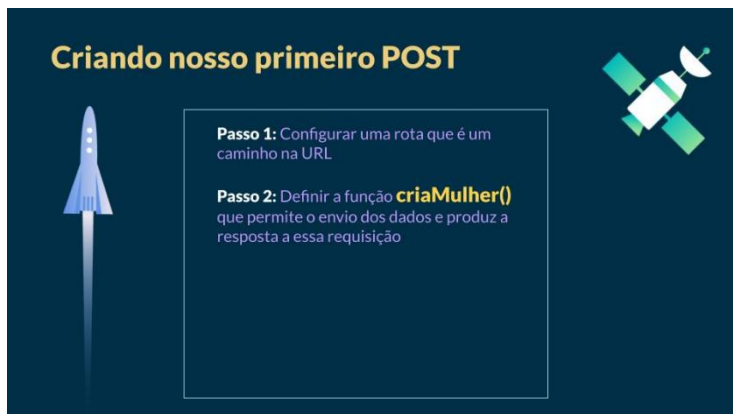


Vamos aprender nesta aula como permitir a criação de uma nova mulher para nossa lista. Tá preparade? Aperte o cinto 🚀

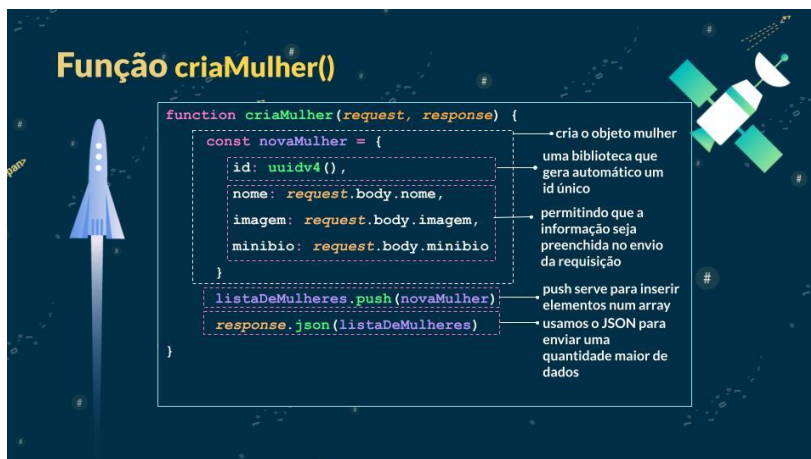
O que é POST

Dentro do conjunto de regras do HTTP, o verbo que realiza criação de informações é chamado de POST, que na tradução para português seria "publicar" 💻

Como criar o POST



No mesmo arquivo, onde já temos o GET criado, podemos também criar o POST através do corpo da requisição, então, desta vez, além da response, vamos usar também a request dentro da nossa função criaMulher(). 📱



Sendo assim, precisamos executar na nossa função os 3 passos a seguir para permitir a criação de uma nova mulher:

✓ Vamos criar cada campo do objeto mulher e em cada valor iremos informar que os dados serão preenchidos no ato da criação de uma nova mulher quando a requisição for feita. Perceba que a partir de agora temos o campo id tanto na nova mulher, como na lista de mulheres, pois ele nos ajuda a lidar com listas que possuem muitos itens.

Estamos usando uma biblioteca chamada **uuid** que gera automaticamente um valor único de id para as novas mulheres que forem criadas.

✓ Vamos atualizar a lista atual de mulheres, mantendo todas as mulheres existentes e incluindo a nova mulher que está sendo criada. Para fazer isso, vamos usar algo chamado **push()**, que é uma função do javascript que serve para inserir um novo item ao final do array. Ou seja, estamos pedindo para a novaMulher que será criada, ser inserida no array de listaDeMulheres.

✓ Por fim, vamos pedir para enviar como **resposta** a lista atualizada com sucesso.

Finalizando a configuração, seguindo a mesma regra que usamos no GET, chegou a hora de pedir para o **aplicativo criar o caminho** e usar o método em questão, além de pedir também para ele chamar a nossa função criaMulher. 🤖



Vamos confirmar se deu tudo certo? 🤔

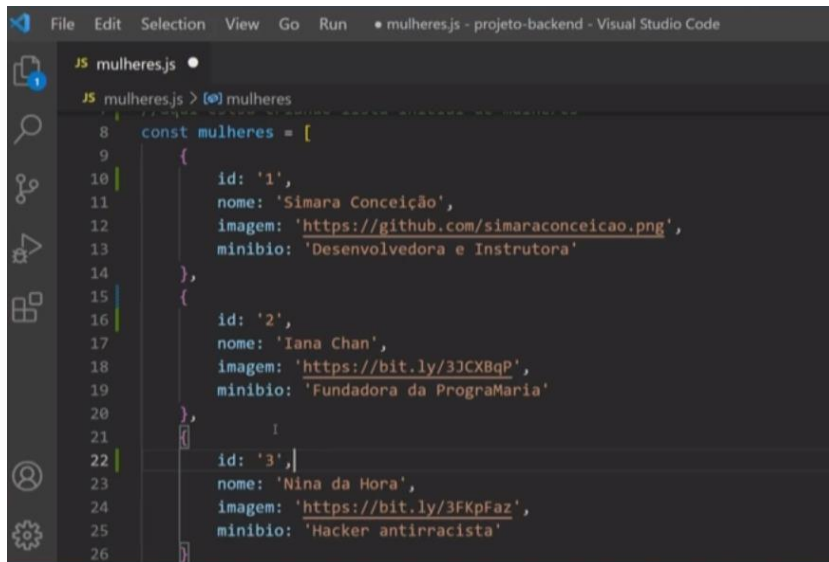
No arquivo **mulheres.js** (já criado para o GET), abaixo da function mostraMulher adicione a função **criaMulher** do POST (por enquanto, sem complementar):

```
File Edit Selection View Go Run • mulheres.js - projeto-backend - Visual Studio Code

JS mulheres.js •
JS mulheres.js > criaMulher > [?] novaMulher > minibio

8 > const mulheres = [ ...
24 ]
25
26 //GET
27 > function mostraMulheres(request, response) { ...
29 }
30
31 //POST
32 function criaMulher(request, response) {
33   const novaMulher = {
34     id: '',
35     nome: '',
36     imagem: '',
37     minibio: ''
38   }
39 }
40
41 //PORTA
42 > function mostraPorta() { ...
```

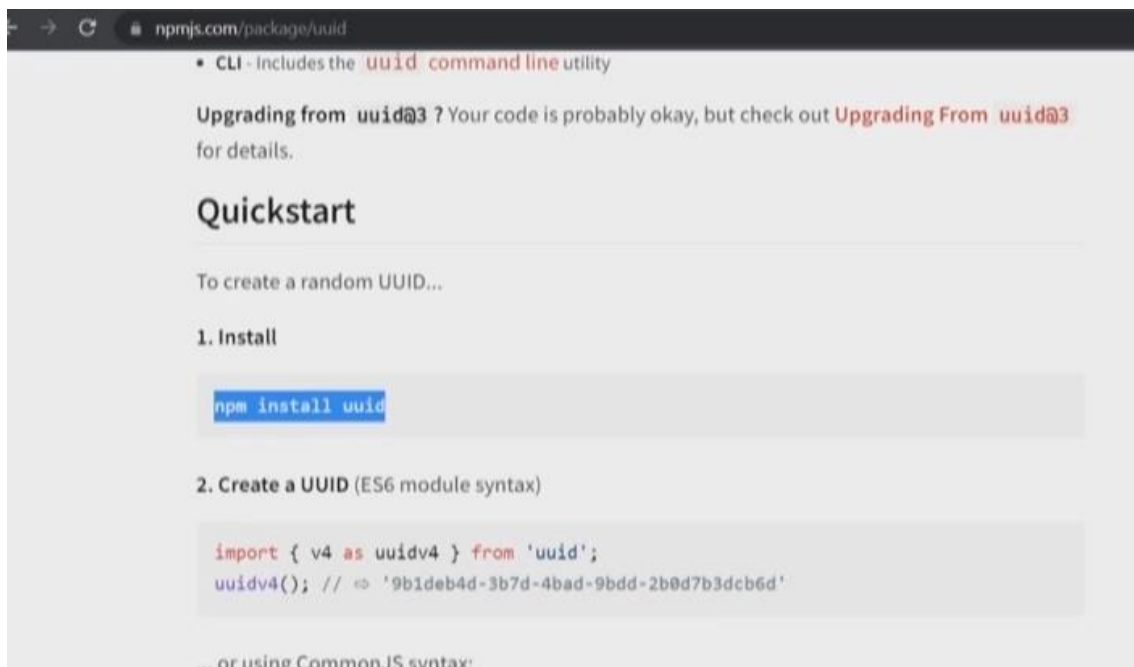
Vá até const mulheres e inclua o **id** em cada uma, acima do nome:



```
JS mulheres.js
JS mulheres.js > [0] mulheres
8 const mulheres = [
9   {
10    id: '1',
11    nome: 'Simara Conceição',
12    imagem: 'https://github.com/simaraconceicao.png',
13    minibio: 'Desenvolvedora e Instrutora'
14  },
15  {
16    id: '2',
17    nome: 'Iana Chan',
18    imagem: 'https://bit.ly/3JCXBqP',
19    minibio: 'Fundadora da PrograMaria'
20  },
21  {
22    id: '3',
23    nome: 'Nina da Hora',
24    imagem: 'https://bit.ly/3FKpFaz',
25    minibio: 'Hacker antirracista'
26  }
27 ]
```

