



Universidade Federal do Rio Grande do Norte – UFRN

Escola Agrícola de Jundiaí – EAJ

Tecnólogo em Análise e Desenvolvimento de Sistemas

Middleware Para Iot Visando Economia de Energia

Manual do Usuário

Macaíba – RN

08/03/2019



## Sumário

1 – Middleware .....	04
2 – Mqtt Protocol .....	04
2.1 - Brooker Mqtt Eclipse Mosquito .....	04
3 – Módulo NodeMCU.....	09
4 – Serviço Web.....	13
4.1 - Serviço Web (Recurso esp) .....	18
4.2 - Serviço Web (Recurso politica) .....	24

## 1 – Middleware

Podemos entender um Middleware como sendo um Software que tem como principal função oferecer serviços para outras aplicações, além das que o S.O fornece.

Seu principal objetivo aqui é facilitar a entrada e saída de informações. Muito útil, pois, permite a troca de informações entre aplicações com diferentes protocolos de comunicação.

## 2 - Mqtt Protocol

Protocolo de comunicação de mensagens leves entre sensores e outros dispositivos móveis, otimizado para redes TCP/IP não confiáveis ou de alta falência. O esquema de troca de informações funciona no modelo Publicador-Subscritor, de maneira geral, podemos entender que o subscritor é aquele que assinará um tópico qualquer onde o publicador pública informações (Wikipédia, pt.wikipedia.org).

### 2.1 – Brooker Mqtt Eclipse Mosquito

O Eclipse Mosquitto é um intermediador de mensagens de código aberto, que aplica as versões 3.1 e 3.1.1 do protocolo Mqtt.

Vamos ao download e configuração do arquivo.

1 – Acessar a página oficial do Eclipse mosquito e execute o download <https://mosquitto.org/>.

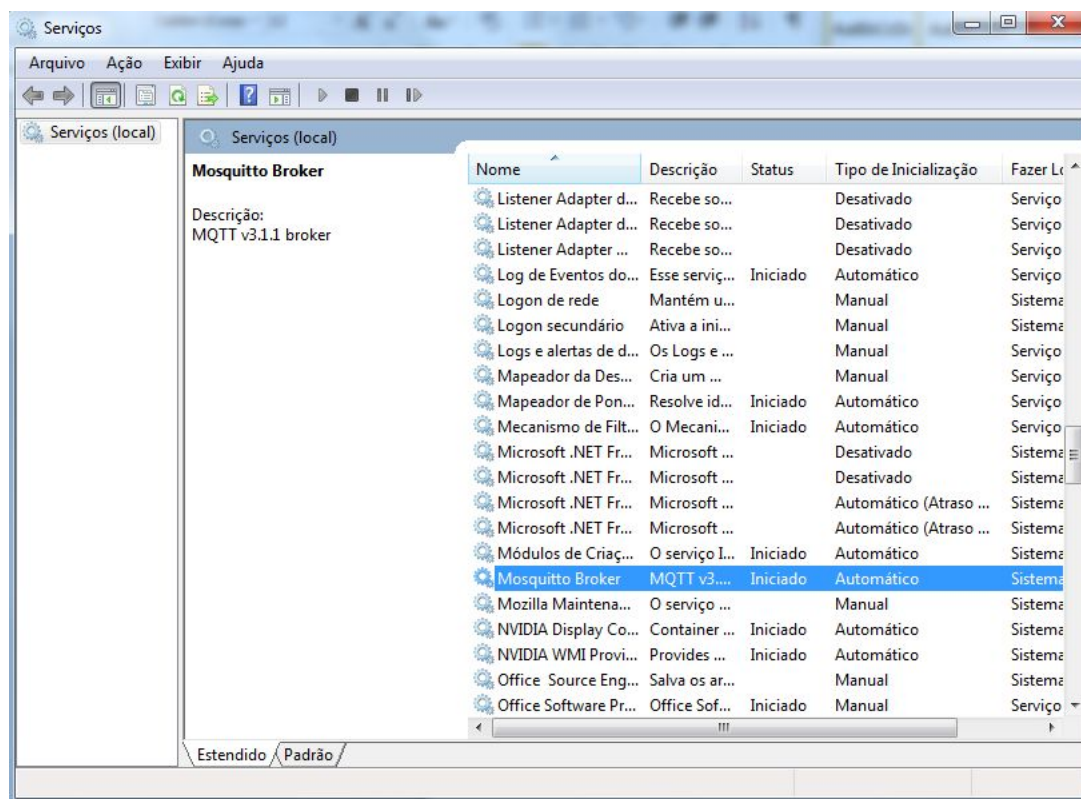
2 – baixe também os arquivos de dll e OpenSSL que serão necessários para o funcionamento do Mosquitto, <http://www.mediafire.com/folder/o78dh3no7hlpo/pthreadVC2>.

3 – Instale o arquivo OpenSSL 1.1.0i e em seguida execute a instalação do Eclipse Mosquitto.

4 – Após a instalação, vá até o diretório onde foi instalado o Mosquitto, copie o arquivo pthreadVC2.dll que você efetuou o download e cole na pasta do Mosquitto, o sistema operacional informará que já existe um arquivo com o mesmo nome, não se preocupe, clique na opção “Colar e Substituir”.

5 – Após seguir os passos acima, podemos conferir se o serviço do mosquito está em funcionamento, para isso, vamos acessar a área de serviços do Windows, basta ir no menu iniciar e digitar “serviços”, será aberto uma tela que lista todos os serviços no sistema operacional, com isso, podemos procurar o serviço do nosso Eclipse Mosquitto.

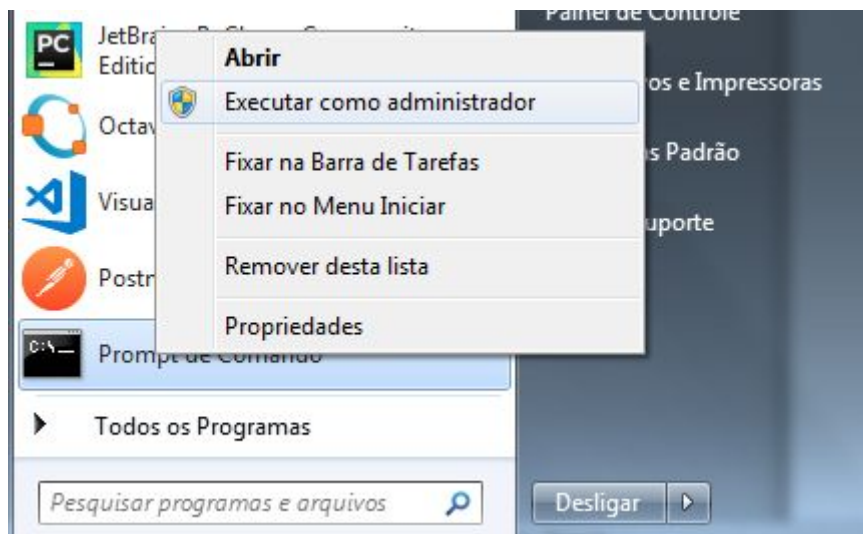
**Figura 01 - Tela de serviços do Windows.**



6 – Neste caso, o serviço já está iniciado, porém pode acontecer dele está parado, então inicie o serviço, é recomendado que reinicie sua máquina. Após isso verifique novamente se ele está iniciado. Com tudo pronto, podemos testar o nosso mediador de mensagens, e verificar se tudo está funcionando da maneira correta.

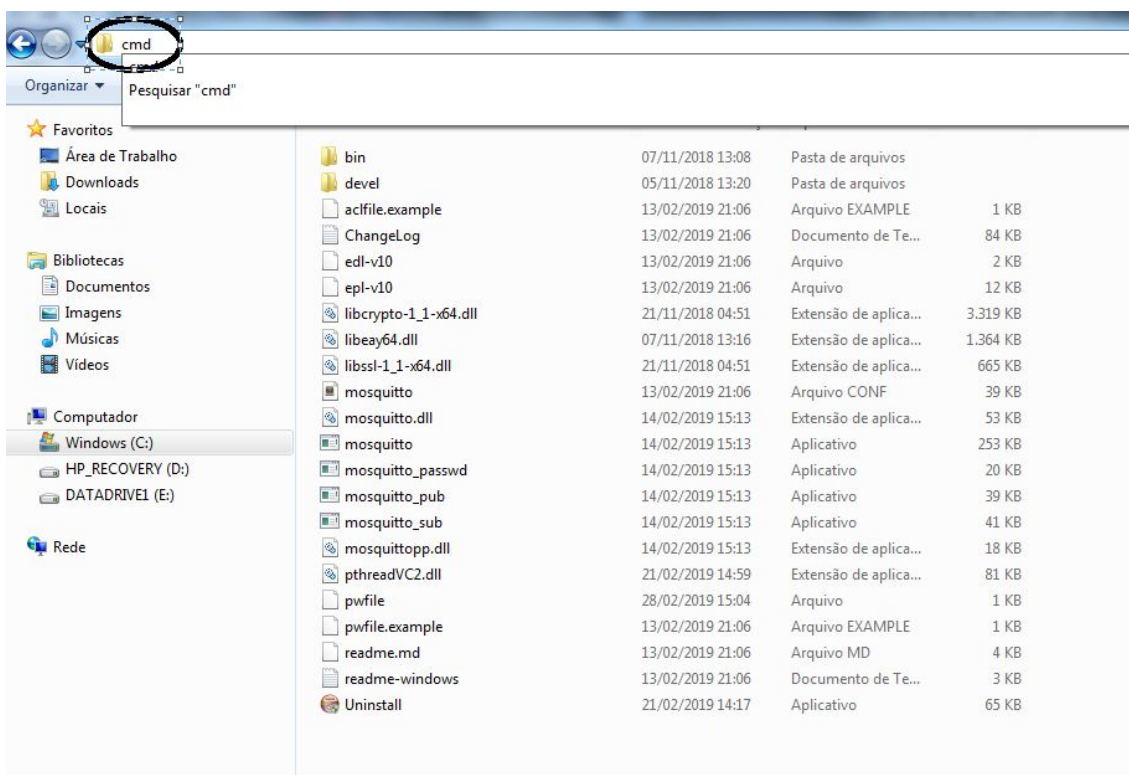
7 – Abra o terminal de comando (CMD) como modo administrador.

