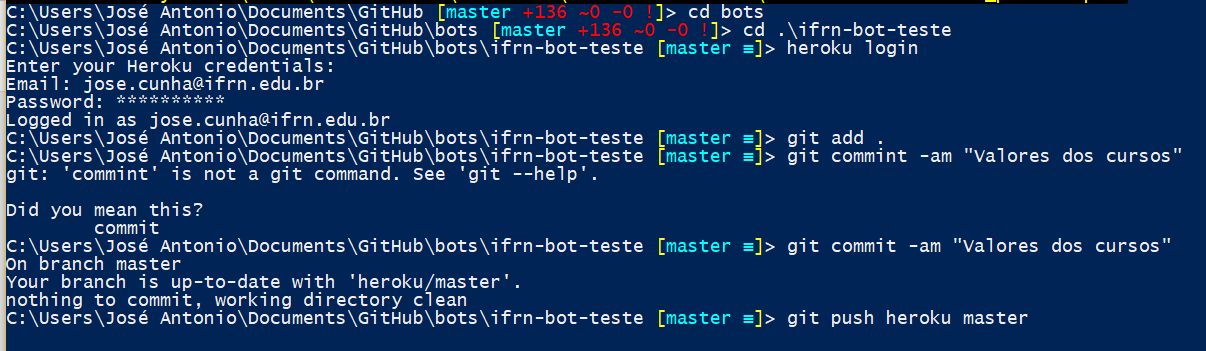
**cd bots**

**cd ifrn-bot-teste**

**git add .**

**git commit -am “Valores dos cursos”**

**git push heroku máster**



Se tudo correr bem! Você agora pode testar seu bot diretamente da ferramenta DialogFlow ou Postman ou de seu site.

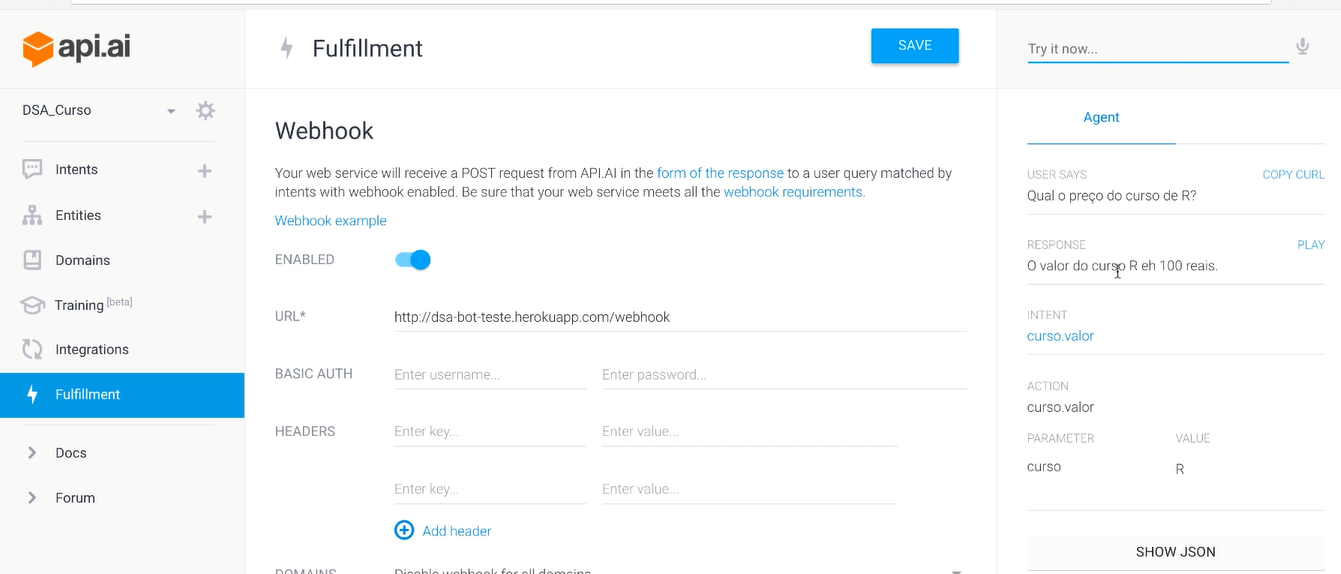


Figura 11. No Postman

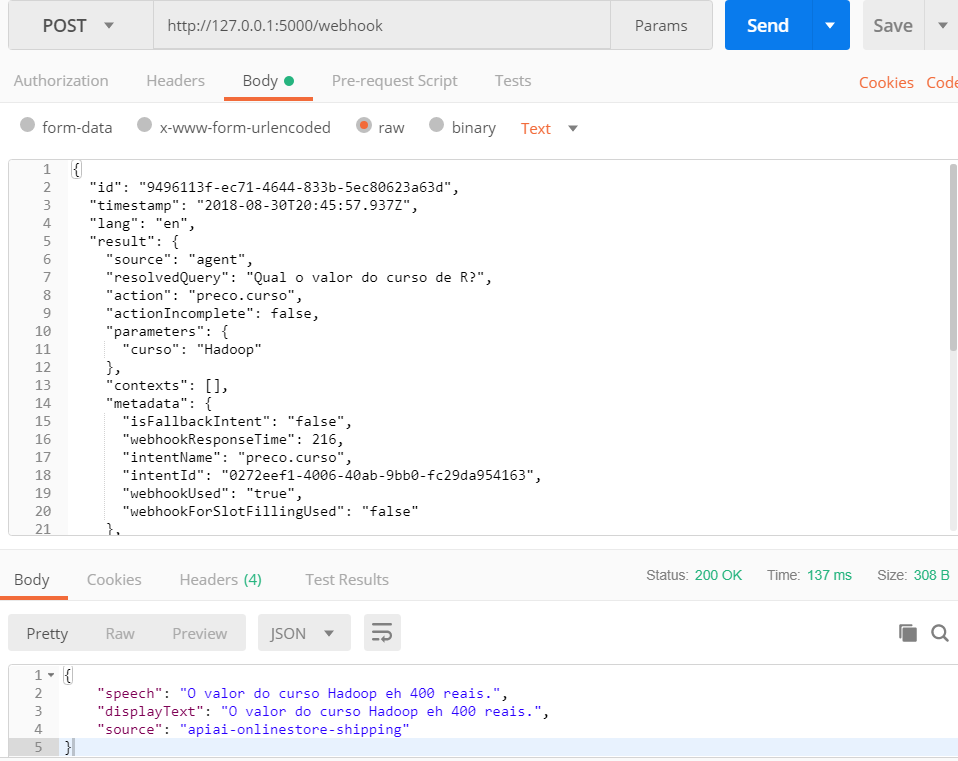


Figura 12.

**WebHook for Slot Filling**

Nesta seção, vamos usar WebHook para fazer Slot Filling. Mais o que é Slot Filling?

Imagine que o usuário pergunte, qual o valor do curso? Nós queremos que essa expressão seja mapeada para a intente preco.curso. No entanto, se você testar o bot com essa expressão “qual o valor do curso” não dará match, pois a expressão não tem o parâmetro curso, o bot fica sem saber qual é o curso.

Uma primeira tentativa seria tornar o parâmetro curso como obrigatório e definir um prompt “Qual curso? Escolha entre R, Python, Machine Learning e Hadoop:”

Essa é uma ótima prática. Só que a cada semana são lançados novos cursos e, eu não quero vir alterar o prompt acima, a cada atualização que ocorrer.

