

# Gerenciador de Bibliotecas


# Trabalho Final de Curso Back End

## Dupla: Anderson e Luiz





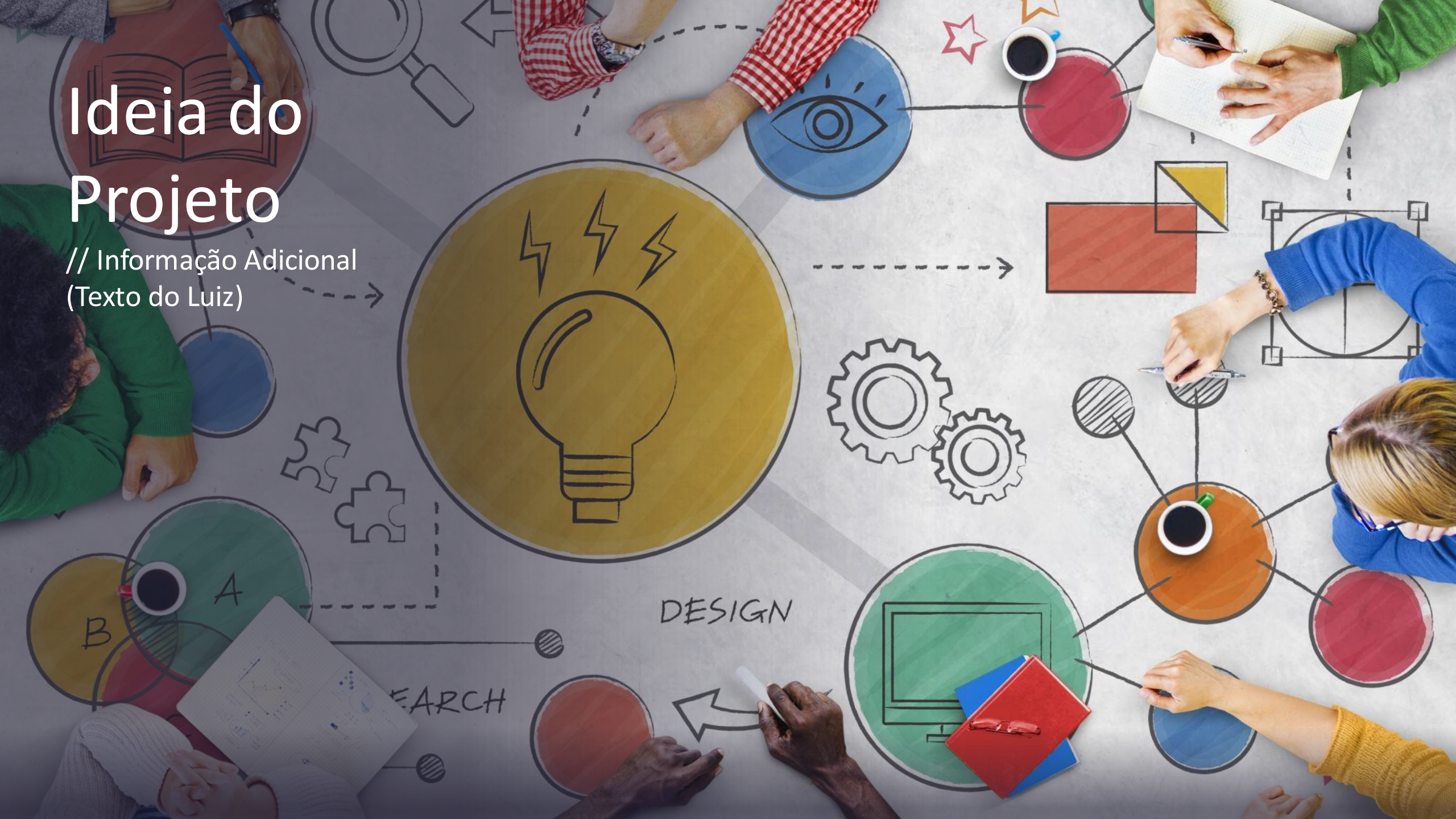
# Introdução

- Ideia do Projeto
    - Problema e Solução
    - Diagrama de Entidades
  - Linhas de Código
    - Explicação
    - Requisições
  - Referências
  - Contracapa do Trabalho
- 



# Ideia do Projeto

// Informação Adicional  
(Texto do Luiz)



# Descrição

Nosso trabalho final foi desenvolvido para melhorar o desempenho de uma biblioteca que deseja utilizar o Banco de Dados MySQL para armazenar os registros da entrada e saída de livros, através de uma API podemos definir rotas que realizem as operações CRUD (Create, Read, Update, Delete). Desta maneira o atendente da biblioteca pode facilmente monitorar os dados e atualiza-los sempre que necessário para que possa obter um resultado satisfatório em seu dia de trabalho.