**Figura 02 - Abrindo cmd em modo administrador**



8 – No terminal, vá até a pasta raiz do Eclipse Mosquitto, caso não saiba fazer este procedimento pelo terminal, você também pode ir até a pasta pelo Windows Explorer e digitar URL no caminho a palavra “cmd”, automaticamente será aberto uma aba do cmd já na raiz do caminho do mosquito.

**Figura 03 -Abrindo o cmd através da raiz do arquivo**



9 – Agora vamos simular um encaminhamento de mensagens, primeiro vamos criar um Subscritor:

Comando > **mosquitto\_sub -h 127.0.0.1 -p 1883 -t "teste"**

-h > endereço onde o serviço está rodando.

-p > Porta padrão do Eclipse Mosquitto.

-t > define o tópico que este Subscritor está inscrito.

Repita o passo 8, não feche o cmd atual, agora iremos criar o Publicador:

Comando > **mosquitto\_pub -h 127.0.0.1 -p 1883 -t "teste" -m "Olá subscritor"**

-m > mensagem publicada.

-t > tópico onde este será publicado a mensagem.

No fim, perceba que a mensagem chegará até a aba do cmd onde foi criado o Subscritor daquele tópico.

**Figura 04 - Criando um sub**

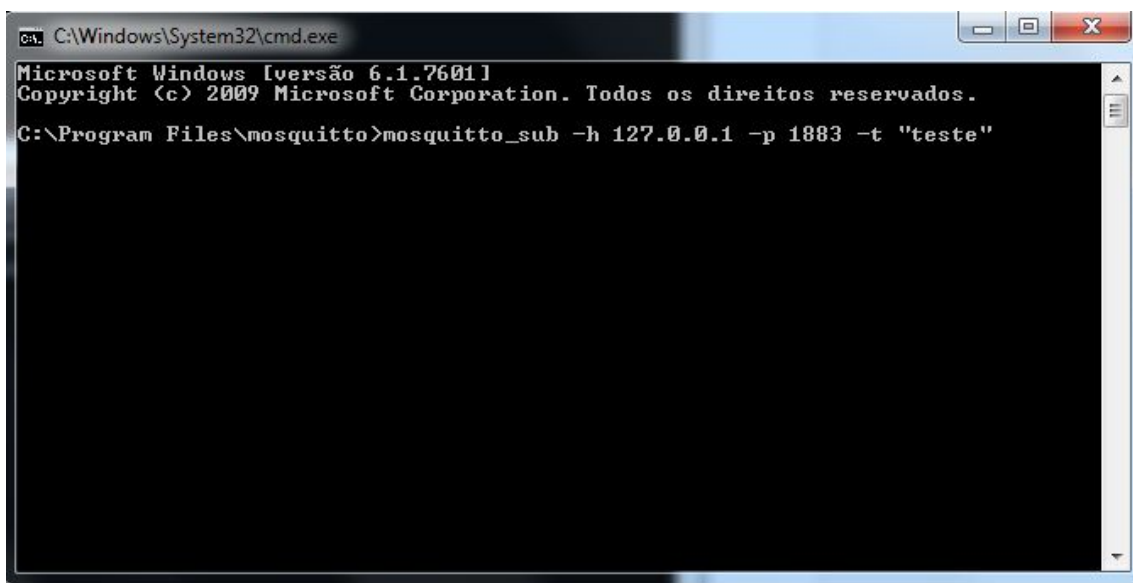
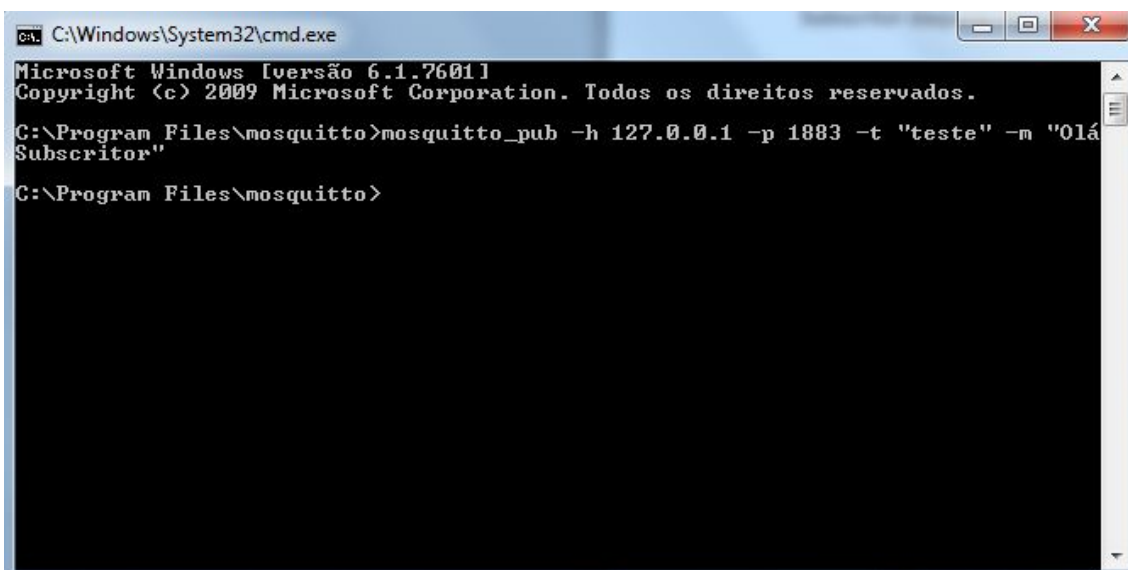


Figura 05 - Criando um publicador e mandando uma mensagem

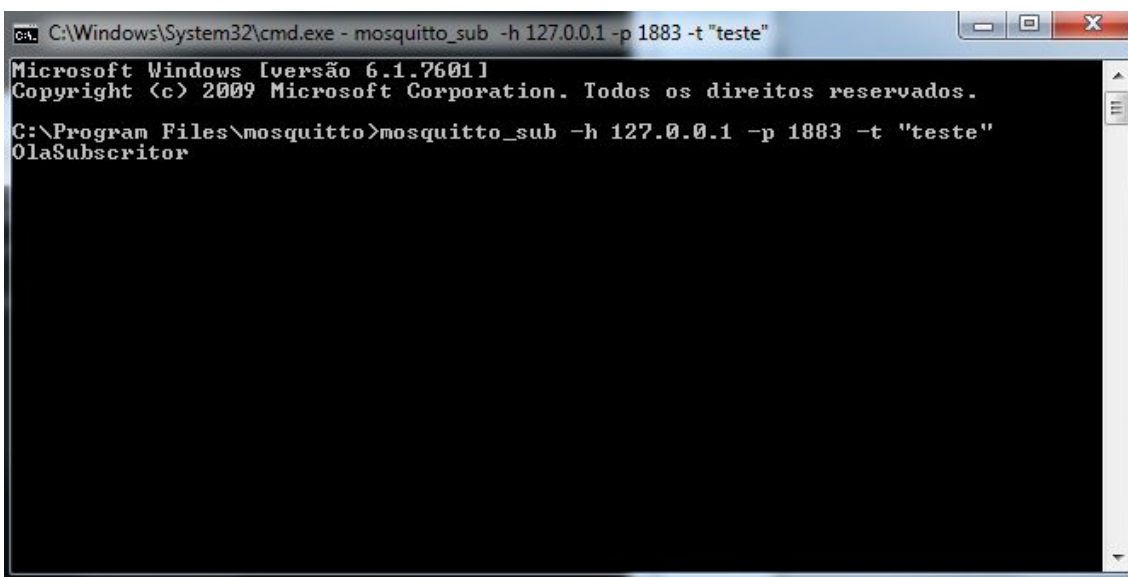


```
C:\Windows\System32\cmd.exe
Microsoft Windows [versão 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Todos os direitos reservados.

C:\Program Files\mosquitto>mosquitto_pub -h 127.0.0.1 -p 1883 -t "teste" -m "Olá
Subscritor"

C:\Program Files\mosquitto>
```

Figura 06 - Recebimento da mensagem



```
C:\Windows\System32\cmd.exe - mosquitto_sub -h 127.0.0.1 -p 1883 -t "teste"
Microsoft Windows [versão 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Todos os direitos reservados.

C:\Program Files\mosquitto>mosquitto_sub -h 127.0.0.1 -p 1883 -t "teste"
OláSubscritor
```



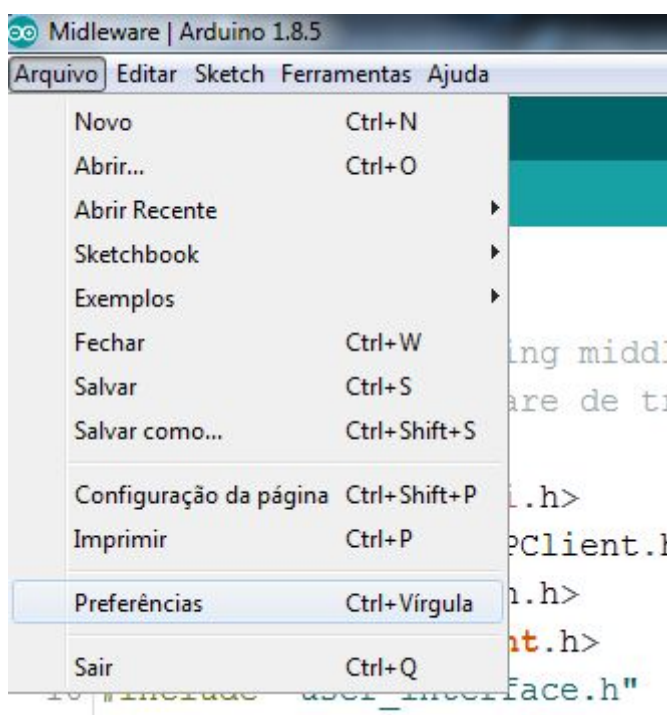
### 3 – Módulo nodeMCU

NodeMcu é uma plataforma de prototipagem que contém o microcontrolador Esp8266, diferente do Atmega do Arduino, este suporta a transmissão de dados através da rede Wifi, no projeto ele é usado por controlar os nós sensores, é ele que envia informações coletadas pelos nós e também é o subscritor do tópico, para receber informações sobre o modo de funcionamento, relacionado a economia de energia.