Observe que no Prompt “Qual curso? Escolha entre R, Python, Machine Learning e Hadoop:”, nós apenas estamos fornecendo informações para o usuário completar sua solicitação.

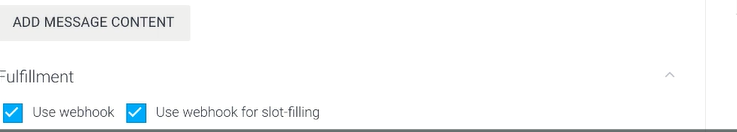
Se você testar sem usar slot filling, o json retornado fica como mostra aFigura 13.



**Figura 13. Sem WebHook Slot Filling**

Observe que a propriedade actionIncomplete está “false”. Mais, no entanto, a ação não está incompleta, basta observar que o parâmetro curso está com o valor “R”. isso aconteceu por que a opção “use Webhook slot filling” está desmarcada.

Agora, marque a opção “use Webhook slot filling” e faça novo teste. O json fica como mostrado na Figura 14.



**Figura 14**. Habilitando WebHook for slot-filling



**Figura 15. Json retornado sem uso do WebHook for slot-filling**

Agora actionIncomplete está “true”, a informação foi enviada ao servidor. No entanto, ainda temos um erro mostrado a Figura 16



**Figura 16.**

Por que o erro? Isso ocorreu por que o nosso BackEnd não está preparado para trabalhar com WebHook Slot Filling. O que vamos fazer na próxima seção.

Nessa seção vamos alterar nosso código no **BackEnd** para acessar **WebHooks Slot Filling**.

Poderíamos fazer de diversas formas, como por exemplo, poderíamos analisar se o parâmetro curso está vazio e retornar uma informação para o bot ou então, como já foi mostrado, através do parâmetro **actionIncomplete** do json enviado. A segunda opção será usada nesse momento, como mostra a Figura 17.

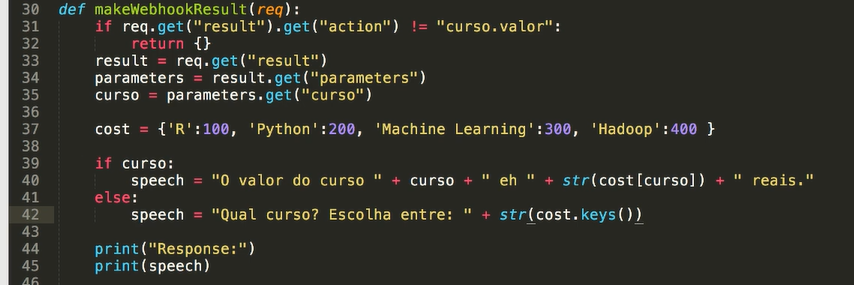


Figura 17. Usando WebHook Slot Filling no BackEnd

Salve e teste. Podemos testar antes de fazer o deploy. Sim, vá até o Postman e mude o valor da propriedade actionIncomplete para true, como mostra a Figura 18.

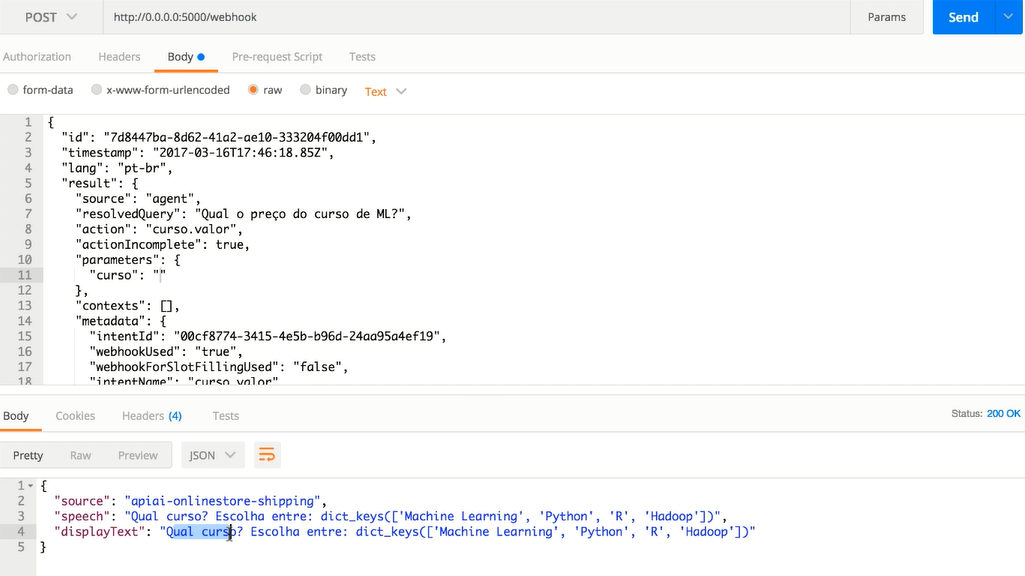


Figura 18. Rodando no Postman.

Veja que o prompt retornado foi do BackEnd.

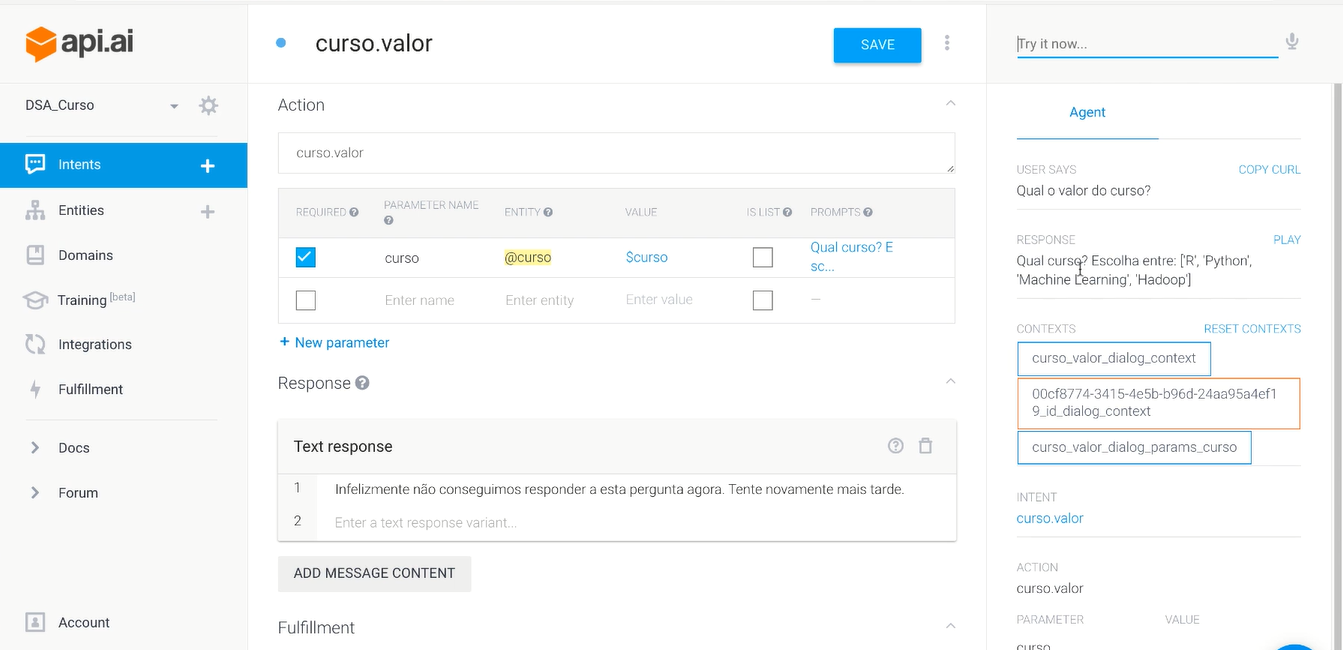
Agora vamos atualizar nosso código no servidor, fazendo um novo deploy. Abra o terminal e digite:

**git add .**

**git commit -am “Curso valor slot filling”**

**git push heroku master**

Testando no DialogFlow

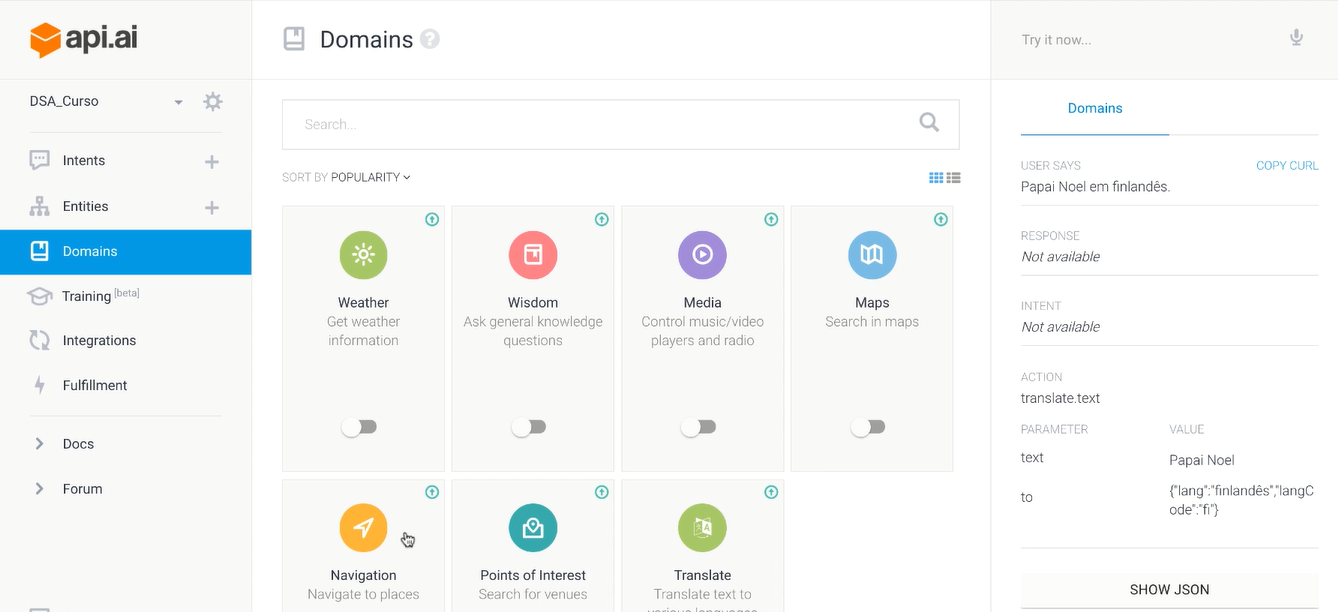


**Figura 19**. Testando as alterações no DialogFlow.

Observe que o prompt exibido para o usuário é do servidor.

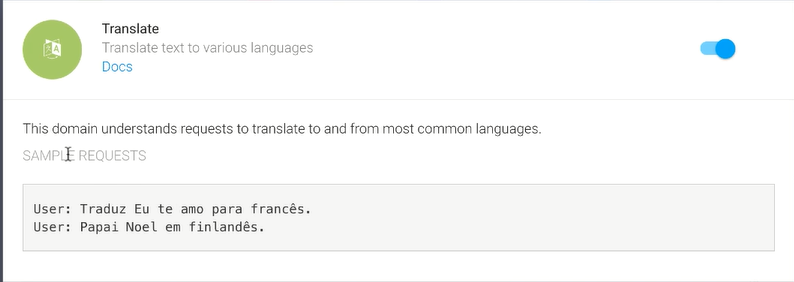
**WebHook for Domains**

Nesta seção vamos aprender usar WebHooks para domínios. Mais o que são domínios. Domínios são pacotes de conhecimentos já pré-definidos por áreas onde nós podemos ativar, e nesses domínios existem várias expressões que serão mapeadas para determinadas ações já extraindo os valores dos parâmetros. Por exemplo, se você quer construir um bot que vai auxiliar o usuário em uma navegação, você não precisa fazer todo o caminho da construção de seu bot. Nesse caso, você pode fazer tudo apenas habilitando o Domínio Navegation, como mostra a Figura 20.



**Figura 20.** Biblioteca de Domínios do DialogFlow

Para testar, vamos habilitar o Domínio **Translate** e clique em **VIEW DETAILS**, será mostrado a janela a seguir, onde são exibidas algumas expressões como mostra a Figura 21.



**Figura 21**.

Feche essa janela, e faça um teste informando ao bot a frase “Papai Noel em finlandês”.

O resultado deve ser como o da Figura 22.

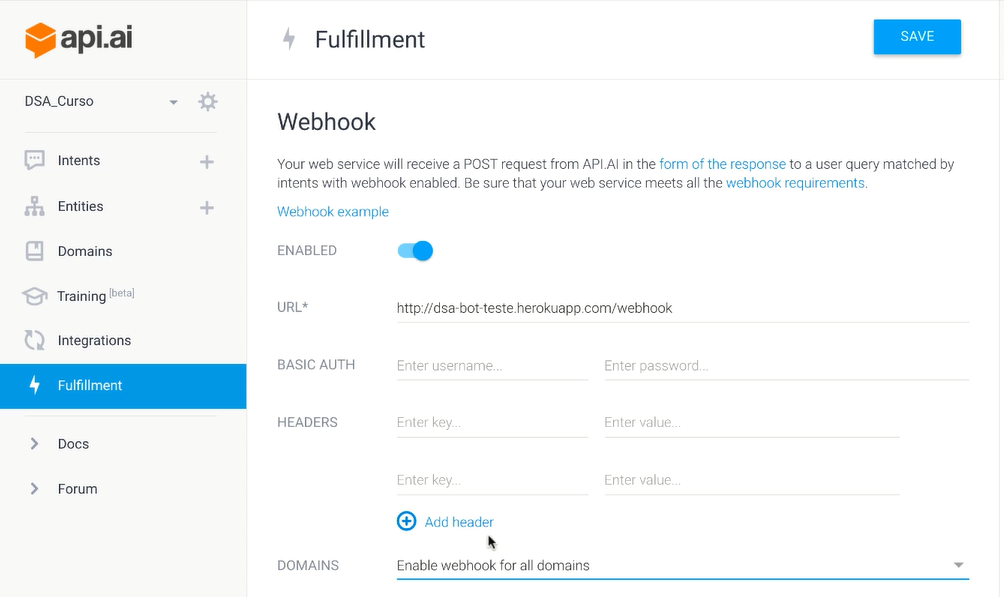


Figura 22.

Na janela do DialogFlow teste, podemos vê a propriedade **text** com a frase que informamos e a propriedade **to** com a tradução da frase informada.

Agora vamos passar essas informações para o BackEnd, como fazemos?

