

AI & CHATBOT

Aula 08.1 – Técnicas de Integração com
Node-RED

Prof. Érick

Slides Adaptados do Prof.
Henrique Ferreira.

FIAP
GRADUAÇÃO

Mais truques com Node-RED

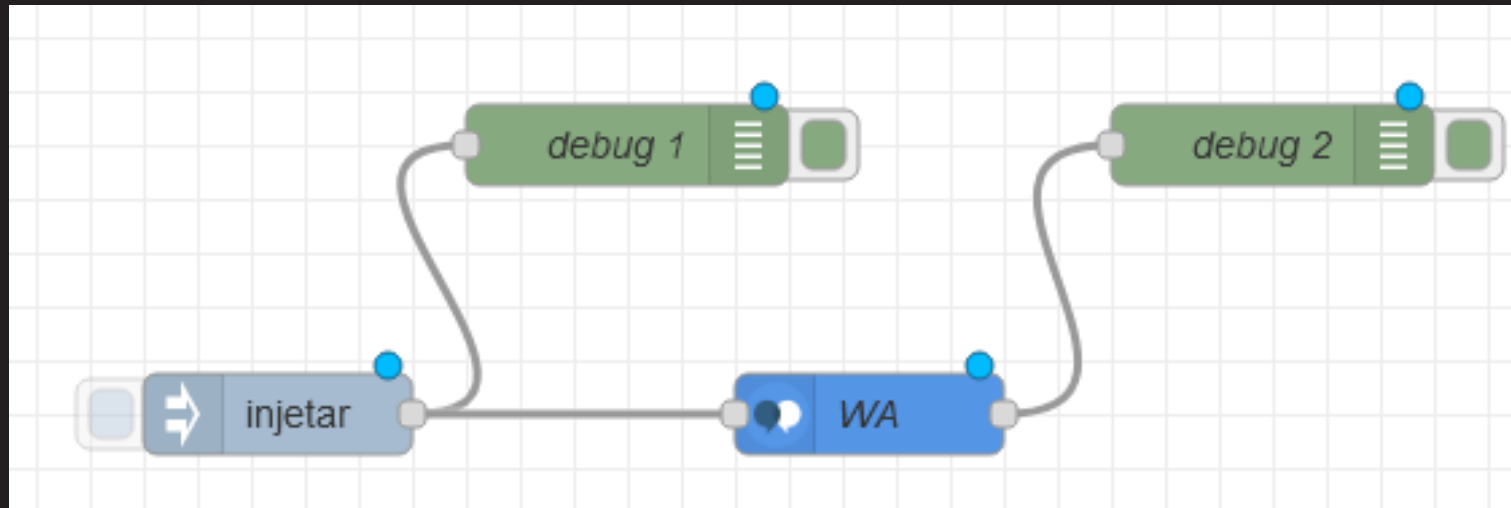
- Nestes slides apresentamos mais formas de usar o Node-RED para criar integrações com outros serviços;
- Cada exemplo explora um aspecto diferente de nós padrões e nós que podemos instalar para realizar integrações específicas;

Manipulando contexto no Watson assistant

Como enviar e receber informações adicionais do WA

Manipulando Contexto - Recebendo

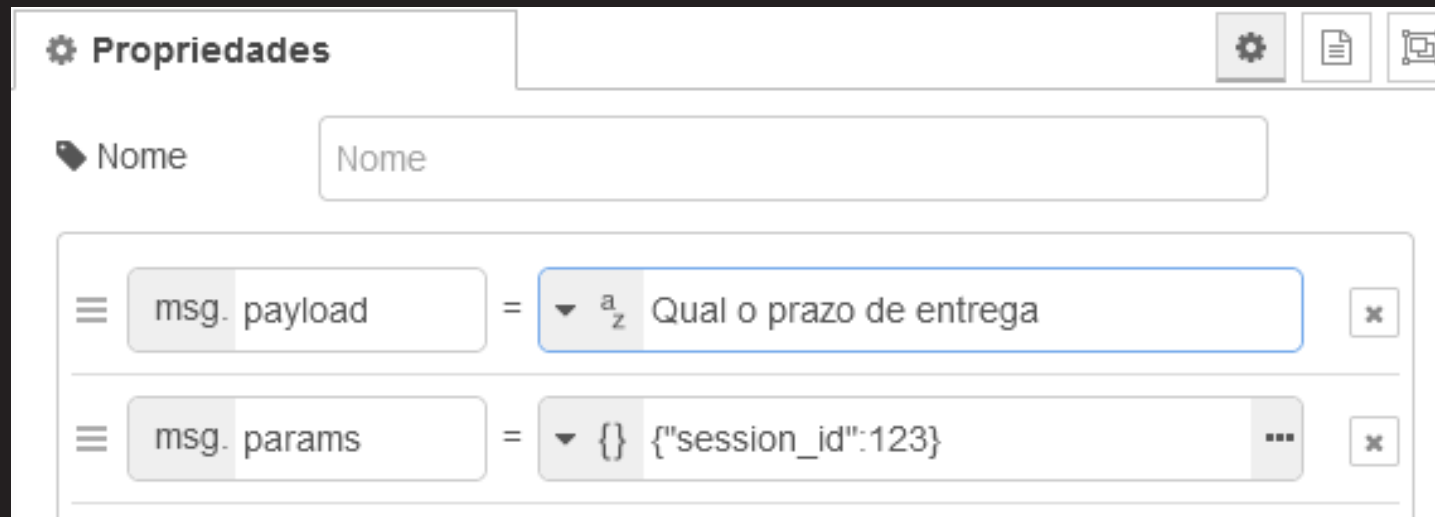
- Construa o seguinte fluxo:



- Configure o WA para acessar o bot de e-commerce que desenvolvemos nas outras aulas.

Manipulando Contexto - Recebendo

- No nó de inject, vamos enviar uma mensagem genérica. Configure a payload para ser a entrada do usuário e o params para ser um json com o session_id:



Manipulando Contexto - Recebendo

- A resposta será:

```
06/05/2024, 21:41:03  nó: debug 1
```

```
msg.payload : string[23]
```

```
"Qual o prazo de entrega"
```

```
06/05/2024, 21:41:04  nó: debug 2
```

```
msg.payload : Object
```

```
▼ object
```

```
▶ output: object
```

```
  user_id: "c1320a10-9de8-450b-9479-7b1e892c76bd"
```

```
▶ context: object
```

```
  session_id: "c1320a10-9de8-450b-9479-7b1e892c76bd"
```

Manipulando Contexto - Recebendo

- Vamos expandir o contexto o observar se algum contexto foi criado nessa interação:

```
▼ context: object
  ▼ global: object
    ▶ system: object
      session_id: "c1320a10-9de8-450b-9479-7b1e892c76bd"
  ▼ skills: object
    ▼ main skill: object
      ▶ system: object
```

- Estamos interessados no main skill onde deve aparecer os dados de criados pelas variáveis de contexto que criamos na Skill do WA

Manipulando Contexto - Recebendo

- Alteremos o nó de inject para criar uma interação que gera contexto:

Editar inject nó

Deletar Cancelar Feito

Propriedades

Nome inject

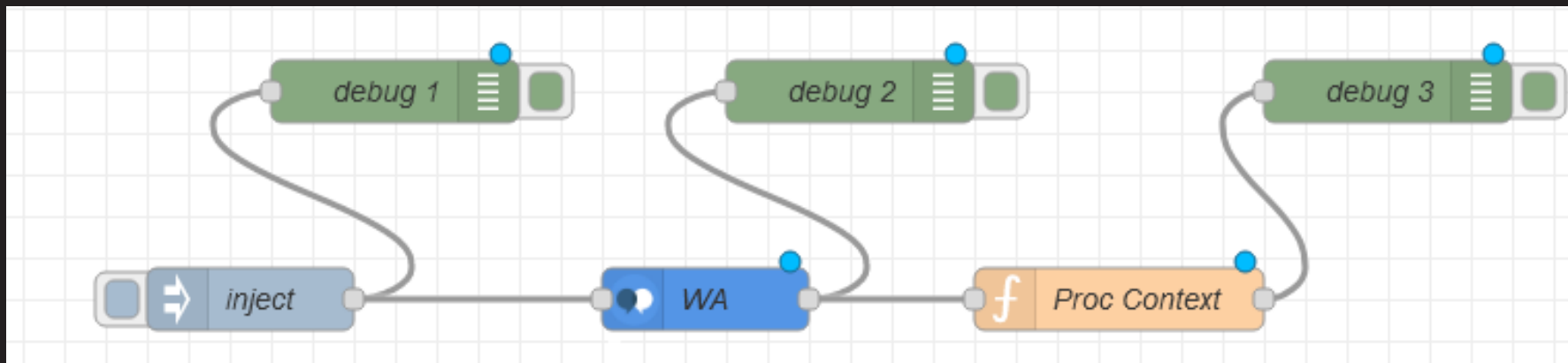
msg. payload = a_z Quero me cadastrar. Meu cep é 12345-123

msg. params = {} {"session_id":123}



Manipulando Contexto - Recebendo

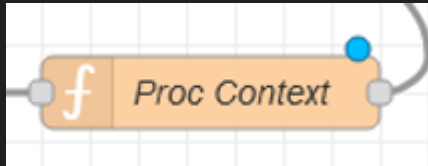
- Vamos criar uma function para processar o contexto:



- Como nem sempre os campos de contexto vem definidos (as variáveis ainda não foram criadas na interação dentro do WA), se tentarmos **acessar uma chave do JSON que não existe, teremos um erro**. Para isso não ocorrer iremos realizar uma **validação da existência** da chave com um pouco de **JavaScript**.

Manipulando Contexto - Recebendo

- Na função vamos escrever:



```
Nome Proc Context
Configurar No início Na mensagem Na parada

1 //Verifica se a chave user_defined existe no dicionario
2 if('user_defined' in msg.payload.context.skills['main skill']){
3
4     //Verifica se cep existe
5     if ('cep' in msg.payload.context.skills['main skill'].user_defined){
6         var cep=msg.payload.context.skills['main skill'].user_defined.cep;
7     }else{
8         var cep = null;
9     }
10
11     //Verifica se telefone existe
12     if ('telefone' in msg.payload.context.skills['main skill'].user_defined){
13         var telefone = msg.payload.context.skills['main skill'].user_defined.telefone;
14     }else{
15         var telefone = null;
16     }
17
18     //Veritica se email existe
19     if ('email' in msg.payload.context.skills['main skill'].user_defined){
20         var email = msg.payload.context.skills['main skill'].user_defined.email;
21     }else{
22         var email=null;
23     }
24 }
25
26 // Cria um novo dicionario com os dados de contexto
27 msg.payload = {
28     'cep':cep,
29     'telefone':telefone,
30     'email':email
31 }
```

Manipulando Contexto - Recebendo



```
//Verifica se a chave user_defined existe no dicionario
if('user_defined' in msg.payload.context.skills['main skill'])){

    //Verifica se cep existe
    if ('cep' in msg.payload.context.skills['main skill'].user_defined){
        var cep=msg.payload.context.skills['main skill'].user_defined.cep;
    }else{
        var cep = null;
    }

    //Verifica se telefone existe
    if ('telefone' in msg.payload.context.skills['main skill'].user_defined){
        var telefone = msg.payload.context.skills['main skill']. user_defined.telefone;
    }else{
        var telefone = null;
    }

    //Veritica se email existe
    if ('email' in msg.payload.context.skills['main skill'].user_defined){
        var email = msg.payload.context.skills['main skill'].user_defined.email;
    }else{
        var email=null;
    }
}
```

Manipulando Contexto - Recebendo



```
// Cria um novo dicionario com os dados de contexto
msg.payload = {
  'cep':cep,
  'telefone':telefone,
  'email':email
}
return msg;
```

- Essa função verifica se cada uma das chaves de dicionário de cadastro existem, atribuindo o valor que vem do contexto caso existirem ou um valor nulo, caso não existam.
- Vejamos o resultado:

Manipulando Contexto - Recebendo

Propriedades

Nome: inject

msg. payload = a_z Quero me cadastrar. Meu cep é 12344-122

msg. params = {} {"session_id":123}



06/05/2024, 21:59:56 nó: debug 1

msg.payload : string[39]

"Quero me cadastrar. Meu cep é 12344-122"

06/05/2024, 21:59:57 nó: debug 2

msg.payload : Object

```
▶ { output: object, user_id: "6e96d571-0b36-4377-a7a6-a59236...", context: object, session_id: "6e96d571-0b36-4377-a7a6-a59236..." }
```

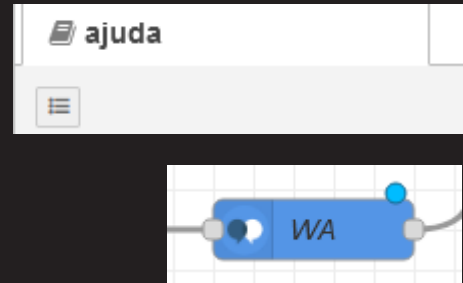
06/05/2024, 21:59:57 nó: debug 3

msg.payload : Object

```
▶ { cep: "12344-122", telefone: null, email: null }
```

Manipulando Contexto - Enviando

- Como já vimos em outras aulas, o `params.session_id` garante que o WA estará mantendo sessão de conversa com o mesmo usuário.
- Agora vamos explorar o `additional_context` para enviarmos uma variável de contexto adicional para o bot.



assistant V2

With the IBM Watson™ Assistant V2 service you can create cognitive agents – virtual agents that combine machine learning, natural language understanding, and integrated dialog scripting tools to provide outstanding customer engagements.