No seu navegador, acesse: <https://www.npmjs.com/package/uuid> e siga os passos para instalar uma biblioteca:

No seu terminal, use o **primeiro comando do site**:



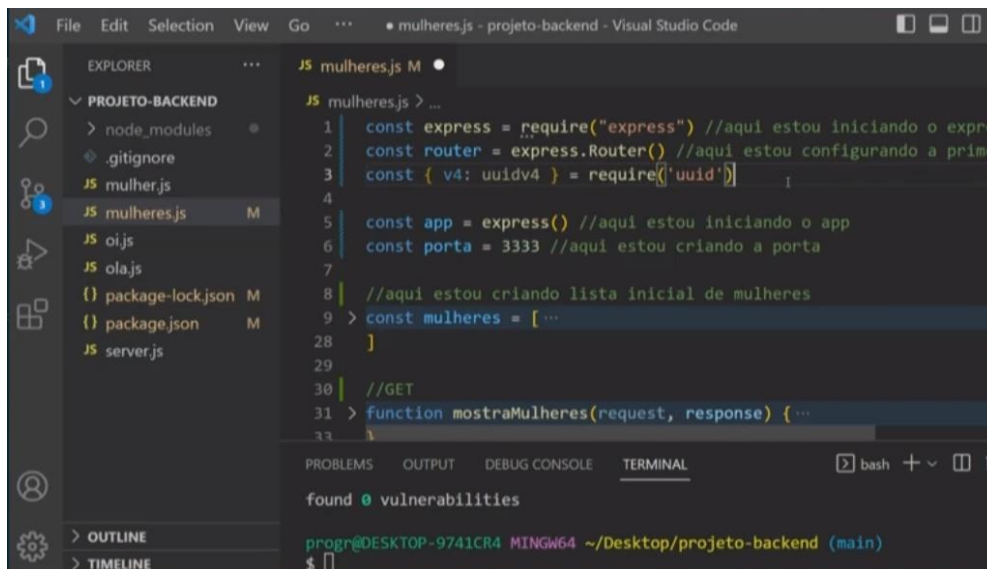
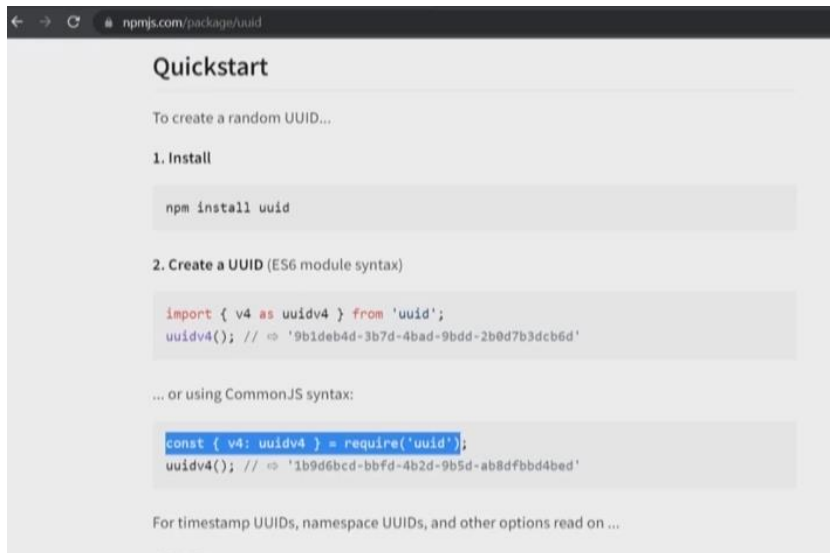
The screenshot shows the Visual Studio Code interface with a file named `mulheres.js` open. The code defines a `criaMulher` function that creates a new user object with fields `id`, `nome`, `imagem`, and `minibio`. Below the code editor, the TERMINAL panel is active, showing the command `npm install uuid` being executed. The terminal output shows the command prompt `progr@DESKTOP-9741CR4 MINGW64 ~/Desktop/projeto-backend (main)` and the command `$ npm install uuid`.

This screenshot shows the same Visual Studio Code interface, but the TERMINAL panel now displays the output of the `npm install uuid` command. The output includes: `added 1 package, and audited 59 packages in 766ms`, `7 packages are looking for funding`, `run 'npm fund' for details`, and `found 0 vulnerabilities`. The command prompt is now `progr@DESKTOP-9741CR4 MINGW64 ~/Desktop/projeto-backend (main)` with a new line for input.

Para confirmar se a biblioteca foi instalada, vá em **package.json** e veja dentro de dependências se está o **uuid**:

The screenshot shows the Visual Studio Code interface with the `package.json` file open. The `dependencies` section of the file is visible, showing that `express` and `uuid` are listed as dependencies. The `uuid` version is specified as `"^9.0.0"`. The TERMINAL panel at the bottom shows the same output as the previous screenshot, confirming the successful installation of the package.

Volte no site e use o **segundo comando**, entre `const router` e `const app`:



Preencha a função do POST:

⚠️ Atenção: lembre-se das vírgulas, parênteses, dois pontos, ponto final e colchetes.

⚠️ Lembre-se de salvar suas alterações!

```
File Edit Selection View Go ... • mulheres.js - projeto-backend - Visual Studio Code

EXPLORER
PROJETO-BACKEND
  > node_modules
  > .gitignore
  JS mulheres.js
  JS mulheres.js M
  JS oi.js
  JS ola.js
  {} package-lock.json M
  {} package.json M
  JS server.js

JS mulheres.js M
JS mulheres.js > criaMulher > [0] novaMulher
33 }
34
35 //POST
36 function criaMulher(request, response) {
37   const novaMulher = {
38     id: uuidv4(),
39     nome: request.body.nome,
40     imagem: request.body.imagem,
41     minibio: request.body.minibio
42   }
43 }
44
45 //PORTA
46 > function mostraPorta() { ...
```

Abaixo adicione a **função push()** e a **resposta**:

```
File Edit Selection View Go ... • mulheres.js - projeto-backend - Visual Studio Code

EXPLORER
PROJETO-BACKEND
  > node_modules
  > .gitignore
  JS mulher.js
  JS mulheres.js M
  JS oi.js
  JS ola.js
  {} package-lock.json M
  {} package.json M
  JS server.js

JS mulheres.js M
JS mulheres.js > criaMulher
40   imagem: request.body.imagem,
41   minibio: request.body.minibio
42 }
43
44 mulheres.push(novaMulher)
45
46 response.json(mulheres)
47 }
48
49 //PORTA
50 > function mostraPorta() { ...
52 }
53
```


Adicione a **rota do POST** abaixo do app.use do GET:

```
File Edit Selection View Go ... • mulheres.js - projeto-backend - Visual Studio Code

JS mulheres.js M
JS mulheres.js > ...
50 > function mostraPorta() { ...
52 }
53
54 app.use(router.get('/mulheres', mostraMulheres)) //configurei rota GET /mulheres
55 app.use(router.post('/mulheres', criaMulher)) // configurei rota POST /mulheres
56 app.listen(porta, mostraPorta) //servidor ouvindo a porta
```

⚠ **Lembre-se de salvar suas alterações!**

Enquanto nosso backend não está conectado com nenhum front-end precisamos de uma ferramenta para simular o lado do cliente 🤖

Nós vamos usar a ferramenta chamada Insomnia. Acompanhe na videoaula como baixá-la e utilizá-la 



Se do lado direito do insomnia aparecer a palavra error e a frase:

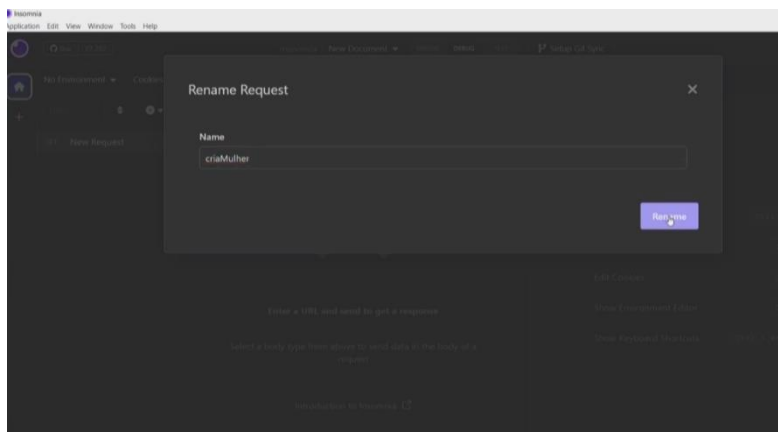
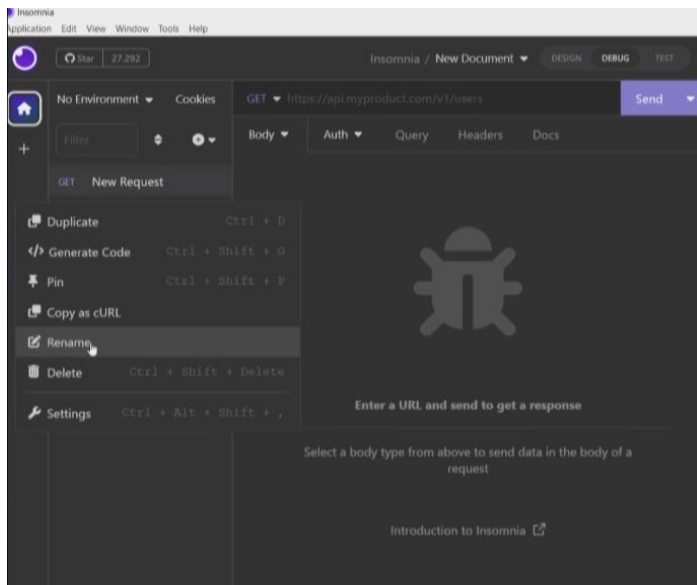
"Error: Couldn't connect to server." você precisa perceber se escreveu errado o endereço até a parte do número da PORTA ou se não rodou o projeto lá no terminal do VSCode.