Você pode obter acesso ao projeto acessando o github do criador, [https://github.com/lucasbernardo95/mpm2ee\\_Node](https://github.com/lucasbernardo95/mpm2ee_Node), efetuado o download, agora vamos configurar o ambiente Arduino Ide para podemos usar o NodeMCU, é bem simples.

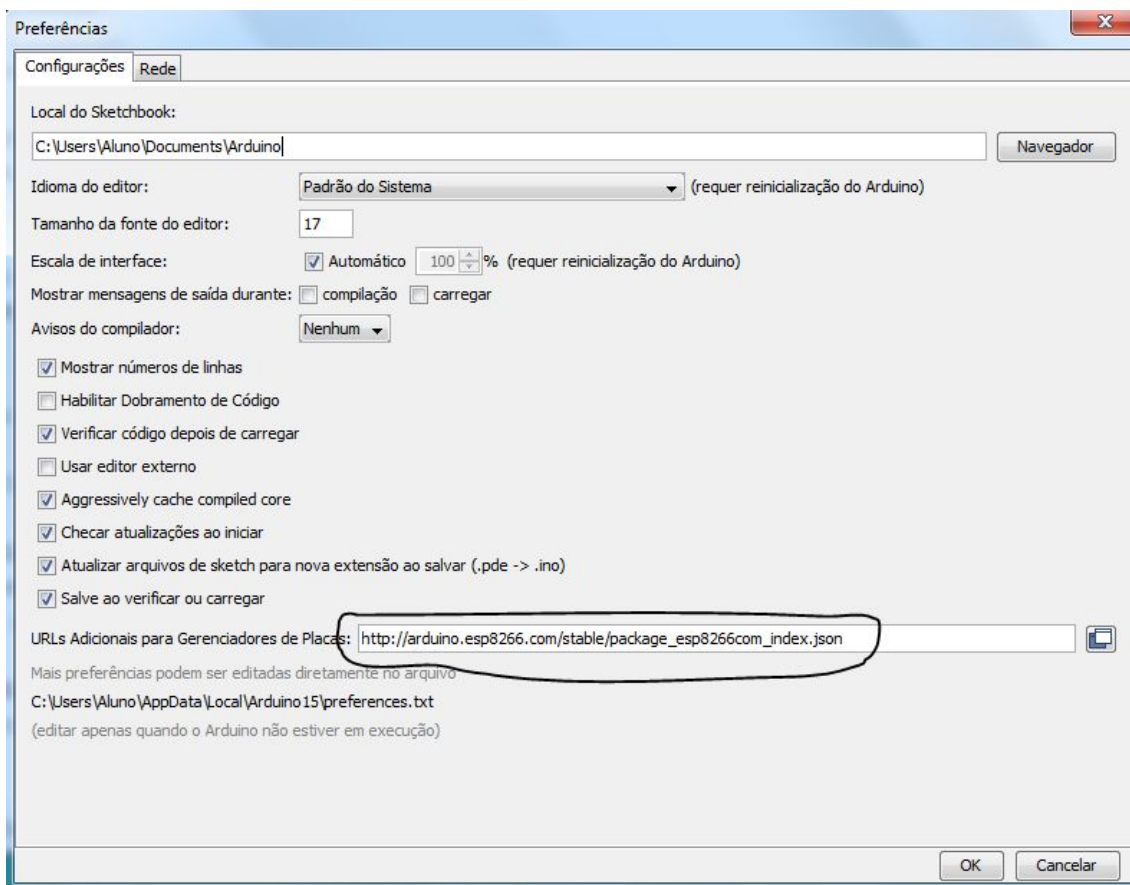
1 – Abra o Arduino Ide, acesse a opção em **Arquivo > Preferências**.

**Figura 07 - Abrindo o menu Preferências do Arduino IDE**



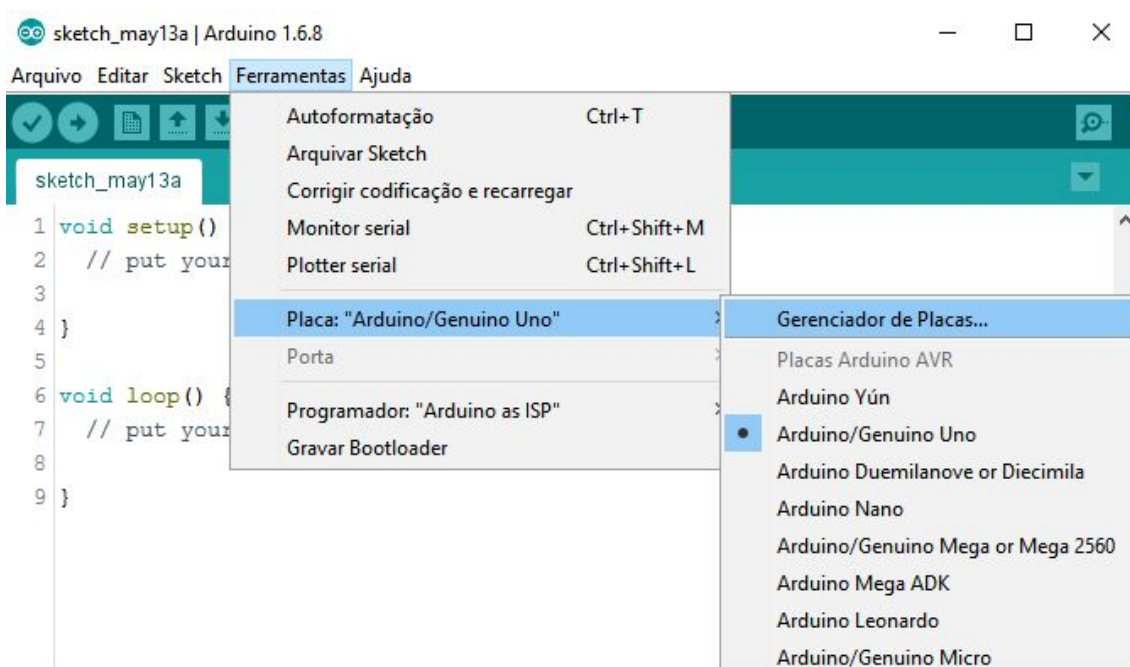
2 – Na seguinte tela, você irá digitar em “Urls adicionais de Gerenciamento de Placas” a seguinte URL > [http://arduino.esp8266.com/stable/package\\_esp8266com\\_index.json](http://arduino.esp8266.com/stable/package_esp8266com_index.json), após feito isso, pode clicar em OK.

**Figura 07 - Adicionando URL do esp8266 ao Gerenciador de Placas**



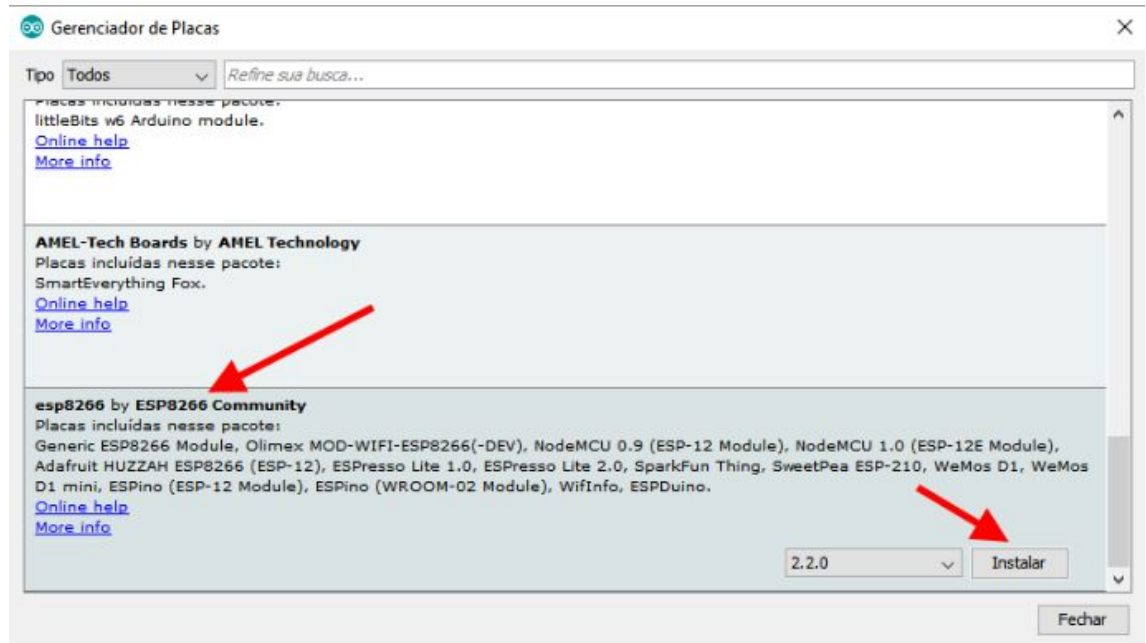
3 - Acesse a aba **Ferramentas > Placa > Gerenciador de Placas**.