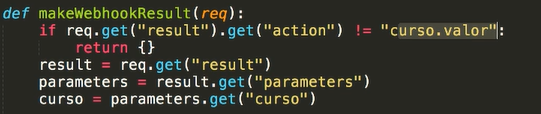
Vamos até Fulfilment e vamos habilitar o WebHook para todos os domínios, como mostra a Figura 23.



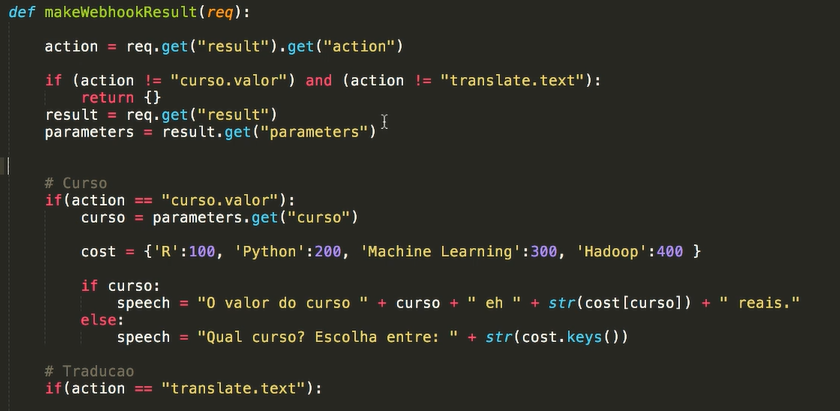
**Figura 23. Habilitando WebHook para todos os Domínios**

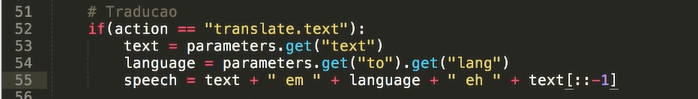
Salve.

Só que agora nosso BackEnd tem que processar uma ação chamada “**translate.text**”., que espera dois parâmetros “text” e “to”. Não entanto, até o momento, nosso backEnd só processa o parâmetro curso.valor.



Vamos alterar nosso código no BackEnd para utilizar Domínios. Como isso, vamos habilitar nosso bot para processar ações do tipo translate.text.





Aqui estou apenas imprimindo o texto na ordem inversa. Mas claro, que você poderia usar uma **API**, por exemplo, do Google Tradutor e fazer a tradução correta.

Salve e teste via Postman antes de fazer o deploy para o servidor.

Vá até o DialogFlow para obter um json de tradução copie e cole no body do Postman e execute.

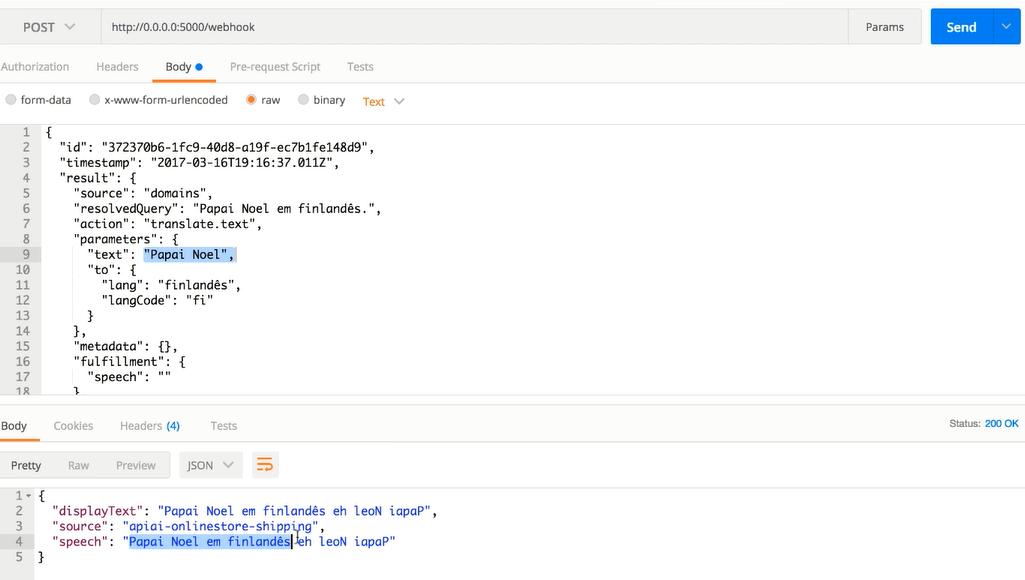
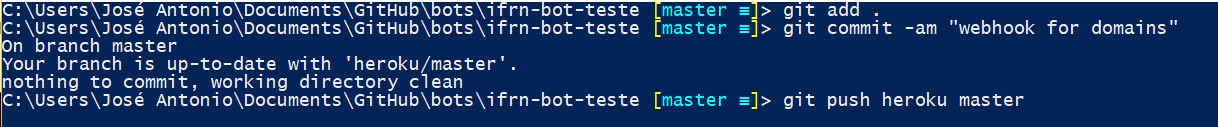


Figura 24.

Agora vamos fazer o deploy para o servidor



Faça alguns testes no DialogFlow

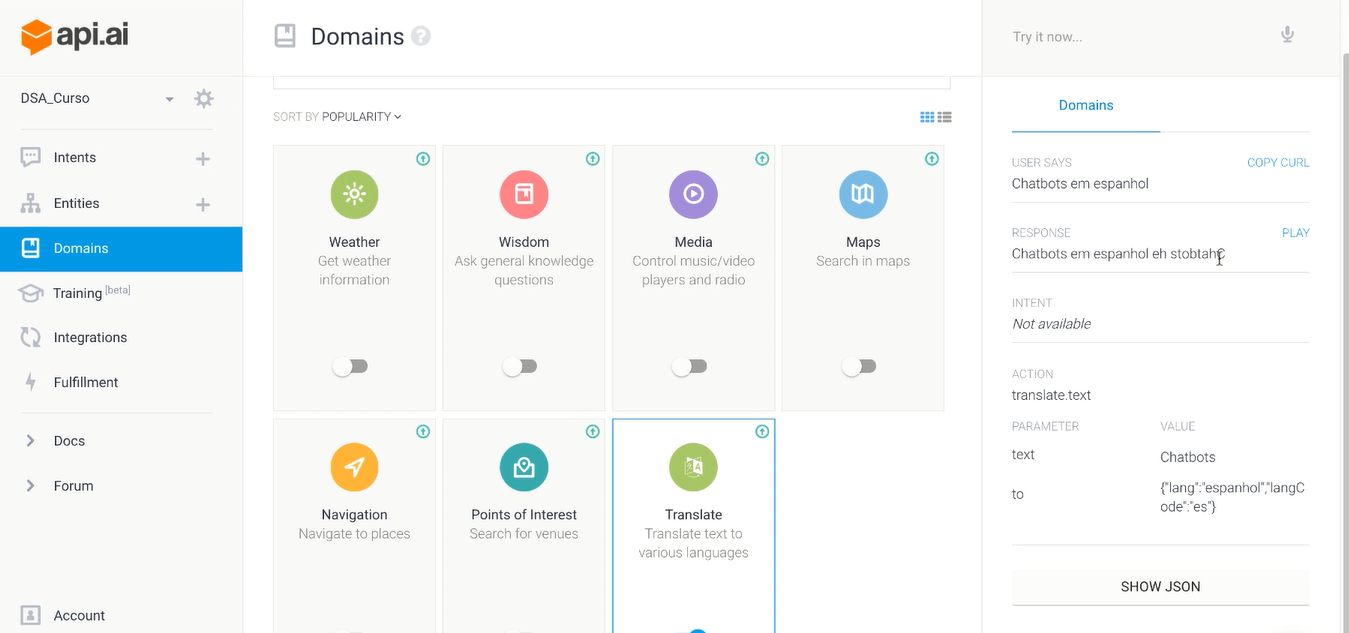


Figura 25.