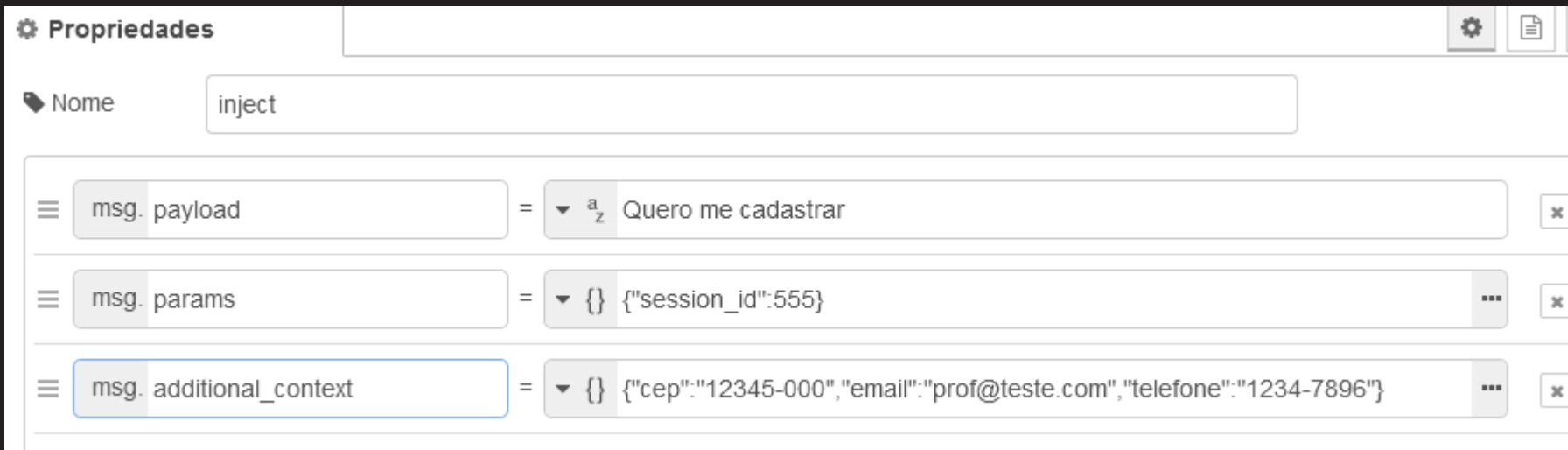
Usage

This node should be provided in input :

- `msg.payload` : the message of the Assistant to analyse. Format: String
- `msg.params.session_id` (optional): unique identifier for the conversation session. If this field is not provided then the node will generate a new `session_id` and return it as part of the response. If the node is not used in multi-session mode, then a `session_id` need not be provided, to reset the `session_id` in single-session mode send a null value as the `session_id`. Format: String
- `msg.params.reset_session` (optional): Will force a session reset Format: Any
- `msg.params.assistant_id` : unique identifier of the assistant to be used. Could be also configured in the node. Format: String
- `msg.params.timeout` (optional): The timeout period (in millisecond) for Watson request. Leave empty or set to 0 to disable.
- `msg.params.alternate_intents` (optional) : whether to return more than one intent. Default is false. Set to true to return all matching intents. For example, return all intents when the confidence is not high to allow users to choose their intent.
- `msg.params.entities` (optional) : see API documentation
- `msg.params.intents` (optional) : see API documentation
- `msg.params.return_context` (optional) : see API documentation
- `msg.params.debug` (optional) : see API documentation
- `msg.params.restart` (optional) : see API documentation
- `msg.additional_context` (optional) : additional properties that will be added to the context object. Format: Object
- `msg.params.username` : If provided will be used as the username credential for the Assistant service.

Manipulando Contexto - Enviando

- Agora que já aprendemos a manipular o **contexto recebido do WA**, vamos aprender a **como enviar contexto para o WA**
- Altere o nó de inject para:



- **msg.additional_context** (optional) : additional properties that will be added to the context object. Format: Object

```
1  {  
2    "cep": "12345-000",  
3    "email": "prof@teste.com",  
4    "telefone": "1234-7896"  
5  }
```

Manipulando Contexto - Enviando

06/05/2024, 22:05:46 nó: debug 1

msg.payload : string[18]

"Quero me cadastrar"

06/05/2024, 22:05:48 nó: debug 2

msg.payload : Object

```
▼ object
  ▼ output: object
    ▶ intents: array[1]
      entities: array[0]
    ▼ generic: array[1]
      ▼ 0: object
        response_type: "text"
        text: "Você poderia confirmar que seu CEP é 12345-000, o seu telefone é 1234-7896 e o seu email é prof@teste.com?"
  user_id: "8de0d55e-58a6-4b2f-a73d-429f003175dc"
  ▼ context: object
    ▶ global: object
    ▼ skills: object
      ▼ main skill: object
        ▶ user_defined: object
        ▶ system: object
  session_id: "8de0d55e-58a6-4b2f-a73d-429f003175dc"
```

06/05/2024, 22:05:48 nó: debug 3

msg.payload : Object

```
▶ { cep: "12345-000", telefone: "1234-7896", email: "prof@teste.com" }
```

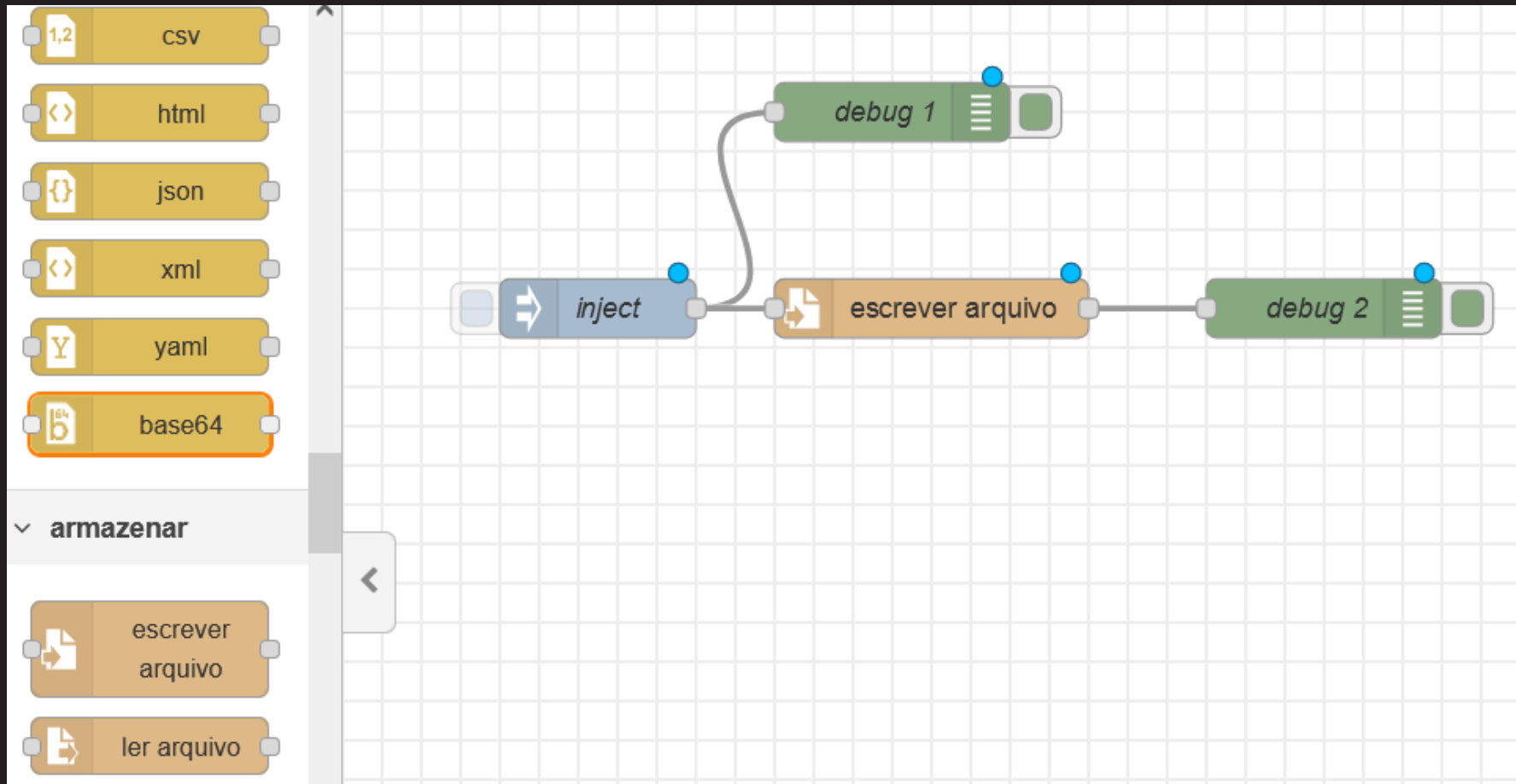
- Perceba que o WA já responde como se soubesse dos dados cadastrais mesmo sendo a primeira interação com esse session_id:

Salvando dados em um arquivo

Como salvar informações em um arquivo usando Node-RED

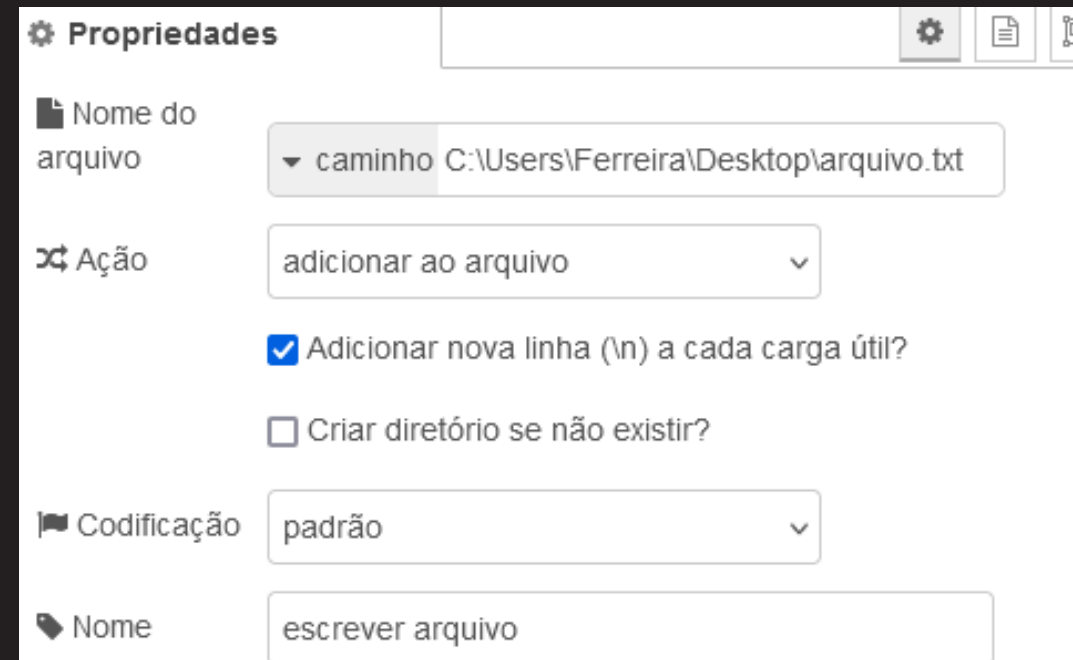
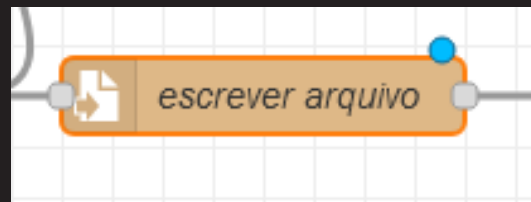
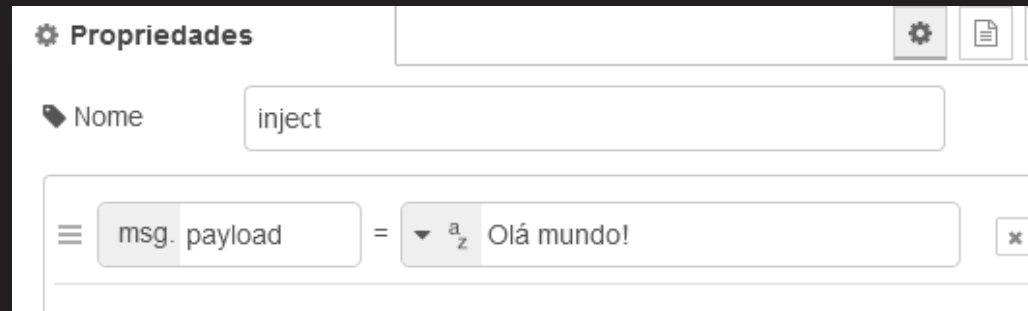
Salvando um arquivo

- Construa o seguinte fluxo:



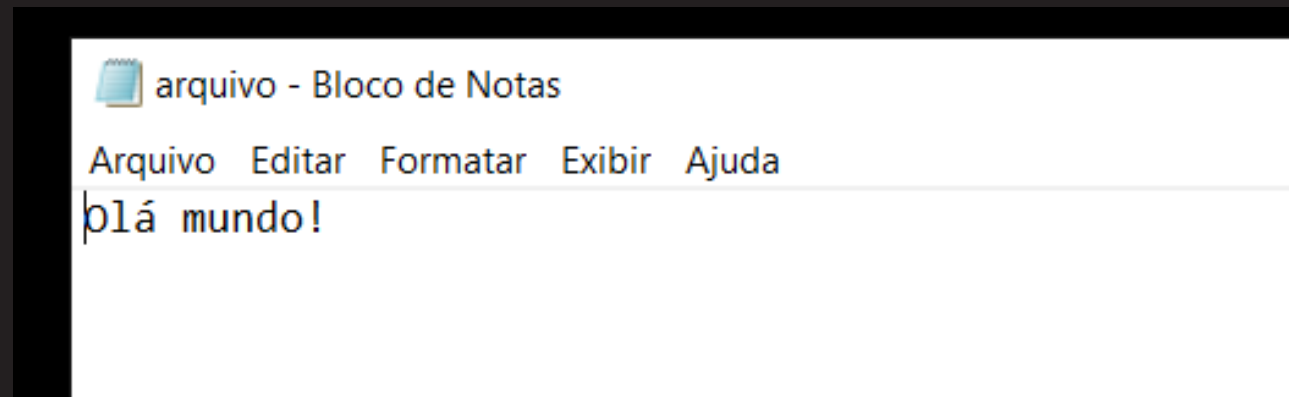
Salvando um arquivo

- As configurações dos nós são:



Salvando um arquivo

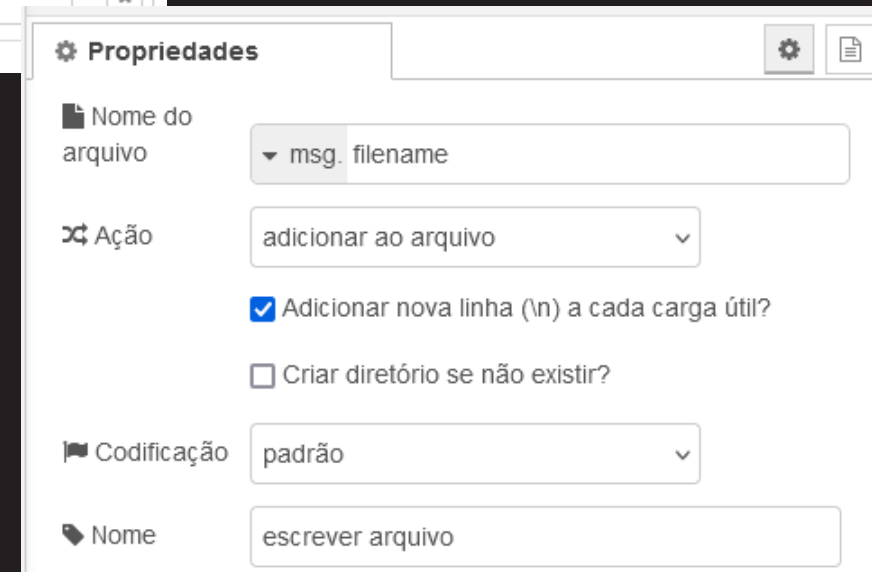
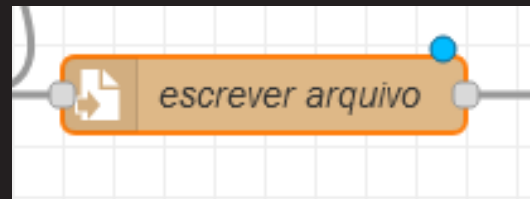
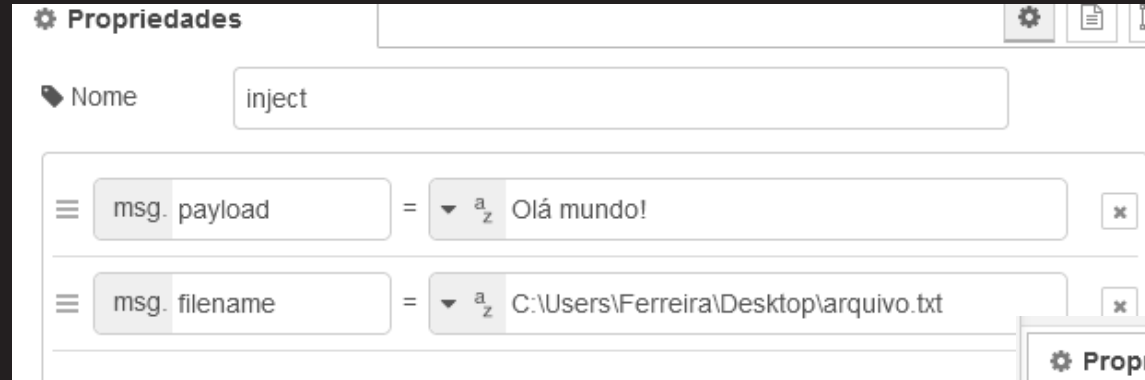
- Injete a mensagem e observe o arquivo criado no caminho (path) especificado no nó de escrever arquivo.