**Figura 08 - Acessando Gerenciador de Placas**



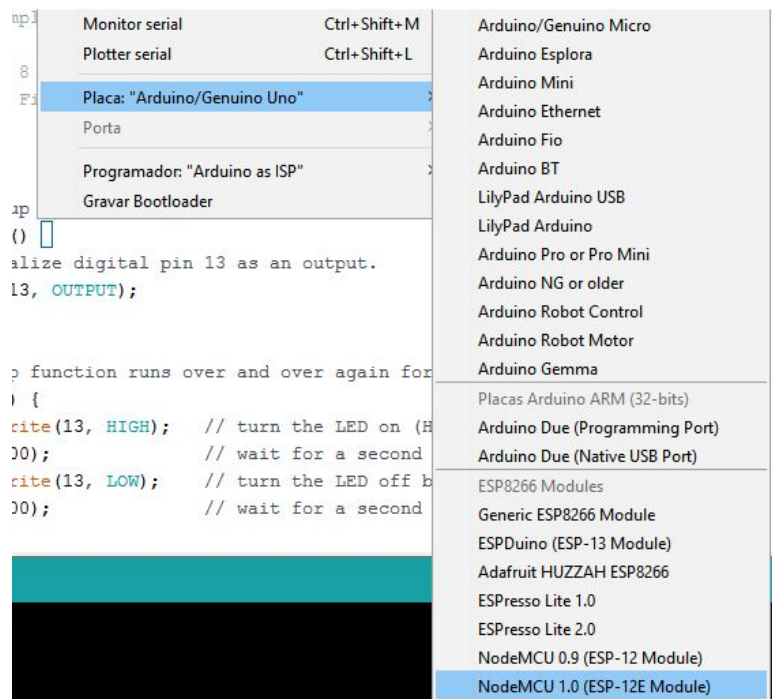
4 – Na barra de pesquisar, pesquise por esp8266

**Figura 09 - Download do Esp8266 Community**



5 – Feita a instalação, basta redefinir o modelo de placa em **Ferramentas > Placas**, selecione a placa NodeMCU 1.0 (ESP-12E module).

**Figura 10 - Selecionando o modelo de placa**



6 – Abra o arquivo *mpm2ee.ino* do projeto baixado do github no Arduino Ide, será necessário algumas alterações para que funcione de forma correta.

**Figura 11 - Método tryConnectWifi - projeto Node**

```

50 //Tenta conectar à rede wifi, se deu certo retorna true
51 bool tryConnectWifi() {
52     Serial.println("====begin tryConnectWifi=====\n");
53     int cont = 0, tempo = 1000;
54     const char* SSID      = "???"; // Nome da rede wifi
55     const char* PASSWD    = "???";
56     //WiFi.config(ip, dns, gateway, subnet);
57     WiFi.begin(SSID, PASSWD);
58     while (WiFi.status() != WL_CONNECTED) {
59         Serial.println("tentando reconectar");
60         delay(tempo);
61         cont += tempo;
62         if (cont == (tempo * 8)) { //tenta por 4s
63             Serial.println("ERRO AO CONECTAR");
64             return false;
65         }
66     }
67     Serial.println("====END tryConnectWifi=====\n");
68     Serial.println("CONEXÃO BEM SUCEDIDA");
69     return true;

```

No método TryConnectWifi, será necessário que informe o SSID e a senha da rede Wifi que você está utilizando para conectar-se.

**Figura 12 - Método startMqtt - projeto node**

```

40 /*Seta os parâmetros para conexão com o servidor broker*/
41 void startMqtt() {
42     Serial.println("====begin startMqtt=====\n");
43     const char* BROKER    = "10.77.34.241"; //Endereço do broker
44     const int   PORT      = 1883;          //porta padrão usada pelo mqtt para o broker 1883
45     client.setServer(BROKER, PORT);        //seta a porta e o endereço do broker
46     client.setCallback(callBackMqtt);      //faz uma chamada ao broker
47     Serial.println("====END startMqtt=====\n");
48 }

```

No método startMqtt, informe o endereço Ip local da máquina que está sendo utilizada.

Após feitas as alterações, rode o projeto e verifique se tudo ocorreu da maneira adequada.

#### 4 - Serviço Web

Agora que concluímos a configuração do nosso mediador Mosquitto e nosso projeto do Node já está rodando da maneira correta, podemos então colocar o serviço para funcionar.

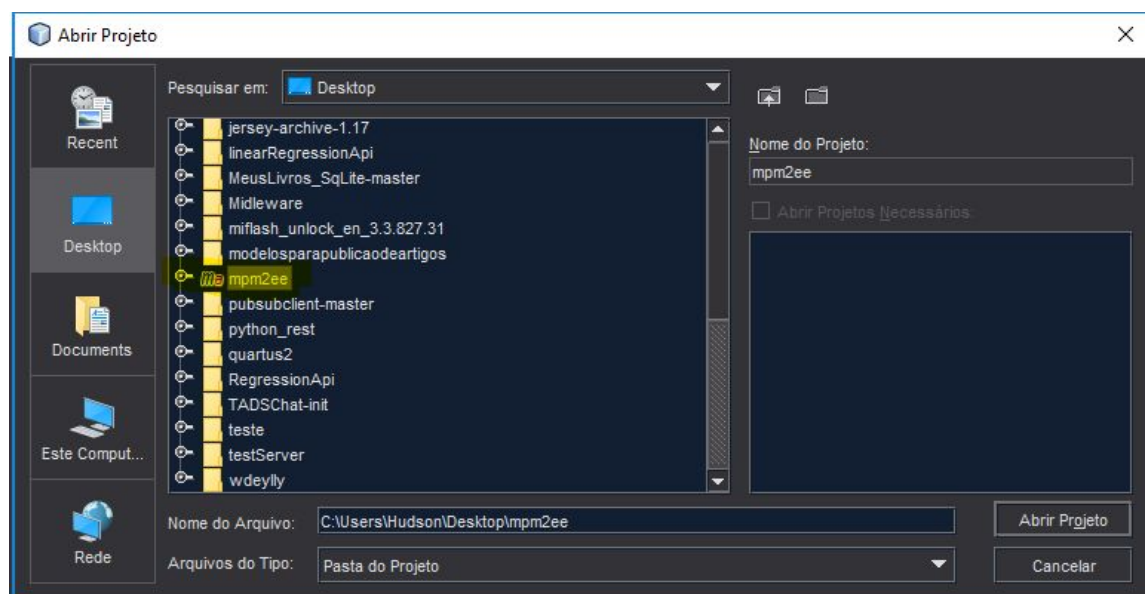
Neste projeto foi criado um serviço web baseado no padrão arquitetural Rest (Representational State Transfer), (Wikipédia, pt.wikipedia.org).

1 - Faça o download do projeto acessando <https://github.com/lucasbernardo95/mpm2ee>, github do criador do projeto.

2 - Abra o projeto na sua Ide, neste tutorial, estou utilizando o Netbeans Ide 8.2 (Netbeans, netbeans.org/).

Acesse a aba **File > Open Project** e vá até onde o projeto se encontra e depois clique em Open Project.

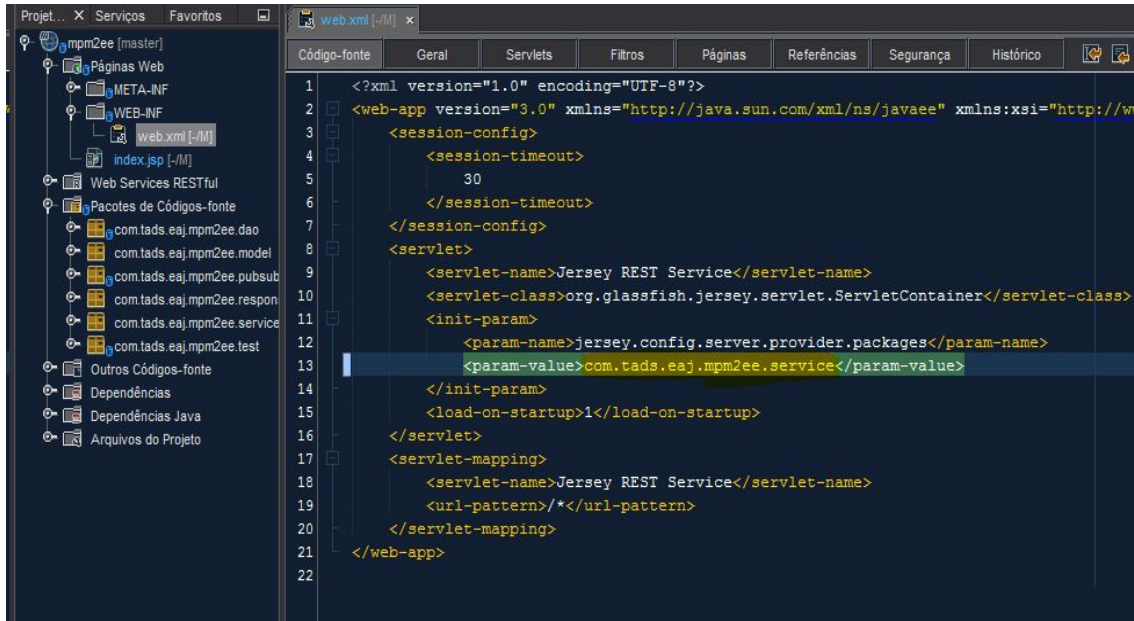
Figura 12 - Abrindo o projeto “mpm2ee”





