

Criando nossa primeira aplicação no Sequelize

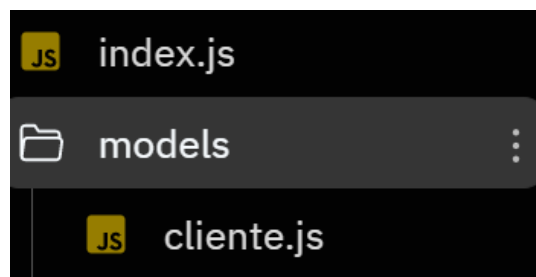
Iniciando as rotas e o create.

Para começar, vamos instalar via shell o **ejs**. O **ejs** dará várias funções ao nosso código html, sendo a principal o retorno de variáveis do node para nossas páginas html e a configuração para o nosso modelo com o uso de **views**.

Para instalar utilizamos o seguinte comando:

```
npm install ej
```

Vamos criar uma pasta chamada **models** e coloque o arquivo cliente.js dentro dela.



Após a alteração do lugar do arquivo **cliente.js**, precisamos também fazer a alteração onde ele está sendo referenciado.

```
//Criando o banco de dados
(async () => {
  const database = require('./db');
  const Cliente = require('./cliente');
  try {
    const resultado = await database.sync();
    console.log(resultado);
  } catch (error) {
    console.log(error);
  }
})
```

Altere para o caminho:

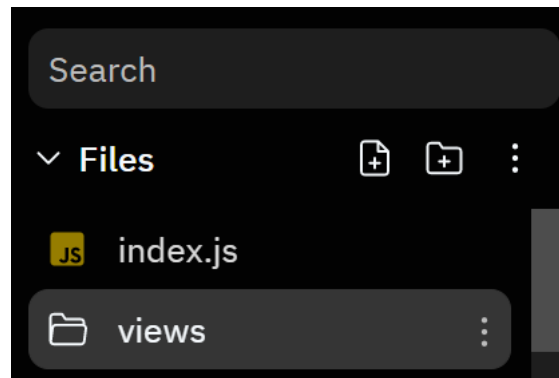
```
const Cliente = require('./models/cliente');
```

```
//Criando o banco de dados
(async () => {
  const database = require('./db');
  const Cliente = require('./models/cliente');
  try {
    const resultado = await database.sync();
    console.log(resultado);
  }
}
```

No arquivo cliente.js, coloque mais um ponto no caminho do arquivo db.js.

```
const Sequelize = require('sequelize');
const database = require('../db');
```

O próximo passo é criar uma pasta chamada **views** dentro do nosso projeto. A pasta views será responsável por todas as nossas páginas que serão mostradas para nosso usuário.



É importante que você guarde bem o nome views, ela faz parte de uma metodologia de desenvolvimento que iniciarei agora com você a **MVC**.

MVC é a sigla **M**odel, **V**iew e **C**ontroller, já utilizamos o model e agora vamos utilizar a view.

Vamos falar bastante ainda durante o curso sobre o assunto, mas precisamos ir seguindo um passo a passo para entendermos bem o conceito.

Dentro da pasta views, vamos criar uma página HTML chamada **formCliente.html**.

```
<!DOCTYPE html>
<html>

<head>
```

```

<meta charset="utf-8">
<meta name="viewport" content="width=device-width">
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-
alpha3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-
KK94CHFLLe+nY2dmCWGMq91rCGa5gtU4mk92HdvYe+M/SXH301p5ILy+dN9+nJO
Z" crossorigin="anonymous">
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-
alpha3/dist/js/bootstrap.bundle.min.js" integrity="sha384-
ENjdO4Dr2bkBIFxQpeoTz1Hlcje39Wm4jDKdf19U8gl4ddQ3GYNS7NTKfAdVQSZe"
crossorigin="anonymous"></script>
<title>Teste bootstrap 5replit</title>

</head>

<body>
<center><H4><p>Cadastro de Clientes</p></h4></center>

<form action="/addcliente" method="POST">
<div class="row justify-content-center">
<div class="col-8">
<label for="nome2" class="form-label">Nome:</label>
<input type="text" name="nome" class="form-control" id="nome">
</div>
<div class="col-8">
<label for="nascimento" class="form-label">Data de nascimento</label>
<input type="text" name="nascimento" class="form-control" id="nascimento"
aria-describedby="nascimentoHelp">
<small id="nascimentoHelp" class="form-text text-muted">Por favor, preencha
a data no formato AAAA-MM-DD</small>
</div>
<div class="col-8">
<label for="cidade" class="form-label">Cidade:</label>
<input type="text" name="cidade" class="form-control" id="cidade">
</div>
<div class="col-8">
<label for="telefone" class="form-label">Telefone:</label>
<input type="text" name="telefone" class="form-control" id="telefone">
</div>
<div class="col-8">
<button type="submit" class="btn btn-primary">Cadastrar</button>
</div>
</div>
</form>
</body>
</html>