- O que acontece quando você clica em injetar de novo?

Salvando um arquivo

- Também podemos escrever dinamicamente o caminho do arquivo usando uma variável no nó de inject. Vejamos as novas configurações dos nós:

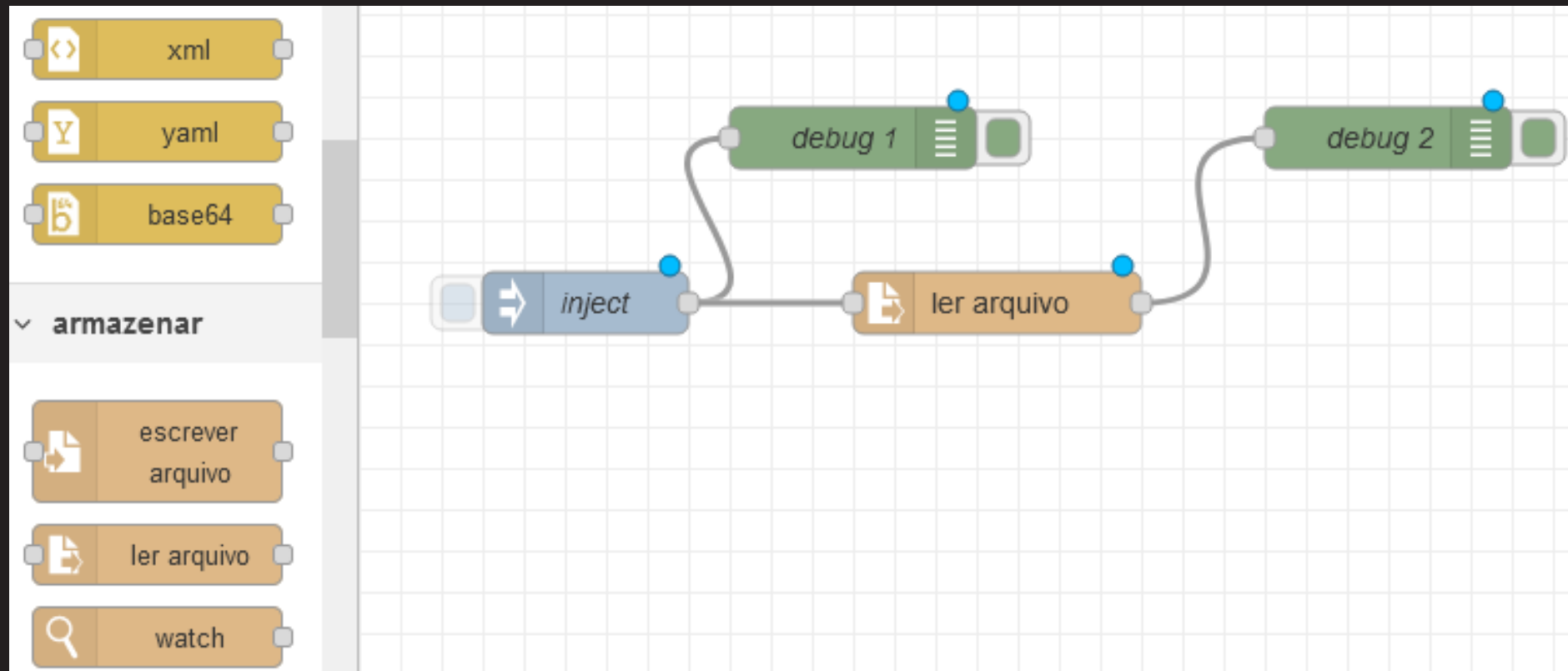


Lendo dados em um arquivo

Como carregar informações de um arquivo usando Node-RED

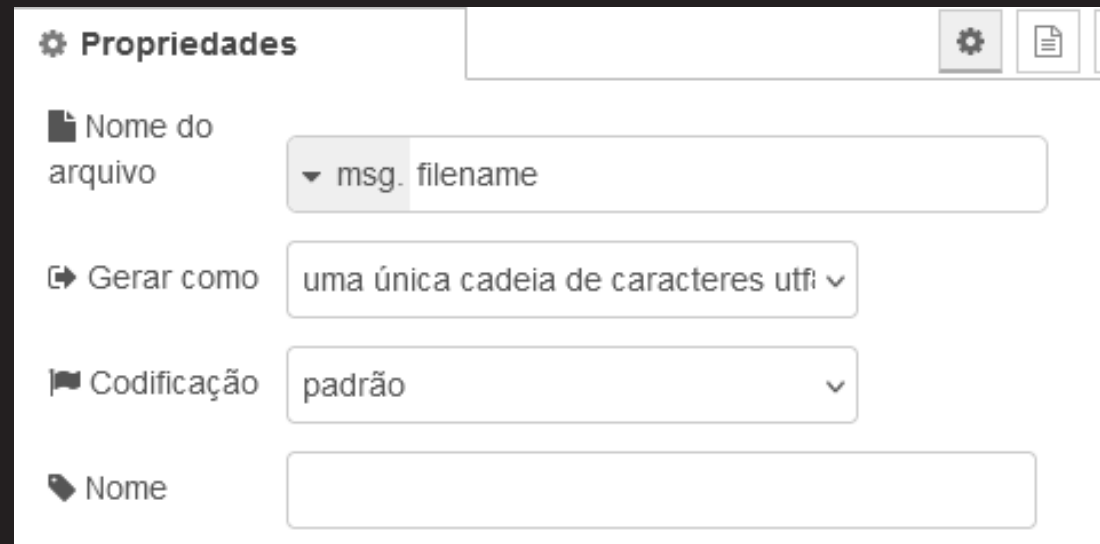
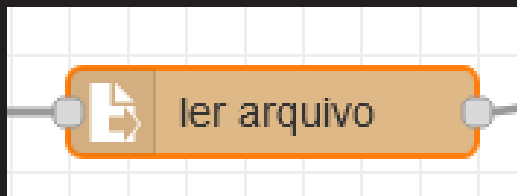
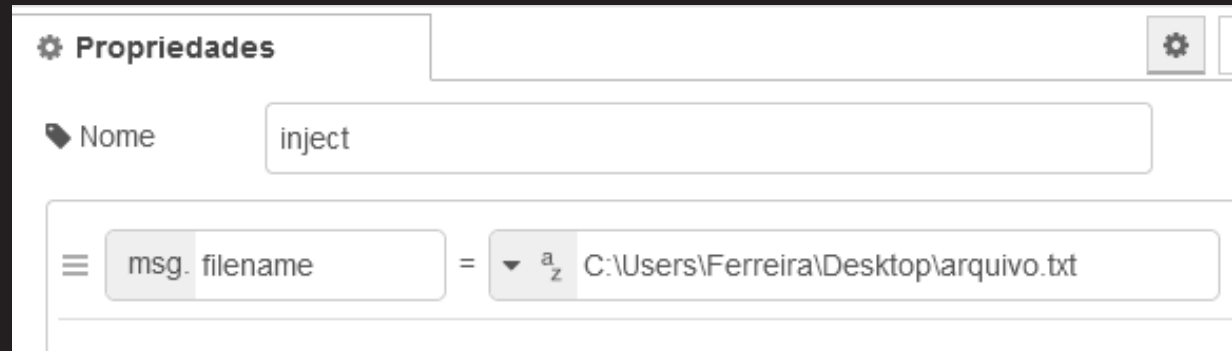
Lendo um arquivo .txt

- Construa o seguinte fluxo:



Lendo um arquivo .txt

- As configurações dos nós são:



Lendo um arquivo .txt

- Injete a mensagem e observe o resultado na janela de debug:



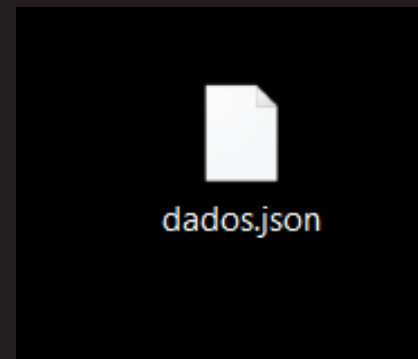
- Perceba que o primeiro debug mostra uma mensagem não definida já que não definimos o `msg.payload` no nó de inject. O segundo debug mostra o que está escrito no arquivo.

Lendo um arquivo .json

- Agora vamos tentar ler um arquivo .json usando os mesmos nós. Inicialmente vamos preparar nosso arquivo:

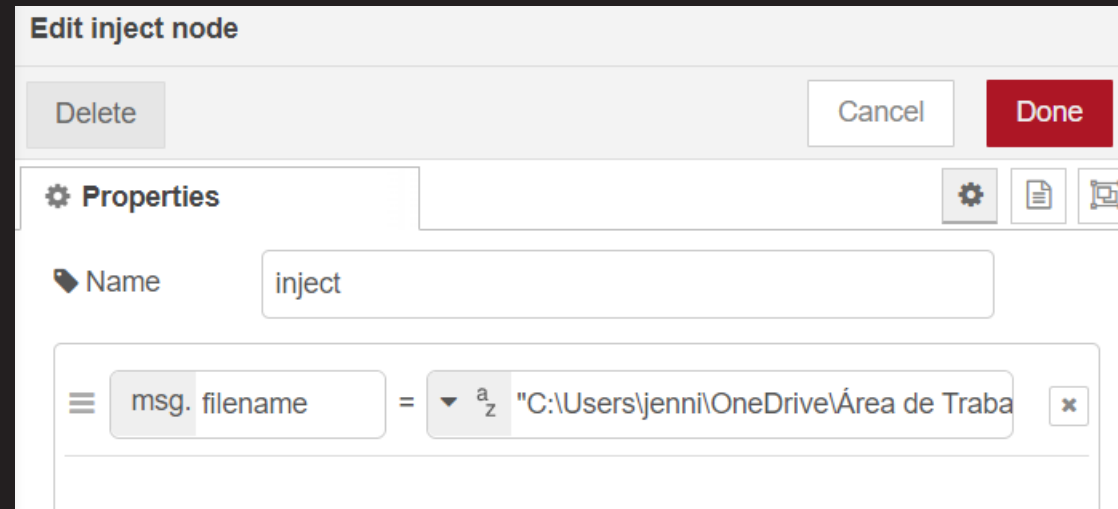
`{}` Dados.json > ...

```
1  [  
2      {  
3          "user": "prof Erick",  
4          "cep": "12345-123",  
5          "email": "prof@fiap.com.br",  
6          "telefone": "1234-1234"  
7      }  
8  ]
```



Lendo um arquivo .json

- Modificando o inject para ler o arquivo desejado:

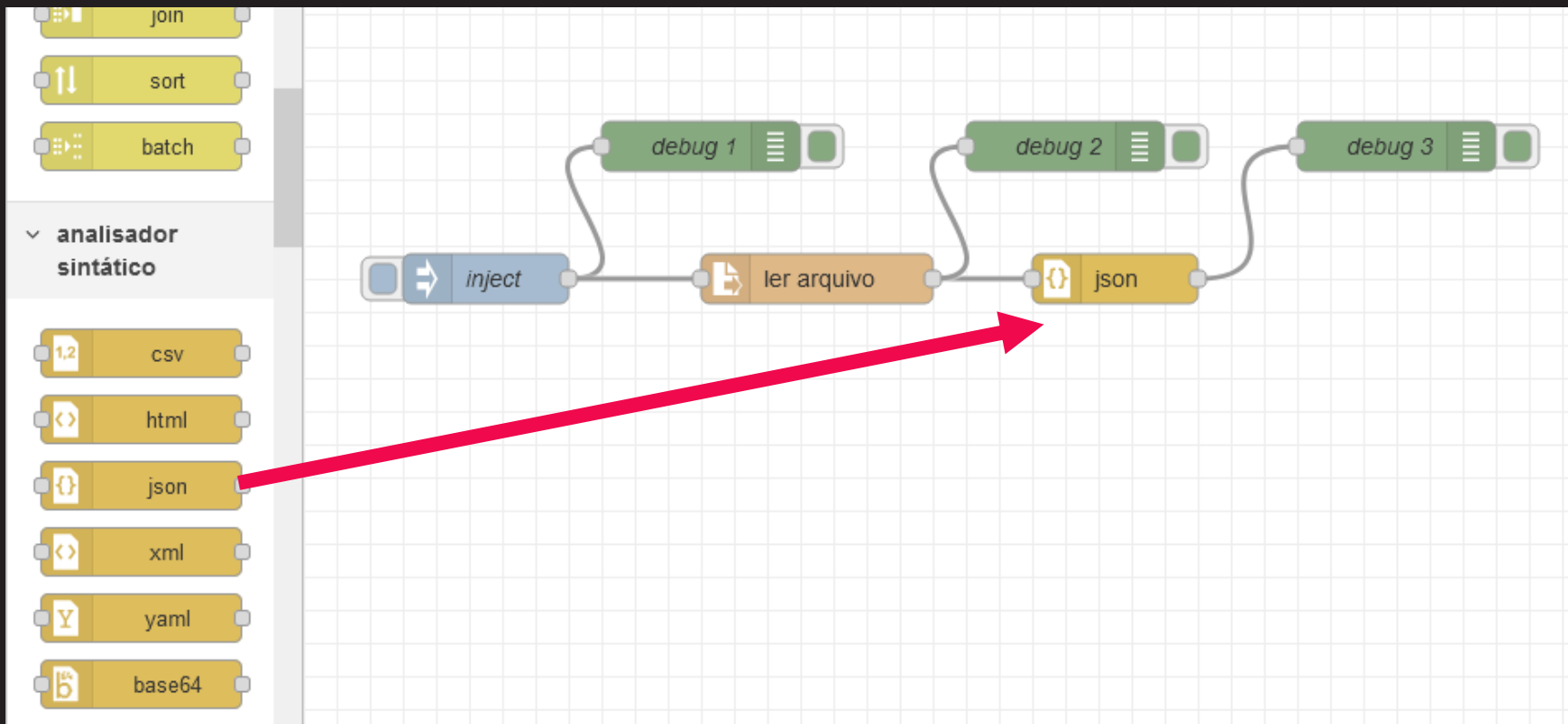


O resultado ao
injetar deve ser:

```
msg.payload : string[149]
▼ string[149]
[
  {
    "user": "prof Erick",
    "cep": "12345-123",
    "email": "prof@fiap.com.br",
    "telefone": "1234-1234"
  }
]
```