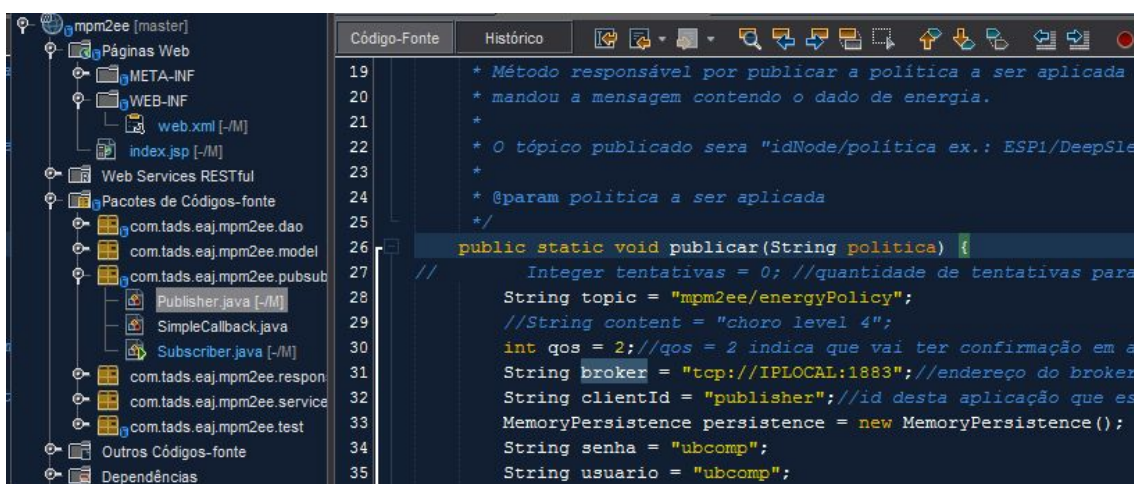
3 - Com o projeto aberto, vá até **Páginas Web > WEB-INF > web.xml**, certifique se na linha 13 em Param-value está escrito desta forma, senão, altere o conteúdo para (com.tads.eaj.mpm2ee.service).

Figura 13 - Arquivo web.xml



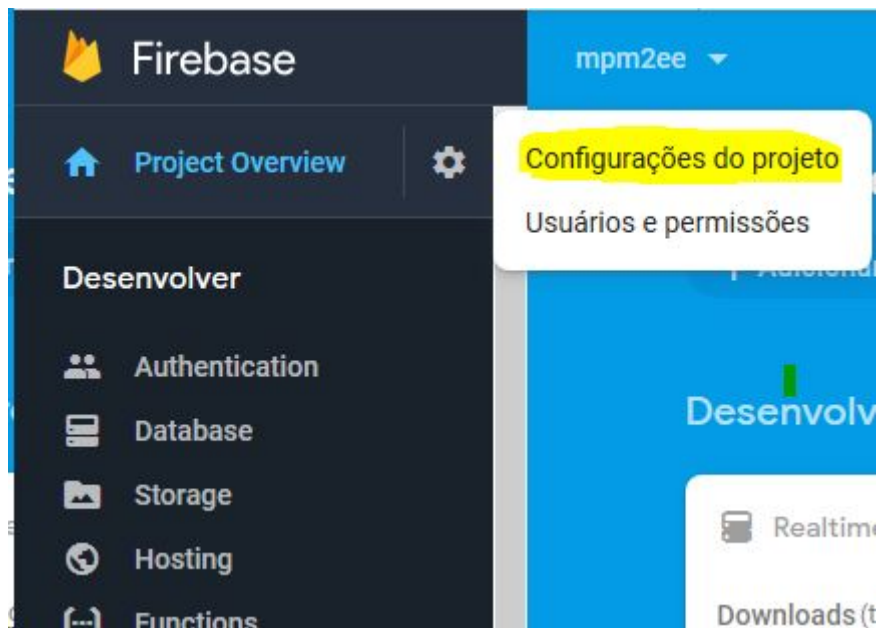
4 - Em Páginas de Código-fonte > com.tads.eaj.mpm2ee.pubsub > **Publisher.java** Na linha 31 do método Publicar(), você deve colocar o Ip local da máquina, da mesma forma que foi feito no método startMqtt() no projeto do Node.

Figura 14 - Classe Publisher



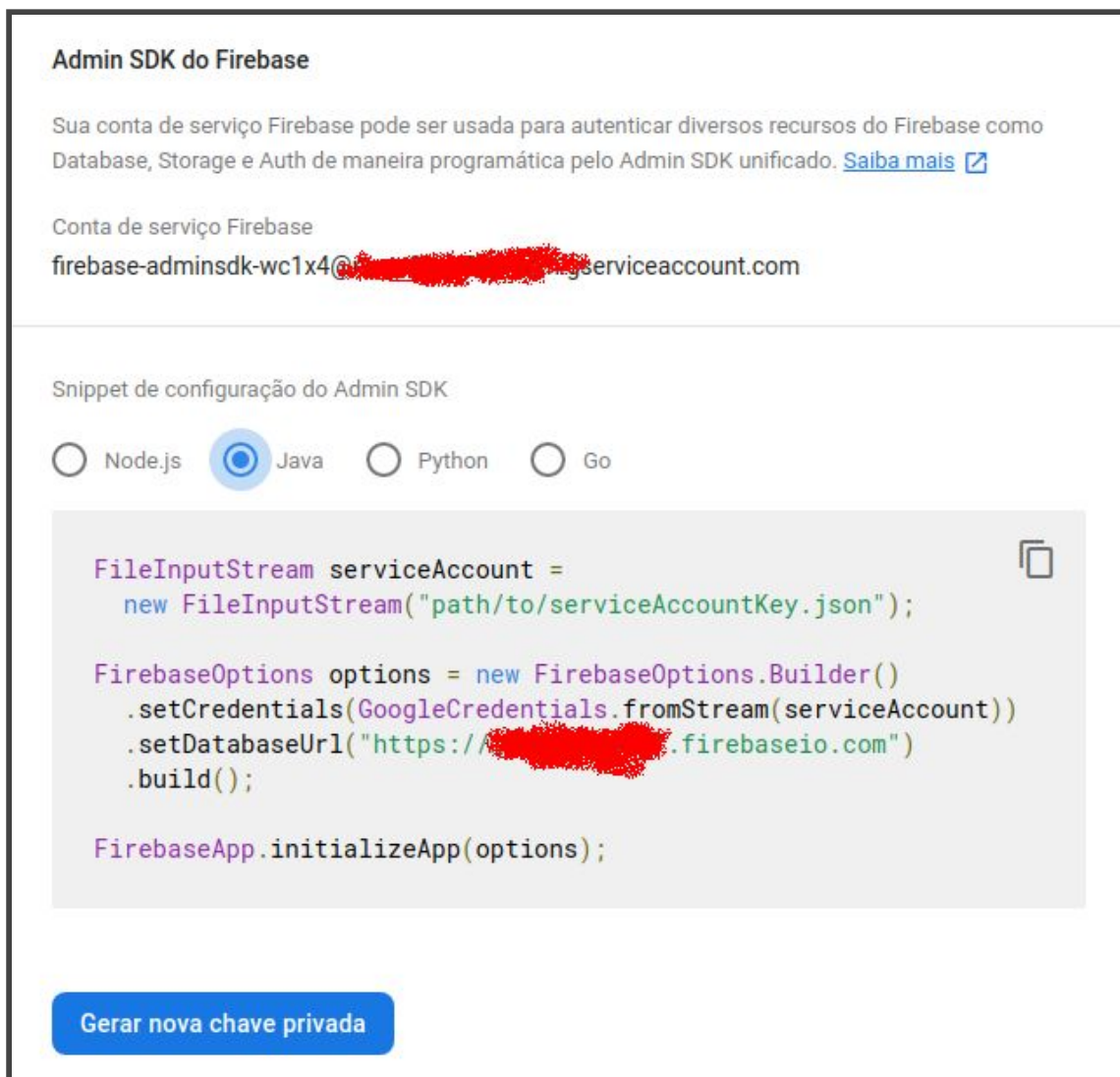
5 - Agora vamos redefinir as configurações de Autenticação ao Realtime Database do Firebase, para isso vamos acessar nosso projeto do Firebase, na tela do projeto vá em **Settings > Configurações do Projeto**.

Figura 15 - Tela de configurações de projeto Firebase



6 - Na tela de configurações, vá até a aba **Contas e Serviços**, em **Admin SDK do Firebase**, selecione a opção **Java**, e depois clique em **Gerar nova chave privada**. Será baixado um arquivo no formato Json com as credenciais de acesso ao projeto do firebase.

Figura 16 - Admin Sdk do firebase



7 - Agora precisamos referenciar o arquivo de credenciais no projeto de serviço, para isso vamos ao projeto novamente, vá em **Pacotes de Código-fonte > com.tads.eaj.mpm2ee.dao > AuthFactory.java**, vá até o método `authenticateWithPrivileges()` na linha 283 e altere na linha 287 o diretório onde encontra-se o arquivo com as credenciais que foram baixados na página do projeto no firebase.

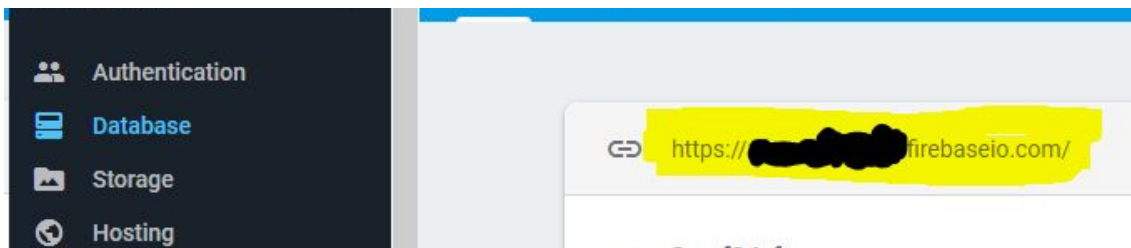
Figura 17 - Método `authenticateWithPrivileges()` linha 287





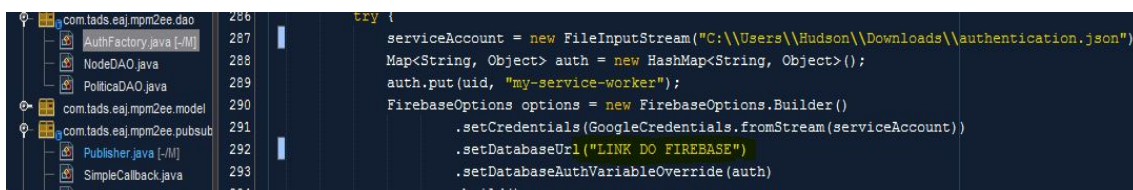
Após feita esta alteração, também é necessário referenciar o link do modo Realtime Database do seu projeto, pra isso na aba do projeto acesse **Database**, copie e cole o link na linha 292 no projeto do serviço.

