Iremos criar neste tutorial um CRUD de cadastro de livros.

Neste projeto vamos trabalhar:

1. Configurando o Node.js
2. Configurando o MongoDB
3. Conectando no MongoDB com Node
4. Cadastrando no banco
5. Atualizando livros
6. Excluindo livros

Passo 1: Criação do Projeto em TypeScript com MongoDB

- Crie uma nova pasta para o projeto:

```
mkdir crud-livros
cd crud-livros
```

- Inicialize o projeto com o npm:

```
npm init -y
```

- Instale as dependências necessárias:

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

npm install express mongoose body-parser cors

- express: Um framework web popular para Node.js, utilizado para criar servidores HTTP e definir rotas para suas aplicações.
- mongoose: Uma biblioteca ODM (Object Data Modeling) para MongoDB. Ela permite que você interaja com o banco de dados MongoDB de uma forma mais orientada a objetos.
- body-parser: Um middleware que analisa o corpo das requisições HTTP. Isso é útil para extrair dados de formulários HTML ou JSON.
- cors: Um middleware que permite que sua aplicação receba requisições de outros domínios. Isso é crucial para aplicações que precisam se comunicar com diferentes serviços na web.

npm install typescript ts-node nodemon @types/node @types/express --save-dev

- typescript: Um transpiler que converte código TypeScript em JavaScript.
- ts-node: Uma ferramenta que permite executar diretamente código TypeScript sem a necessidade de pré-compilação.
- nodemon: Um utilitário que reinicia automaticamente seu servidor Node.js quando ocorrem alterações no código.
- @types/node: Declarações de tipos para a biblioteca padrão do Node.js, fornecendo informações sobre tipos e funções para o TypeScript.
- @types/express: Declarações de tipos para o framework Express, facilitando o desenvolvimento de código TypeScript com Express.
- --save-dev: indica que esses pacotes são dependências de desenvolvimento, ou seja, são necessários para o desenvolvimento e teste da sua aplicação, mas não são necessários para a execução da aplicação final.

- Configure o TypeScript:

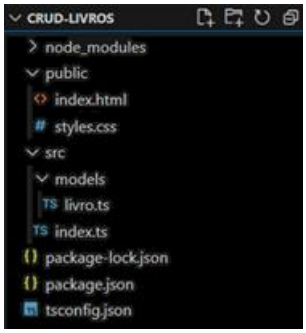
```
npx tsc –init
```

Isso gerará um arquivo `tsconfig.json`. Certifique-se de que as opções abaixo estejam habilitadas no arquivo:

```
"target": "es6",
"module": "commonjs",
"rootDir": "./src",
"outDir": "./dist",
"strict": true,
"esModuleInterop": true,
```

Passo 2: Estrutura de Diretórios

Crie a seguinte estrutura de diretórios e arquivos:



Passo 3 – Criando o Banco de Dados no MongoDB Compass

- Abra o MongoDB Compass no seu computador.
- Clique no botão Connect para conectar ao MongoDB local.

- Criar um Novo Banco de Dados:
    - Após a conexão, você verá uma lista de bancos de dados existentes no lado esquerdo.
    - Na parte superior da interface, clique em "Create Database".

- Nomear o Banco de Dados e a Coleção:

Uma janela aparecerá pedindo para inserir o nome do novo banco de dados e uma coleção inicial.

- Nome do banco de dados: crud_livros
- Nome da coleção: livros

Clique em Create Database.

- Verificar o Banco de Dados Criado:

Agora, você verá o novo banco de dados crud_livros na lista de bancos de dados no lado esquerdo da tela. Clicando nele, você verá a coleção livros, que é onde os dados dos livros que você cadastrar no CRUD serão armazenados.

Passo 4: Configurar o Backend (Express, Mongoose)

- Configurar o index.ts:

No arquivo `src/index.ts`, adicione o seguinte código para configurar o servidor Express e a conexão com o MongoDB:

```ts
import express from 'express';
import mongoose from 'mongoose';
import bodyParser from 'body-parser';
import cors from 'cors';
import Livro from './models/livro';

const app = express();
const PORT = 3000;
const MONGODB_URI = 'mongodb://localhost:27017/crud_livros';

// Middleware
app.use(bodyParser.json());
app.use(cors());
app.use(express.static('public'));
app.use(express.json());

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

```

```ts
// Rota para cadastrar um livro
app.post('/livros', async (req, res) => {
    try {
      const novoLivro = new Livro({
        titulo: req.body.titulo,
        autor: req.body.autor,
        ano: req.body.ano
      });
      const livroSalvo = await novoLivro.save();
      res.status(201).json(livroSalvo); // Certifique-se de retornar o livro cadastrado
    } catch (error) {
      res.status(500).json({ message: 'Erro ao cadastrar livro', error });
    }
  });

// Rota para listar todos os livros
app.get('/livros', async (req, res) => {
    try {
        const livros = await Livro.find();
        res.json(livros);
    } catch (error) {
        res.status(500).json({ error: 'Erro ao buscar livros' });
    }
});