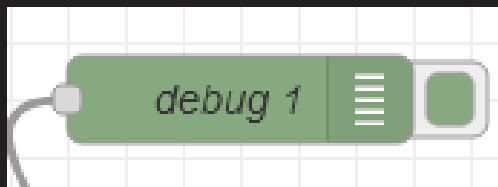
Lendo um arquivo .json

- Perceba entretanto que o arquivo foi lido e passado para uma variável do **tipo String**. Isso impede que manipulemos os campos do **dicionário JSON** de maneira programática. Vamos usar um PARSER para arrumar isso:



Lendo um arquivo .json

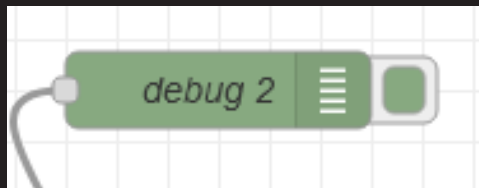
- Nenhuma configuração adicional precisa ser feita. Clique em injetar e observe o resultado no debug 3:



07/05/2024, 09:10:04 node: debug 1

msg.payload : undefined

undefined



07/05/2024, 09:10:04 node: debug 2

msg.payload : string[149]

```
▶ "[{"user": "prof  
Érick", "cep": "12345-123",  
"email": "prof@fiap.com.br",  
"telefone": "1234-1234"}]"
```



07/05/2024, 09:10:04 node: debug 3

msg.payload : array[1]

▼ array[1]

▼ 0: object

user: "prof Erick"

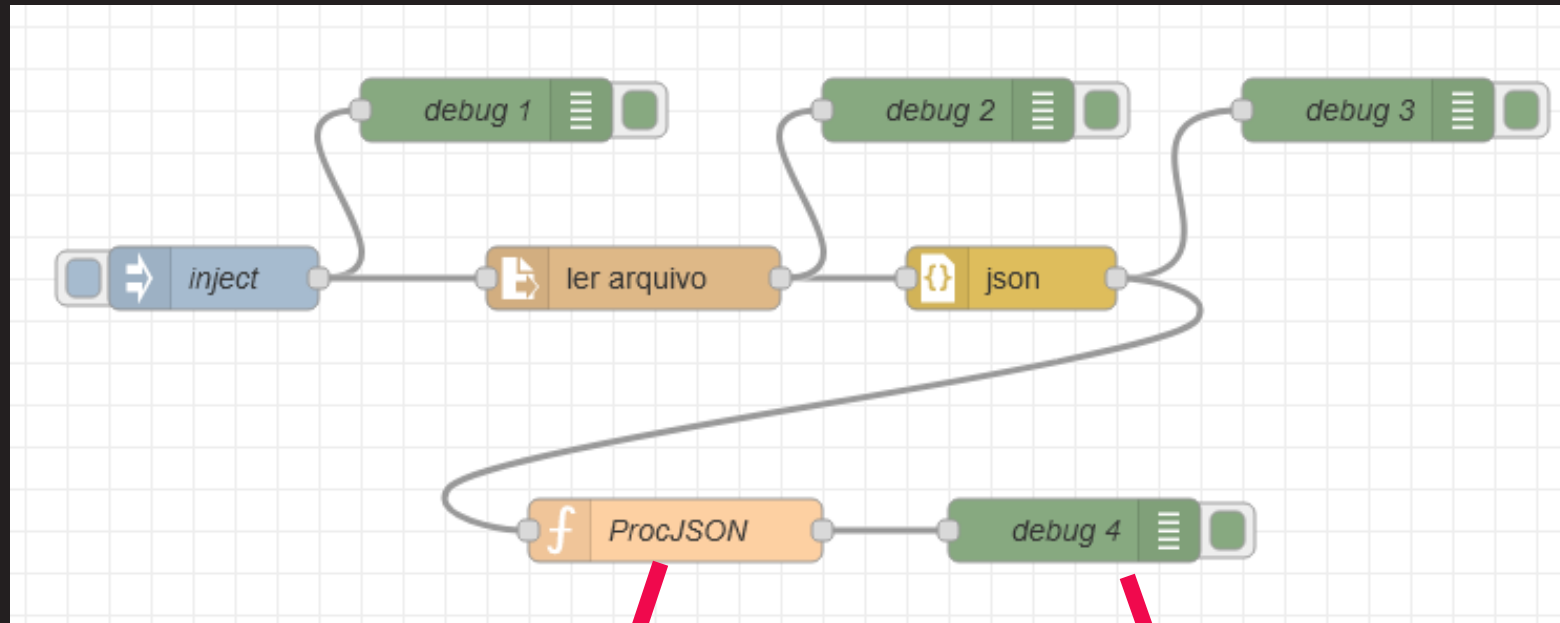
cep: "12345-123"

email: "prof@fiap.com.br"

telefone: "1234-1234"

Lendo um arquivo .json

- Agora podemos manipular os dados usando uma function:



Name ProcJSON

Setup On Start On Message On Stop

```
1 msg.payload = msg.payload[0].user;
2 return msg;
```

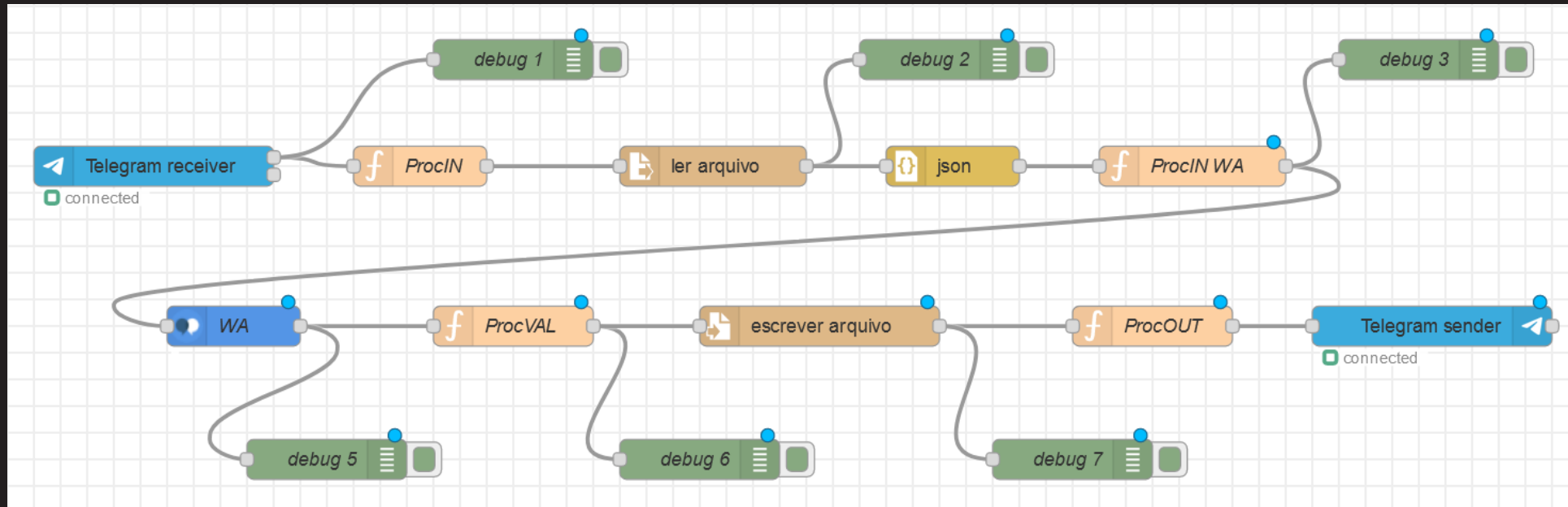
msg.payload : string[10]
"prof Erick"

Criando banco de dados rudimentar

Salvando e lendo dados de cadastro do bot e-commerce

Banco de dados rudimentar

Construa o seguinte fluxo:



Além de usar o chatId do Telegram como session_id do Watson Assistant, vamos usá-lo como chave de usuário no nosso banco de dados.

Banco de dados rudimentar

Configurando os nós:



Propriedades

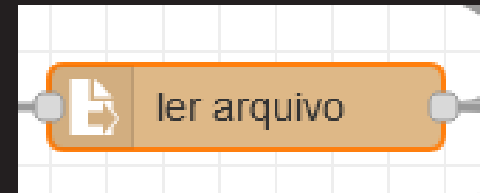
Nome: ProcIN


Configurar No início Na mensagem




```
1 msg.filename = "C:/Users/Ferreira/Desktop/base_dados.json";
2 msg.params={
3     "session_id":msg.payload.chatId
4 }
5 msg.entrada_usuario = msg.payload.content;
6 return msg;
```


Banco de dados rudimentar


Configurando os nós:





 **Propriedades**


  


 Nome do arquivo


 msg. filename

 Gerar como

uma única cadeia de caracteres utf: 

 Codificação

padrão 

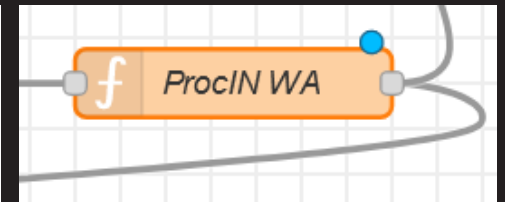
 Nome

Banco de dados rudimentar

Configurando os nós:

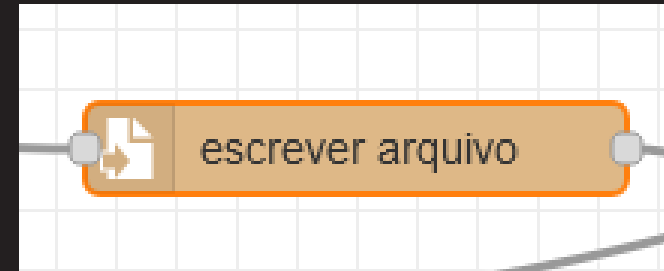
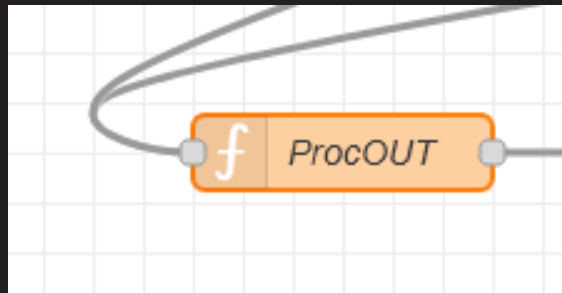
```
// Percorrendo a lista de dicionários carregada o arquivo JSON
if (msg.payload.length==0){
    msg.index_usuario = "sem cadastro";
}else{
    for (let i = 0; i < msg.payload.length; i++) {
        // Verificando se a chave 'user' é igual ao chatId/session_id atual
        if (msg.payload[i].user === msg.params.session_id) {
            msg.additional_context = {
                "cep":msg.payload[i].cep,
                "email":msg.payload[i].email,
                "telefone":msg.payload[i].telefone,
            }
            msg.index_usuario=i;
            break;
        }else{
            // Se não encontrar, então o usuário ainda não foi cadastrado
            msg.index_usuario = "sem cadastro";
        }
    }
}

node.warn(msg.index_usuario) // Print sem usar debug
msg.arquivo = msg.payload; // Salvando o arquivo lido em outra variável
msg.payload = msg.entrada_usuario;
return msg;
```



Banco de dados rudimentar

Configurando os nós:



Nome ProcOUT

Configurar No início

```
1 msg.payload = {
2   'chatId':msg.params.session_id,
3   'content':msg.saida_wa,
4   'type':"message"
5 }
6 return msg;
```

Propriedades

Nome do arquivo msg. filename

Ação sobrescrever file

☐ Adicionar nova linha (\n) a cada carga útil?

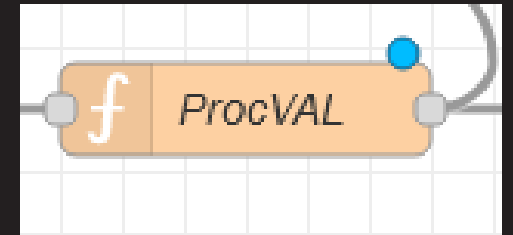
☐ Criar diretório se não existir?