"404 not found Error: Cannot POST /mulhere"

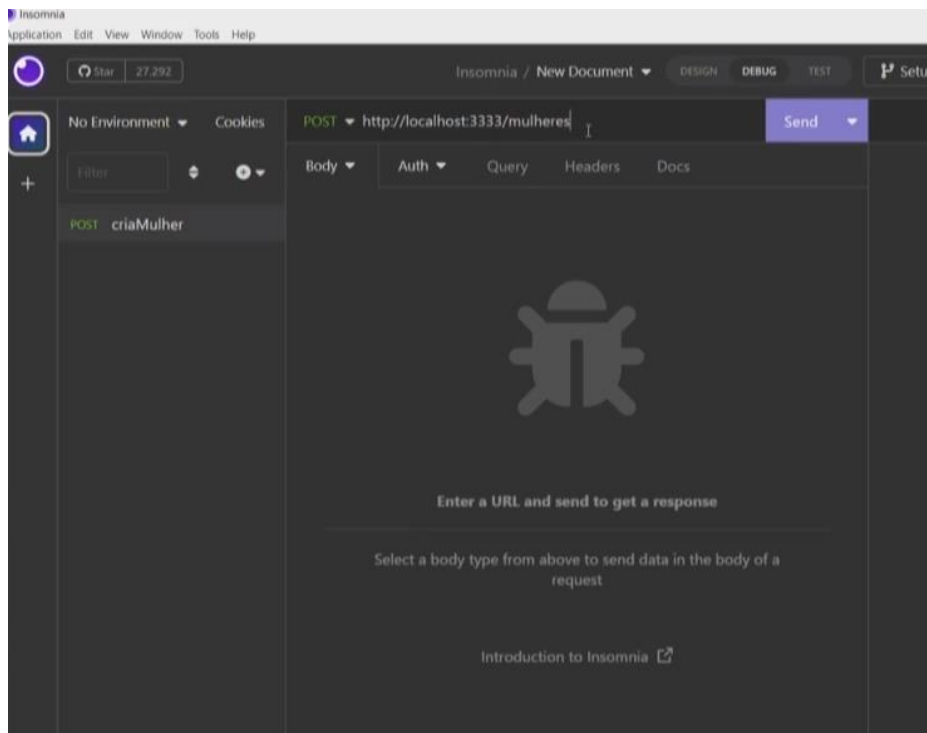
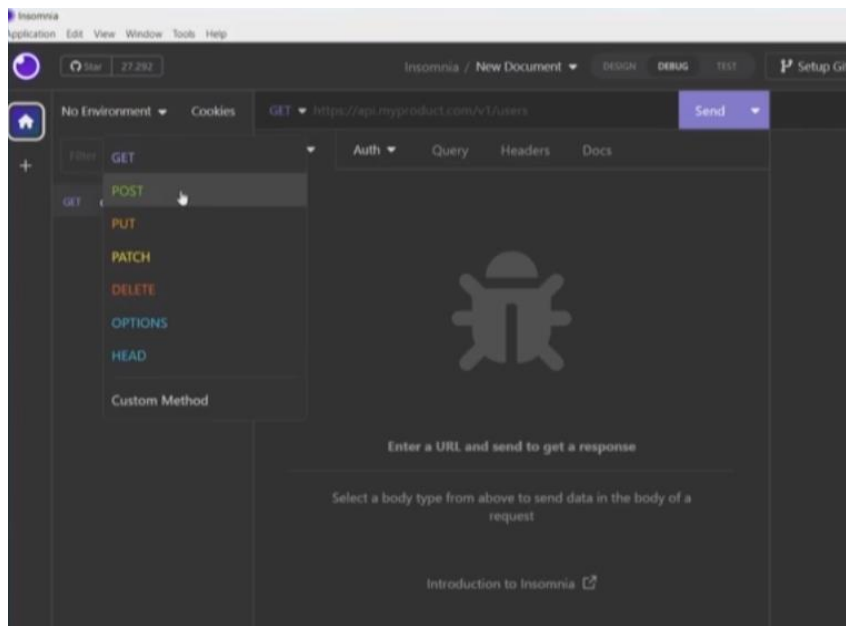
⚠ Significa que escrevemos o endereço errado depois da barra.

TESTANDO O POST NO INSOMNIA

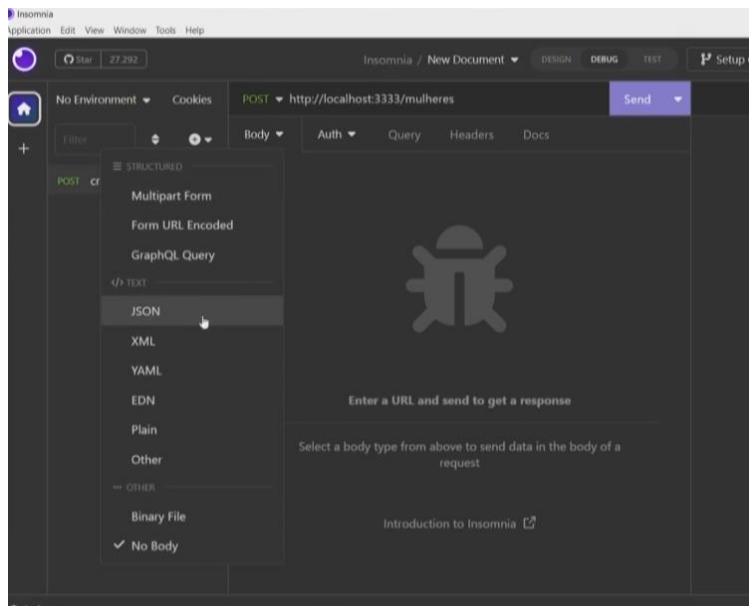
No canto superior esquerdo, clique em **GET New Request** para **renomear**. Use o nome da requisição **criaMulher**:



Altere o GET para **POST** e **adicione a nossa URL**:

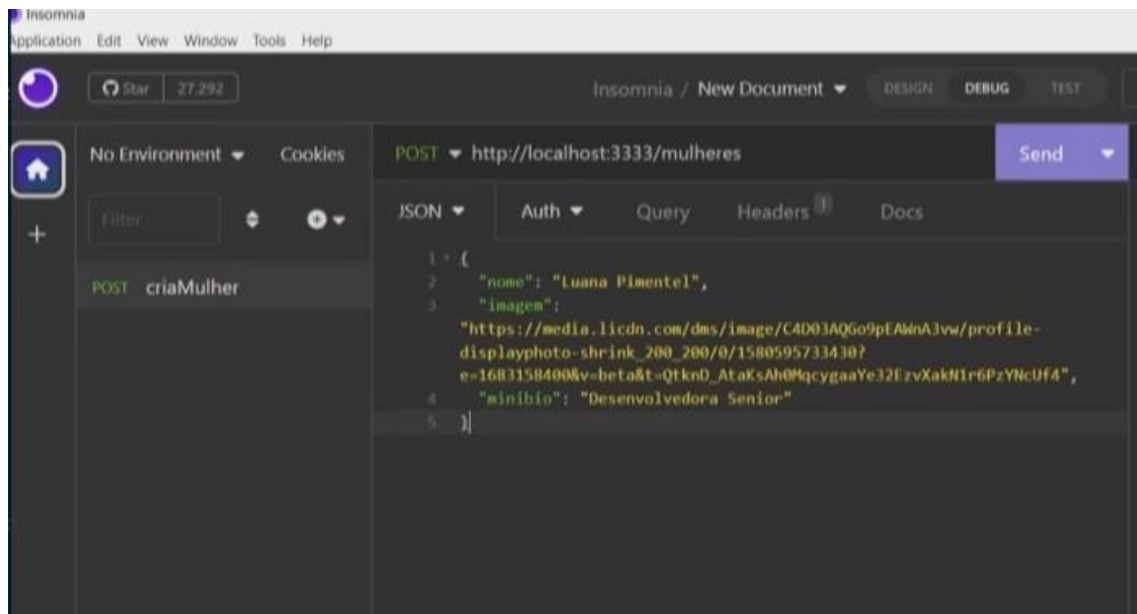


Altere o Body para **JSON**:



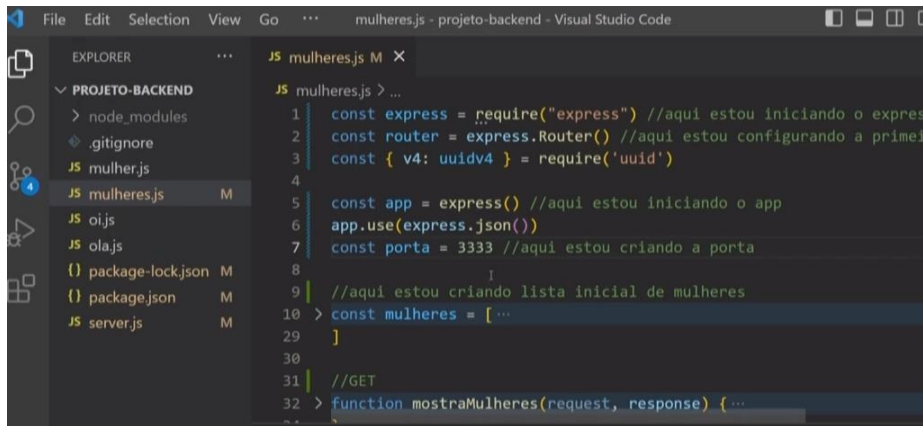
Faça o **corpo da requisição** com **nome**, **imagem** (link) e **minibio**

Não clique em Send (enviar) por enquanto:

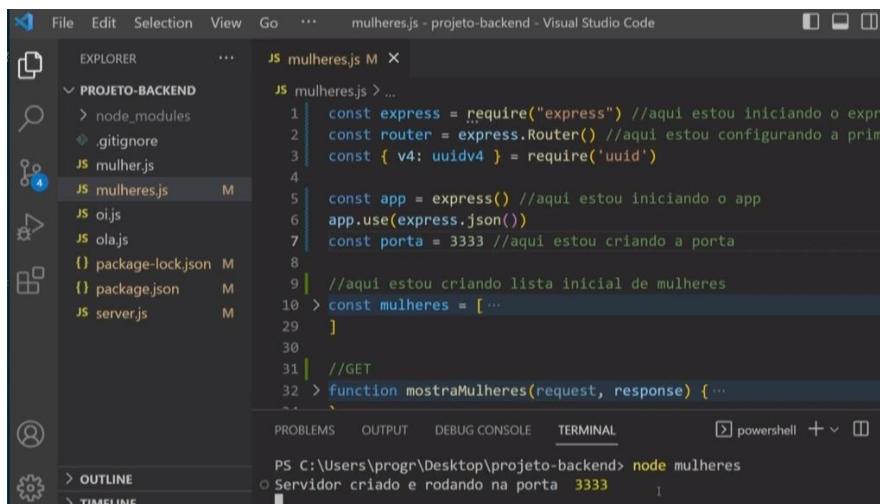


Volte no VSCODE e complemente seu código com **app.use(express.json())** abaixo de **const app**. Pois precisamos dizer que os dados que vão trafegar estão no formato JSON.

Em seguida, rode no terminal **node mulheres.js**



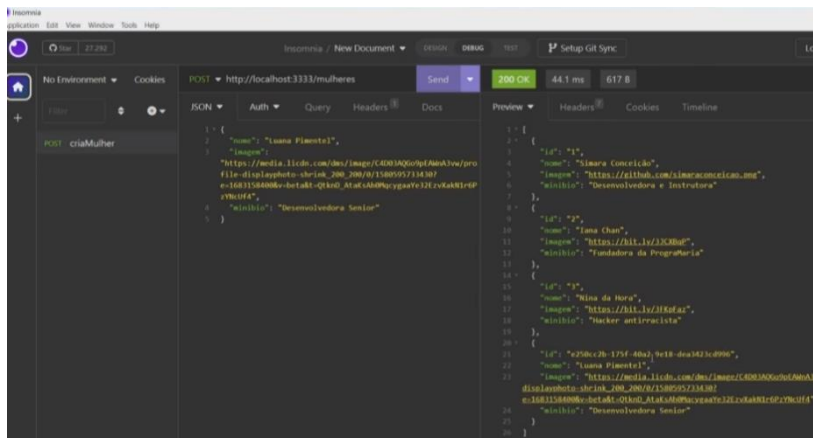
```
JS mulheres.js M X
JS mulheres.js > ...
1 const express = require("express") //aqui estou iniciando o expres
2 const router = express.Router() //aqui estou configurando a primei
3 const { v4: uuidv4 } = require('uuid')
4
5 const app = express() //aqui estou iniciando o app
6 app.use(express.json())
7 const porta = 3333 //aqui estou criando a porta
8
9 //aqui estou criando lista inicial de mulheres
10 > const mulheres = [ ...
29 ]
30
31 //GET
32 > function mostraMulheres(request, response) { ...
```



```
JS mulheres.js M X
JS mulheres.js > ...
1 const express = require("express") //aqui estou iniciando o expres
2 const router = express.Router() //aqui estou configurando a primei
3 const { v4: uuidv4 } = require('uuid')
4
5 const app = express() //aqui estou iniciando o app
6 app.use(express.json())
7 const porta = 3333 //aqui estou criando a porta
8
9 //aqui estou criando lista inicial de mulheres
10 > const mulheres = [ ...
29 ]
30
31 //GET
32 > function mostraMulheres(request, response) { ...

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
powershell + v
PS C:\Users\progr\Desktop\projeto-backend> node mulheres
Servidor criado e rodando na porta 3333
```

Volte no Insomnia e **clique em Send (enviar)**:



E se quiséssemos corrigir os dados feitos na aula anterior, como seria? 🤔

Bora aprender como alterar a informação de uma mulher já cadastrada!? 🧐

O que é PATCH?

PATCH, que traduzido do inglês quer dizer correção. E é exatamente isso que ele faz, corrige a informação no objeto que apontarmos. 🖥️

PATCH /mulheres/:id

Ao acessar o endereço `localhost:3333/mulheres/:id`, ao preencher o id da mulher no final da rota, os dados que queremos corrigir no corpo da requisição, esperamos o resultado da lista atual incluindo os dados novos da mulher que acabamos de alterar.

Criando o PATCH

Criando nosso primeiro PATCH

Passo 1: Configurar uma rota que é um caminho na URL

Passo 2: Definir a função `corrigirMulher()` que permite o envio dos dados e produz a resposta a essa requisição

Configuração da rota

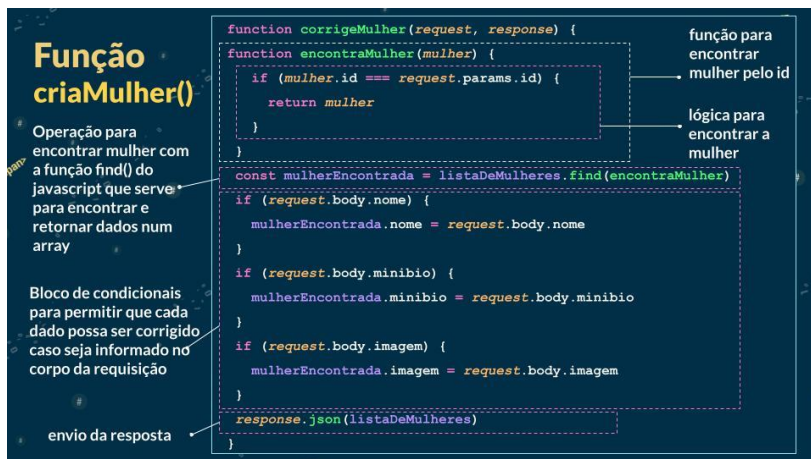


É necessário manter toda a estrutura de criação do servidor que aprendemos nas aulas passadas e vamos incluir as informações de configuração da rota POST /mulheres.

A primeira etapa da configuração da rota só precisa ser feita uma vez e já fizemos no GET

```
app.use(router.patch('/:id', corrigirMulher))
```

Agora estamos dizendo para o aplicativo responder o verbo PATCH na rota /mulheres/:id e chamar a função corrigirMulher



Neste caso, para pensar na lógica e desenhar o passo a passo que vamos seguir precisamos:

✓ Informar o objeto que iremos alterar a partir de uma informação única, como o **id** por exemplo.

Para encontrar um objeto dentro de um array podemos utilizar uma função do javascript chamada **find**, que traduzindo para o português significa encontrar.

O objeto `request` tem várias propriedades neste momento, estamos aprendendo o `params`.

Para utilizar essa função devemos informar em qual lista queremos encontrar o item e depois qual função será chamada que possui a condição para que o item seja retornado.

Para criar condições, vamos usar a palavra reservada **if**, que significa *se*. Ou seja, vamos perguntar se uma condição é verdadeira e caso a resposta seja sim, teremos retornado o objeto mulher que queremos alterar.

Para fazer a comparação vamos usar um sinal com três iguais `===` que significa igualdade estrita no javascript. Lembre-se que apenas um igual é atribuição e não igualdade.

✓ Informar um ou mais propriedades que podem ser corrigidas no momento da requisição, também a partir do corpo/body da `request`.

✓ E por fim podemos retornar a lista completa com o item já corrigido.

Pausa para o Javascript



Condicional **if**

É a palavra reservada que usamos quando precisamos que o computador execute uma ou mais instruções caso uma condição seja atendida.

Operador **===**

Diferente do operador com apenas um igual(=) que significa atribuir, o operados com três iguais representa igualdade estrita, ou seja um valor é exatamente igual a.

Função **find()**

É a função que serve para percorrer um array, encontrar e retornar uma informação de acordo com uma condição.

