MongoDB é um banco de dados NoSQL, orientado a documentos JSON(-like), o mesmo formato de dados usado no JavaScript, além de ser utilizado através de funções com sintaxe muito similar ao JS também, diferente dos bancos SQL.
Ambos tem o JavaScript em comum e isso facilita bastante o uso destas duas tecnologias em conjunto, motivo de ser uma dupla bem frequente para projetos de todos os tamanhos.
Nossa ideia é justamente lhe dar um primeiro contato prático com ambos, fazendo o famoso CRUD (Create, Retrieve, Update e Delete).

Neste projeto vamos trabalhar:

1. Configurando o Node.js
2. Configurando o MongoDB
3. Conectando no MongoDB com Node
4. Cadastrando no banco

Passo 1: Configuração do Ambiente

- Criando o projeto

Crie uma pasta para o projeto:

```
mkdir cadastro-clientes
cd cadastro-clientes
```

Vamos iniciar o projeto através do npm:

```
npm init -y
```

- Instalar dependências

Precisamos do Express (para o backend), Mongoose (para a conexão com o MongoDB), e algumas bibliotecas adicionais.

```
npm install express mongoose body-parser
npm install --save-dev typescript ts-node @types/express @types/node nodemon
```

Agora, vamos Configurar o TypeScript. Gere o arquivo de configuração tsconfig.json com o comando:

```
npx tsc --init
```

- Vamos configurar o script de inicialização

Edite o `package.json` para adicionar um script de inicialização com Nodemon:

```
"scripts": {
  "start": "nodemon src/index.ts"
}
```

Passo 2: Criar um Banco de Dados no MongoDB Compass

- Abrir o MongoDB Compass:

Inicie o MongoDB Compass no seu computador, e vamos criar o Banco de Dados:

Após conectar, clique no botão "Create Database" no canto superior esquerdo.

Escolha o nome do banco de dados e da coleção inicial (no caso, pode ser algo como clientes).

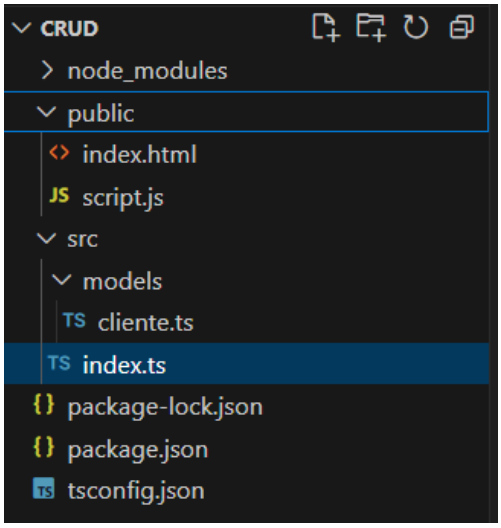
- Exemplo de nome do banco: crud_clientes
- Nome da coleção: clientes

Clique em "Create Database".

Agora você tem um banco de dados local criado e está pronto para conectar seu projeto a ele.

Passo 3: Configuração do Ambiente

Vamos criar a seguinte estrutura do nosso projeto:



- Vamos verificar o Compilador TypeScript:

Às vezes, o TypeScript pode não encontrar o módulo corretamente se o caminho estiver errado ou se a configuração `tsconfig.json` estiver errada. Garanta que o arquivo `tsconfig.json` tenha as seguintes opções mínimas:

```
{
  "compilerOptions": {
    "module": "commonjs",
    "target": "es6",
    "outDir": "./dist",
    "rootDir": "./src",
    "esModuleInterop": true
  },
  "include": ["src/**/*.ts"],
  "exclude": ["node_modules"]
}
```

Em src, no arquivo `index.ts` e adicione o seguinte código para configurar a conexão com o MongoDB:

```
import express from 'express';
import mongoose from 'mongoose';
import bodyParser from 'body-parser';

const app = express();
app.use(bodyParser.json());

const PORT = 3000;
const MONGODB_URI = 'mongodb://localhost:27017/crud_clientes';

mongoose.connect(MONGODB_URI)
    .then(() => console.log('MongoDB conectado'))
    .catch(err => console.log('Erro ao conectar ao MongoDB:', err));

app.listen(PORT, () => {
    console.log(`Servidor rodando em http://localhost:${PORT}`);
});