**Firebase**

Vamos voltar ao problema de valores de um curso. Nós temos um problema, toda vez que o valor de um curso alterar, nós temos que alterar nosso código fonte e fazer o deploy. E isso, não é bom. Mas você devia ter imaginado que nós poderíamos utilizar um banco de dados para armazenar esses valores. Pode ser um banco de dados relacional ou não. Para o nosso estudo de caso de exemplo, vamos utilizar o Firebase.

No Firebase temos diversas formas de armazenamento de dados, porém nós utilizaremos para esse exemplo o **RealTime Database**, como mostra a Figura 26.

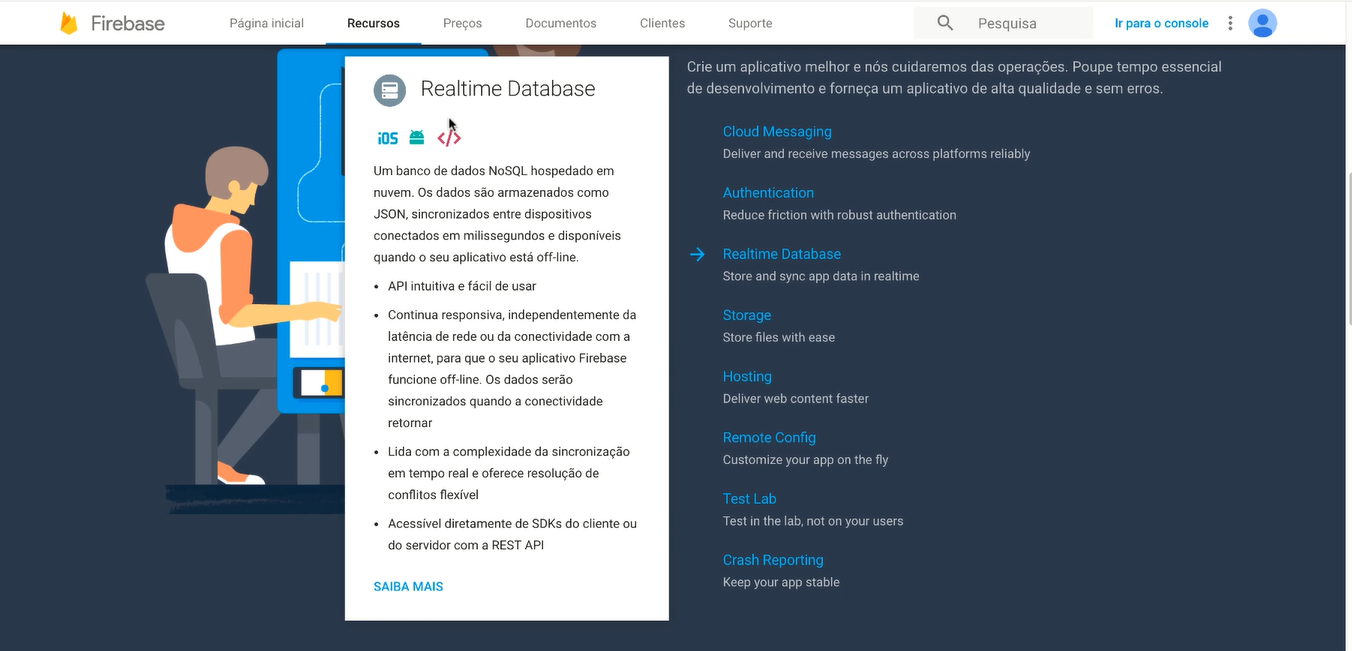
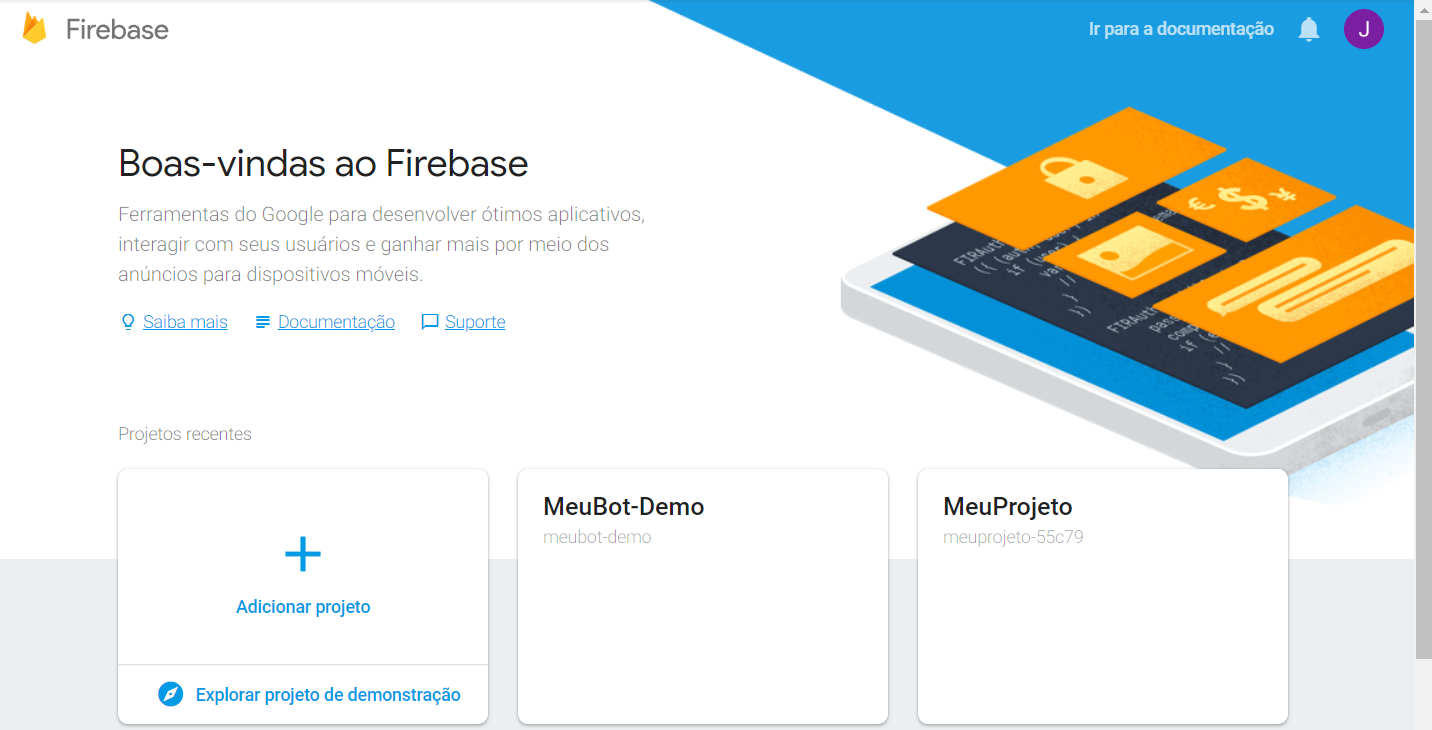


Figura 26.