**Figura 18 - Aba do Database Firebase**



**Figura 19 - Método authenticateWithPrivileges() linha 292**

Feito isso, você já pode testar, basta adicionar sua aplicação a um servidor web e depois rodar a aplicação, se tudo der certo podemos utilizar a URL localhost:8080/mpm2ee/esp para acessar os métodos do recurso “esp”.



## 5 - Http Methods

Quando estamos usando um serviço web, estamos nada mais do que fazendo solicitações Http, protocolo que define um conjunto de **métodos de requisição** responsáveis por indicar a ação a ser executada para um dado recurso. Embora esses métodos possam ser descritos como substantivos, eles também são comumente referenciados como **HTTP Verbs (Verbos HTTP)**. Cada um deles implementa uma semântica diferente, mas alguns recursos são compartilhados por um grupo deles, como por exemplo, qualquer método de requisição pode ser do tipo safe, idempotent ou cacheable (MDN Web Docs, developer.mozilla.org/pt-BR/docs/Web/HTTP/Methods).

**GET:** O método GET solicita a representação de um recurso específico. Requisições utilizando o método GET devem retornar apenas dados.

**POST:** O método POST é utilizado para submeter uma entidade a um recurso específico, frequentemente causando uma mudança no estado do recurso ou efeitos colaterais no servidor.

**PUT:** O método PUT substitui todas as atuais representações do recurso de destino pela carga de dados da requisição.

**DELETE:** O método DELETE remove um recurso específico.

Existem mais alguns outros, porém não daremos foco a eles, já que não foram utilizados no projeto.

No projeto nosso serviço contém dois recursos, sendo eles, “esp” e “politica”, ambos contém suas próprias implementações dos métodos Http.

Para mostrar o resultado de cada tipo de chamada para os recursos existentes,, vamos utilizar o software Postman (<https://www.getpostman.com/>), para que possamos testar cada método e suas alterações e retornos.

#### 4.1 Serviço Web (Recurso esp)

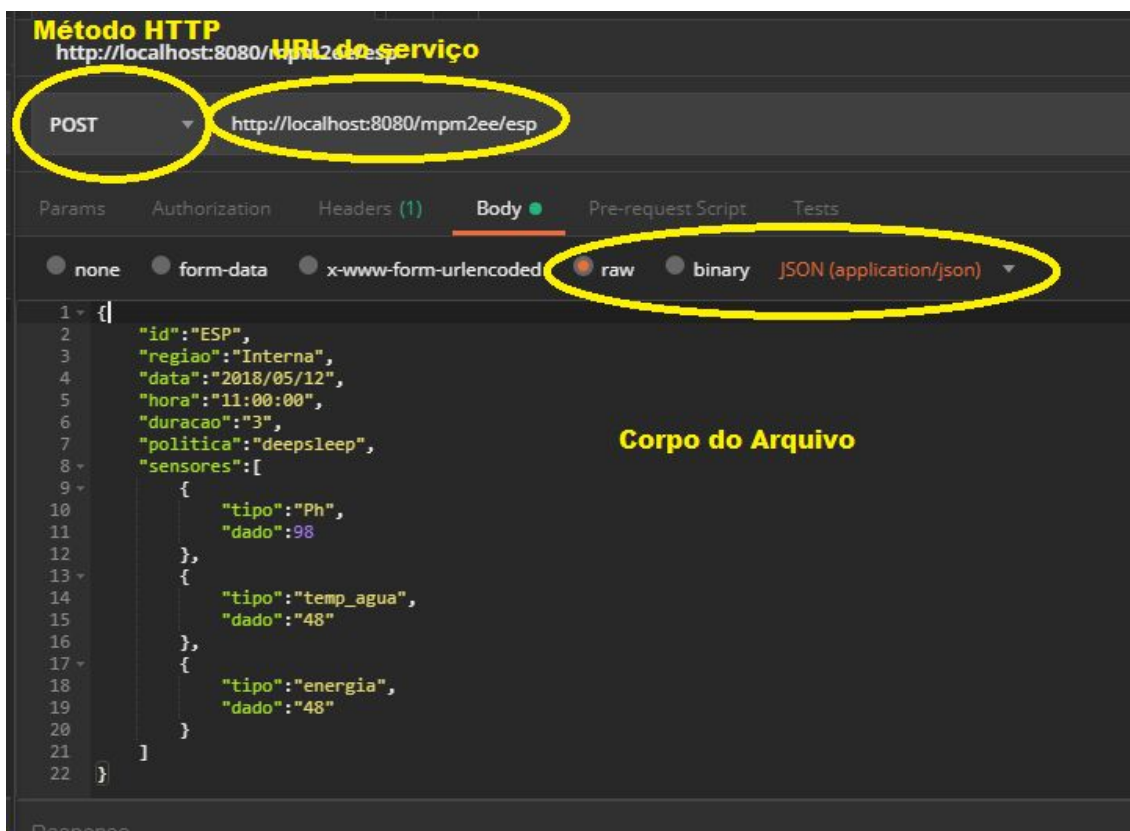
Método POST, bom primeiro vamos postar uma informação ao nosso banco de dados, com o seu serviço já em execução, vamos utilizar o Postman para criar o corpo de um arquivo do tipo Json, que será nossa entrada com os dados que queremos postar no banco, consequentemente ele retorna também um Json com os dados que foram passados na requisição.

Todas as requisições do tipo POST deve obedecer a estrutura correta do arquivo Json que será dado como entrada, observe padrão na figura 20.

Figura 20 - Estrutura do arquivo Json de entrada

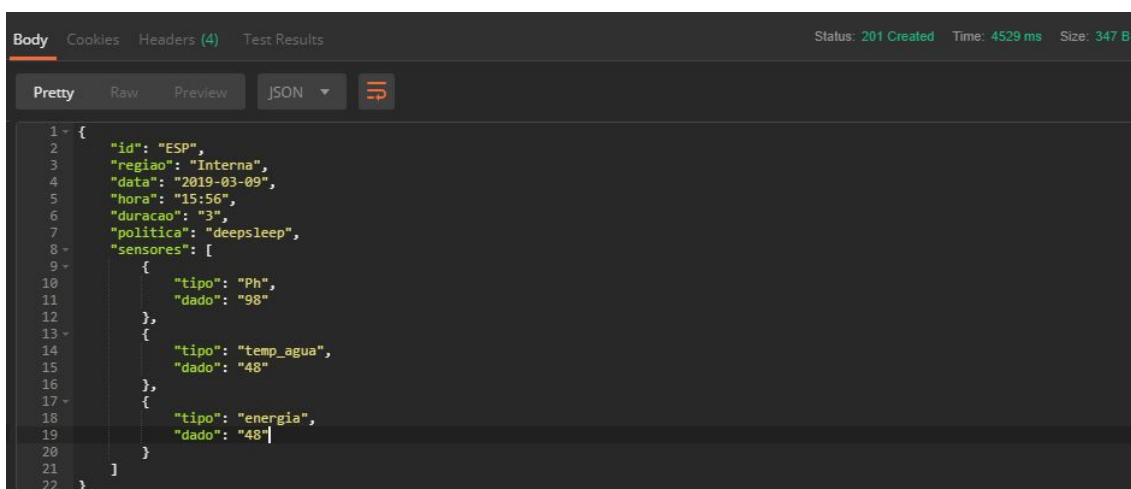
```
{
  "id": "idsensor",
  "regiao": "externo",
  "data": "10/10/2010",
  "duracao": "5",
  "politica": "deepsleep",
  "sensores": [
    {
      "Tipo": "Ph",
      "dado": "20"
    },
    {
      "Tipo": "temp_agua",
      "dado": "20"
    },
    {
      "Tipo": "energia",
      "dado": "20"
    }
  ]
}
```

Figura 20 - Configurando o postman para uma requisição do tipo POST



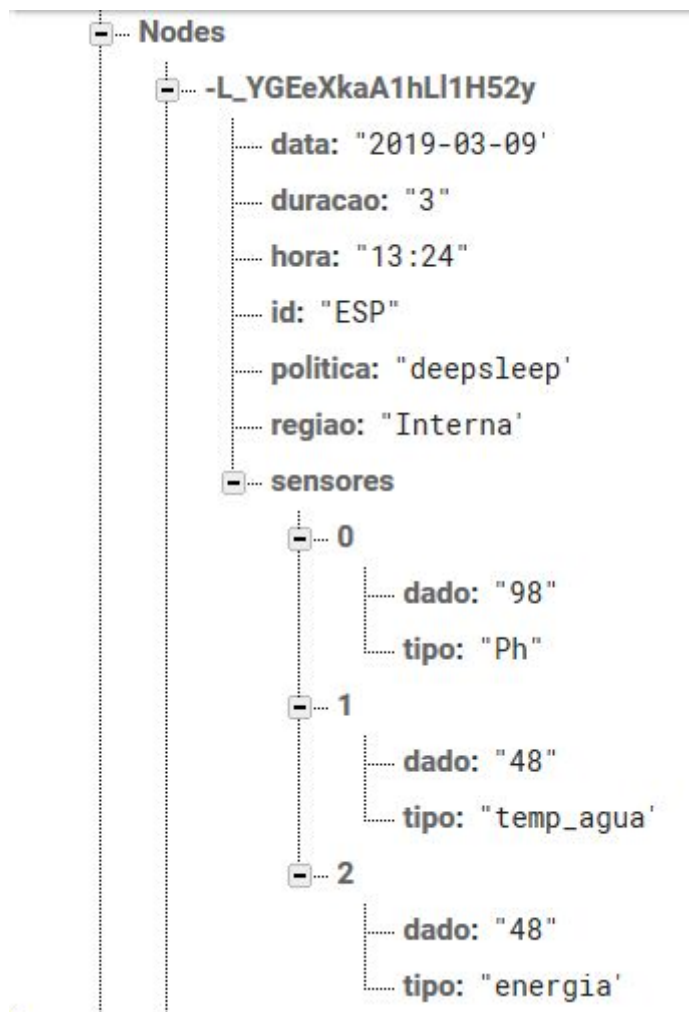
Após feito as configurações no Postman, clique no botão **Send** que fica a direita do campo da **URL do serviço**, a aplicação retorna para o usuário em formato Json as informações de entrada e a mensagem **201 Created**, para informar ao usuário que tudo ocorreu da maneira correta e que o arquivo foi salvo no Realtime Database do Firebase, caso contrário ele enviará a mensagem **500 Objeto Inválido**, a estrutura do arquivo não tenha sido atendida da forma correta.