```

A **formCliente.html** será responsável pelo cadastro dos dados via formulário dos clientes.

Pronto, criamos nossa página HTML, agora vamos criar nosso servidor e configurar nosso index.js com a rota que será utilizada.

No arquivo index.js, vamos criar nosso servidor node.js.

```

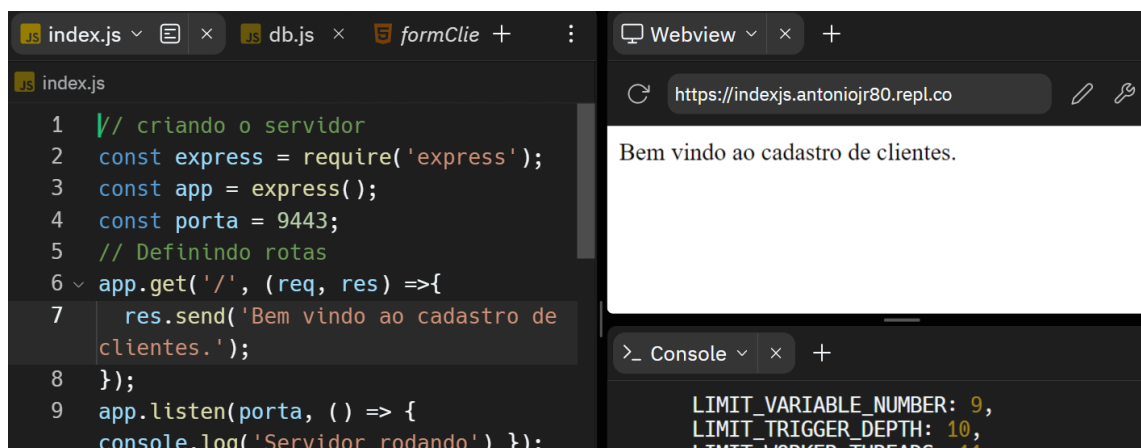
// criando o servidor
const express = require('express');
const app = express();
const porta = 9443;
// Definindo rotas

app.get('/', (req, res) =>{
  res.send('Bem vindo ao cadastro de clientes.');
```

```

});
app.listen(porta, () => { console.log('Servidor rodando') });
```

Vamos executar para ver se o servidor está funcionando.



The screenshot shows a code editor with the following code in index.js:

```

1 // criando o servidor
2 const express = require('express');
3 const app = express();
4 const porta = 9443;
5 // Definindo rotas
6 app.get('/', (req, res) =>{
7   res.send('Bem vindo ao cadastro de
   clientes.');
```

```

8   });
9   app.listen(porta, () => {
   console.log('Servidor rodando') });
```

The browser window shows the URL <https://indexjs.antoniojr80.repl.co> and the response "Bem vindo ao cadastro de clientes." The console shows the following output:

```

LIMIT_VARIABLE_NUMBER: 9,
LIMIT_TRIGGER_DEPTH: 10,
LIMIT_WORKER_THREADS: 11
```

Retornando ao código para criação do formulário de cadastro de clientes, após a criação da constante com a porta do servidor, vamos criar a constante **bodyParser**.

```
const bodyParser = require('body-parser');
```

```

const app = express();
const porta = 9443;
const bodyParser = require('body-parser');
```

A constante **bodyParser** informa que utilizaremos o body-parser.