Codificação padrão

Nome

Banco de dados rudimentar

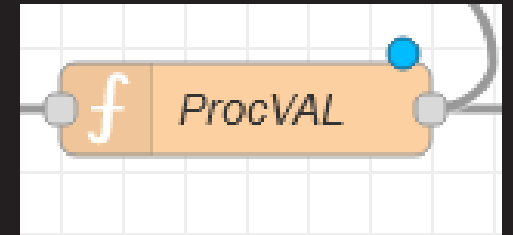
```
//Pegando as informacoes do contexto do WA
if ("user_defined" in msg.payload.context.skills["main skill"]) {
    var temp = msg.payload.context.skills["main skill"].user_defined;
    if("cep" in temp){
        var cep = temp.cep;
    }else{
        var cep = null;
    }
    if("telefone" in temp){
        var telefone = temp.telefone;
    }else{
        var telefone = null;
    }
    if("email" in temp){
        var email = temp.email;
    }else{
        var email = null;
    }
}
```



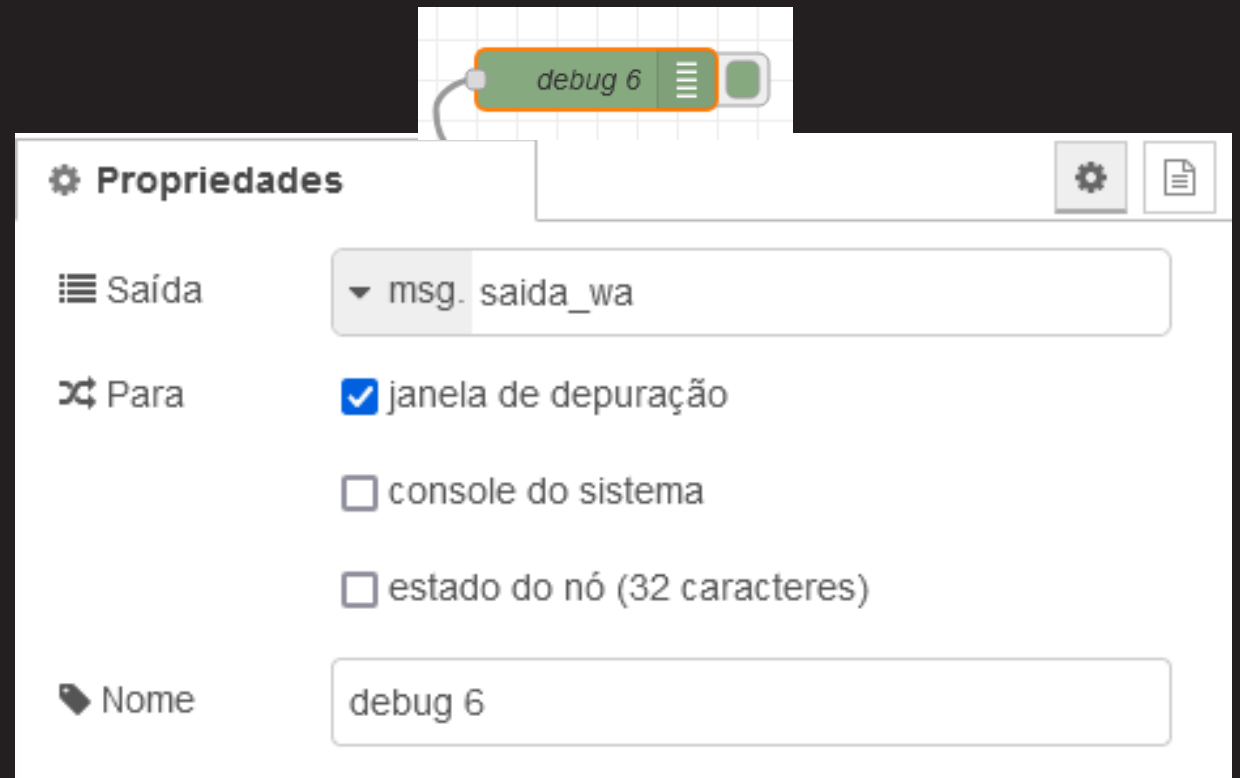
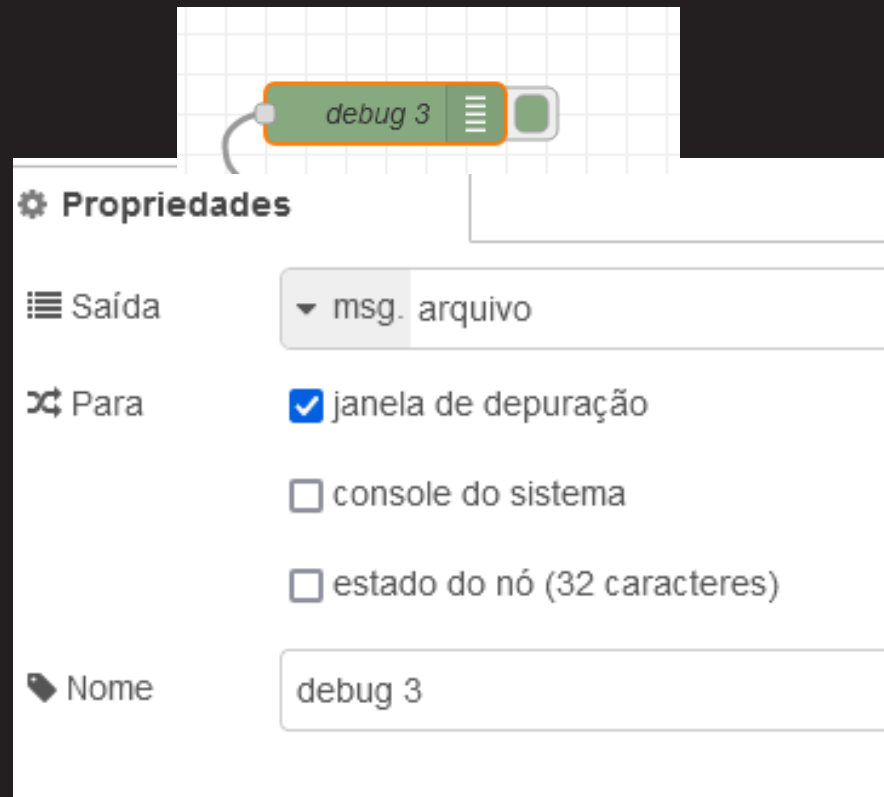
Banco de dados rudimentar

```
// Realizando o cadastro no arquivo base_dados
const novoObjeto = {
  'user':msg.params.session_id,
  'cep':cep,
  'email':email,
  'telefone':telefone
};

if(msg.index_usuario == "sem cadastro"){
  //Se o usuario nao tiver cadastro ainda
  msg.arquivo.push(novoObjeto)
}
else{
  //Se o usuario já estiver cadastrado
  msg.arquivo[msg.index_usuario]=novoObjeto
}
msg.saida_wa = msg.payload.output.generic[0].text;
msg.payload = msg.arquivo;
return msg;
```



Banco de dados rudimentar

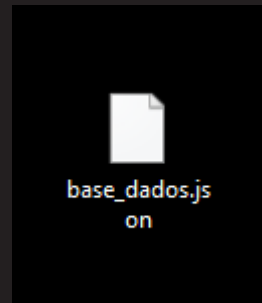
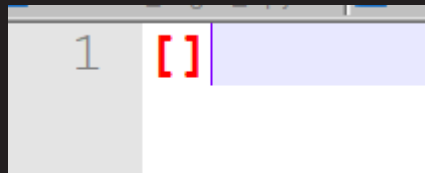


Todos os outros debugs são padrão msg.payload.

Banco de dados rudimentar

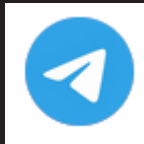
Agora crie um arquivo chamado `base_dados.json`

O arquivo deve conter uma lista vazia, isto é, abrir e fechar colchetes: `[]`

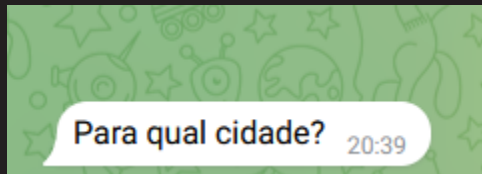
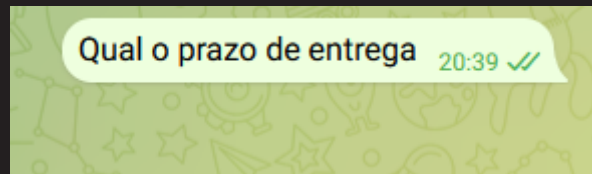


Converse com o bot via Telegram. Tente realizar o cadastro. Observe como o arquivo `base_dados.json` é alterado a cada nova interação.

Banco de dados rudimentar

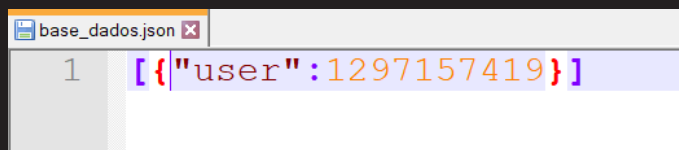


No Telegram:



base_dados.js
on

No arquivo JSON:



```
06/05/2024, 20:39:34  nó: debug 1
msg.payload : Object
  ▶ { chatId: 1297157419, messageId: 2494, type: "message", content:
    "Qual o prazo de entrega", date: 1715038772 }

06/05/2024, 20:39:35  nó: debug 2
msg.payload : string[2]
  "[]"

06/05/2024, 20:39:36  nó: ProcIN WA com cadastro
function : (warn)
  "sem cadastro"

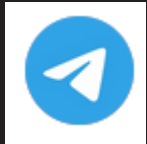
06/05/2024, 20:39:37  nó: debug 3
msg.arquivo : array[0]
  [ empty ]

06/05/2024, 20:39:38  nó: debug 5
msg.payload : Object
  ▶ { output: object, user_id: "9f1aa082-e247-4a4f-bebe-7a03a1...",
    context: object, session_id: "9f1aa082-e247-4a4f-bebe-7a03a1..." }

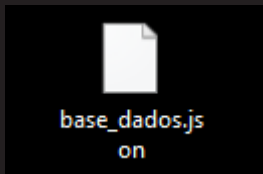
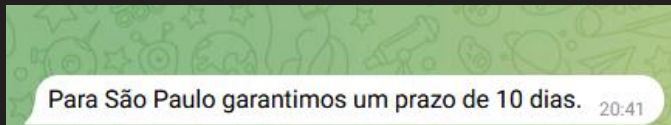
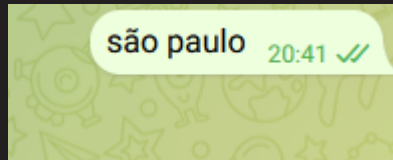
06/05/2024, 20:39:40  nó: debug 6
msg.saida_wa : string[17]
  "Para qual cidade?"

06/05/2024, 20:39:40  nó: debug 7
msg.payload : array[1]
  ▶ [ object ]
```

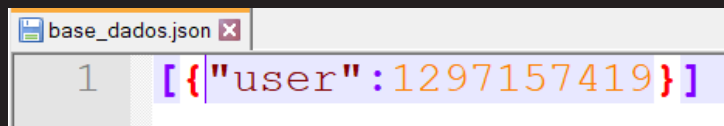
Banco de dados rudimentar



No Telegram:



No arquivo JSON:



06/05/2024, 20:41:49 nó: debug 1

msg.payload : Object

```
▶ { chatId: 1297157419, messageId: 2496, type: "message",  
  content: "são paulo", date: 1715038907 }
```

06/05/2024, 20:41:50 nó: debug 2

msg.payload : string[21]

```
"[{"user":1297157419}]"
```

06/05/2024, 20:41:51 nó: ProcIN WA com cadastro

function : (warn)

0

Perceba que agora
há entrada no banco

06/05/2024, 20:41:52 nó: debug 3

msg.arquivo : array[1]

```
▶ [ object ]
```

06/05/2024, 20:41:52 nó: debug 5

msg.payload : Object

```
▶ { output: object, user_id: "9f1aa082-e247-4a4f-bebe-7a03a1...",  
  context: object, session_id: "9f1aa082-e247-4a4f-bebe-7a03a1..." }
```

06/05/2024, 20:41:52 nó: debug 6

msg.saida_wa : string[46]

```
"Para São Paulo garantimos um prazo de 10 dias."
```

06/05/2024, 20:41:52 nó: debug 7

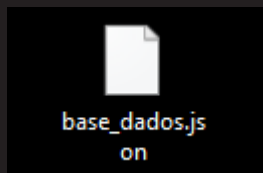
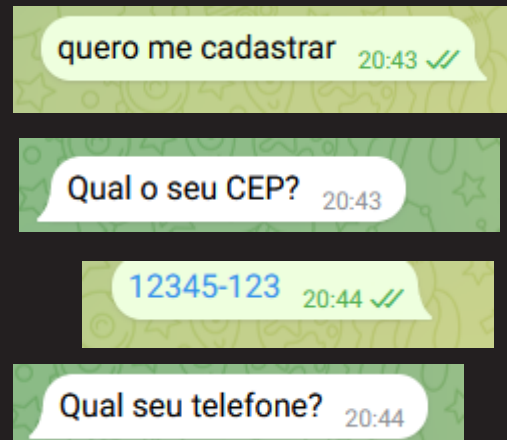
msg.payload : array[1]

```
▶ [ object ]
```

Banco de dados rudimentar



No Telegram:



No arquivo JSON:

```
base_dados.json x
1 [{"user":1297157419,"cep":"12345-123","email":null,"telefone":null}]
```

06/05/2024, 20:44:06 nó: ProclIN WA com cadastro

function : (warn)

0

06/05/2024, 20:44:07 nó: debug 3

msg.arquivo : array[1]

▶ [object]

06/05/2024, 20:44:08 nó: debug 5

msg.payload : Object

▶ { output: object, user_id: "9f1aa082-e247-4a4f-bebe-7a03a1...",
context: object, session_id: "9f1aa082-e247-4a4f-bebe-7a03a1..." }

06/05/2024, 20:44:09 nó: debug 6

msg.saida_wa : string[18]

"Qual seu telefone?"

06/05/2024, 20:44:10 nó: debug 7

msg.payload : array[1]

▼ array[1]

▼ 0: object

user: 1297157419

cep: "12345-123"

email: null

telefone: null

Banco de dados rudimentar



No Telegram:



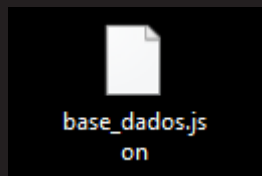
base_dados.js
on

No arquivo JSON:

```
1 [{"user":1297157419,  
2   "cep":"12345-123",  
3   "email":"prof@fiap.com",  
4   "telefone":"1234-1234"}]
```

Banco de dados rudimentar

Se outra pessoa interagir com seu bot, sua base de dados continuar a ser incrementada:



No arquivo JSON:

```
1 [{"user":1297157419,"cep":"12345-123","email":"prof@fiap.com","telefone":"1234-1234"},
2 {"user":1800298072,"cep":"23345-890","email":null,"telefone":null}]
```

```
06/05/2024, 20:51:35  nó: debug 1
msg.payload : Object
  ▶ { chatId: 1800298072, messageId: 2516, type: "message",
    content: "11967882345", date: 1715039493 }

06/05/2024, 20:51:35  nó: debug 2
msg.payload : string[153]
  "[{"user":1297157419,
  "cep":"12345-123","email":"prof@fiap.com","telefone":"1234-1234"
  },{"user":1800298072,
  "cep":"23345-890","email":null,"telefone":null}]"

06/05/2024, 20:51:35  nó: ProclN WA
function : (warn)
  1

06/05/2024, 20:51:35  nó: debug 3
msg.arquivo : array[2]
  ▶ [ object, object ]

06/05/2024, 20:51:36  nó: debug 5
msg.payload : Object
  ▶ { output: object, user_id: "5c87982c-aed3-40ce-a7e0-92c3b1...",
    context: object, session_id: "5c87982c-aed3-40ce-a7e0-92c3b1..." }

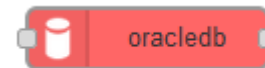
06/05/2024, 20:51:36  nó: debug 6
msg.saida_wa : string[18]
  "Qual seu telefone?"

06/05/2024, 20:51:36  nó: debug 7
msg.payload : array[2]
  ▶ [ object, object ]
```

Banco de dados reais

Neste exemplo estudamos como usar arquivo no ambiente local para emular um banco de dados. De fato, no mundo real é necessário usar aplicações mais robustas, como software de banco de dados dedicados. Você pode buscar nós adicionais de Node-RED para usar essas aplicações:

node-red-contrib-oracledb 0.5.1



Node-RED oracle database nodes

```
npm install node-red-contrib-oracledb
```

`node-red-contrib-oracledb` is a **Node-RED** package that connects directly to an Oracle database server. It currently contains a query and a configuration node to connect to Oracle databases for Node-RED storage.

