

Plano de aula 07 – WebHook (Python)

Nesta aula falaremos sobre WebHooks com Python.

Passo 1

Criando uma conta no Heroku

Para criar uma conta no Heroku acesse o site (<https://www.heroku.com>). Não esqueça de selecionar a linguagem primária como sendo Python.

Passo 2

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**.

Clonando o repositório do GitHub.

Acesse seu aplicativo (O que você acabou de criar) no dashboard do aplicativo <https://dashboard.heroku.com/apps/ifrn-bot-teste/deploy/heroku-git>, clique na opção **Deploy** e instale o Heroku CLI. Através da opção **install Heroku CLI**.

Para fazer 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

Crie uma pasta para o bot e, nessa pasta execute o comando

```
mkdir Bots
```

```
cd Bots
```

```
$ heroku git:clone -a nome-de-seu-bot
```

Não se esqueça que o “**nome-de-seu-bot**” é o nome da aplicação que você fez o deploy no heroku.

Você clonou o repositório, porém ele está vazio.

Agora vamos baixar o código fonte para esse diretório, portanto, execute o comando

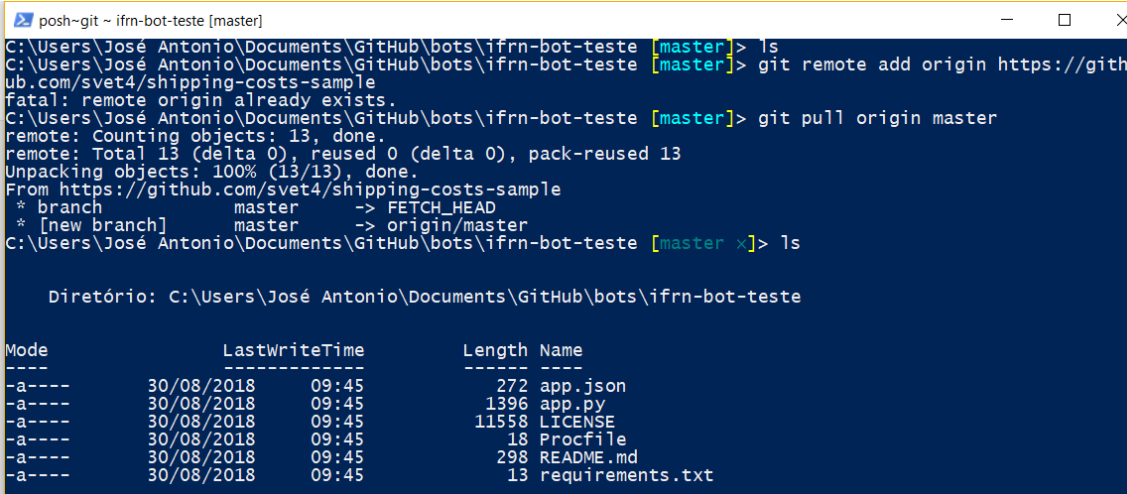
cd ifrn-bot-teste //para entrar para a pasta

git remote add origin <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 a Figura 1.



```
posh-git - ifrn-bot-teste [master]
C:\Users\José Antonio\Documents\GitHub\bots\ifrn-bot-teste [master]> ls
C:\Users\José Antonio\Documents\GitHub\bots\ifrn-bot-teste [master]> git remote add origin https://github.com/svet4/shipping-costs-sample
fatal: remote origin already exists.
C:\Users\José Antonio\Documents\GitHub\bots\ifrn-bot-teste [master]> git pull origin master
remote: Counting objects: 13, done.
remote: Total 13 (delta 0), reused 0 (delta 0), pack-reused 13
Unpacking objects: 100% (13/13), done.
From https://github.com/svet4/shipping-costs-sample
* branch            master       -> FETCH_HEAD
* [new branch]      master       -> origin/master
C:\Users\José Antonio\Documents\GitHub\bots\ifrn-bot-teste [master x]> ls

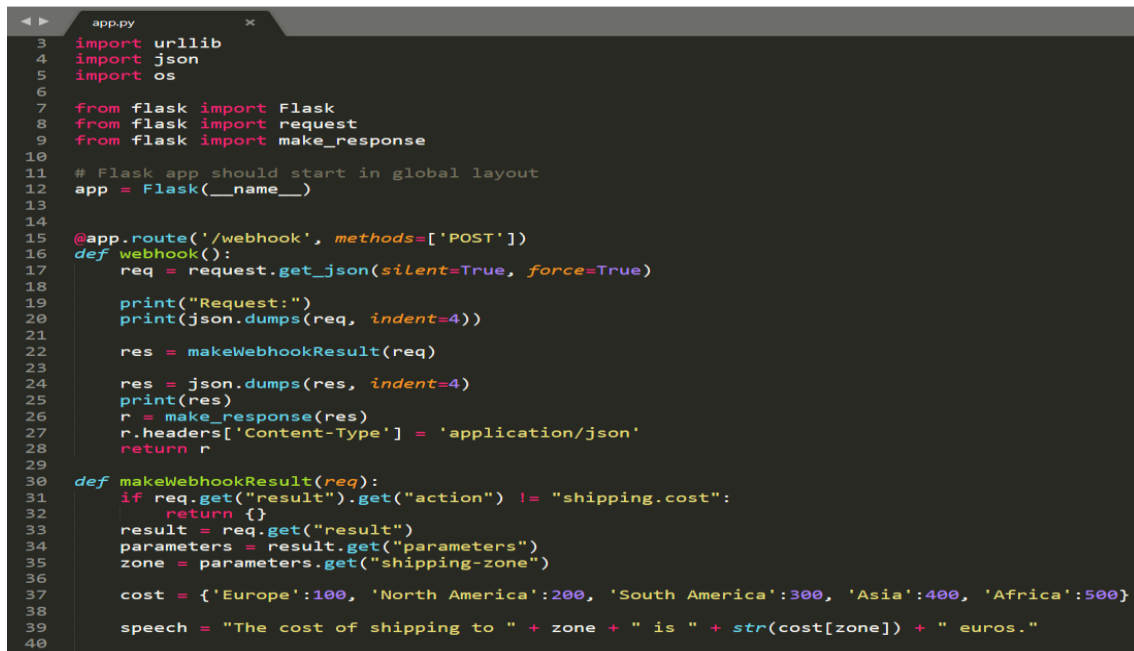
Diretório: C:\Users\José Antonio\Documents\GitHub\bots\ifrn-bot-teste

Mode                LastWriteTime         Length Name
----                -
-a----             30/08/2018     09:45           272 app.json
-a----             30/08/2018     09:45          1396 app.py
-a----             30/08/2018     09:45         11558 LICENSE
-a----             30/08/2018     09:45           18 Procfile
-a----             30/08/2018     09:45          298 README.md
-a----             30/08/2018     09:45           13 requirements.txt
```

Figura 1. Arquivos baixados.

Entendo o código app.py

Abra o arquivo app.py e dê uma olhada no código para entendê-lo. Observe que o mesmo trata de preço de um frete de uma determinada região. Como mostra a Figura 2.



```

1  import urllib
2  import json
3  import os
4
5  from flask import Flask
6  from flask import request
7  from flask import make_response
8
9  # Flask app should start in global layout
10 app = Flask(__name__)
11
12 @app.route('/webhook', methods=['POST'])
13 def webhook():
14     req = request.get_json(silent=True, force=True)
15
16     print("Request:")
17     print(json.dumps(req, indent=4))
18
19     res = makeWebhookResult(req)
20
21     res = json.dumps(res, indent=4)
22     print(res)
23     r = make_response(res)
24     r.headers['Content-Type'] = 'application/json'
25     return r
26
27 def makeWebhookResult(req):
28     if req.get("result").get("action") != "shipping.cost":
29         return {}
30     result = req.get("result")
31     parameters = result.get("parameters")
32     zone = parameters.get("shipping-zone")
33
34     cost = {'Europe':100, 'North America':200, 'South America':300, 'Asia':400, 'Africa':500}
35
36     speech = "The cost of shipping to " + zone + " is " + str(cost[zone]) + " euros."
37
38
39
40

```

Figura 2. Arquivo app.py

Integrando o Dialogflow como o Heroku

No código anterior, podemos ver que está sendo uma verificação se a “action”, é “shipping.cost”. Então, precisamos alterar essa ação para “**preco.curso**”.