⚠ Se liga na sintaxe na demonstração

Anatomia da requisição



Request

⚠ Preencher o id da mulher que deseja corrigir os dados no final endereço

Endereço (params)

localhost:3333/mulheres/:id

Corpo (body)

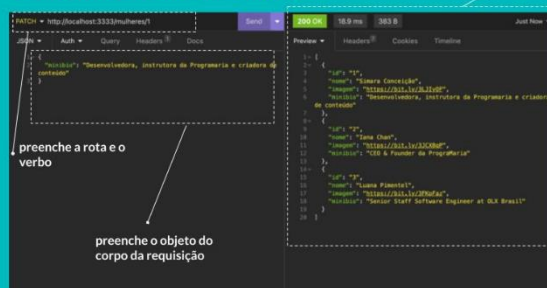
```
{
  "nome": "Maitê Lourenço",
  "imagem": "https://media.licdn.com",
  "minibio": "Criadora da BlackRocks Startups"
}
```

Response

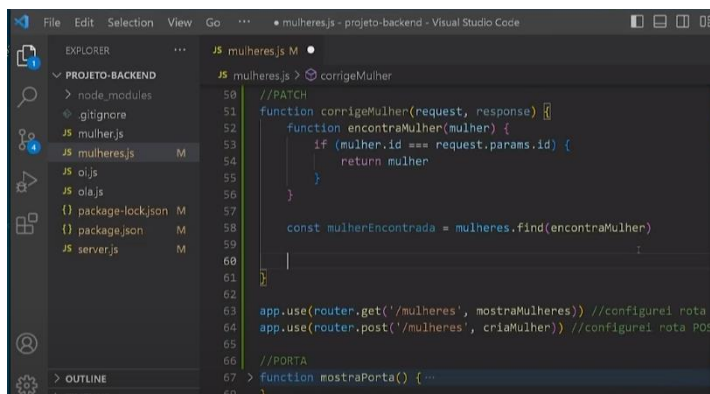
Array das mulheres incluindo dados da mulher corrigida

Testando no Insomnia

Resultado: código e array



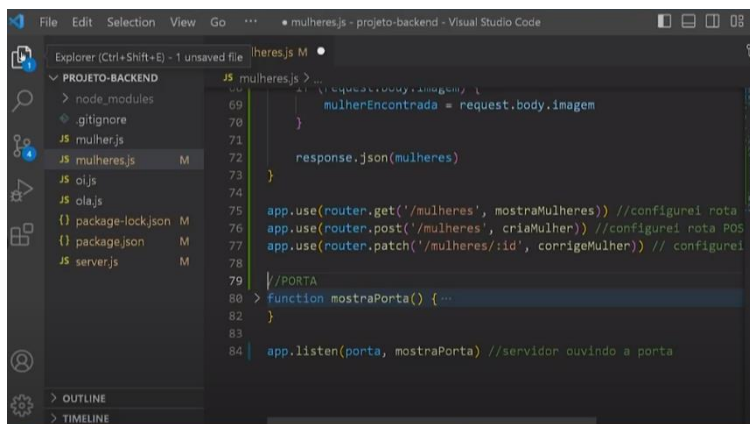
Inclua a **função PATCH** e a **const**:



Adicione o **if (se)** e adicione o **response.json**::

```
mulheres.js > ...
56   }
57
58   const mulherEncontrada = mulheres.find(encontraMulher)
59
60   if (request.body.nome) {
61     mulherEncontrada.nome = request.body.nome
62   }
63
64   if (request.body.minibio) {
65     mulherEncontrada.minibio = request.body.minibio
66   }
67
68   if (request.body.imagem) {
69     mulherEncontrada.imagem = request.body.imagem
70   }
71
72   response.json(mulheres)
73 }
```

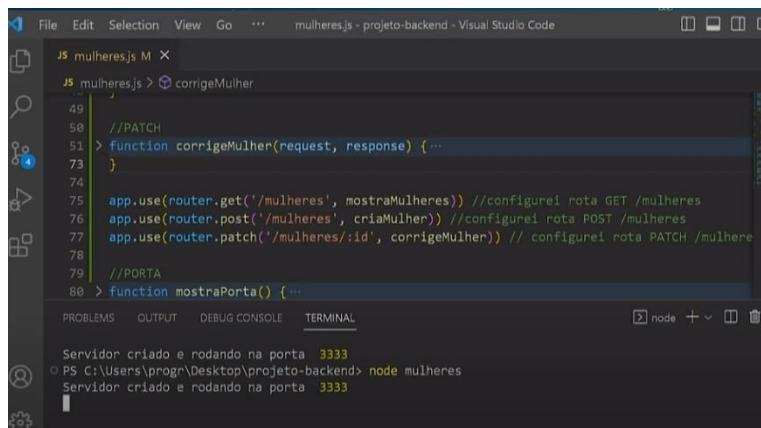
Adicione a **rota**:



```
File Edit Selection View Go ... • mulheres.js - projeto-backend - Visual Studio Code
Explorer (Ctrl+Shift+E) - 1 unsaved file |heres.js M
PROJETO-BACKEND
  > node_modules
  > gitignore
  JS mulher.js
  JS mulheres.js M
  JS oi.js
  JS ola.js
  {} package-lock.json M
  {} package.json M
  JS server.js M
JS mulheres.js > ...
69   if (request.body.imagem) {
70     mulherEncontrada.imagem = request.body.imagem
71   }
72   response.json(mulheres)
73 }
74
75 app.use(router.get('/mulheres', mostraMulheres)) //configurei rota
76 app.use(router.post('/mulheres', criaMulher)) //configurei rota POST
77 app.use(router.patch('/mulheres/:id', corrigeMulher)) // configurei
78
79 //PORTA
80 > function mostraPorta() { ...
82 }
83
84 app.listen(porta, mostraPorta) //servidor ouvindo a porta
```

⚠ **Lembre-se de salvar suas alterações!**

No terminal rode **node mulheres.js**:

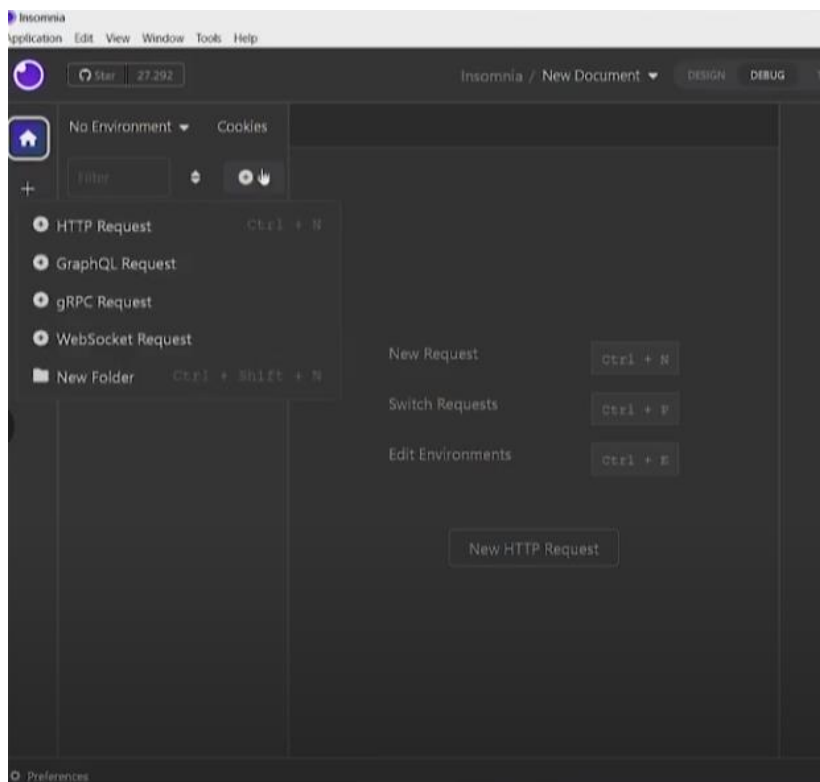


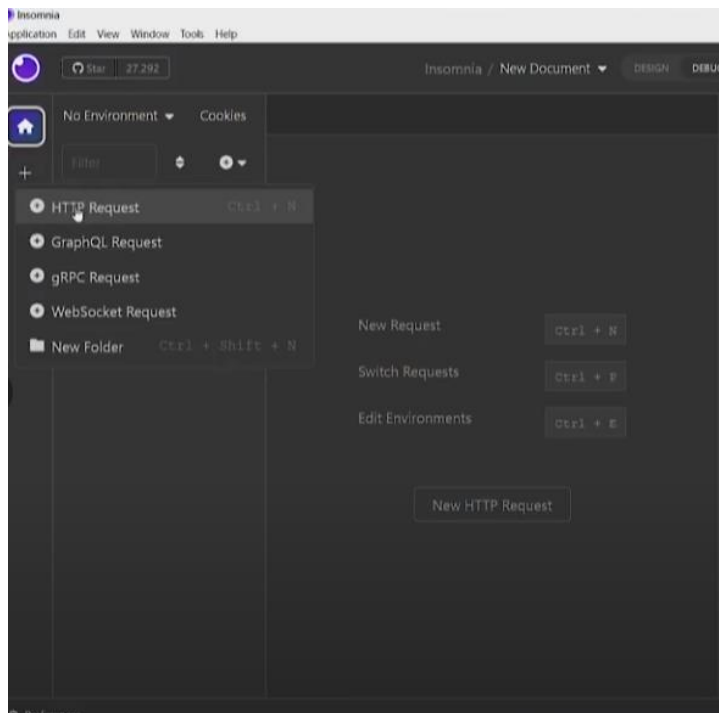
```
File Edit Selection View Go ... mulheres.js - projeto-backend - Visual Studio Code

JS mulheres.js M X
JS mulheres.js > corrigeMulher
49
50 //PATCH
51 > function corrigeMulher(request, response) {...
73 }
74
75 app.use(router.get('/mulheres', mostraMulheres)) //configurei rota GET /mulheres
76 app.use(router.post('/mulheres', criaMulher)) //configurei rota POST /mulheres
77 app.use(router.patch('/mulheres/:id', corrigeMulher)) // configurei rota PATCH /mulheres
78
79 //PORTA
80 > function mostraPorta() {...

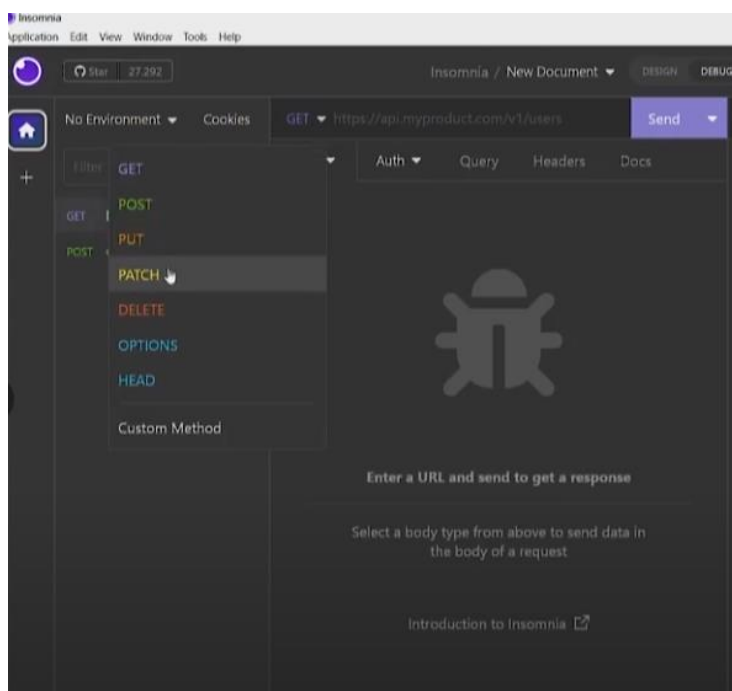
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
node + - -
Servidor criado e rodando na porta 3333
PS C:\Users\progr\Desktop\projeto-backend> node mulheres
Servidor criado e rodando na porta 3333
```

Na ferramenta Insomnia, no lado superior esquerdo clique no + e escolha a opção **HTTP Request**:

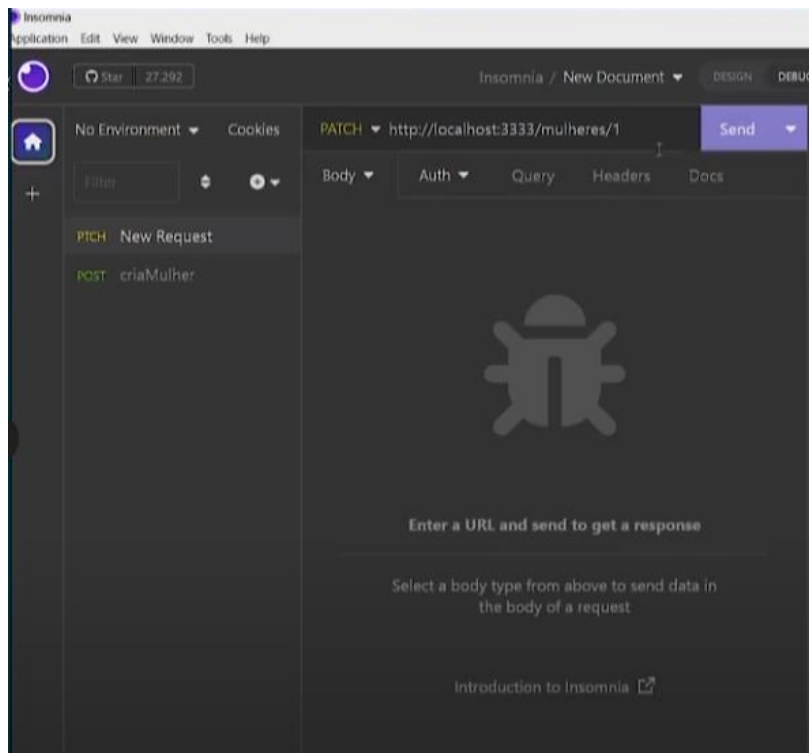




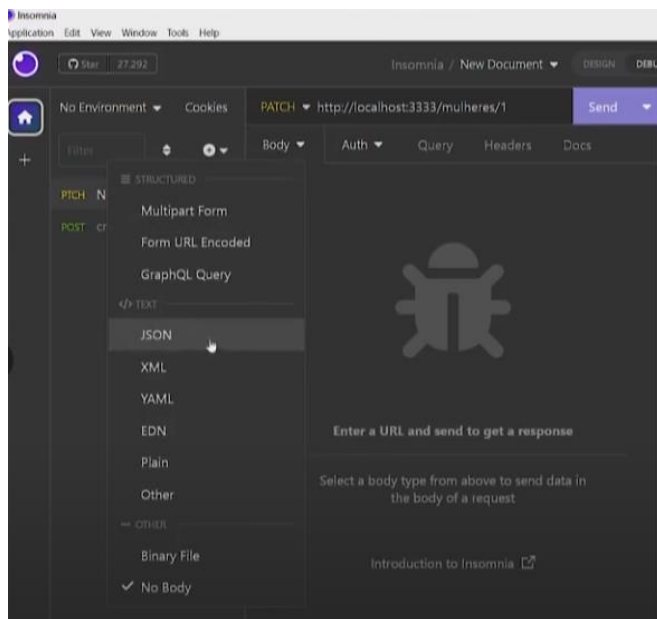
Em seguida mude de GET para **PATCH**:



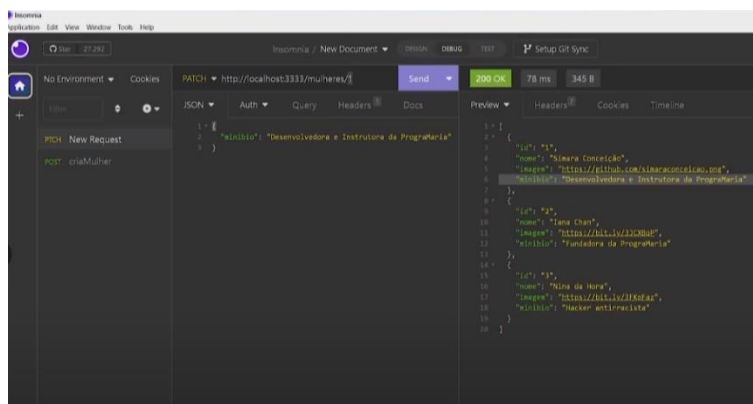
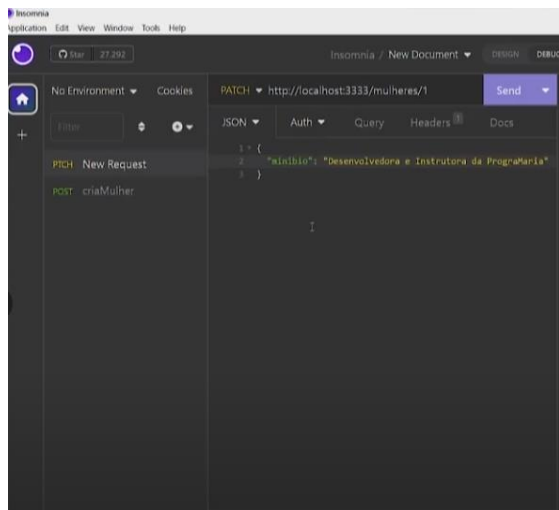
Adicione o **mesmo link com o número do id da mulher a ser corrigida**. No caso estamos corrigindo a **número 1**:



Em **Body**, selecione a opção **Json**:



Monte o objeto com o que deve ser corrigido e em seguida clique em **Send** (enviar). Repare na correção feita:



E se quiséssemos deletar alguma mulher cadastrada, como seria? 🤖

DELETA /mulheres/:id

Ao acessar o endereço `localhost:3333/mulheres/:id`, ao preencher o id da mulher no final da rota, esperamos o resultado da lista atual mantendo todas as outras mulheres e excluindo a indicada na rota

Criando nosso primeiro DELETE



Passo 1: Configurar uma rota que é um caminho na URL

Passo 2: Definir a função **deletaMulher()** que permite o envio do id na rota e produz a resposta a essa requisição



Configuração da rota



É necessário manter toda a estrutura de criação do servidor que aprendemos nas aulas passadas e vamos incluir as informações de configuração da rota POST /mulheres.