Então, vamos criar um projeto no Firebase. Na Figura 27, clique no link Adicionar projeto.



**Figura 27. Criando novo projeto no Firebase**.

Na Figura 28, informe o nome do projeto e clique no botão Criar projeto.

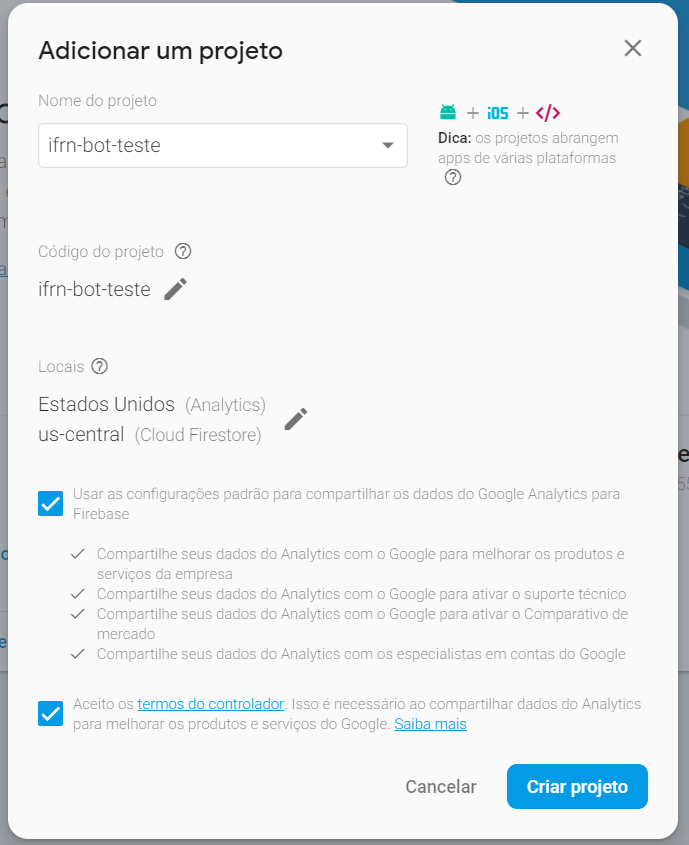


Figura 28.

Após criar o projeto, clique na opção do menu Database, como mostra a figura 29.

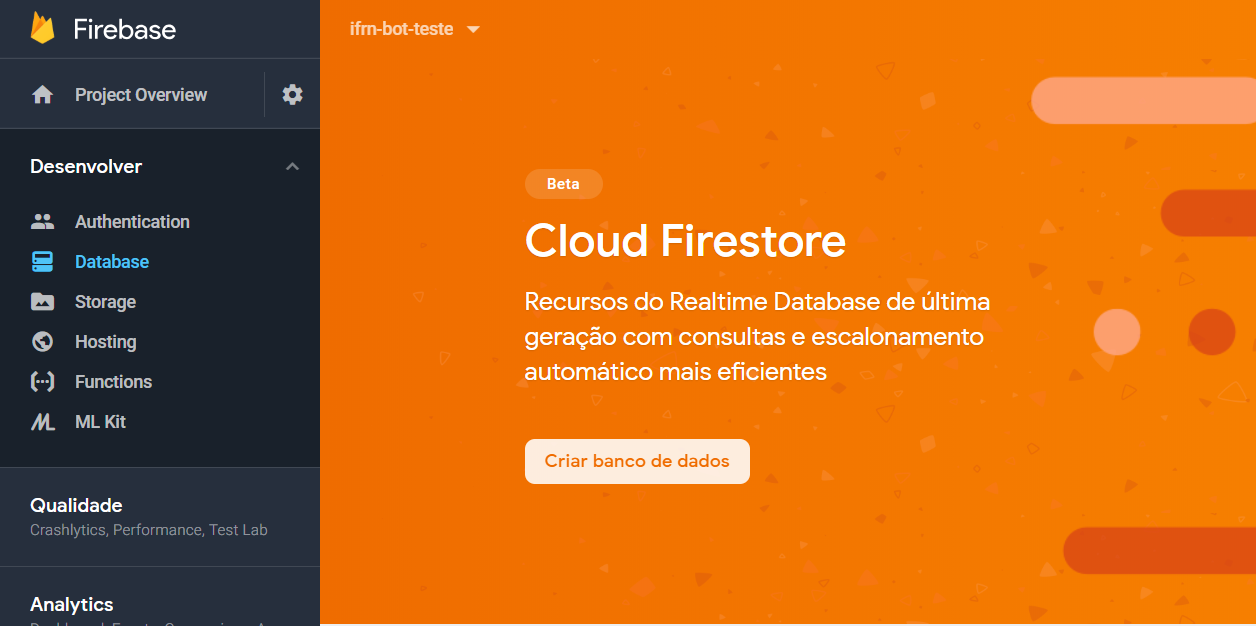


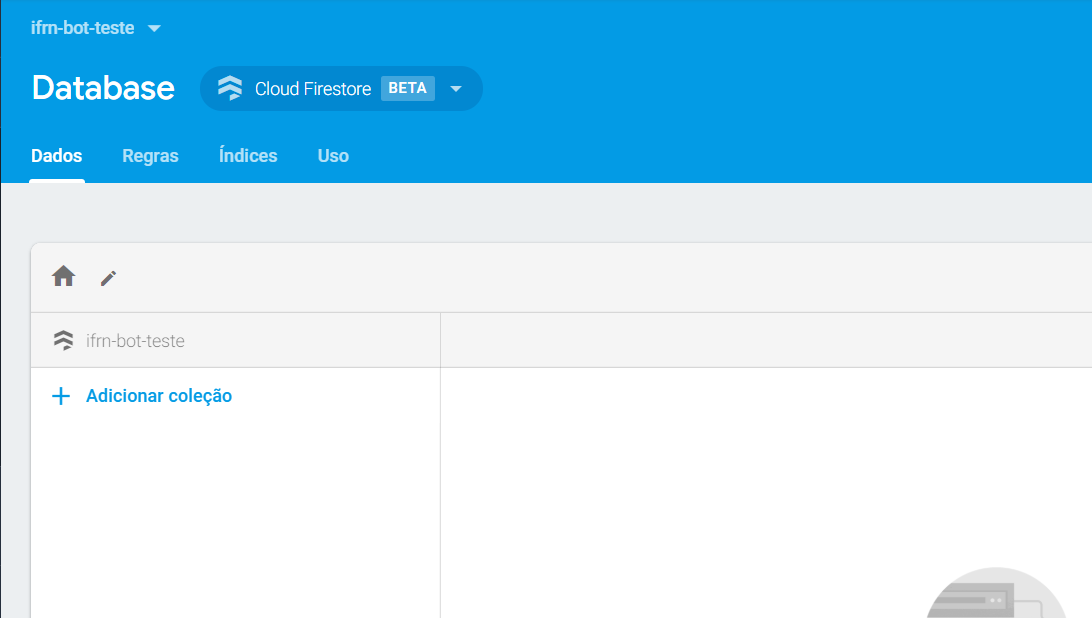
Figura 29.

E clique no botão criar banco de dados e na Figura 30 clique em Ativar.



**Figura 30**.

A figura 31, mostra a tela onde adicionamos novas coleções ao banco de dados.