Figura 21 -Retorno da requisição



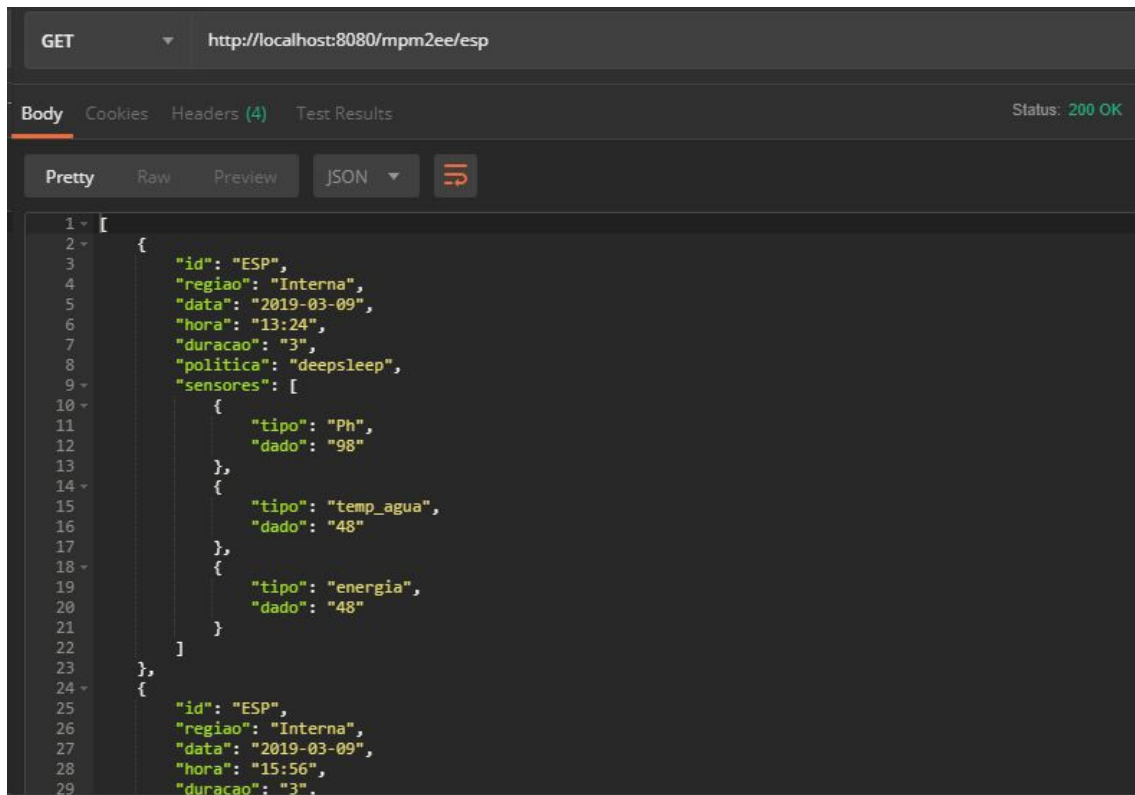
Você também pode conferir no firebase da aba Database se foi criado a raiz Node, dentro dela estarão contidos os objetos Json passados na solicitação.

Figura 22 - Estrutura da raiz no Firebase



Método GET, Funciona de forma bem simples, basicamente a resposta do serviço a esta solicitação é mostrar uma lista com todos os objetos dentro da Raiz Nodes que estão no firebase, retorno no corpo as informações em formato Json e a **200 OK**, caso contrário pode retornar a mensagem **500 Nenhum Objeto Encontrado**.

Figura 23 - Retorno de uma requisição do tipo GET



```
GET http://localhost:8080/mpm2ee/esp

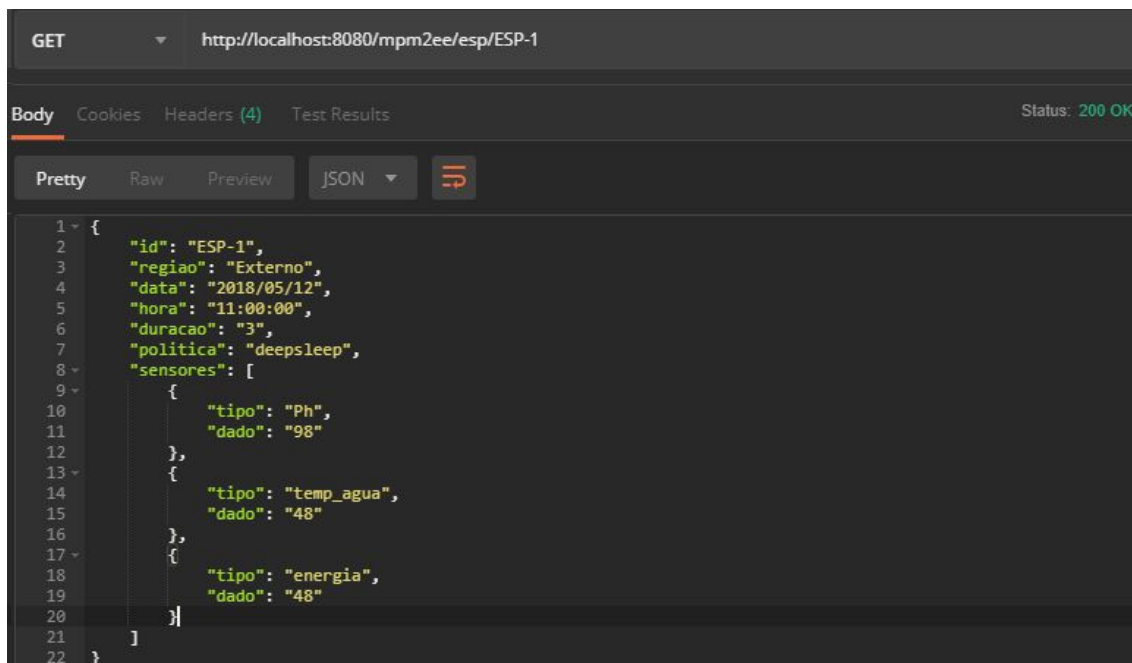
Body Cookies Headers (4) Test Results Status: 200 OK

Pretty Raw Preview JSON

1 [
2   {
3     "id": "ESP",
4     "regiao": "Interna",
5     "data": "2019-03-09",
6     "hora": "13:24",
7     "duracao": "3",
8     "politica": "deepsleep",
9     "sensores": [
10      {
11        "tipo": "Ph",
12        "dado": "98"
13      },
14      {
15        "tipo": "temp_agua",
16        "dado": "48"
17      },
18      {
19        "tipo": "energia",
20        "dado": "48"
21      }
22    ]
23  },
24  {
25    "id": "ESP",
26    "regiao": "Interna",
27    "data": "2019-03-09",
28    "hora": "15:56",
29    "duracao": "3",
```

A também como fazer a mesma solicitação para um objeto específico do bando com a utilização da identificação (Id) daquele objeto, basta que seja passado a URL <http://localhost:8080/mpm2ee/id>.

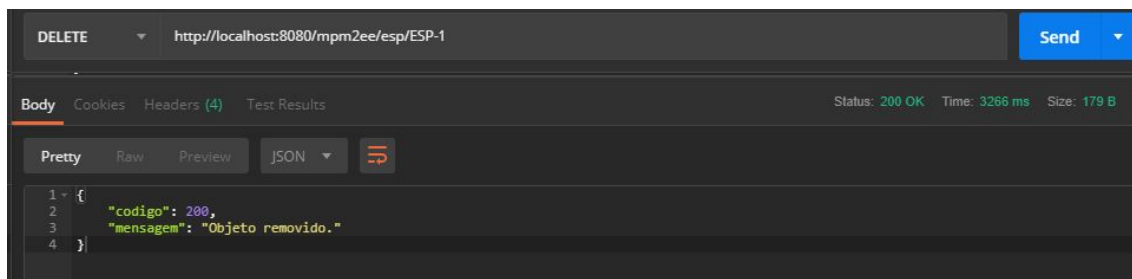
Figura 24 - Retorno de uma requisição GET por Id



É retornado as informações do objeto referente ao Id passado, a mensagem **200 Ok** sinaliza o sucesso da operação, caso contrário a mensagem **500 Id Inválido** é mostrada.

Método DELETE, criado para deletar um objeto do bando através de uma identificação (Id), agora teremos que passar o Id através da URL da requisição, vamos adicionar da seguinte forma: <http://localhost:8080/mpm2ee/esp/id>, onde o id determinará o objeto a ser removido do banco.