Criando uma Intenção para utilização de WebHook

Inicialmente crie uma entidade **curso** para representar os cursos oferecidos.

- Python, Machine Learning → ML, R e Hadoop
- Crie uma intente chamada **curso.valor**, e entre com as seguintes frases para treinamento:
 - 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 faça alguns testes.


Precisamos ir até a aba de fulfilment e habilitar o WebHook e informar a URL de nossa aplicação, como mostra a Figura 3.

Fulfillment

Webhook

ENABLED 

Your web service will receive a POST request from Dialogflow in the form of the response to a user query matched by intents with webhook enabled. Be sure that your web service meets all the [webhook requirements](#) specific to the API version enabled in this agent.

URL*	<input type="text" value="https://dashboard.heroku.com/apps/ifrm-bot-teste/webhook"/>	
BASIC AUTH	<input type="text" value="Enter username"/>	<input type="text" value="Enter password"/>
HEADERS	<input type="text" value="Enter key"/>	<input type="text" value="Enter value"/>
	<input type="text" value="Enter key"/>	<input type="text" value="Enter value"/>
	 Add header	
DOMAINS	<input type="text" value="Disable webhook for all domains"/>	

Inline Editor (Powered by Cloud Functions for Firebase)


DISABLED 

Build and manage fulfillment directly in Dialogflow via Cloud Functions for Firebase. [Docs](#)

SAVE


Figura 3. Habilitando WebHook

Ainda precisamos habilitar o webhook para a intente que queremos utilizá-lo. Nem toda intente usa webhook. Veja a Figura 4.

 preco.curso

SAVE

+ New parameter

Responses 

DEFAULT

GOOGLE ASSISTANT

+

Text response



1


Estamos com problema, volte mais tarde!


2

Enter a text response variant

ADD RESPONSES

 Set this intent as end of conversation 

Fulfillment 

 Enable webhook call for this intent


 Enable webhook call for slot filling

Figura 4. Habilitando a intenção para usar WebHook.

Salve e faça um teste. Observe que no json a seguir temos a **action = preco.curso**



Diagnostic info

RAW API RESPONSE FULFILLMENT REQUEST FULFILLMENT RESPONSE FULFILLMENT STATUS

```
1 {
2   "id": "be0d86d5-f2a1-4a39-81a0-dc6732c28ca0",
3   "timestamp": "2018-09-06T17:56:38.917Z",
4   "lang": "en",
5   "result": {
6     "source": "agent",
7     "resolvedQuery": "Qual o valor do curso de Python?",
8     "action": "preco.curso",
9     "actionIncomplete": false,
10    "parameters": {
11      "curso": "Python"
12    },
13    "contexts": [],
14    "metadata": {
15      "isFallbackIntent": "false",
16      "webhookResponseTime": 239,
17      "intentName": "preco.curso",
18      "intentId": "0272eef1-4006-40ab-9bb0-fc29da954163",
19      "webhookUsed": "true",
20      "webhookForSlotFillingUsed": "false"
21    },
22    "fulfillment": {
23      "speech": "O valor do curso Python eh 200 reais.",
24      "source": "apiai-onlinestore-shipping",
25      "displayText": "O valor do curso Python eh 200 reais.",
26      "messages": [
27        {
28          "type": 0,
29          "speech": "O valor do curso Python eh 200 reais."
30        }
31      ]
32    },
33    "score": 1
34  }
35 }
```

CLOSE COPY FULFILLMENT REQUEST AS CURL COPY RAW RESPONSE

Alterando o Código Fonte do BackEnd

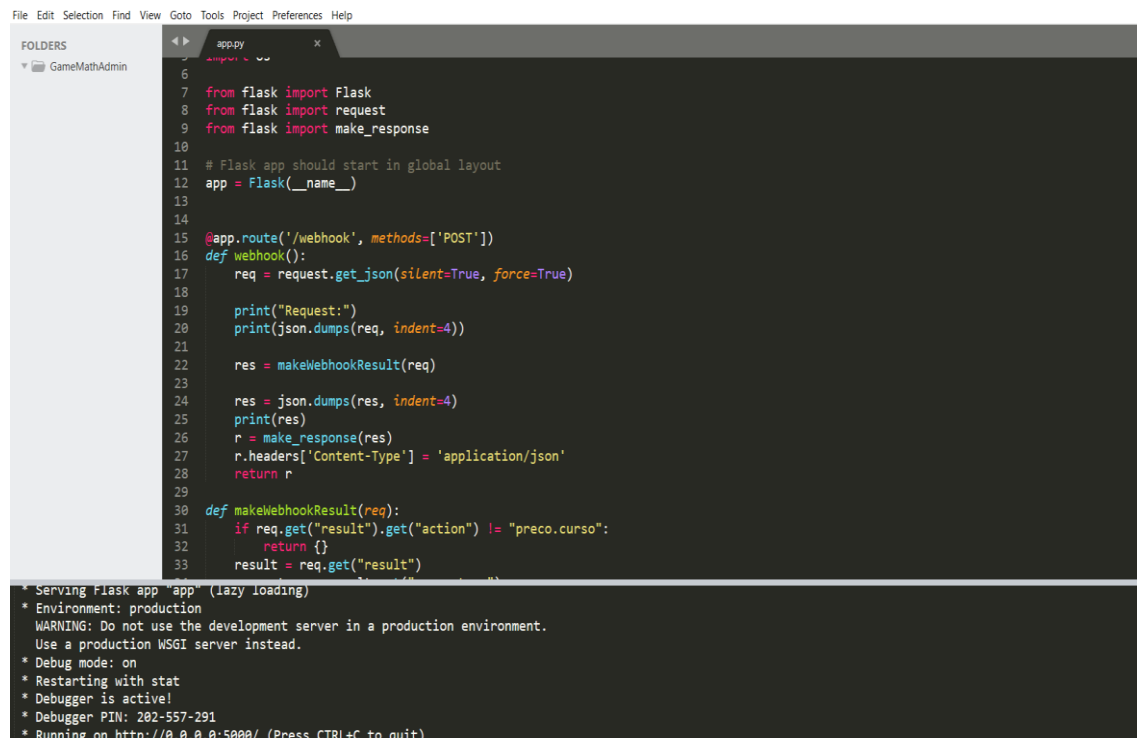
Vamos fazer as seguintes alterações no código fonte, como mostra a Figura 5.

```
30 def makeWebhookResult(req):
31     if req.get("result").get("action") != "preco.curso":
32         return {}
33     result = req.get("result")
34     parameters = result.get("parameters")
35     curso = parameters.get("curso")
36
37     cost = {'R':100, 'Python':200, 'Machine Learning':300, 'Hadoop':400}
38
39     speech = "O valor do curso " + curso + " eh " + str(cost[curso]) + " reais."
40
41     print("Response:")
42     print(speech)
```

Figura 5. Alterando o código fonte para trabalhar com preço de cursos.

Testando as alterações no código fonte antes de fazer o deploy

Se você estiver usando o Sublime text. Você pode compilar a aplicação e testar no Postman, por exemplo. Veja a Figura 6.



```
File Edit Selection Find View Goto Tools Project Preferences Help
FOLDERS
GameMathAdmin
app.py
6
7 from flask import Flask
8 from flask import request
9 from flask import make_response
10
11 # Flask app should start in global layout
12 app = Flask(__name__)
13
14
15 @app.route('/webhook', methods=['POST'])
16 def webhook():
17     req = request.get_json(silent=True, force=True)
18
19     print("Request:")
20     print(json.dumps(req, indent=4))
21
22     res = makeWebhookResult(req)
23
24     res = json.dumps(res, indent=4)
25     print(res)
26     r = make_response(res)
27     r.headers['Content-Type'] = 'application/json'
28     return r
29
30 def makeWebhookResult(req):
31     if req.get("result").get("action") != "preco.curso":
32         return {}
33     result = req.get("result")
34
35 * Serving Flask app "app" (lazy loading)
36 * Environment: production
37   WARNING: Do not use the development server in a production environment.
38   Use a production WSGI server instead.
39 * Debug mode: on
40 * Restarting with stat
41 * Debugger is active!
42 * Debugger PIN: 282-557-291
43 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
```

Figura 6. Compilando a aplicação app.py no Sublime Text.