It uses the `oracledb` library for the Oracle database connectivity.

<https://flows.nodered.org/node/node-red-contrib-oracledb>



Banco de dados reais

node-red-node-mysql 2.0.0

A Node-RED node to read and write to a MySQL database

```
npm install node-red-node-mysql
```

A **Node-RED** node to read and write to a MySQL database.

<https://flows.nodered.org/node/node-red-node-mysql>

Nodes

MySQLdatabase



Banco de dados reais

node-red-node-mongodb 0.2.5

Node-RED nodes to talk to a Mongo database

```
npm install node-red-node-mongodb
```

A **Node-RED** node to save data in a MongoDB database.

Note : This is the same node as was in the core of Node-RED. As of v0.10.8 you will need to install it from here if still required.

Pre-requisite

To run this you need a local MongoDB server running. For details see [the MongoDB site](#).

Nodes

mongodb

mongodb out

mongodb in

<https://flows.nodered.org/node/node-red-node-mongodb>



Banco de dados reais

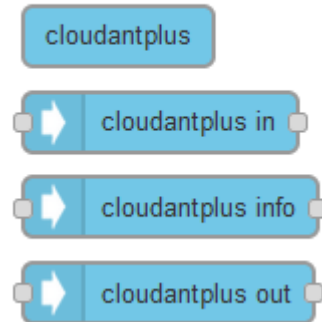
node-red-contrib-cloudantplus 2.0.5

A Node-RED node to access Cloudant and couchdb databases, supports views, query and bulk operations.

```
npm install node-red-contrib-cloudantplus
```

A set of **Node-RED** nodes to work with documents in a **Cloudant** database that is integrated with **IBM Cloud** or an on-premises **CouchDB**. This version is a superset of the functionality in the original Cloudant node and replicates the functionality of the **node-red-contrib-cloudantplus** node and will eventually replace it.

Nodes



<https://flows.nodered.org/node/node-red-contrib-cloudantplus>



Criando uma api http - GET

Como conectar o bot a um cliente HTTP

Configurando conexão HTTP GET

Adicione um nó de **http in**. Depois dê um duplo clique e preencha as propriedades do nó como abaixo. Também adicione um debug.


The screenshot displays the Node-RED web interface. On the left, the 'rede' (network) palette is expanded, showing various nodes. A red arrow points from the 'http in' node in the palette to an 'Entrada' node placed on the workspace grid. The 'Entrada' node is connected to a 'Print Entrada' node. A dialog box titled 'Edit http in node' is open, showing the configuration for the 'Entrada' node. The dialog has tabs for 'Properties' and 'Debug'. The 'Properties' tab is active, showing the following settings:


- Method: GET
- URL: /conversa
- Name: Entrada


The dialog also includes a 'Delete' button, 'Cancel' and 'Done' buttons at the top right, and icons for settings, documentation, and help at the bottom right.

Configurando conexão HTTP GET

Agora adicione um nó de function com nome de ProcIN WA para o Watson. Nas propriedades, digite o seguinte código:

 **Propriedades**

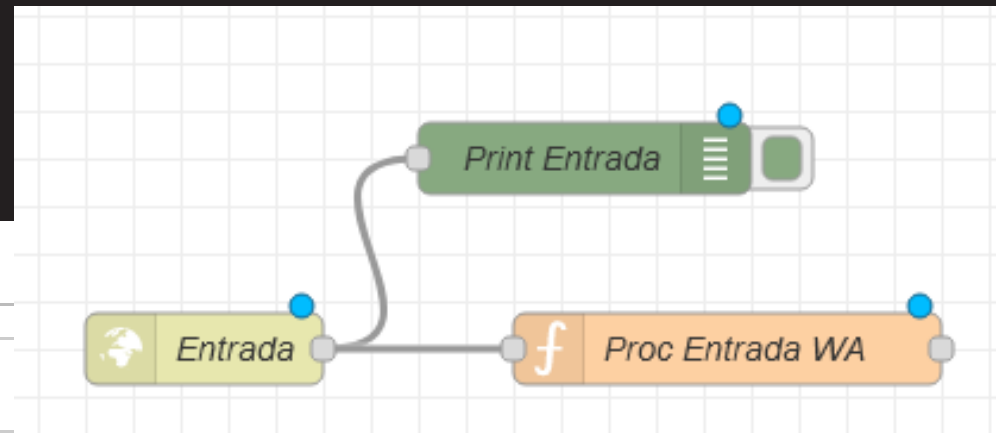
 Nome

 Configurar

No início

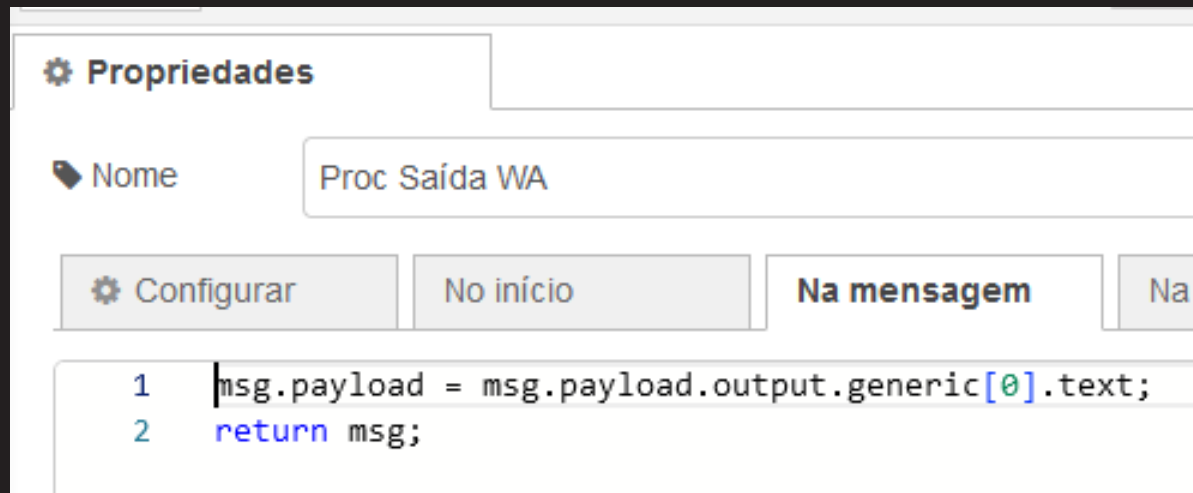
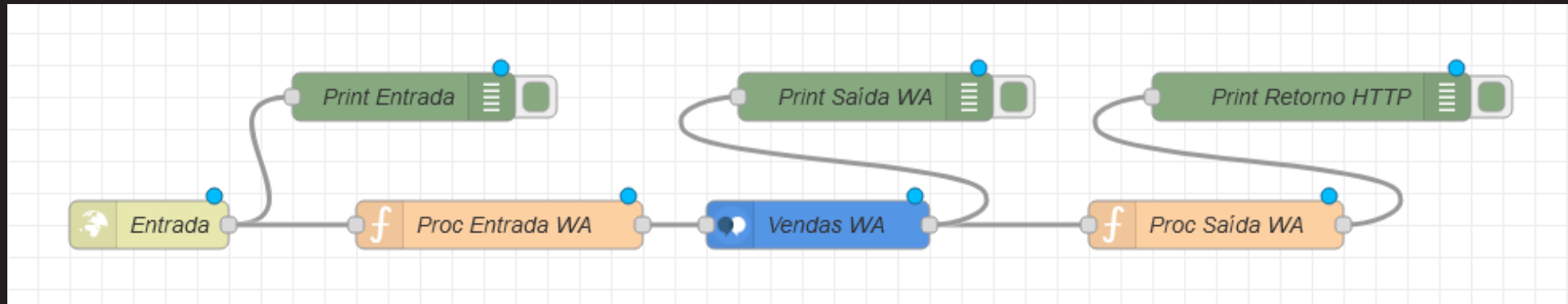
Na mensa

```
1  msg.params = {  
2    "session_id" : 123  
3  };  
4  msg.payload = msg.payload.mensagem;  
5  return msg;
```



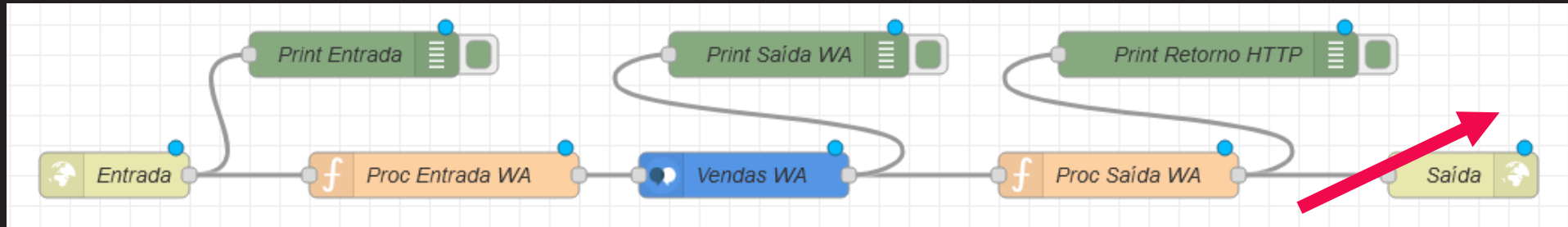
Configurando conexão HTTP GET

Adicione o nó do WA (assistant v2) e um nó de function para processar a saída:



Configurando conexão HTTP GET

Termine o fluxo adicionando um nó de HTTP response no final:



Preencha o
HTTP Response
adequadamente:

Editar http response nó

Deletar

Propriedades

Nome: Saída

Código de estado: msg.statusCode

Cabeçalhos

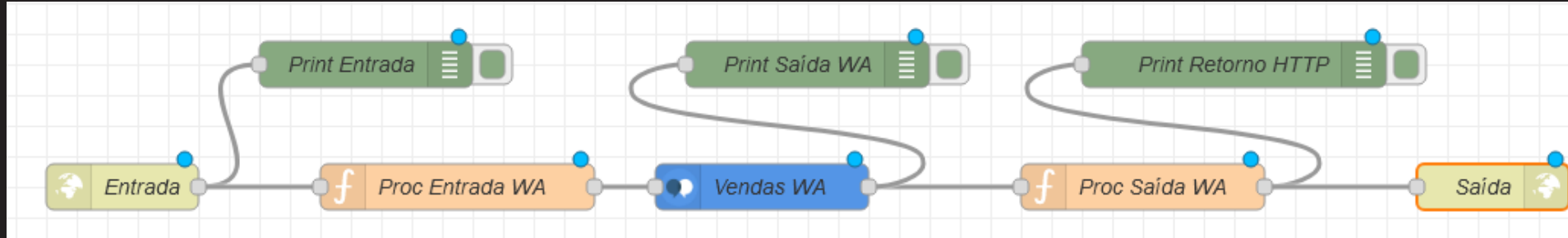
Access-Control-Allow-Origin	*
-----------------------------	---

Configurando conexão HTTP GET



- O cabeçalho "**Access-Control-Allow-Origin: ***" em uma resposta HTTP indica que o recurso sendo solicitado permite solicitações de qualquer origem.
- Quando um servidor envia "Access-Control-Allow-Origin: *" como parte da resposta HTTP, ele está essencialmente dizendo ao navegador que qualquer origem pode fazer solicitações para esse recurso específico. Isso é útil em situações em que você deseja permitir que clientes de diferentes domínios acessem seus recursos, como em uma API pública.

Configurando conexão HTTP GET



Para testar localmente, clique em implementar/deploy e digite no navegador o IP local, seguido da porta do Node-RED / o endereço do nó de entrada.

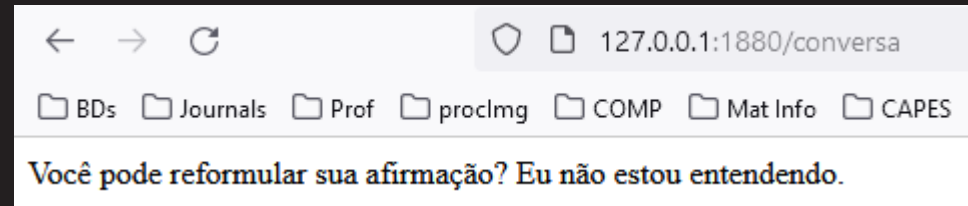


O que aconteceu?

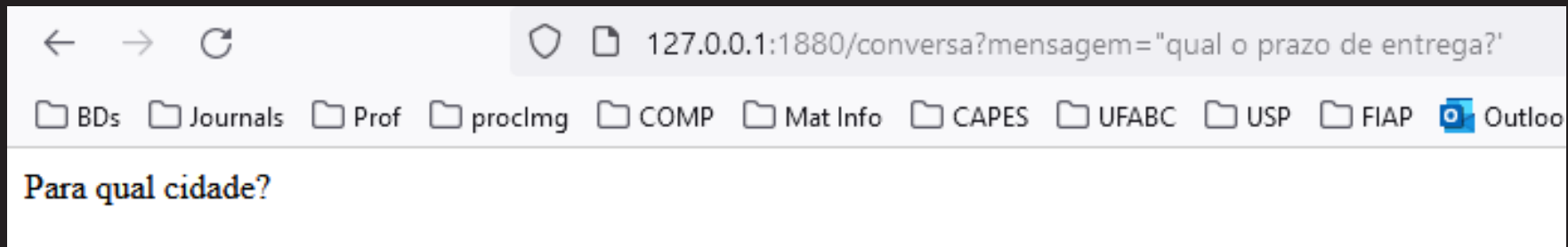
Configurando conexão HTTP GET

Não estamos passando nenhuma entrada no GET, então está indo uma entrada vazia para o WA!