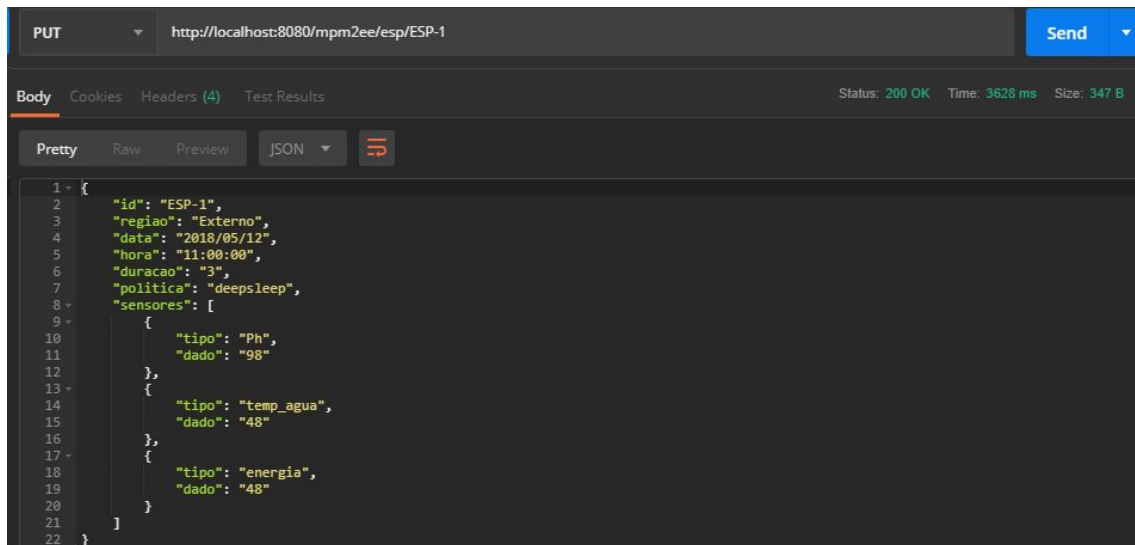
Figura 25 - Retorno de uma requisição DELETE



Retorno é um arquivo em formato Json com o código ou **200 OK** e que o objeto foi removido com sucesso, caso contrário ele retorna mensagem **500**.

Método PUT. da mesma que ocorreu no método DELETE teremos novamente a url com o “/id”, especificando qual o objeto que queremos fazer uma atualização, detalhe é que a estrutura do corpo da mensagem deve ser mantido, caso você mande apenas um campo os demais serão substituídos por valor nulo, no exemplo foi alterado somente o campo “região”, de interno para externo, os demais devem ser mantidos.

Figura 26 - Retorno de uma requisição do tipo PUT



The screenshot shows a REST client interface with a PUT request to `http://localhost:8080/mpm2ee/esp/ESP-1`. The response status is **200 OK**, with a time of 3628 ms and a size of 347 B. The response body is displayed in JSON format, showing a successful update of the resource.

```
1 {
2   "id": "ESP-1",
3   "regiao": "Externo",
4   "data": "2018/05/12",
5   "hora": "11:00:00",
6   "duracao": "3",
7   "politica": "deepsleep",
8   "sensores": [
9     {
10      "tipo": "Ph",
11      "dado": "98"
12     },
13     {
14      "tipo": "temp_agua",
15      "dado": "48"
16     },
17     {
18      "tipo": "energia",
19      "dado": "48"
20     }
21   ]
22 }
```

A mensagem **200 OK**, sinaliza o sucesso da operação, caso contrário a mensagem **500 Id Inválido** é mostrada.

## 4.2 - Serviço Web (Recurso política)

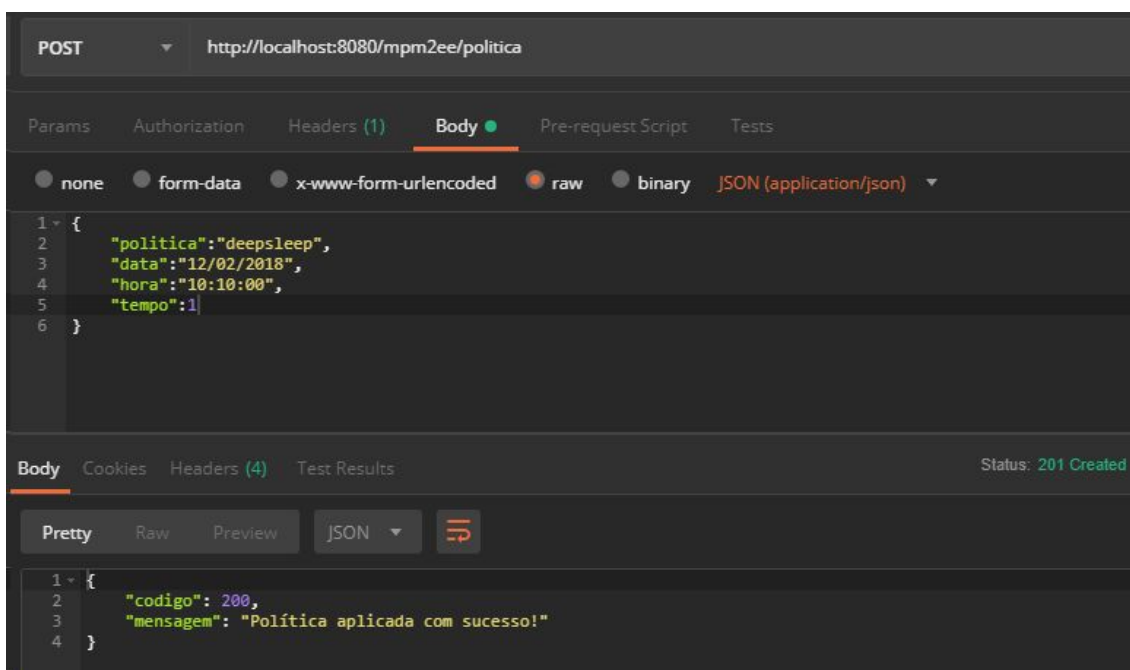
Método POST, assim como no serviço “esp“, vamos começar adicionando algo ao banco de dados com o método POST, também utilizando o Postman para auxiliar no teste., vamos criar novamente um corpo de mensagem no formato Json, que será a entrada com os dados que queremos postar no banco.

Vale lembrar que precisamos seguir o padrão adequado como estávamos criando o corpo do nossa entrada no formato Json, veja a imagem a seguir para evitar os erros.

Figura 27 - Estrutura do arquivo Json de entrada

```
1 {  
2   "politica": "deepsleep",  
3   "data": "12/02/2018",  
4   "hora": "10:10:00",  
5   "tempo": 1  
6 }
```

Figura 28 - Retorno de uma requisição do tipo POST

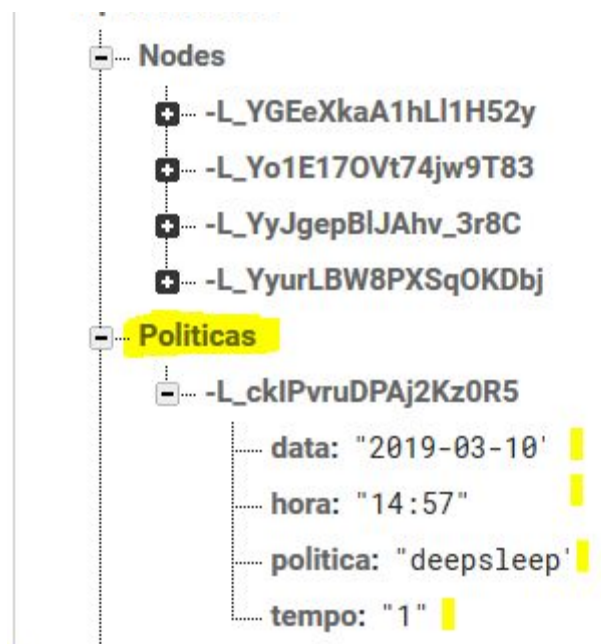


Feita a solicitação o serviço nos devolve um Json informando que a Política foi aplicada com sucesso, e a mensagem **200 Created**, que informa que tudo ocorreu bem e que nossa entrada foi postada. Vamos então verificar o nosso Realtime Database no Firebase para checar se a Raiz “Políticas” foi criada e se a nossa entrada foi salva.



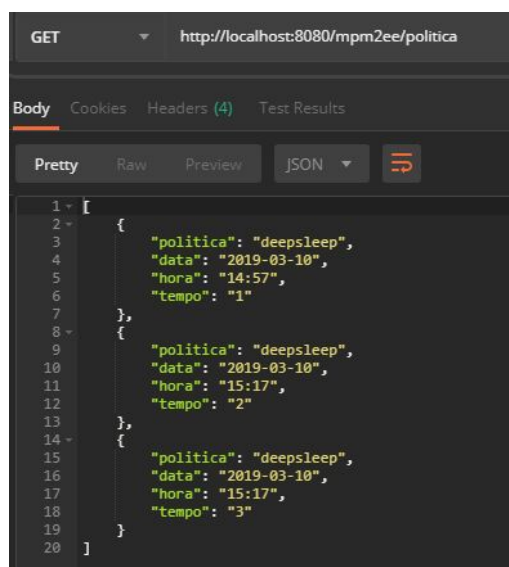
Figura 29 - Estrutura da raiz no Firebase

Método GET, assim como vimos no recurso “esp”, aqui o método GET funciona



da mesma maneira, ele faz uma solicitação ao serviço para que todos os objetos da raiz “Políticas” no firebase sejam listados, esses dados são retornados em formato do tipo Json, diferente do serviço “esp” onde tínhamos um poder de escolher através de uma identificação (Id) o objeto que seria retornado, aqui não temos essa escolha.

Figura 30 - Retorno de uma requisição do tipo GET



Retorno da solicitação, temos a lista de todas as políticas aplicadas e a mensagem **200 OK**, identificando que a solicitação da forma correta.