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**. No Postman, selecione o verbo Post e informe a URL <http://127.0.0.1:5000/webhook> e no Body cole o json copiado do DialogFlow. Veja a Figura 7.

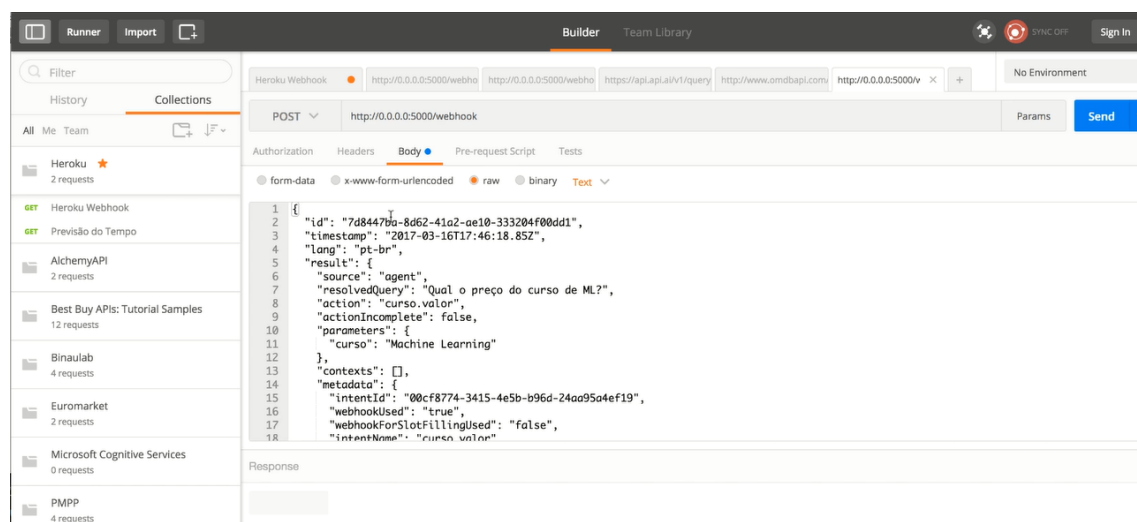


Figura 7. Rodando a aplicação no Postman.

Fazendo o Deploy de uma nova versão

Fazendo o deploy para o servidor.

```
cd bots
```

```
cd ifrn-bot-teste
```

```
git add .
```

```
git commit -am "Valores dos cursos"
```

```
git push heroku master
```

```
C:\Users\José Antonio\Documents\GitHub [master +136 ~0 -0 !]> cd bots
C:\Users\José Antonio\Documents\GitHub\bots [master +136 ~0 -0 !]> cd .\ifrn-bot-teste
C:\Users\José Antonio\Documents\GitHub\bots\ifrn-bot-teste [master ==]> heroku login
Enter your Heroku credentials:
Email: jose.cunha@ifrn.edu.br
Password: *****
Logged in as jose.cunha@ifrn.edu.br
C:\Users\José Antonio\Documents\GitHub\bots\ifrn-bot-teste [master ==]> git add .
C:\Users\José Antonio\Documents\GitHub\bots\ifrn-bot-teste [master ==]> git commint -am "Valores dos cursos"
git: 'commint' is not a git command. See 'git --help'.

Did you mean this?
    commit
C:\Users\José Antonio\Documents\GitHub\bots\ifrn-bot-teste [master ==]> git commit -am "Valores dos cursos"
On branch master
Your branch is up-to-date with 'heroku/master'.
nothing to commit, working directory clean
C:\Users\José Antonio\Documents\GitHub\bots\ifrn-bot-teste [master ==]> git push heroku master
```

Figura 8. Fazendo o deploy da aplicação app.py via terminal.

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

The screenshot shows the DialogFlow Fulfillment interface. On the left, a sidebar lists navigation options: Intents, Entities, Domains, Training (beta), Integrations, Fulfillment (selected), Docs, and Forum. The main area is titled 'Fulfillment' and contains a 'Webhook' section. The 'Webhook' section has a toggle switch set to 'ENABLED'. Below this, the 'URL*' field is populated with 'http://dsa-bot-teste.herokuapp.com/webhook'. The 'BASIC AUTH' section has fields for 'Enter username...' and 'Enter password...'. The 'HEADERS' section has fields for 'Enter key...' and 'Enter value...'. At the bottom, there is a '+ Add header' button. On the right side of the interface, there is a 'Try it now...' button and a section titled 'Agent' which shows a conversation flow: 'USER SAYS: Qual o preço do curso de R?' followed by 'RESPONSE: O valor do curso R eh 100 reais.' and 'INTENT: curso.valor'. Below this, an 'ACTION' table shows 'curso.valor' with a 'PARAMETER' of 'curso' and a 'VALUE' of 'R'. At the bottom right, there is a 'SHOW JSON' button.

Figura 9. Testando o bot no DialogFlow.

WebHook for Slot Filling

Suponha que o usuário não informe o curso. Por exemplo, se o mesmo fizer a seguinte pergunta “qual o valor do 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!”

Agora, marque a opção “use Webhook slot filling”, como mostra a Figura 10.

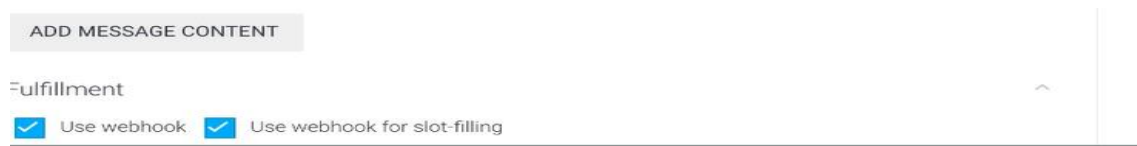


Figura 10. Habilitando WebHook for slot-filling

Habilitando WebHooks Slot Filling no BackEnd

Uma opção é analisar se o parâmetro **curso** está vazio e retornar uma informação para o bot.

```
30 def makeWebhookResult(req):
31     if req.get("result").get("action") != "curso.valor":
32         return {}
33     result = req.get("result")
34     parameters = result.get("parameters")
35     curso = parameters.get("curso")
36
37     cost = {'R':100, 'Python':200, 'Machine Learning':300, 'Hadoop':400 }
38
39     if curso:
40         speech = "O valor do curso " + curso + " eh " + str(cost[curso]) + " reais."
41     else:
42         speech = "Qual curso? Escolha entre: " + str(cost.keys())
43
44     print("Response:")
45     print(speech)
```

Figura 11. 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 12.