O **bodyParser** é utilizado para capturar informações dos formulários.

Começamos a utilizar o processo de MVC em nossos projetos, já criamos dois que são o models e as views.

Sobre as views, devemos informar para a aplicação que utilizaremos views.

Ou seja, tudo que será mostrado no **front-end** será utilizado, compartilhado ou **renderizado**.

Renderizar é começar a juntar nosso back com o front, de uma maneira simples.

Vamos colocar o código abaixo depois da constante bodyParser.

```
// Setar os valores da view e engine
app.set('view engine', 'html');
app.engine('html', require('ejs').renderFile);
```

```
const bodyParser = require('body-parser');

// Setar os valores da view e engine
app.set('view engine', 'html');
app.engine('html', require('ejs').renderFile);
```

Vamos aproveitar e colocar as constantes database e cliente que estão dentro do nosso async do servidor para o início do nosso código.

Faremos isso, pois utilizaremos elas fora do async.

```
(async () => {
  const database = require('./db');
  const Cliente = require('./models/cliente');
```

index.js

```
1 // importando as bibliotecas e arquivos
2 const database = require('./db');
3 const Cliente = require('./models/cliente');
4 |
5 // criando o servidor
6 const express = require('express');
```

Criando as rotas do formulário e cadastro no banco.

Falando sobre o nosso formulário, todos os dados que são passados via formulários, precisam ser convertidos para que possamos utilizar para envio via create para o banco de dados.

```
app.use(bodyParser.json())
app.use(express.urlencoded({extended: true}));
```

Criando a rota para o formulário.

O comando para criação da rota para o formulário é mostrado abaixo:

```
app.get("/cadcliente", function(req, res) {
  res.render('formCliente');
});
```

Quando o usuário chamar /cadcliente, a página mostrada será formCliente.html.

Repare que quando no navegador, você colocar **/cadcliente**, utilizaremos o res.render pedindo para ele mostrar a página HTML **formCliente**. O render entenderá que estamos falando de uma view e automaticamente irá buscar a view formCliente na pasta **views**. Metodologia MVC sendo colocada em prática.

Pronto a rota do formulário foi criada. Agora, vamos fazer o comando para incluir no banco de dados as informações.

Repare que no form da página **formCliente.html** colocamos um action="/addcliente". Assim, toda vez que o formulário for executado, ele irá chamar a rota /addcliente.

A rota addcliente, será responsável pelo create da nossa aplicação.

```
app.post('/addcliente', function(req, res) {  
  Cliente.create({  
    nome: req.body.nome,  
    nascimento: req.body.nascimento,  
    cidade: req.body.cidade,  
    telefone: req.body.telefone  
  }).then(function(){  
    res.send("Cliente cadastrado com sucesso!")  
  })  
})
```

App.post é utilizado quando temos no formulário o method=post.

O método post deixa as informações ocultas para o usuário, ou seja não passa os dados via navegador.

O comando **Cliente.create** já é familiar para você, é o método de inserir registros no banco de dados.

No final do comando utilizamos o **.then(function())** para informar na tela da página que o registro foi inserido com sucesso.

```
.then(function(){  
  res.send("Cliente cadastrado com sucesso!")  
})
```