**Figura 31**. Adicionando coleções ao banco de dados.

Basta agora adicionar as coleções, uma vez que essa é uma estrutura json. Então, clicando no link **Adicionar coleção**, você é direcionado para a tela da Figura 32.

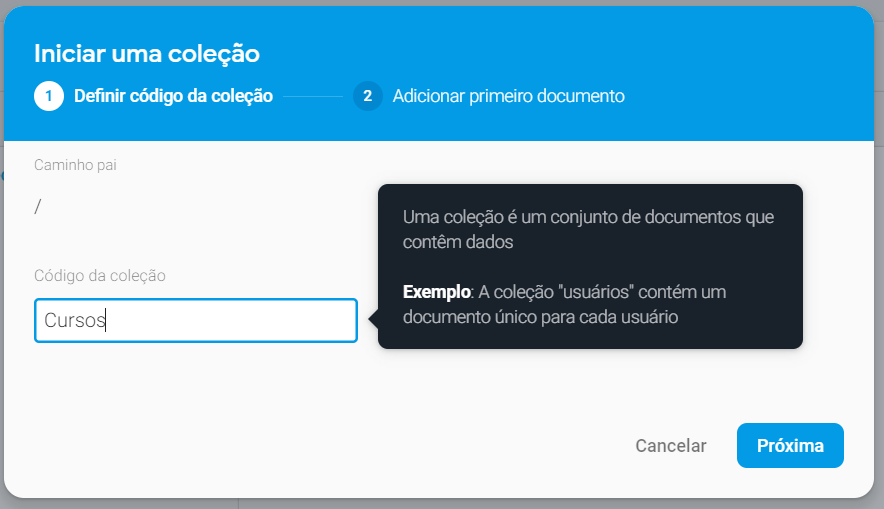


Figura 32.

Em seguida digite os dados dos cursos, como mostra a Figura 33.

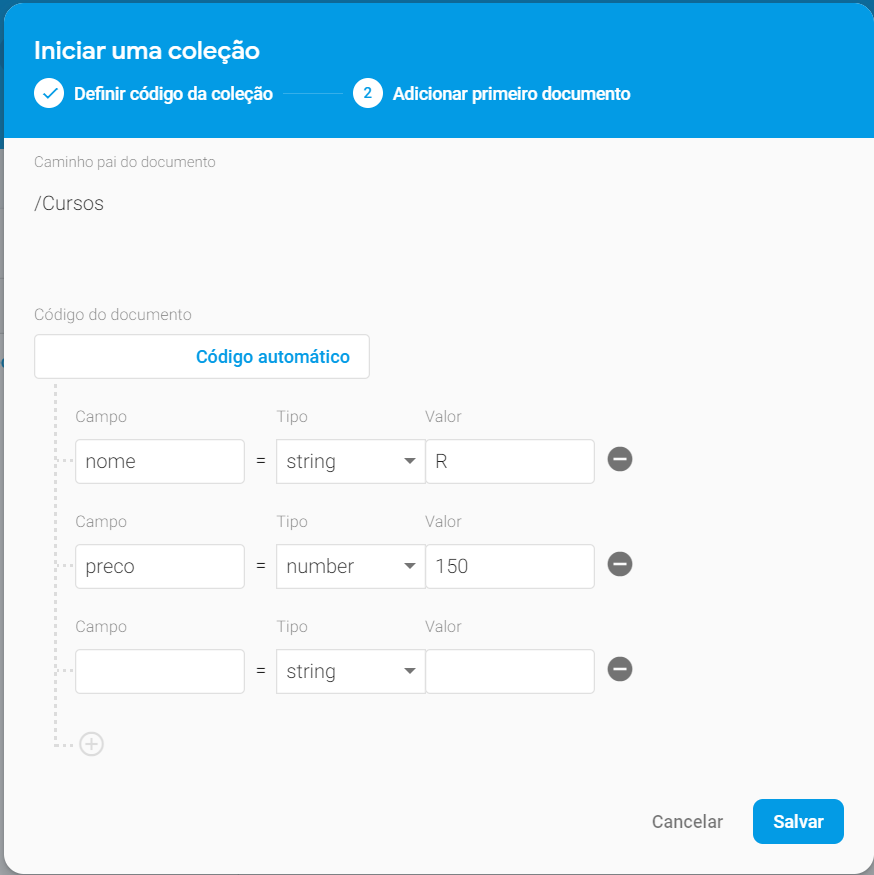


Figura 33.

Nosso banco de dados fica como mostra a Figura 34.

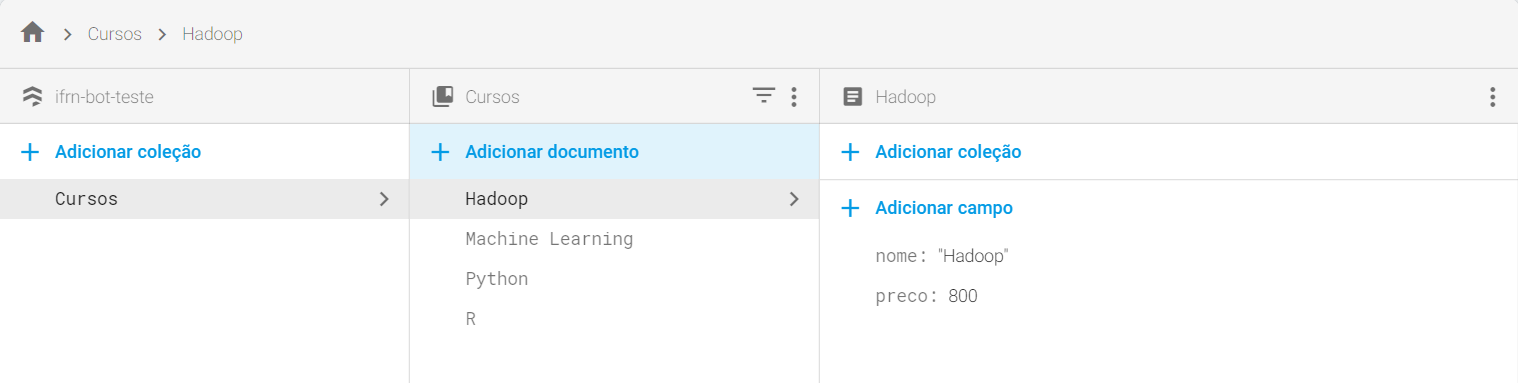
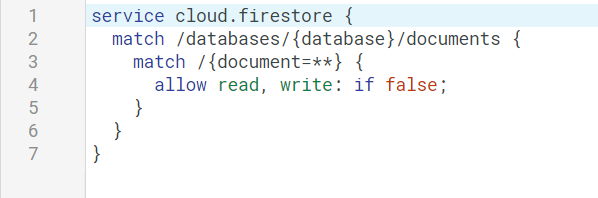


Figura 34.

**Permissões em Banco de Dados Firebase**

Vamos manipular as permissões ou regras no banco de dados Fierbase. A regra mais geral de todas é mostrada a seguir



Essa regra está negando acesso para usuários não autenticados. Faça uma simulação, como mostra a Figura 35.



Figura 35.

Veja que o acesso foi negado

Agora mude o acesso para permitir acesso a todo usuário logado e simule novamente. Veja a Figura 36.



Figura 36.

Publique as modificações e simule ainda sem autenticar o usuário. Veja a Figura 37.