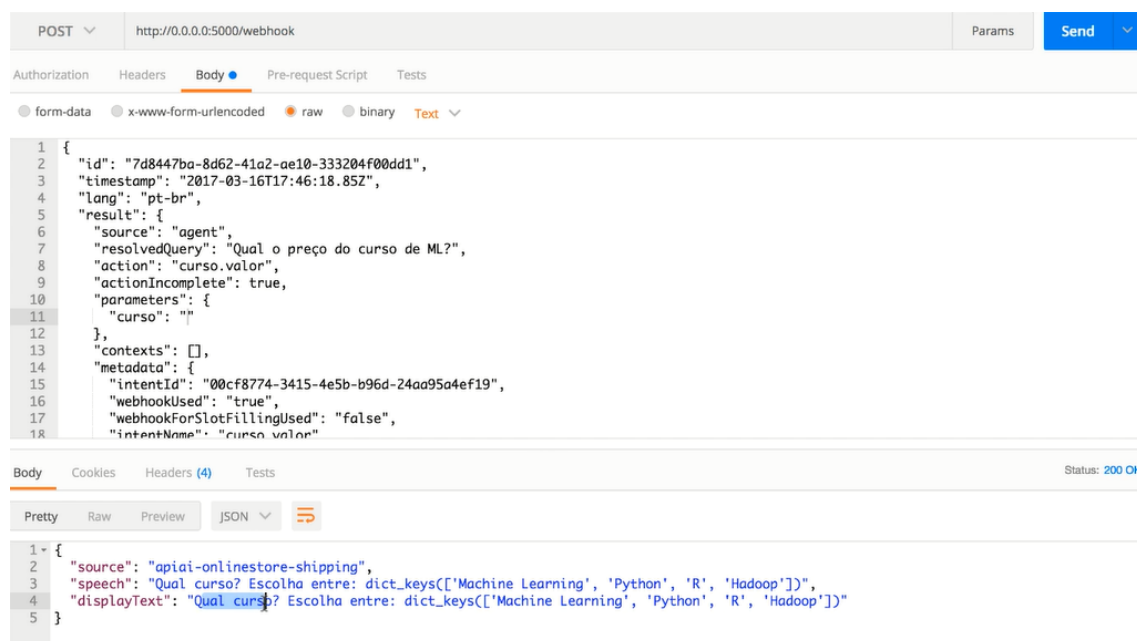


Figura 12. Rodando no Postman.

Veja que o prompt retornado foi do BackEnd.

Fazendo o deploy para atualizar o código no servidor.

git add .

git commit -am "Curso valor slot filling"

git push heroku master

Testando no DialogFlow

The screenshot displays the Google DialogFlow console interface for testing the 'curso.valor' intent. The left sidebar contains navigation links for Intents, Entities, Domains, Training, Integrations, and Fulfillment. The main workspace shows the 'curso.valor' intent configuration. Under the 'Action' tab, a table lists parameters: 'curso' (required, entity '@curso', value '\$curso', is list 'No', prompt 'Qual curso? Escolha entre: [R, Python, Machine Learning, Hadoop]'), 'Enter name', 'Enter entity', 'Enter value', and a placeholder. The 'Response' tab shows two text response variants: '1. Infelizmente não conseguimos responder a esta pergunta agora. Tente novamente mais tarde.' and '2. Enter a text response variant...'. The right sidebar shows the 'Agent' configuration, including the 'USER SAYS' prompt 'Qual o valor do curso?', the 'RESPONSE' list of course options, the 'CONTEXTS' section with 'curso_valor_dialog_context' and a specific context ID, and the 'INTENT' and 'ACTION' sections both set to 'curso.valor'.

Figura 13. Testando as alterações no DialogFlow.

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

WebHook for Domains

Mais o que são domínios. Domínios são pacotes de conhecimentos já pré-definidos por áreas onde nós podemos ativar. 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 14.

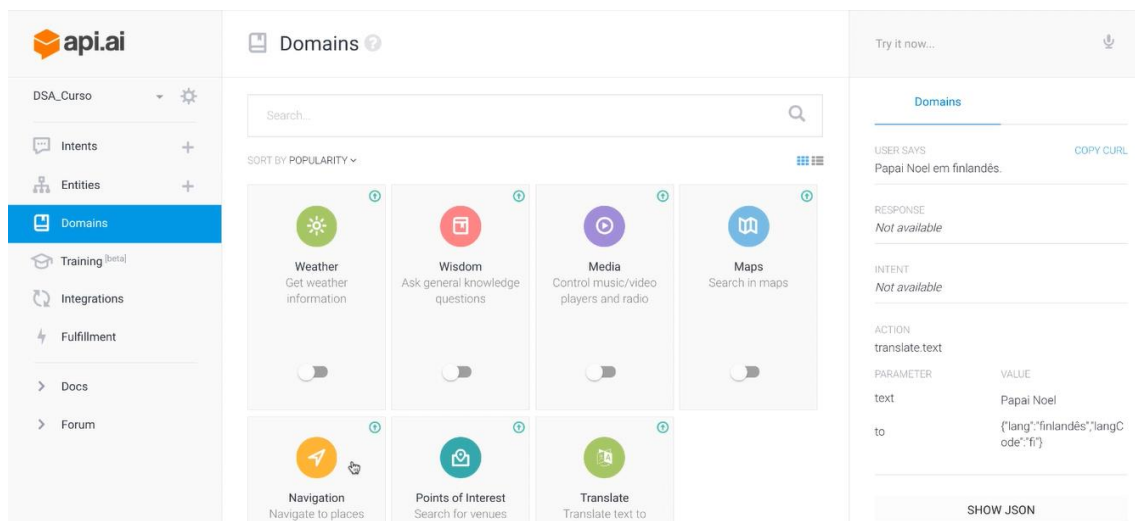


Figura 14. Biblioteca de Domínios do DialogFlow

Para testar, vamos habilitar o Domínio **Translate**. Clique em **VIEW DETAILS**, será mostrado a Figura 15, onde são exibidas algumas expressões.

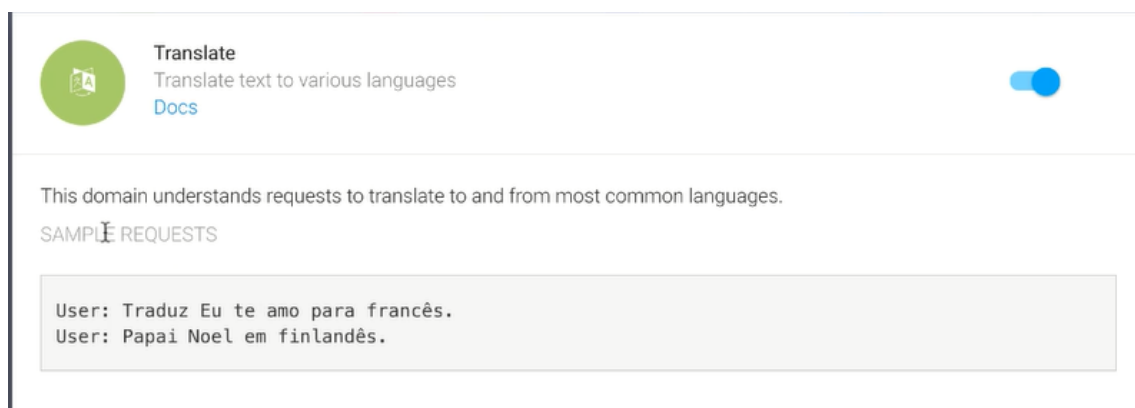



Figura 15.

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

O resultado deve ser como o da Figura 16. Observe as propriedades **text** e **to**.

Try it now... 

Domains

USER SAYS COPY CURL

Papai Noel em finlandês.

RESPONSE

Not available

INTENT

Not available

ACTION

translate.text


PARAMETER	VALUE
Text	Papai Noel
to	{"lang": "finlandês", "langCode": "fi"}

SHOW JSON

Figura 16. Saída apresentada pelo DialogFlow.

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

Vá até Fulfillment e habilite o WebHook para todos os domínios, como mostra a Figura 17.



DSA_Curso

Intents

Entities

Domains

Training ^[beta]

Integrations

Fulfillment

Docs

Forum

Fulfillment SAVE

Webhook

Your web service will receive a POST request from API.AI in the [form of the response](#) to a user query matched by intents with webhook enabled. Be sure that your web service meets all the [webhook requirements](#).

[Webhook example](#)

ENABLED ☒

URL*

http://dsa-bot-teste.herokuapp.com/webhook

BASIC AUTH

Enter username...

Enter password...

HEADERS

Enter key...

Enter value...

Enter key...

Enter value...

+ Add header

DOMAINS

Enable webhook for all domains

Figura 17. 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”.