A primeira etapa da configuração da rota só precisa ser feita uma vez e já fizemos no GET

```
app.use(router.delete('/mulheres/:id', deletaMulher))
```

Agora estamos dizendo para o aplicativo responder o verbo DELETE na rota /mulheres/:id e chamar a função deletaMulher

Função deletaMulher()



```
function deletaMulher(request, response) {  
  function todasMenosEla(mulher) {  
    if (mulher.id !== request.params.id) {  
      return mulher  
    }  
  }  
}
```

Função que cria uma condição para retornar a mulher que deve ser deletada da lista

```
const mulheresQueFicaram = listaDeMulheres.filter(todasMenosEla)  
response.json(mulheresQueFicaram)  
}
```

envio da resposta

Função que serve para filtrar uma lista atendendo a uma condição



Vamos pensar na lógica da exclusão de uma mulher cadastrada indevidamente?

✓ Precisamos apontar uma informação única para identificar qual mulher queremos deletar, sim, você percebeu mais um padrão: vai ser bem parecido com a aula de PATCH, mas em vez da função find, que encontra uma mulher, vamos usar a filter que vai filtrar e retornar a lista de mulheres que não serão deletadas, ou seja, que permanecem após a requisição de deletar.

Desta vez vamos fazer uma comparação utilizando o sinal de diferente de `!=`, para solicitar que todas as mulheres que possuírem o id diferente do informado na requisição deverão permanecer, e vamos usar mais uma vez a condicional `if` para escrever essa lógica.

✓ Agora precisamos deletar o objeto encontrado

Pausa para o Javascript




Operador
!=
Esse é o operador que representa o "diferente de", podemos usá-lo para comparar valores no Javascript.

Função
filter()
É a função que serve para percorrer um array, filtrar e retornar uma nova lista de acordo com uma condição.

⚠ Se liga na sintaxe na demonstração

Anatomia da requisição



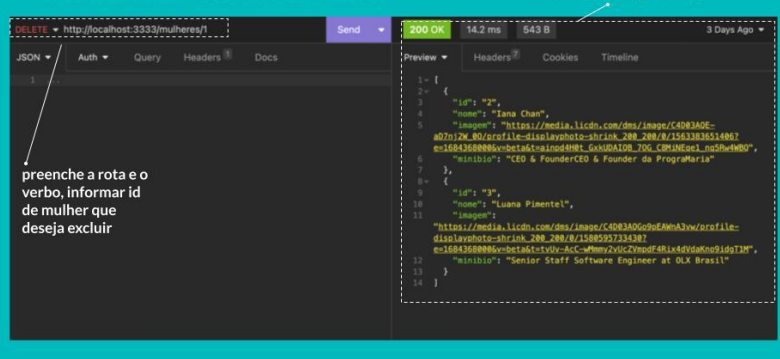
Request
⚠ Preencher o id da mulher que deseja excluir no final endereço

Endereço (params)
localhost:3333/mulheres/id

Response
Array das mulheres excluindo a mulher deletada

Testando no Insomnia

Resultado: código e array

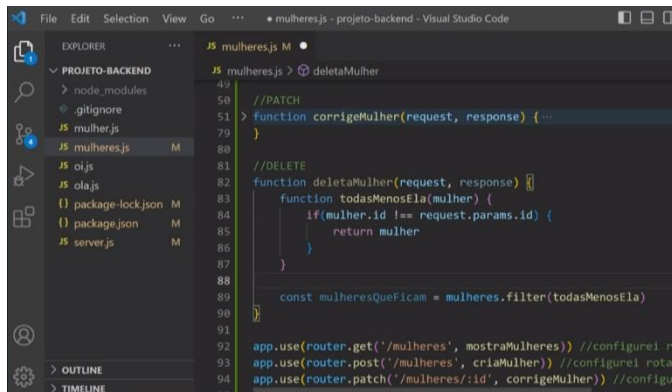


preenche a rota e o verbo, informar id de mulher que deseja excluir

```
DELETE http://localhost:3333/mulheres/1
```

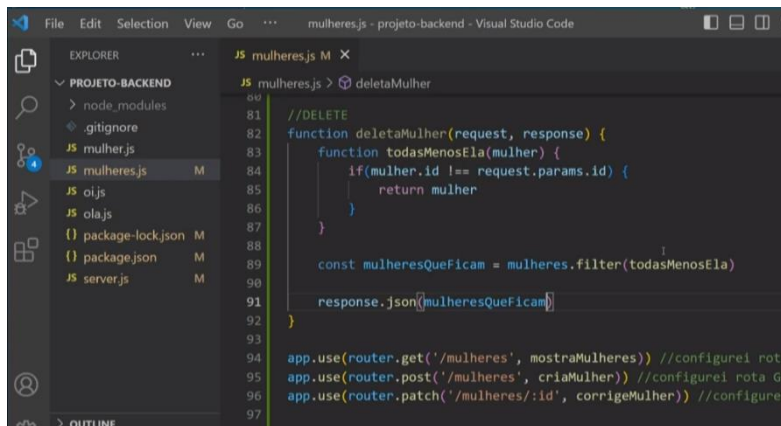
```
{
  "id": "2",
  "name": "Joao Chen",
  "image": "https://media.licdn.com/dms/image/C4083ADE-a07n12w_80/profile-displayphoto-shrink_200_200/0/1563383651486?e=1684368000&v=beta&token=6x50A108_70G_CPMANEst_n5Shw4W0",
  "minibio": "CEO & Founder CEO & Founder da Programaria",
  "id": "3",
  "name": "Luana Pimentel",
  "image": "https://media.licdn.com/dms/image/C40830G6o5dAmA3w/profile-displayphoto-shrink_200_200/0/1568959733430?e=1684368000&v=beta&token=AcC-uMmz2vUcZ/modF48Lx4dVdaKou9IdpTJM",
  "minibio": "Senior Staff Software Engineer at OLX Brasil"
}
```

Adicione a **função DELETE** e o **If (se)**:



```
49 //PATCH
50 //DELETE
51 > function corrigeMulher(request, response) { ...
79 }
80
81 //DELETE
82 function deletaMulher(request, response) {
83   function todasMenosEla(mulher) {
84     if(mulher.id !== request.params.id) {
85       return mulher
86     }
87   }
88
89   const mulheresQueFicam = mulheres.filter(todasMenosEla)
90 }
91
92 app.use(router.get('/mulheres', mostraMulheres)) //configurei rota GET /mulheres
93 app.use(router.post('/mulheres', criaMulher)) //configurei rota POST /mulheres
94 app.use(router.patch('/mulheres/:id', corrigeMulher)) //configurei rota PATCH /mulheres/:id
```

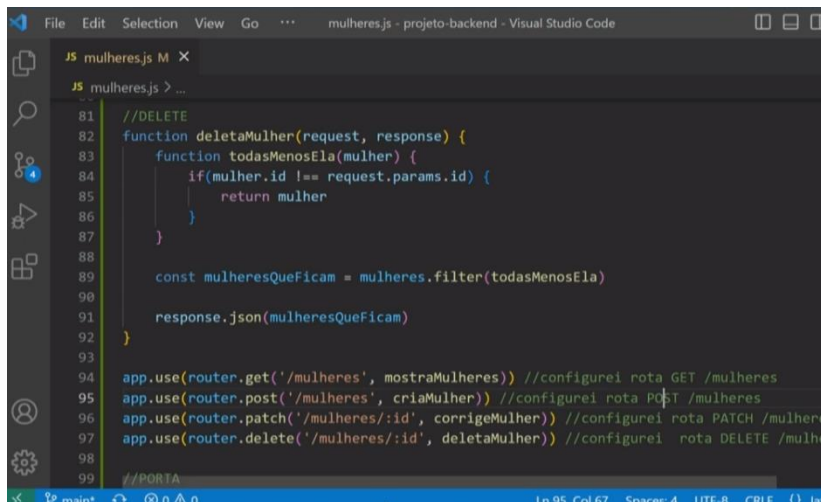
Adicione o `response.json`



```
81 //DELETE
82 function deletaMulher(request, response) {
83   function todasMenosEla(mulher) {
84     if(mulher.id !== request.params.id) {
85       return mulher
86     }
87   }
88
89   const mulheresQueFicam = mulheres.filter(todasMenosEla)
90
91   response.json(mulheresQueFicam)
92 }
93
94 app.use(router.get('/mulheres', mostraMulheres)) //configurei rota GET /mulheres
95 app.use(router.post('/mulheres', criaMulher)) //configurei rota POST /mulheres
96 app.use(router.patch('/mulheres/:id', corrigeMulher)) //configurei rota PATCH /mulheres/:id
97 app.use(router.delete('/mulheres/:id', deletaMulher)) //configurei rota DELETE /mulheres/:id
```

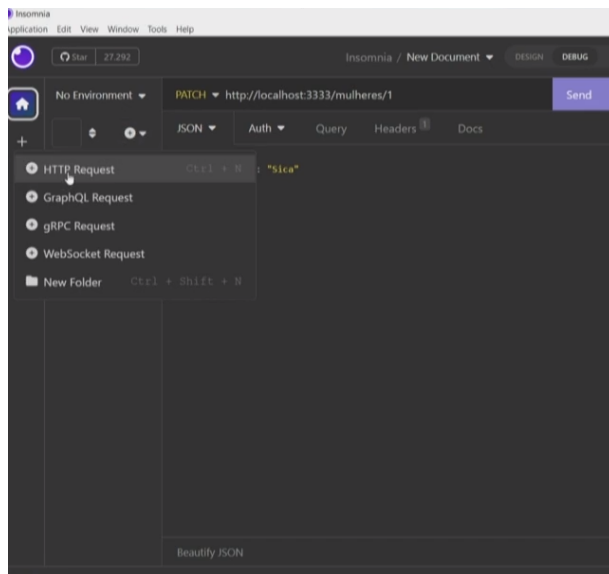
Adicione a **rota**:

⚠ **Lembre-se de salvar suas alterações!**

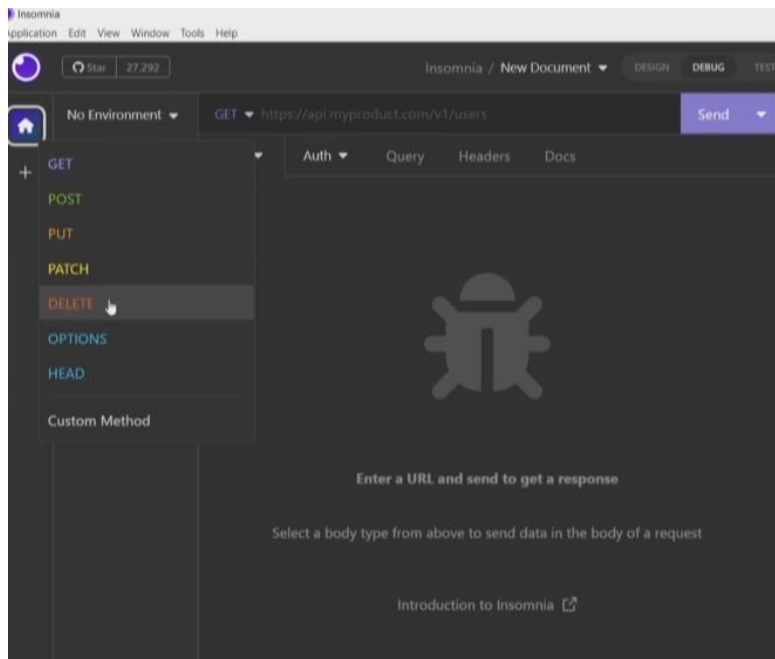


```
81 //DELETE
82 function deletaMulher(request, response) {
83   function todasMenosEla(mulher) {
84     if(mulher.id !== request.params.id) {
85       return mulher
86     }
87   }
88
89   const mulheresQueFicam = mulheres.filter(todasMenosEla)
90
91   response.json(mulheresQueFicam)
92 }
93
94 app.use(router.get('/mulheres', mostraMulheres)) //configurei rota GET /mulheres
95 app.use(router.post('/mulheres', criaMulher)) //configurei rota POST /mulheres
96 app.use(router.patch('/mulheres/:id', corrigeMulher)) //configurei rota PATCH /mulheres/:id
97 app.use(router.delete('/mulheres/:id', deletaMulher)) //configurei rota DELETE /mulheres/:id
98
99 //PORTA
```

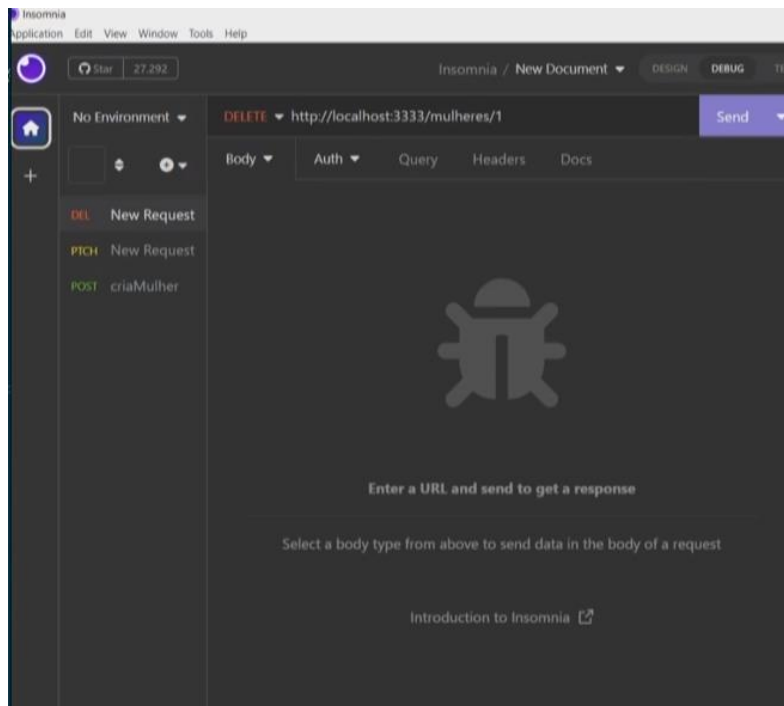
Na **ferramenta insomnia**, clique no + e em **HTTP Request**:



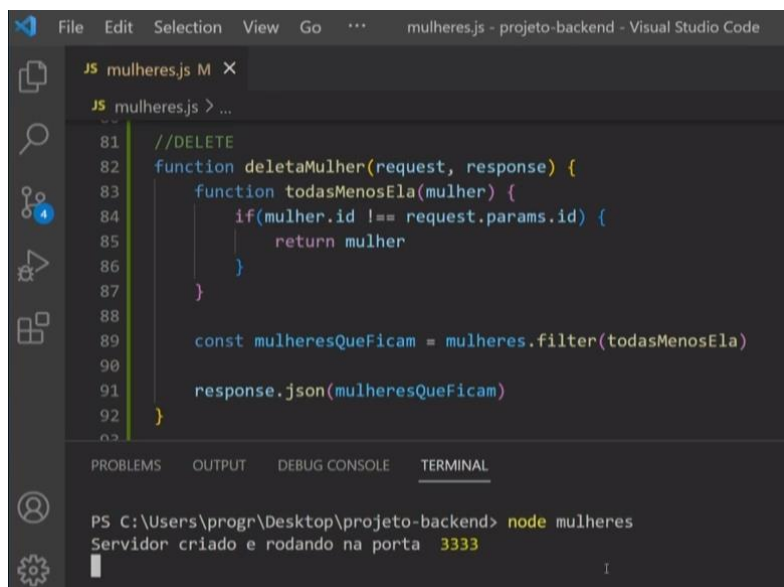
Mude o verbo para **DELETE**:



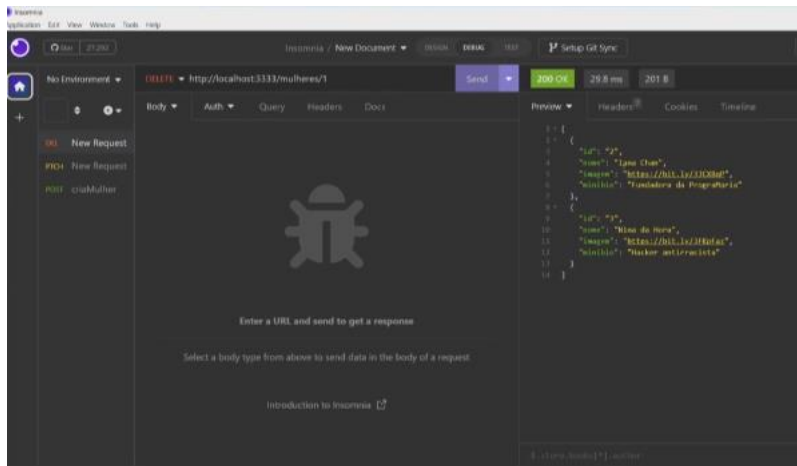
Adicione o **link** com o **número do id da mulher a ser deletada**:



Volte no VSCODE e rode **node mulheres.js** no terminal:



De volta a ferramenta Insomnia clique em **Send (enviar)** e veja a alteração:



Acabamos de criar a primeira versão do nosso projeto, além disso aprendemos com a mão no código alguns conceitos dentro do javascript, treinamos nosso pensamento computacional e aprendemos também na prática sobre o conjunto de regras do HTTP.



Para ajudar a entender o código, fizemos uma prática que pessoas desenvolvedoras utilizam em seus códigos: incluir mensagens para nós seres humanos e que o computador ignora enquanto código.

Os comentários são excelentes para nos guiar durante os estudos. Para comentar seu código, usamos duas barras //, O texto que vem depois é um comentário para nós seres humanos, ele até fica em branco, indicando que este trecho é um comentário e será ignorado! Vamos usar mais!

Olha quanta coisa legal a gente aprendeu

- Sempre dividimos nosso problema em pequenos passos lógicos
- Padrão em todas as chamadas: verbo, request, response, endereço, uma lógica pra permitir receber os dados da request e enviar resposta
- Ao final de cada código sempre testamos
- Nossa configuração de servidor é feita apenas uma vez
- Toda vez que usamos um verbo precisamos permitir o app usar
- Sempre precisa rodar nosso servidor pra testar
- O único que permite testar no navegador é o GET, o resto é numa ferramenta como Insomnia

Batizando nosso projeto

Acabamos de fazer nossa primeira versão de API. Mas o que é API?

Interface entre aplicativos e programação. Se uma interface de um front é criada para o usuário final, a API é desenvolvida para que um sistema possa usar as funcionalidades de outro sistema. Interface ideal para que um sistema se comunique com outro sistema. Ou seja, criamos nosso back-end para que o front-end possa usá-lo! 💖