Nossa function Proc Entrada WA estabelece uma variável específica para ir para o WA. Vamos usar ela como parâmetro do GET:



```
4 msg.payload = msg.payload.mensagem;  
5 return msg;
```



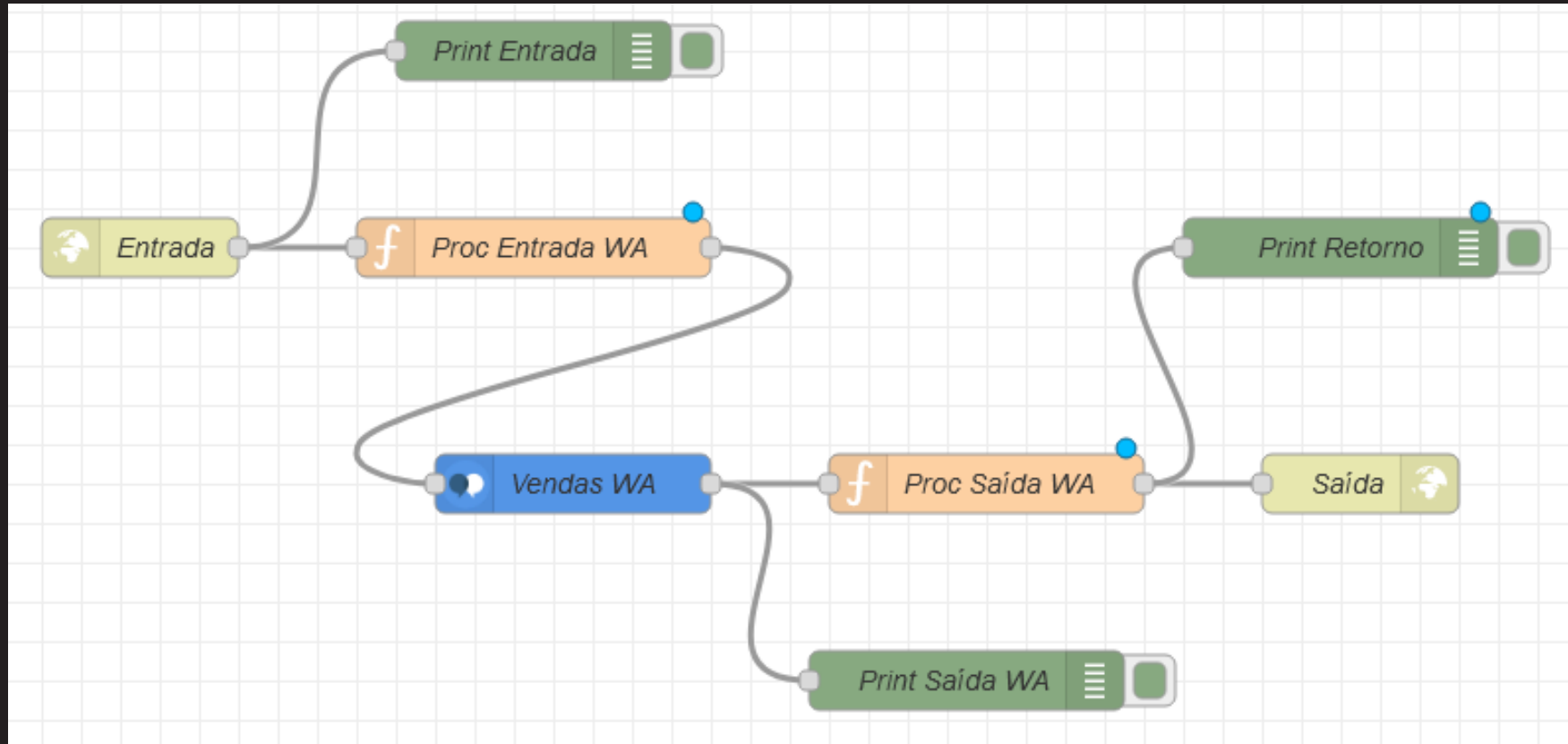
Agora obtivemos a resposta esperada!

Criando uma api http - POST

Como conectar o bot a um formulário HTML via POST

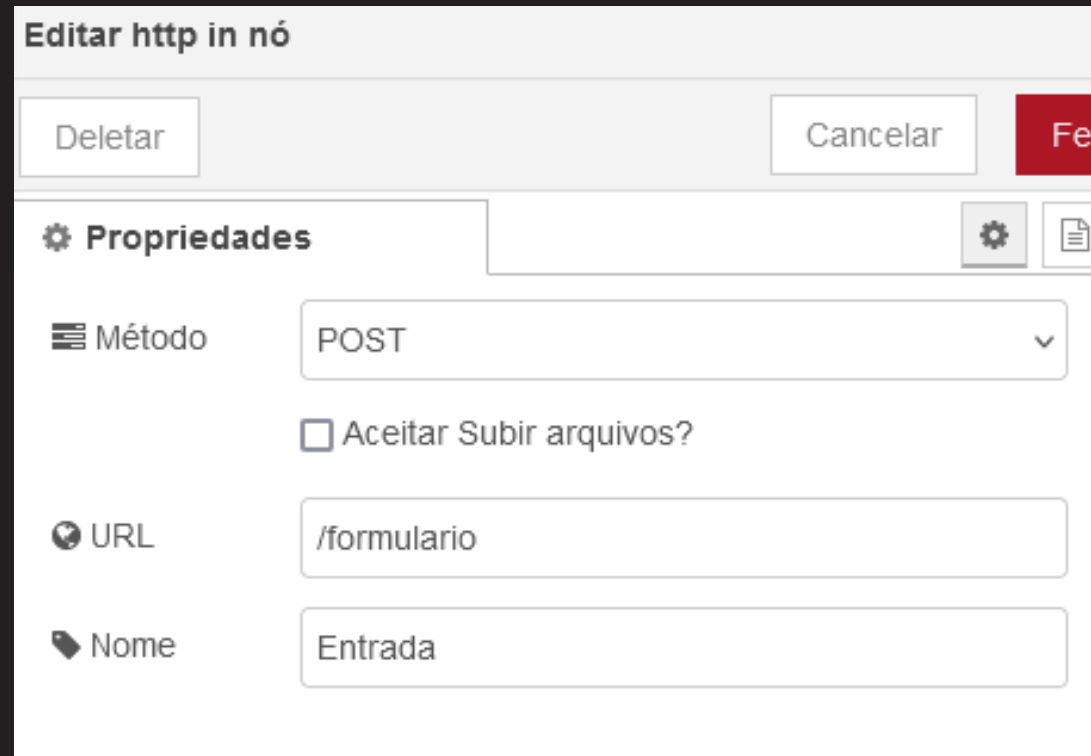
Configurando conexão HTTP POST

Façamos o mesmo fluxo do exemplo HTTP GET



Configurando conexão HTTP POST

Agora iremos trocar a configuração do http in para o método POST:



The screenshot shows a dialog box titled "Editar http in nó". At the top, there are three buttons: "Deletar", "Cancelar", and "Feito" (partially visible). Below the buttons is a section titled "Propriedades" with a gear icon. Inside this section, there are four configuration items:

- Método:** A dropdown menu showing "POST".
- Aceitar Subir arquivos?:** A checkbox that is currently unchecked.
- URL:** A text input field containing "/formulario".
- Nome:** A text input field containing "Entrada".

Também trocamos a rota o URL para formulário.

Configurando conexão HTTP POST

Vamos começar com um formulário geral, que recebe qualquer entrada de texto. Assim como na versão com POST, nossa função *Proc Entrada WA* está esperando uma variável chamada mensagem. Coloquemos isso no id e name do campo do formulário:

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Formulário</title>
</head>
<body>

<form action="#" method="post">
  <label for="mensagem">Mensagem:</label><br>
  <input type="text" id="mensagem" name="mensagem"><br><br>
  <input type="submit" value="Enviar">
</form>

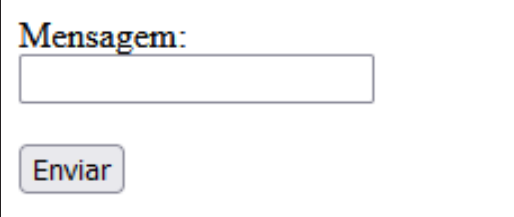
</body>
</html>
```

Configurando conexão HTTP POST

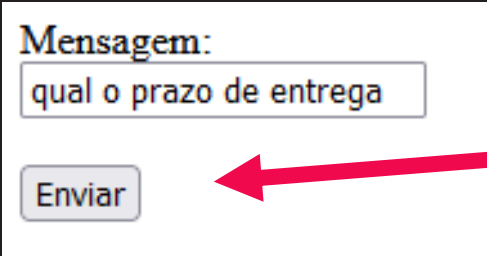
Altere a **rota de action** do formulário para o script do Node-RED. De fato o Node-RED estará agindo como nosso **back-end** nesse exemplo:

```
<form action="http://127.0.0.1:1880/formulario" method="post">CRLF
...<label for="mensagem">Mensagem:</label><br>CRLF
...<input type="text" id="mensagem" name="mensagem"><br><br>CRLF
...<input type="submit" value="Enviar">CRLF
```

Abra o formulário no seu navegador e digite uma mensagem para o bot:



Mensagem:



Mensagem:

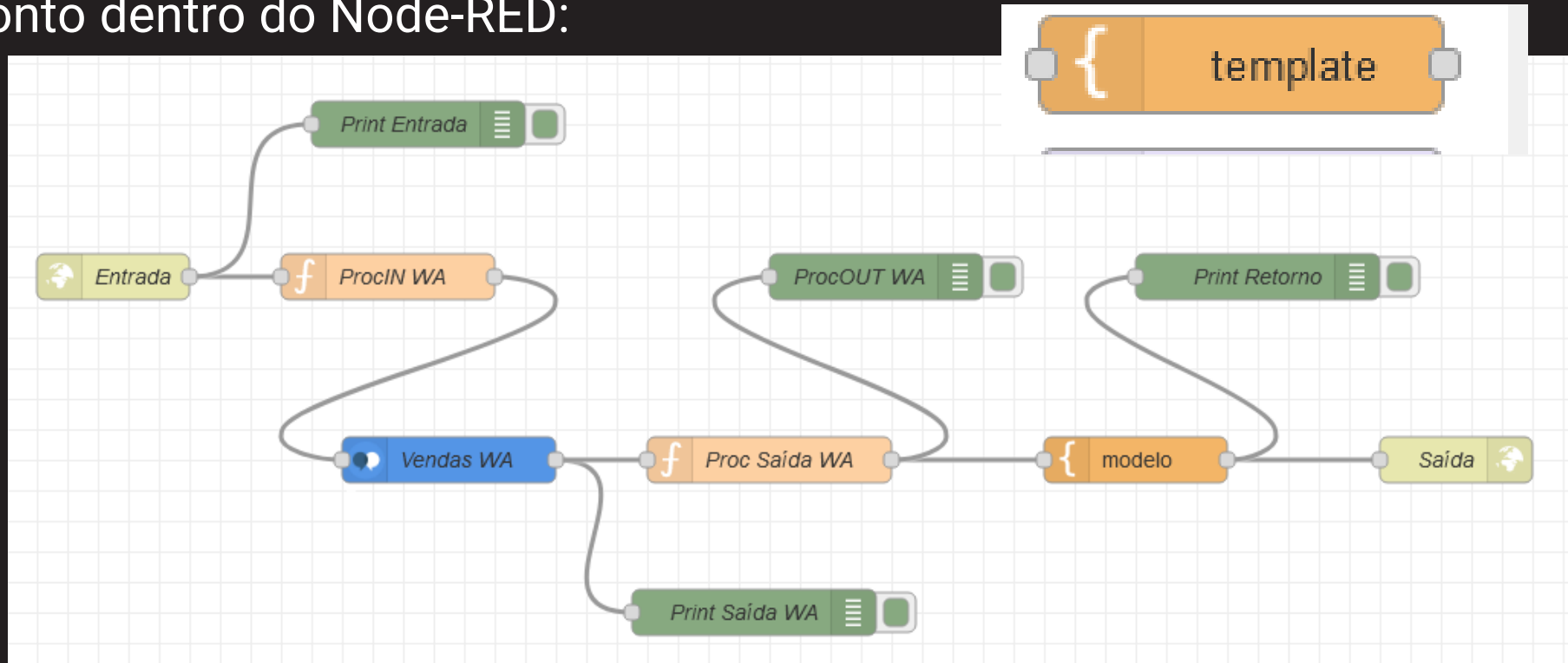
Clique em enviar. Qual o resultado?

Configurando conexão HTTP POST

Você deve ter recebido a resposta:

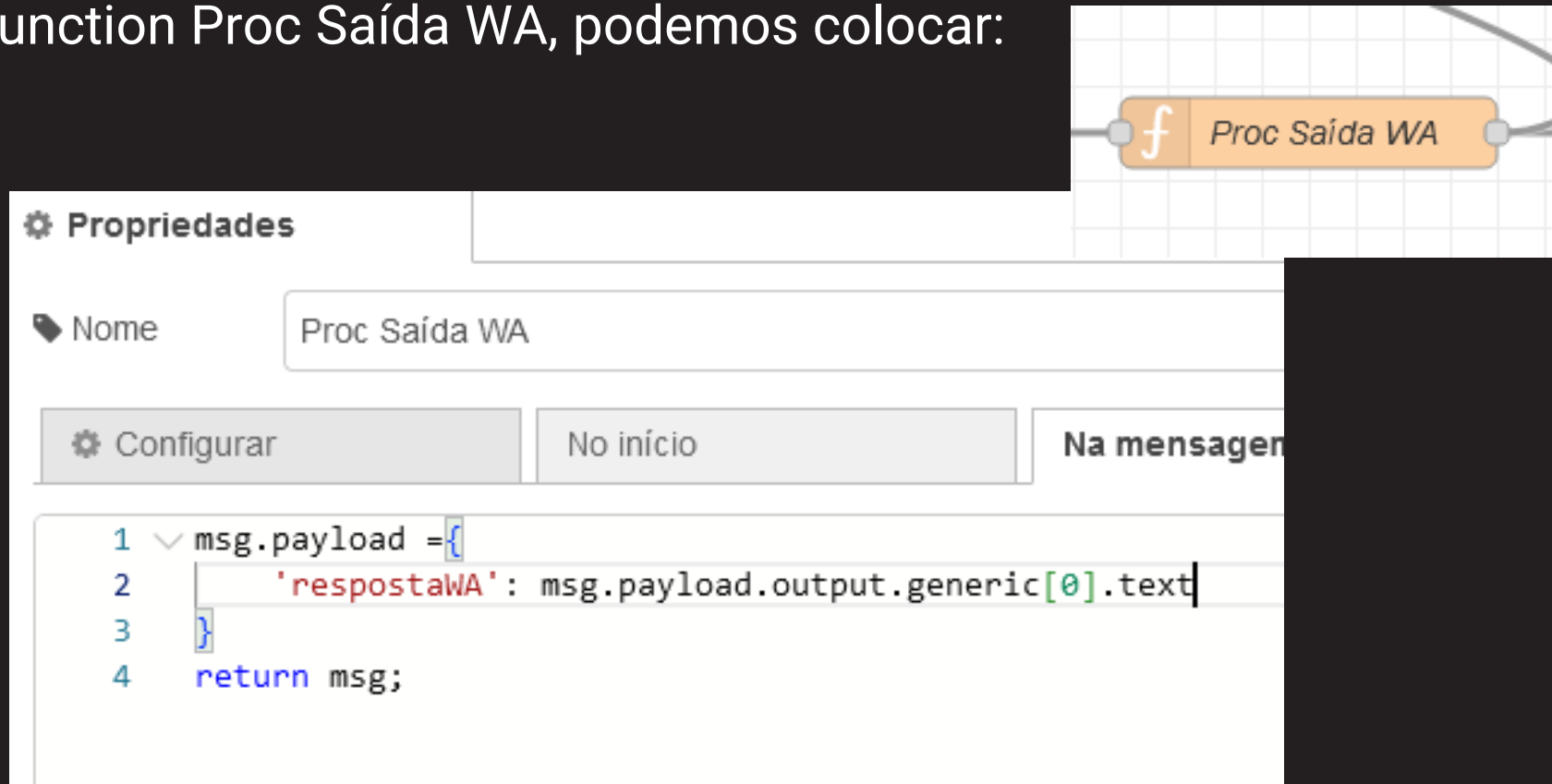
Para qual cidade?

Agora, como continuar a interação? Precisamos que o back-end gere uma nova página HTML com o campo e formulário como antes. Podemos utilizar o nó de Template para criar um HTML pré-pronto dentro do Node-RED:



Configurando conexão HTTP POST

No nó de function Proc Saída WA, podemos colocar:



Isso arrumará a saída que recebemos do WA para facilitar na hora de usar no template.

Configurando conexão HTTP POST

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Formulário</title>
<style>
  .campo-azul {
    background-color: #e0f0ff;
    padding: 10px;
    margin-bottom: 10px;
  }
  .campo-verde {
    background-color: #d9fbd9;
    padding: 10px;
  }
  .campo-texto {
    font-weight: bold;
    margin-bottom: 5px;
  }
</style>
</head>
<body>
```



Agora, dentro do Template vamos escrever.