Alterando o BackEnd para utilizar Domínios. Veja a Figura 18.

```
def makeWebhookResult(req):  
    action = req.get("result").get("action")  
  
    if (action != "curso.valor") and (action != "translate.text"):  
        return {}  
    result = req.get("result")  
    parameters = result.get("parameters")  
  
    # Curso  
    if (action == "curso.valor"):  
        curso = parameters.get("curso")  
  
        cost = {'R':100, 'Python':200, 'Machine Learning':300, 'Hadoop':400 }  
  
        if curso:  
            speech = "O valor do curso " + curso + " eh " + str(cost[curso]) + " reais."  
        else:  
            speech = "Qual curso? Escolha entre: " + str(cost.keys())  
  
    # Traducaao  
    if (action == "translate.text"):  
  
51     # Traducaao  
52     if (action == "translate.text"):  
53         text = parameters.get("text")  
54         language = parameters.get("to").get("lang")  
55         speech = text + " em " + language + " eh " + text[::-1]  
56
```

Figura 18. Código alterado para trabalhar com Domínios.

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.

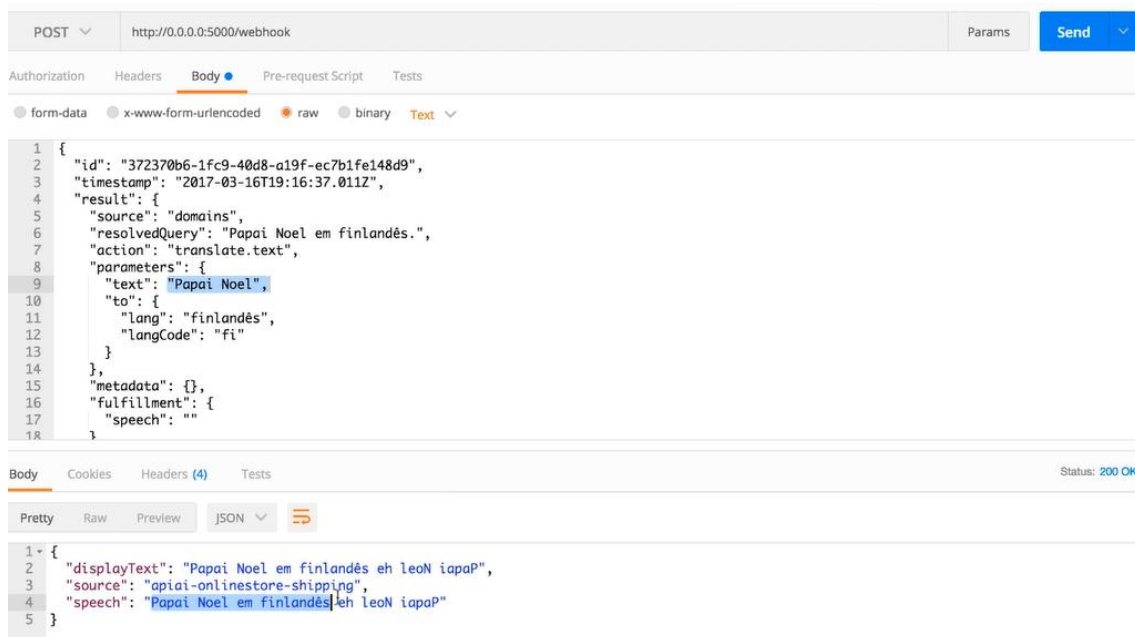


Figura 19. Testando a aplicação via Postman.

Faça o deploy e teste no DialogFlow.

```
C:\Users\José Antonio\Documents\GitHub\bots\ifrn-bot-teste [master =>] > git add .
C:\Users\José Antonio\Documents\GitHub\bots\ifrn-bot-teste [master =>] > git commit -am "webhook for domains"
On branch master
Your branch is up-to-date with 'heroku/master'.
nothing to commit, working directory clean
C:\Users\José Antonio\Documents\GitHub\bots\ifrn-bot-teste [master =>] > git push heroku master
```

Firestore

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. Mais 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 Firestore.

No Firestore temos diversas formas de armazenamento de dados, porém nós utilizaremos para esse exemplo o **RealTime Database**, como mostra a Figura 20.

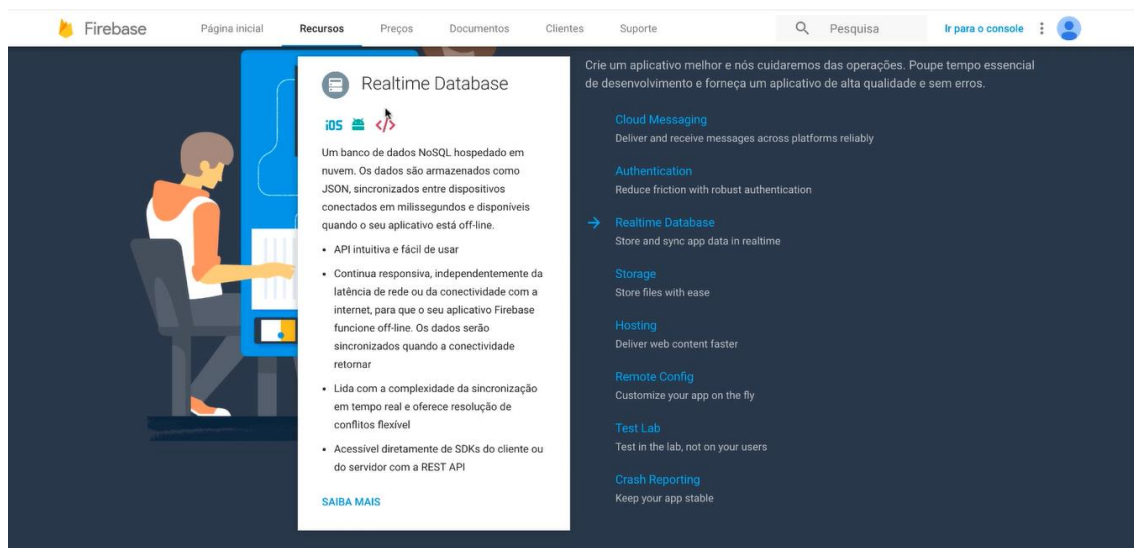


Figura 20.

Então, vamos criar um projeto no Firestore. Na Figura 21, clique no link Adicionar projeto.

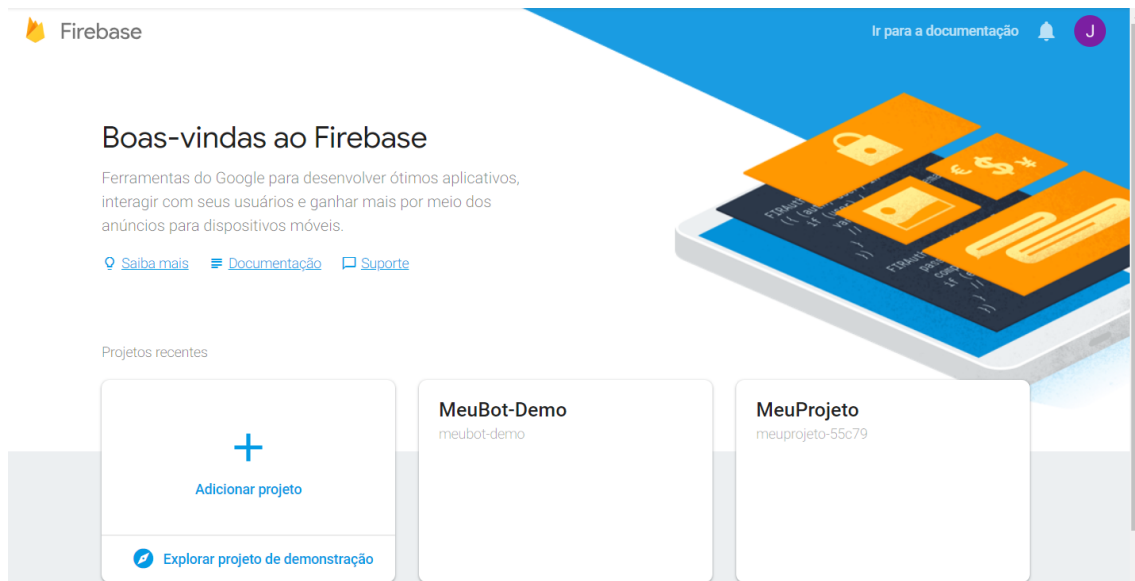


Figura 21. Criando novo projeto no Firebase.

Na Figura 22, informe o nome do projeto e clique no botão Criar projeto.