```

```
// Rota para atualizar um livro
app.put('/livros/:id', async (req, res) => {
    const { id } = req.params;
    const { titulo, autor, anoPublicacao } = req.body;
    try {
        const livroAtualizado = await Livro.findByIdAndUpdate(id, { titulo, autor, anoPublicacao },
            { new: true });
        if (!livroAtualizado) {
            return res.status(404).json({ error: 'Livro não encontrado' });
        }
        res.json(livroAtualizado);
    } catch (error) {
        res.status(400).json({ error: 'Erro ao atualizar livro' });
    }
});

// Rota para deletar um livro
app.delete('/livros/:id', async (req, res) => {
    const { id } = req.params;
    try {
        const livroDeletado = await Livro.findByIdAndDelete(id);
        if (!livroDeletado) {
            return res.status(404).json({ error: 'Livro não encontrado' });
        }
        res.status(204).send();
    } catch (error) {
        res.status(500).json({ error: 'Erro ao deletar livro' });
    }
});
```

Passo 5 – Frontend

HTML (`public/index.html`):

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Cadastro de Livros</title>
  <link rel="stylesheet" href="styles.css">
</head>

<body>
  <h1>Cadastro de Livros</h1>
  <form id="formLivro">
    <input type="text" id="titulo" placeholder="Título" required /><br />
    <input type="text" id="autor" placeholder="Autor" required /><br />
    <input type="number" id="ano" placeholder="Ano" required /><br />
    <button type="submit">Cadastrar Livro</button>
  </form>

  <h2>Livros Cadastrados</h2>
  <div id="listaLivros" class="livros"></div>
```

```
<script>
  // Função para cadastrar livro
  async function cadastrarLivro(event) {
    event.preventDefault();

    const livro = {
      titulo: document.getElementById('titulo').value,
      autor: document.getElementById('autor').value,
      ano: parseInt(document.getElementById('ano').value),
    };

    const response = await fetch('http://localhost:3000/livros', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify(livro),
    });

    const data = await response.json();
    console.log('Livro cadastrado:', data);

```

```
    // Atualizar a lista de livros
    atualizarListaDeLivros();
  }

  // Função para exibir a lista de livros
  async function atualizarListaDeLivros() {
    const response = await fetch('http://localhost:3000/livros');
    const livros = await response.json();
    const listaLivros = document.getElementById('listaLivros');

    listaLivros.innerHTML = ''; // Limpar a lista antes de atualizá-la

    livros.forEach((livro) => {
      const div = document.createElement('div');
      div.className = 'livro-item';
      div.innerHTML = `
        <strong>${livro.titulo}</strong> - ${livro.autor} (${livro.ano})
        <button class="btn btn-editar" onclick="editarLivro('${livro._id}')">Alterar</button>
        <button class="btn btn-excluir" onclick="excluirLivro('${livro._id}')">Excluir</button>
      `;
      listaLivros.appendChild(div);
    });
  }
```

```
    // Função para editar livro
    async function editarLivro(id) {
      const novoTitulo = prompt("Novo título:");
      const novoAutor = prompt("Novo autor:");
      const novoAno = prompt("Novo ano:");

      if (novoTitulo && novoAutor && novoAno) {
        const livroAtualizado = {
          titulo: novoTitulo,
          autor: novoAutor,
          ano: parseInt(novoAno),
        };

        await fetch(`http://localhost:3000/livros/${id}`, {
          method: 'PUT',
          headers: {
            'Content-Type': 'application/json',
          },
          body: JSON.stringify(livroAtualizado),
        });

        atualizarListaDeLivros();
      }
    }

    // Função para excluir livro
    async function excluirLivro(id) {
      if (confirm('Tem certeza que deseja excluir este livro?')) {
        await fetch(`http://localhost:3000/livros/${id}`, {
          method: 'DELETE',
        });

        atualizarListaDeLivros();
      }
    }

    // Vincular o evento de submit do formulário à função de cadastro
    document.getElementById('formLivro').addEventListener('submit', cadastrarLivro);

    // Carregar a lista de livros ao carregar a página
    window.onload = atualizarListaDeLivros;
  </script>
</body>
</html>
```

- Verifique seu `package.json`

```
{
  "name": "crud-livros",
  "version": "1.0.0",
  "main": "index.js",
  ▷Debug
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "nodemon src/index.ts"
  },
```

Passo 6 – Criando o arquivo do modelo

- Modelo do Livro (`models/livro.ts`):

O arquivo `models/livro.ts` define o esquema do MongoDB para o livro:

```ts
import mongoose, { Schema, Document } from 'mongoose';

// Interface para o livro
export interface ILivro extends Document {
  titulo: string;
  autor: string;
  ano: number;
}

// Schema do livro
const LivroSchema: Schema = new Schema({
  titulo: { type: String, required: true },
  autor: { type: String, required: true },
  ano: { type: Number, required: true },
});

// Exporta o modelo
export default mongoose.model<ILivro>('Livro', LivroSchema);
```

Passo 7 – Criando o css da aplicação

- Criando o Estilo da aplicação (`public/styles.css`)

Personalize ao seu gosto a sua aplicação.

Passo 8 – Rodar o Projeto

Agora que o banco de dados está configurado, vamos compilar o projeto e depois vamos rodar o servidor:

`npx tsc` – Ao compilar o projeto, irá criar a parta dist.

`npm start`