```

Passo 4: Criar o Model para Clientes

No arquivo `models/cliente.ts` vamos adicionar o schema do MongoDB para o cliente:

```
import mongoose from 'mongoose';

const ClienteSchema = new mongoose.Schema({
    nome: { type: String, required: true },
    email: { type: String, required: true, unique: true }
});

const Cliente = mongoose.model('Cliente', ClienteSchema);
export default Cliente;

```

Passo 5: Criar as Rotas para o um futuro CRUD

No arquivo `src/index.ts`, adicione as rotas para criar, ler, atualizar e deletar clientes:

```
// Criar cliente
app.post('/clientes', async (req, res) => {
    const { nome, email } = req.body;
    try {
        const novoCliente = new Cliente({ nome, email });
        await novoCliente.save();
        res.status(201).json(novoCliente);
    } catch (error) {
        res.status(400).json({ error: 'Erro ao criar cliente' });
    }
});
```

```
// Ler todos os clientes
app.get('/clientes', async (req, res) => {
    try {
        const clientes = await Cliente.find();
        res.json(clientes);
    } catch (error) {
        res.status(500).json({ error: 'Erro ao buscar clientes' });
    }
});
```

```
// Atualizar cliente
app.put('/clientes/:id', async (req, res) => {
    const { id } = req.params;
    const { nome, email } = req.body;
    try {
        const clienteAtualizado = await Cliente.findByIdAndUpdate(id, { nome, email }, { new: true });
        res.json(clienteAtualizado);
    } catch (error) {
        res.status(400).json({ error: 'Erro ao atualizar cliente' });
    }
});
```

```
// Deletar cliente
app.delete('/clientes/:id', async (req, res) => {
    const { id } = req.params;
    try {
        await Cliente.findByIdAndDelete(id);
        res.status(204).send();
    } catch (error) {
        res.status(500).json({ error: 'Erro ao deletar cliente' });
    }
});
```

Passo 6: Criar a Interface Visual

Vamos usar o HTML simples e o JavaScript para a interface visual.

No arquivo `public/index.html`:

```html
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Cadastro de Clientes</title>
</head>
<body>
    <h1>Cadastro de Clientes</h1>
    <form id="form-cliente">
        <input type="text" id="nome" placeholder="Nome" required />
        <input type="email" id="email" placeholder="Email" required />
        <button type="submit">Cadastrar</button>
    </form>

    <h2>Lista de Clientes</h2>
    <ul id="clientes-lista"></ul>

    <script src="script.js"></script>
</body>
</html>
```

- Criar o script frontend:

No arquivo `public/script.js`, adicione o código para interagir com o backend:

```javascript
const form = document.getElementById('form-cliente');
const listaClientes = document.getElementById('clientes-lista');

// Carregar clientes
async function carregarClientes() {
    const resposta = await fetch('/clientes');
    const clientes = await resposta.json();
    listaClientes.innerHTML = '';
    clientes.forEach(cliente => {
        const li = document.createElement('li');
        li.textContent = `${cliente.nome} - ${cliente.email}`;
        listaClientes.appendChild(li);
    });
}
```

```javascript
// Adicionar cliente
form.addEventListener('submit', async (e) => {
    e.preventDefault();
    const nome = document.getElementById('nome').value;
    const email = document.getElementById('email').value;

    const resposta = await fetch('/clientes', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ nome, email })
    });

    if (resposta.ok) {
        carregarClientes();
    }
});

carregarClientes();
```

Passo 7: Servir Arquivos Estáticos

No `index.ts`, adicione a linha para servir a pasta pública:

```
app.use(express.static('public'));
```

Passo 8: Atualizar o Frontend para Alterar e Excluir

No arquivo `public/index.html`, adicione botões para editar e excluir cada cliente da lista:

```html
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Cadastro de Clientes</title>
</head>
<body>
    <h1>Cadastro de Clientes</h1>
    <form id="form-cliente">
        <input type="text" id="nome" placeholder="Nome" required />
        <input type="email" id="email" placeholder="Email" required />
        <button type="submit">Cadastrar</button>
        <button type="button" id="atualizar" style="display: none;">Atualizar Cliente</button>
    </form>

    <h2>Lista de Clientes</h2>
    <ul id="clientes-lista"></ul>

    <script src="script.js"></script>
</body>
</html>
```

Passo 9: Rodar o Projeto

Agora que o banco de dados está configurado, rode o servidor:

```
npm start
```