Figura 22.

Após criar o projeto, clique na opção do menu Database, como mostra a figura 23.

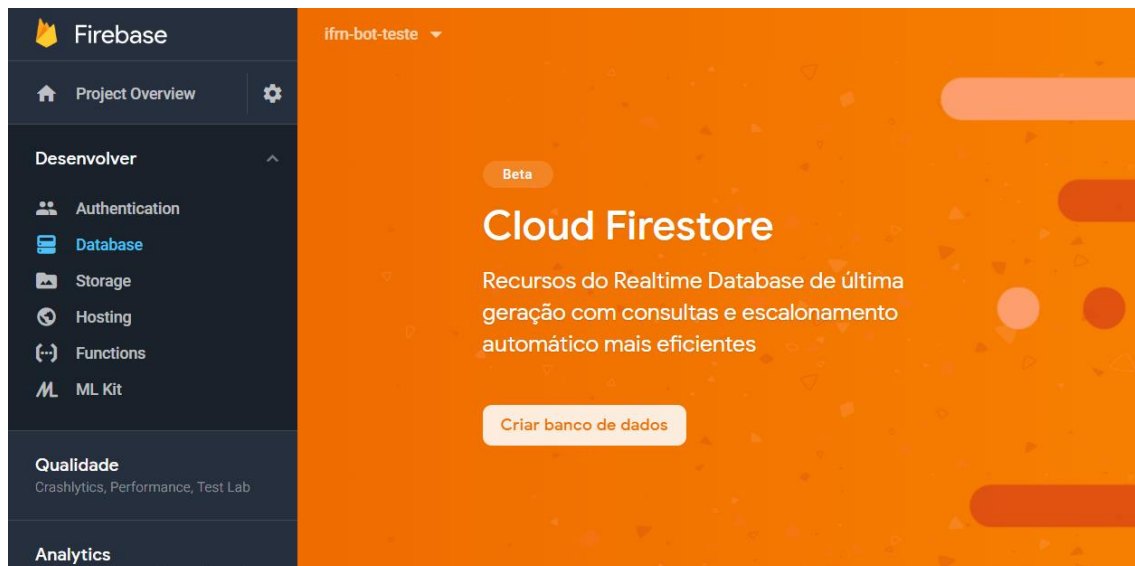


Figura 23.

E clique no botão criar banco de dados e na Figura 24 clique em Ativar.



Figura 24.

A figura 25, mostra a tela onde adicionamos novas coleções ao banco de dados.

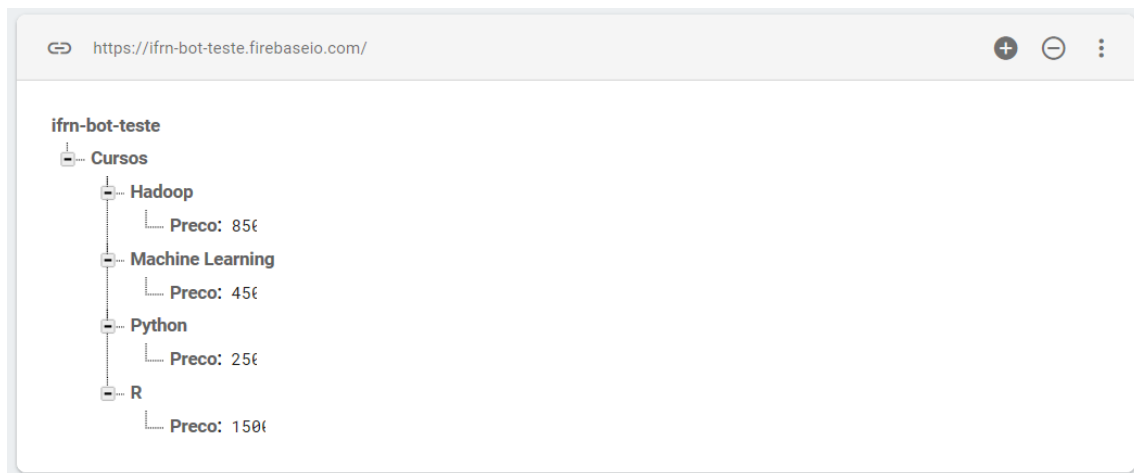


Figura 25. Adicionando coleções ao banco de dados.

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 na Figura 26.

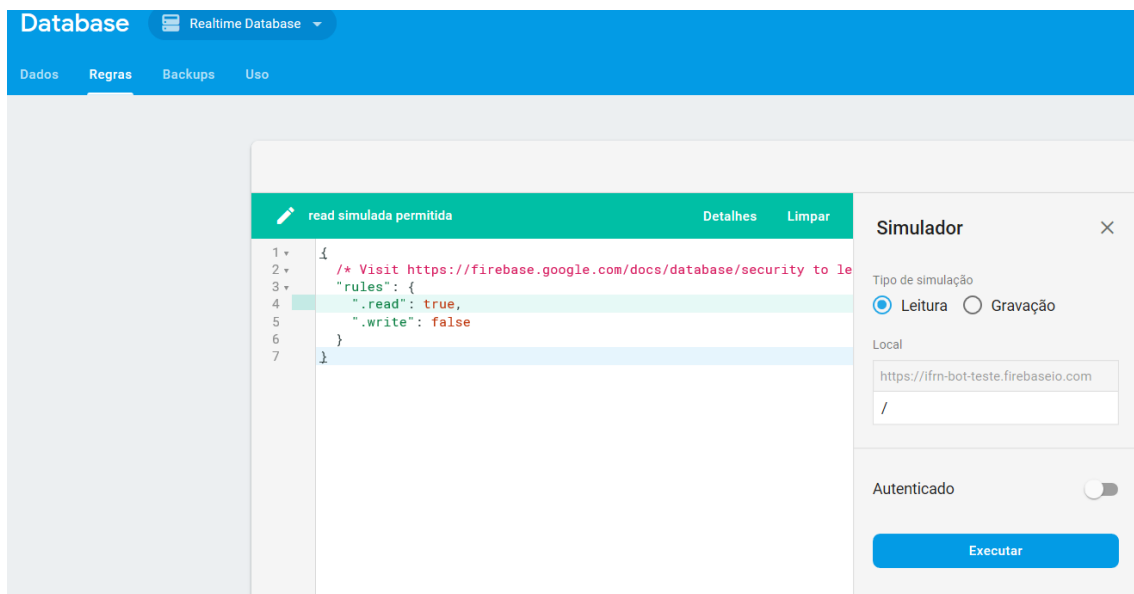


Figura 26. Regras de permissões.

Quando você muda uma regra de permissão, você deve publicá-la. Veja a Figura 27.

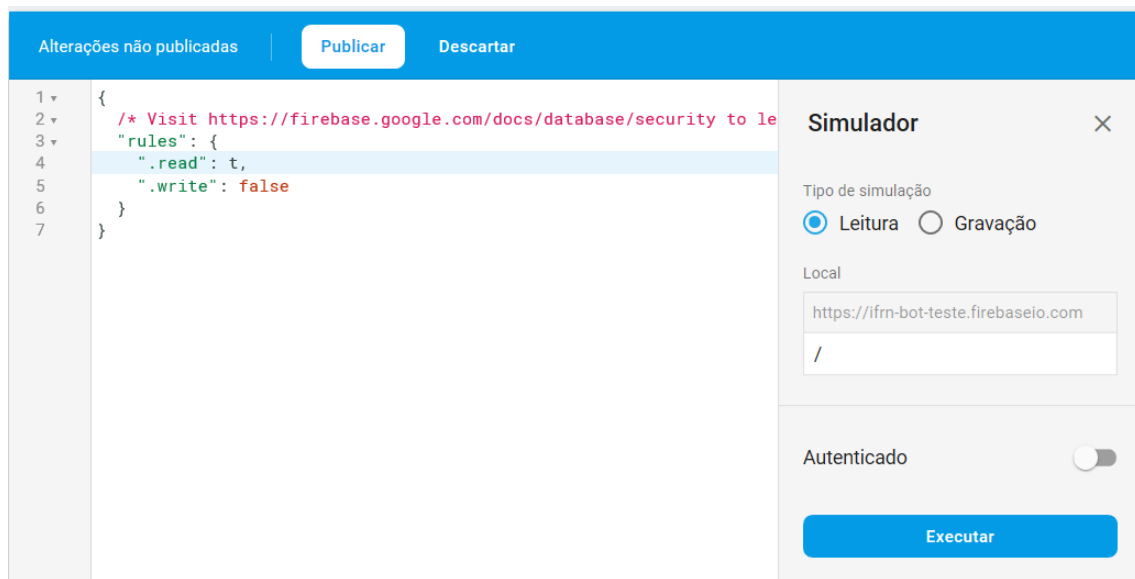


Figura 27. Publicar uma regra de permissão alterada.