Configurando conexão HTTP POST



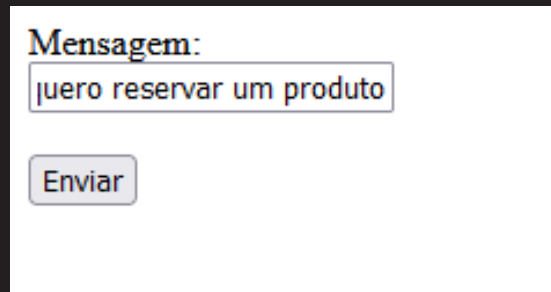
```
<div class="campo-azul">
  <div class="campo-texto">Chatbot</div>
  <div id="mensagem-usuario">{{payload.respostaWA}}</div>
</div>
```

Agora, dentro do Template vamos escrever.

```
<div class="campo-verde">
  <div class="campo-texto">Você</div>
  <form action="http://127.0.0.1:1880\formulario"
method="post">
    <label for="mensagem">Mensagem:</label><br>
    <input type="text" id="mensagem"
name="mensagem"><br><br>
    <input type="submit" value="Enviar">
  </form>
</div>
</body>
</html>
```

Configurando conexão HTTP POST

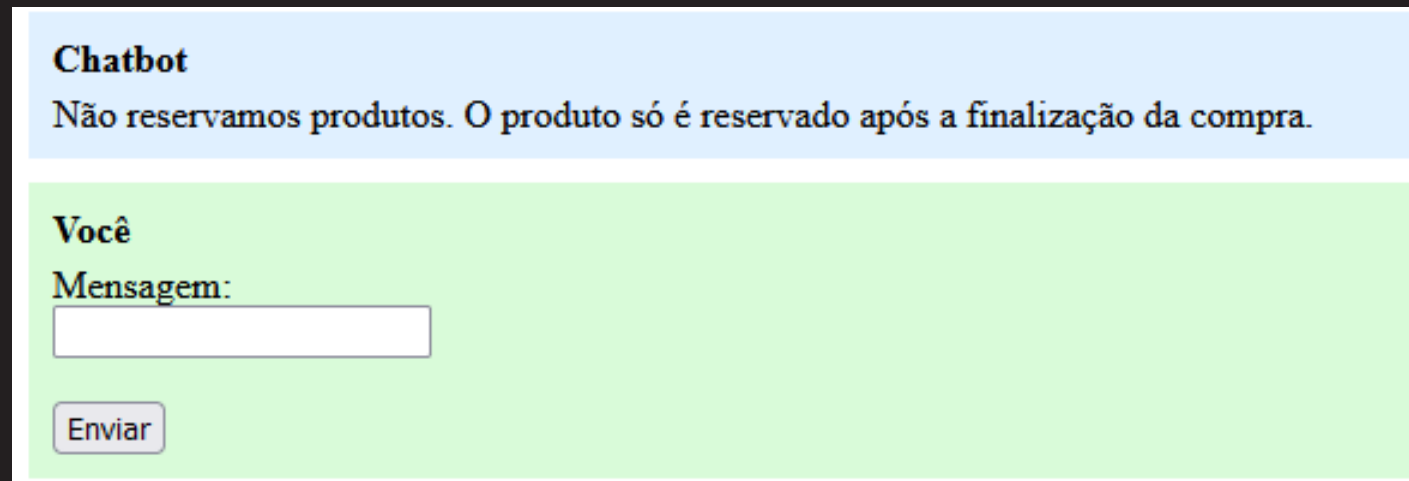
Abra o formulário no navegador e comece uma nova interação:



Mensagem:

Enviar

Agora nosso template do Node-RED retorna um HTML formatado com CSS e que reescreve o formulário para novas entradas. Você pode continuar a interação:



Chatbot

Não reservamos produtos. O produto só é reservado após a finalização da compra.

Você

Mensagem:

Enviar

Configurando conexão HTTP POST

1)

Chatbot
Não reservamos produtos. O produto só é reservado após a finalização da compra.

Você
Mensagem:

2)

Chatbot
Para qual cidade?

Você
Mensagem:

3)

Chatbot
Para São Paulo garantimos um prazo de 10 dias.

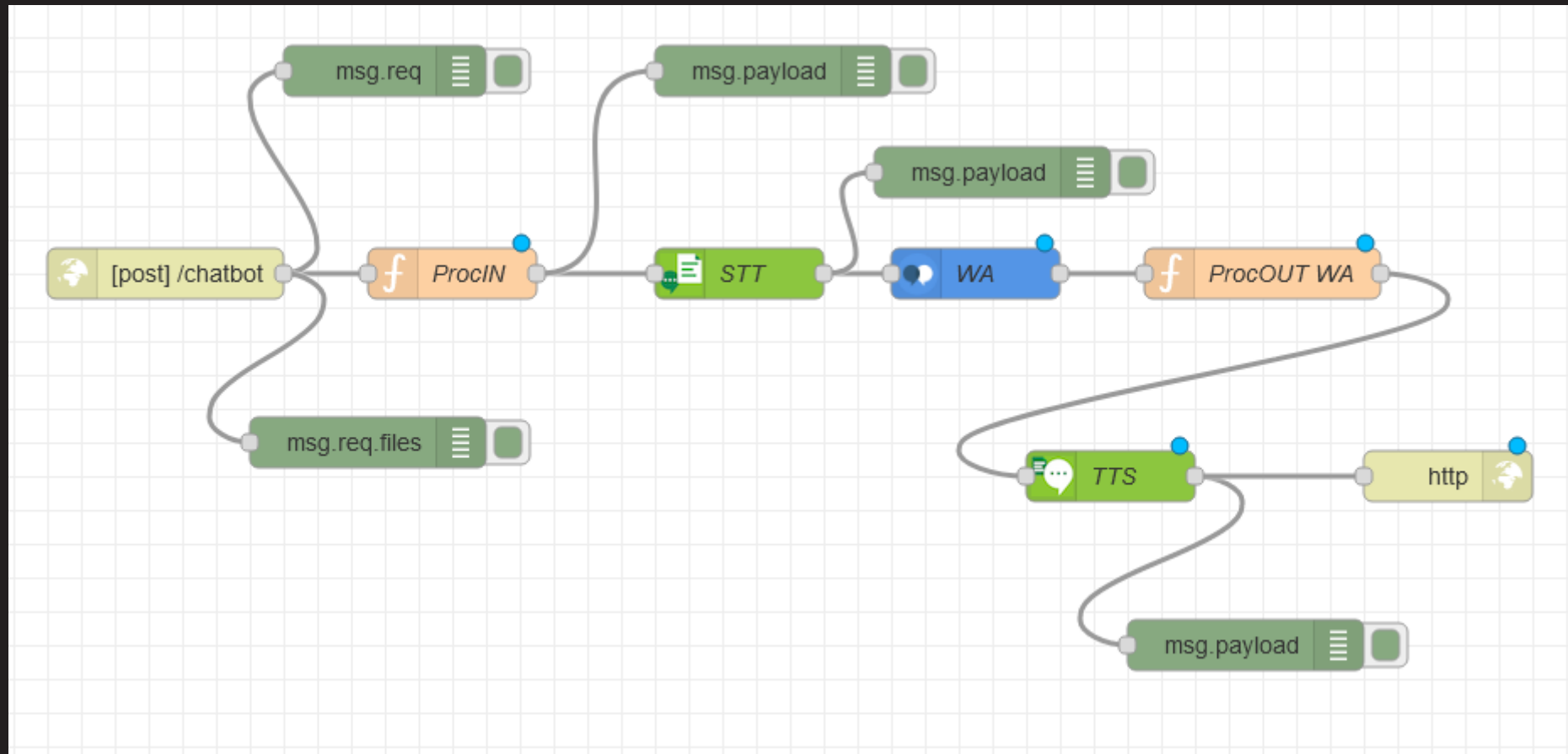
Você
Mensagem:

Criando uma api http POST com o STT e o TTS

Usando um formulário HTML para enviar Áudio via POST

Configurando conexão HTTP POST

Construa o seguinte fluxo:



Configurando conexão HTTP POST

Configurando os nós:



Editar http in nó

Deletar Cancelar

Propriedades

Método POST

☒ Aceitar Subir arquivos?

URL /chatbot

Nome Nome



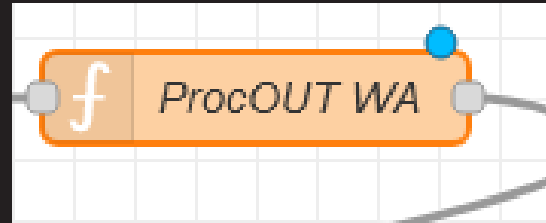
Nome ProcIN

Configurar No início

```
1 msg.params={
2   'session_id': '123'
3 }
4 msg.payload = msg.req.files[0].buffer;
5 return msg;
```

Configurando conexão HTTP POST

Configurando os nós:



Nome ProcOUT WA

Configurar No início Na mens

```
1 msg.payload = msg.payload.output.generic[0].text;  
2 return msg;
```

Agora vamos criar uma página HTML para enviar o arquivo de áudio.

Configurando conexão HTTP POST

Use o bloco de notas e salve como arquivo .html:

```
<html>
<head>
Enviar audio
</head>
<body>
<form enctype="multipart/form-data" method="post"
action="http://127.0.0.1:1880/chatbot">
  <p>
    <label>Add file: </label><br/>
    <input type="file" id="uploaded_audio" name="uploaded_audio"/>
  </p>

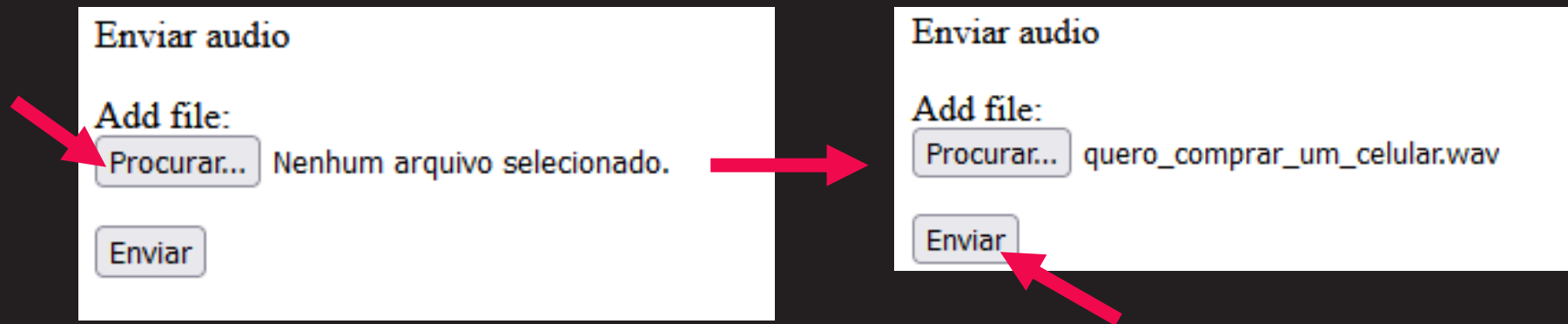
  <p>
    <input type="submit" value="Enviar"/>
  </p>
</form>
</body>
</html>
```



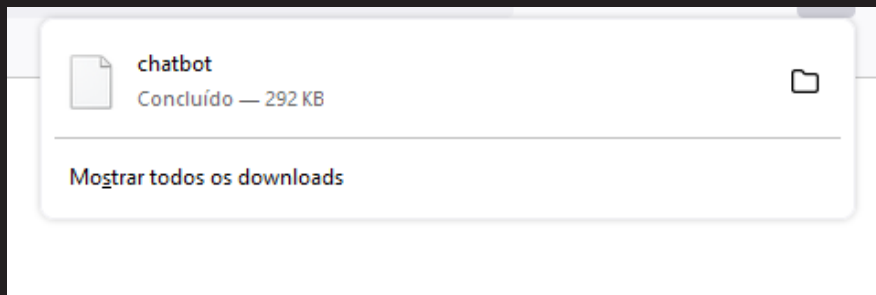
formulario_enviar_audio

Configurando conexão HTTP POST

Clique em Procurar e selecione um arquivo de áudio contendo apenas fala humana. Em seguida clique em Enviar:



O download de um arquivo de áudio deve iniciar na sequência:



Modifique a extensão do arquivo para terminar com .wav. O bot respondeu de acordo com o esperado?

Exercícios

Colocando em prática os conceitos aprendidos

Exercícios Avançados

1. Altere o fluxo de Banco de Dados Rudimentar para que os dados cadastrais só sejam salvos após o usuário confirmar o cadastro. Você pode usar uma variável de contexto adicional criada no nó de diálogo do Watson Assistant como um booleano de disparo para atualizar o banco.
2. Faça um fluxo que salve e leia os dados cadastrar do bot de e-commerce do WA. Este fluxo deve funcionar por texto e áudio, de forma que ao receber áudio, o bot responderá em áudio e ao receber texto, o bot responderá em texto.
3. Junte o exemplo do HTTP Post com Template e o HTTP Post STT/TTS para que seja possível interagir via áudio continuamente em uma página HTML aberta no navegador.

Copyright © 2024

**Slides do Prof. Érick Yamamoto, com adaptações dos
slides dos Prof. Henrique Ferreira- FIAP**

Todos direitos reservados. Reprodução ou divulgação total ou parcial deste documento é expressamente proibido sem o consentimento formal, por escrito, do Professor (autor).