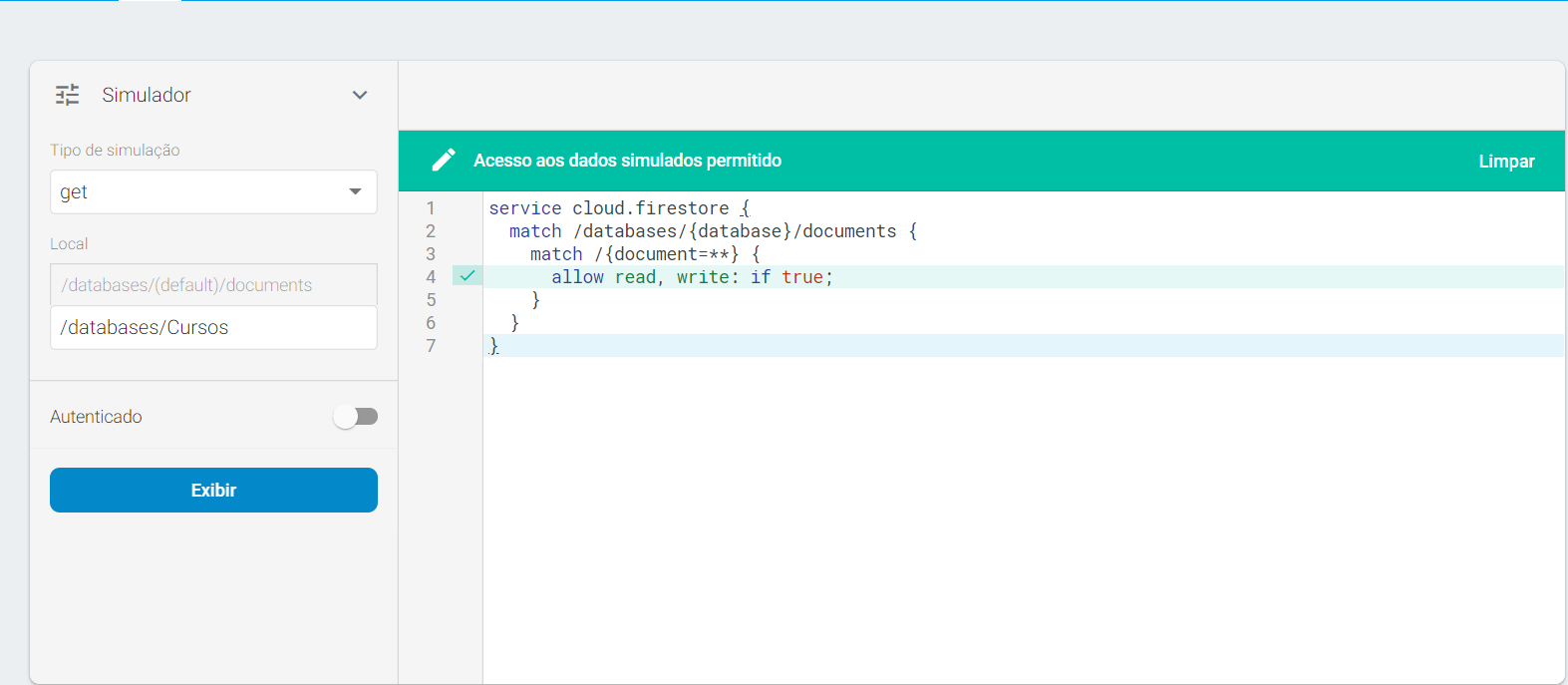


Figura 37.

Como podemos testar isso? Podemos fazer o teste em uma janela anônima. Pegue o endereço URL do banco de dados Firebase em Configurações, como mostra a Figura 38.

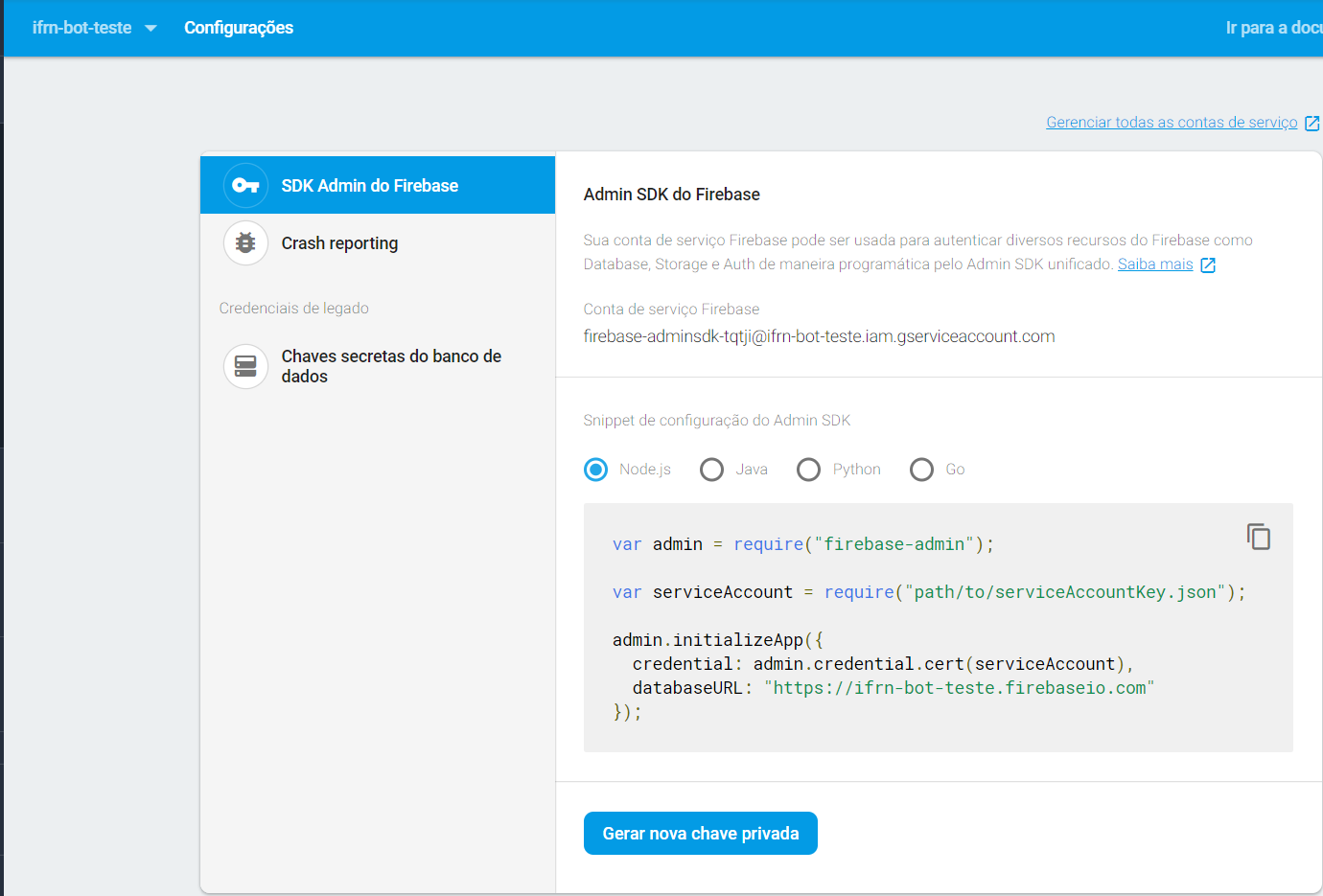


Figura 38.