Publique as modificações e simule ainda sem autenticar o usuário. Veja a Figura 28.

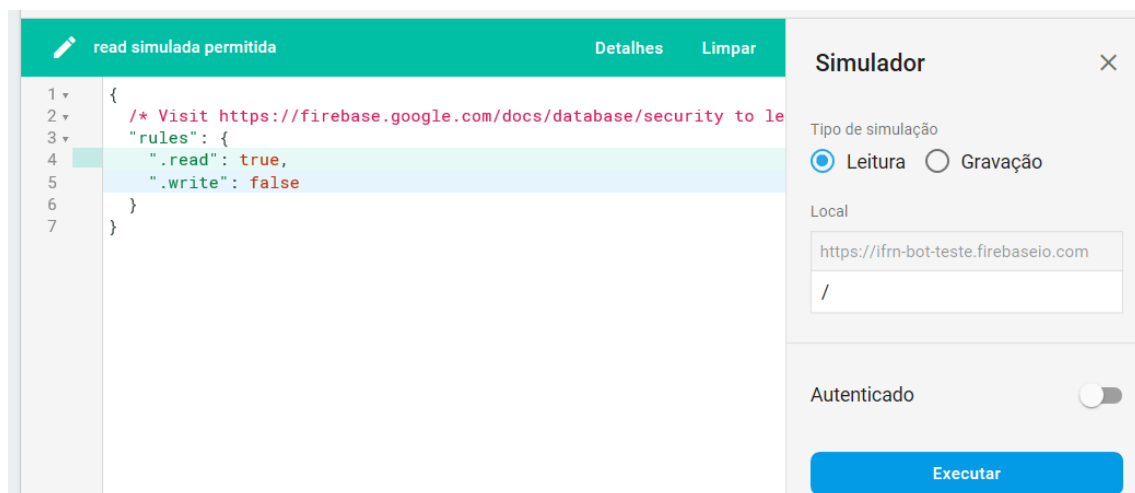


Figura 28.

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 29.

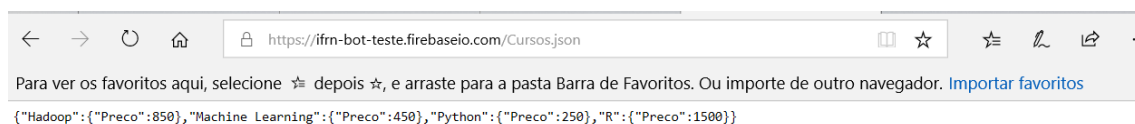
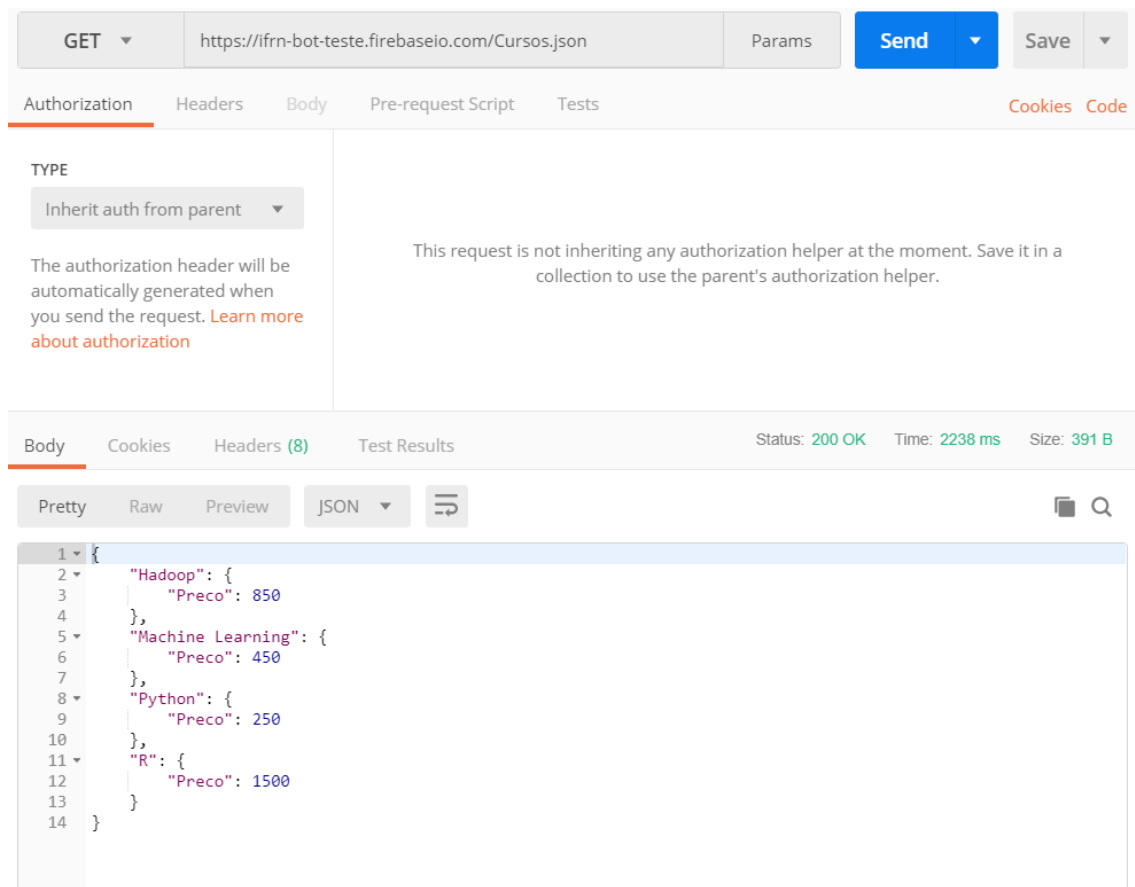


Figura 29.

Você também pode testar via Postman. Veja a Figura 30.



Recuperando valores do banco de dados

Antes de tudo, é necessário instalar localmente na sua máquina o pacote `python-firebase` e depois instalá-lo no heroku.

Então, abra um terminal e execute o comando

`pip install python-firebase` //Instala na máquina local

após instalação do pacote `python-firebase`, vamos fazer as alterações no backend. Observe que, na linha 6 fazemos a importação do pacote `firebase` e da linha 49 até a linha 51, estamos fazendo a leitura dos dados do banco de dados *“Realtime Database”*

```

1  #!/usr/bin/env python
2
3  import urllib
4  import json
5  import os
6  import firebase
7  import math
8
9  import urllib3
10
11 from flask import Flask
12 from flask import request
13 from flask import make_response
14
15 # Flask app should start in global layout
16 app = Flask(__name__)
17
18
19 @app.route('/webhook', methods=['POST'])
20 def webhook():
21     req = request.get_json(silent=True, force=True)
22
23     print("Request:")
24     print(json.dumps(req, indent=4))
25
26     res = makeWebhookResult(req)
27
28     res = json.dumps(res, indent=4)
29     print(res)
30     r = make_response(res)
31     r.headers['Content-Type'] = 'application/json'
32     return r