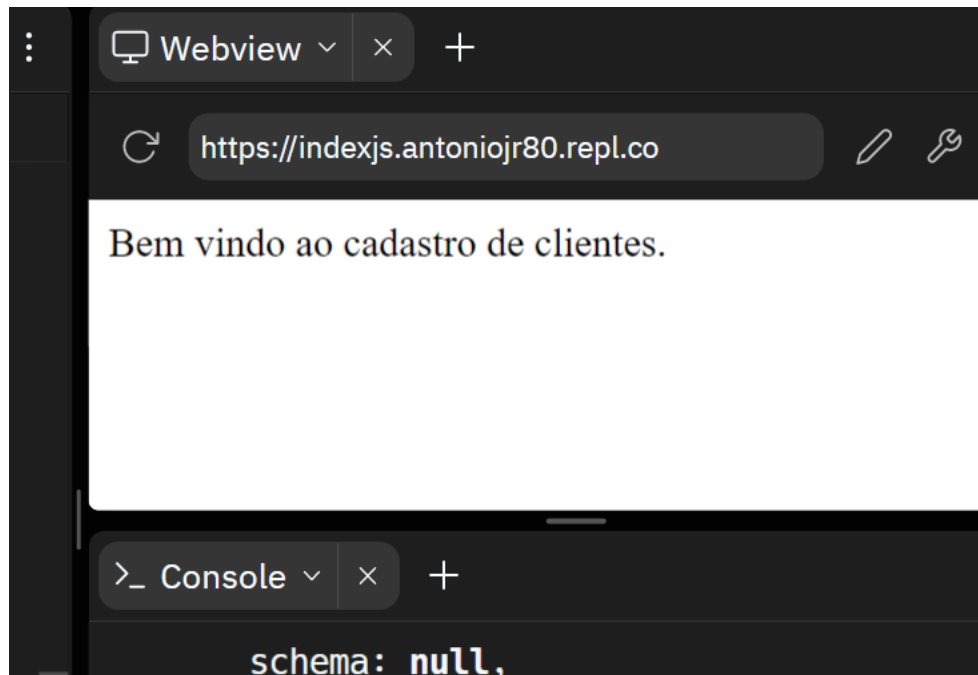
Agora dentro do async do banco de dados, crie o read para listar no console os dados cadastrados no banco.

Utilizaremos o findall().

```
const clientes = await Cliente.findAll();  
onsole.log("Lista de Clientes \n",clientes);
```

Vamos testar nosso código?

Execute o arquivo index.js.



Clique na opção Open in a new tab para abrimos no nosso navegador.

Coloque o /cadcliente



Faça uma cadastro, lembre que temos que preencher todos os campos e por enquanto nossa data de nascimento está no formato AAA-MM-DD,

Cadastro de Clientes

Nome:

Data de nascimento

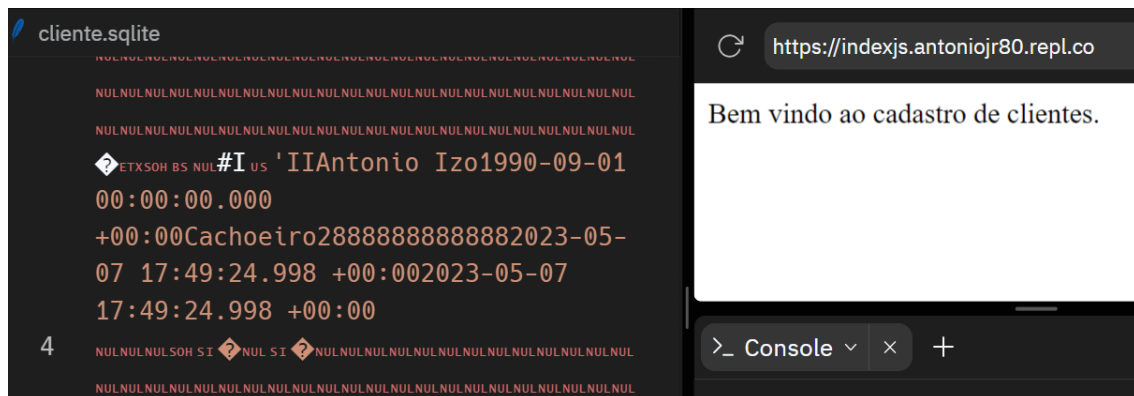
Por favor, preencha a data no formato AAAA-MM-DD

Cidade:

Telefone:

Cliente cadastrado com sucesso!

Vamos dar uma olhada no arquivo do banco para verificar se foi inserido o registro.



Vamos parar e executar nosso projeto novamente, repare que o registro será mostrado no findall do console.

```
Lista de Clientes
[
  cliente {
    dataValues: {
      id: 1,
      nome: 'Antonio Izo',
      nascimento: 1990-09-01T00:00:00.000Z,
      cidade: 'Cachoeiro',
      telefone: '2888888888888',
      createdAt: 2023-05-07T17:49:24.998Z,
      updatedAt: 2023-05-07T17:49:24.998Z
    }
  },
]
```

Finalizamos assim nossa terceira semana.