```

```

31 def makeWebhookResult(req):
32
33     action = req.get("result").get("action");
34
35     if (action != "preco.curso") and (action != "pedido.gravar"):
36         return {}
37     result = req.get("result")
38     parameters = result.get("parameters")
39
40     #cost = {'R':100, 'Python':200, 'Machine Learning':300, 'Hadoop':400}
41
42     # Curso
43     if (action == "preco.curso"):
44         curso = parameters.get("curso")
45
46         #cost = {'R':100, 'Python':200, 'Machine Learning':300, 'Hadoop':400}
47         from firebase import firebase
48         firebase = firebase.FirebaseApplication('https://ifrn-bot-teste.firebaseio.com', None)
49         preco = firebase.get("/Cursos", curso + "/Preco")
50
51         if curso:
52             speech = "O valor do curso " + curso + " é " + str(preco) + " reais."
53         else:
54             speech = "Qual curso? Escolha entre: " + str(cost.keys())
55

```

Faça o deploy para o heroku

git add .

git commit -am "firebase"

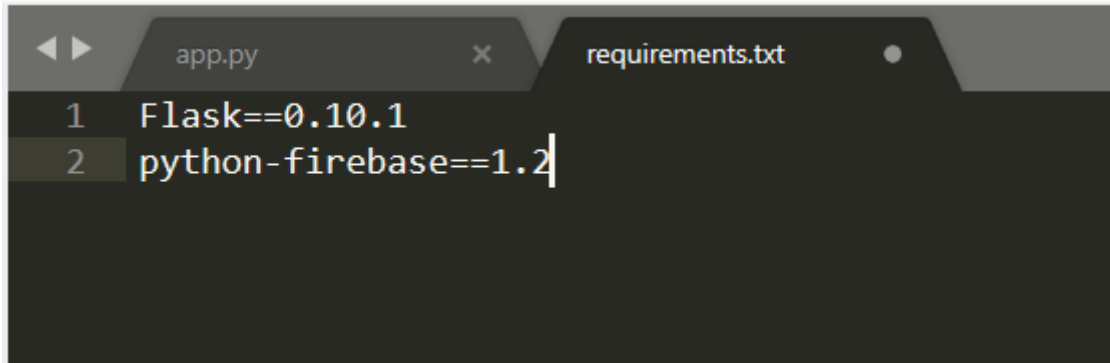
git push heroku master

Teste no DialogFlow e no Postman

Precisamos ainda instalar o firebase no heroku.

Instalando pacotes Python no Heroku

Abra o arquivo `requirements.txt` na pasta onde está seu arquivo `app.py` e, acrescente o código da linha 2, como mostra a Figura 31.



```
1 Flask==0.10.1
2 python-firebase==1.2
```

Figura 31. Instalando pacote Python no Heroku.

Faça novamente o deploy e teste.

Testando no Postman, Figura 32.

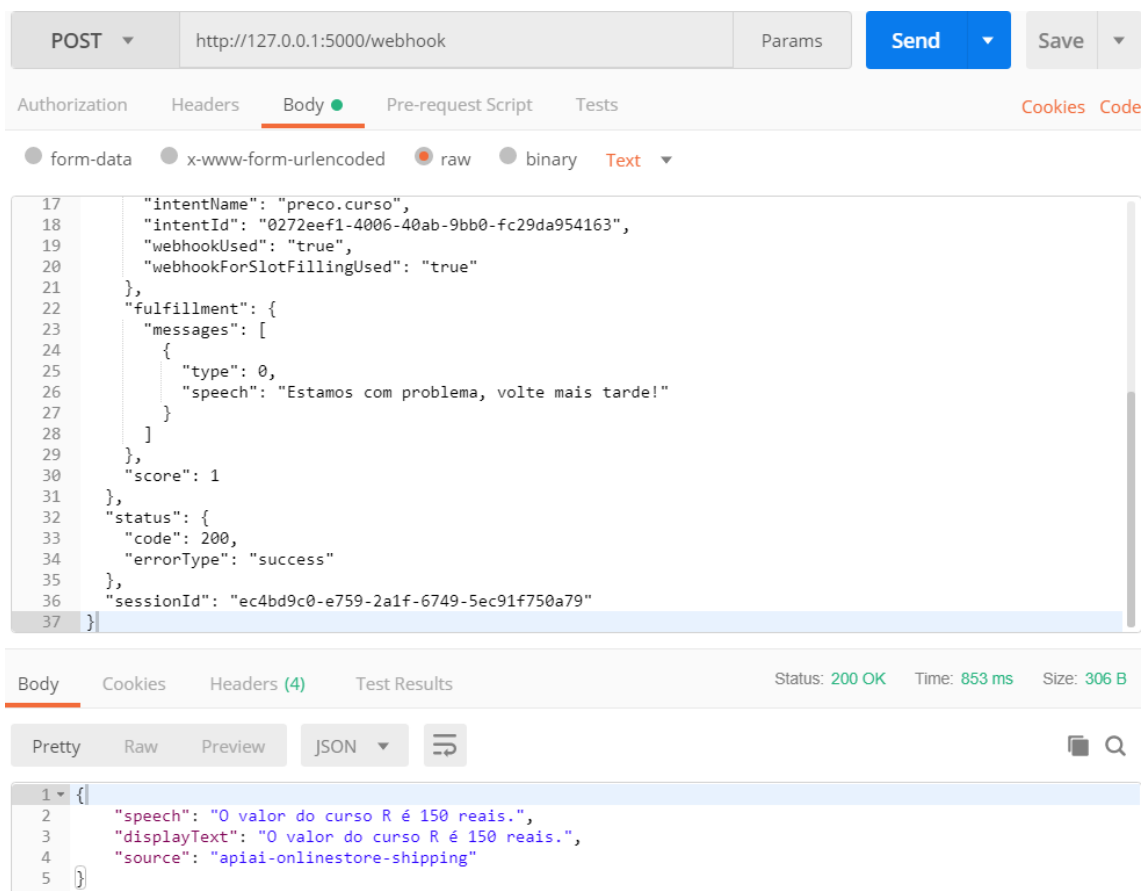





Figura 32.

Testando no DialogFlow, Figura 33.



 See how it works in [Google Assistant](#). 



Agent

Domains

USER SAYS

[COPY CURL](#)

Qual o valor do curso de R?

 DEFAULT RESPONSE 

O valor do curso R é 150 reais.

INTENT

[preco.curso](#)

ACTION

[preco.curso](#)

PARAMETER	VALUE
curso	R

DIAGNOSTIC INFO

Figura 33.

Na seção anterior, fizemos apenas a leitura dos dados do banco de dados, que não precisa fazer autenticação. Agora vamos fazer a gravação de dados, onde é necessário se autenticar no banco de dados.

Gravando dados no banco de dados

Você deve criar uma intenção pedido.confirmacao-sim. Nessa intenção defina a ação (action) como pedido.gravar. Uma resposta do tipo “Estamos com problema, volte mais tarde!” e habilite essa intenção para usar WebHook.

Altere o código no backend para gravar o pedido.

```
31 def makeWebhookResult(req):
32
33     action = req.get("result").get("action");
34
35     if (action != "preco.curso") and (action != "pedido.gravar"):
36         return {}
37     result = req.get("result")
38     parameters = result.get("parameters")

# Gravando Pedido
if (action == "pedido.gravar"):
    from firebase import firebase
    from firebase.firebase import FirebaseApplication, FirebaseAuthentication

    authentication = FirebaseAuthentication('3c455ba0-e846-46ee-9bce-a1dfc59fd608', True, True)
    firebase = firebase.FirebaseApplication('https://ifrn-bot-teste.firebaseio.com', authentication)

    parameters = result.get("contexts")[0].get("parameters")
    nome = parameters.get("nome")
    tamanho_pao = parameters.get("tamanho_pao")
    tipo_pao = parameters.get("tipo_pao")
    recheio = parameters.get("recheio")
    queijo = "Não"
    dobro_queijo = "Não"
    if (parameters.get("dobro_queijo")):
        queijo = "Sim"
        dobro_queijo = parameters.get("dobro_queijo")

    pedido = {"nome": nome, "tamanho_pao": tamanho_pao, "tipo_pao": tipo_pao, "queijo": queijo, "dobro_queijo": dobro_queijo}
    result = firebase.post("/Pedidos", pedido)
    speech = "Anoto seu pedido: {}. Volte sempre!".format(result['name'])
```