// Explicação do CRUD (Texto do Luiz)

//

//



# Problema e Solução

// Explicação do Problema (Texto do Luiz)

//

//

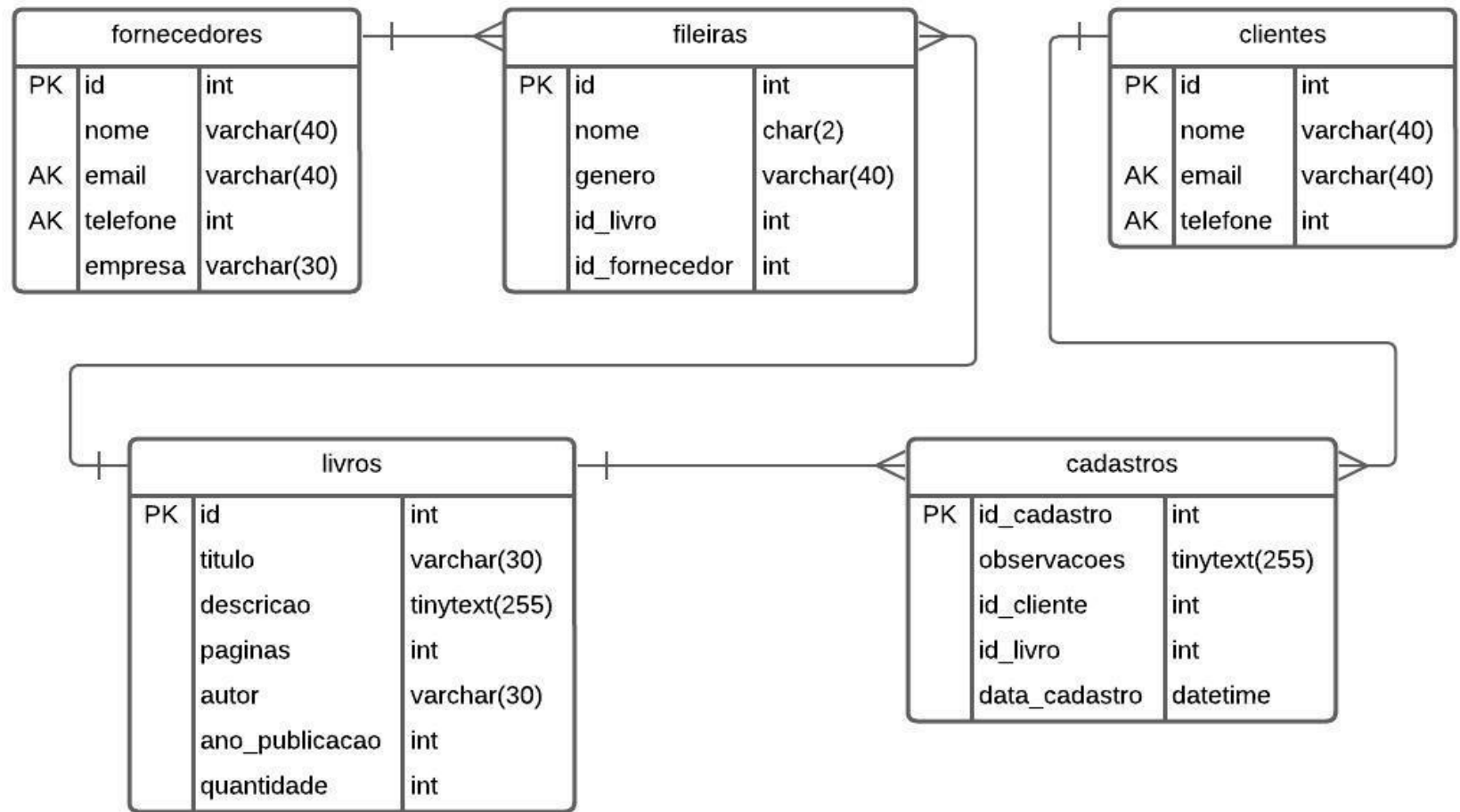
Por este motivo, propomos a substituição da ferramenta Excel feita para inserir registros em planilhas pelo Banco de Dados MySQL que vem a ser usado para o mesmo fim, armazenar de forma organizada em linhas e colunas diferentes tipos de dados. Em adição, temos que comentar sobre como este recurso permitiria a vantagem de agilidade na inserção de registros e uma busca eficiência devido as relações entre as tabelas pelas chaves estrangeiras, resultando em uma consulta totalmente precisa.





# Diagrama

Para maior entendimento de nosso trabalho, criamos um diagrama do Banco de Dados com as interações entre cada tabela.



# Linhas de Código

Esse gerenciador de bibliotecas foi desenvolvido com a linguagem Typescript usando recursos de bibliotecas do NPM (gerenciador de pacotes para NodeJS)

# Primeiros Passos

Para começar foi necessário instalar as dependências que serão usadas durante o desenvolvimento, estas que são 'express', 'typescript', 'ts-node-dev', 'mysql2', '@types/express' e '@types/node'.

- Express: Um framework web para NodeJS que facilita a criação de aplicações web e APIs, fornecendo um conjunto robusto de funcionalidades para desenvolvimento de servidores web.
- Typescript: Um superconjunto de Javascript que adiciona tipagem estática opcional ao código, ajudando a detectar erros durante o desenvolvimento e melhorando a manutenção do código.
- TS-Node-Dev: Uma ferramenta que combina TS-Node e Nodemon, permitindo executar e reiniciar automaticamente aplicações NodeJS escritas em Typescript sempre que há mudanças no código.
- MySQL2: Um cliente MySQL para NodeJS que oferece suporte a promessas e async/await, além de ser mais rápido e eficiente em comparação com o módulo MySQL original.
- @types/express: Pacote de definições de tipos para o Express, permitindo que você use o Express com Typescript e tenha suporte a tipagem estática.
- @types/node: Pacote de definições de tipos para NodeJS, fornecendo tipagem estática para as APIs do NodeJS quando usado com Typescript.



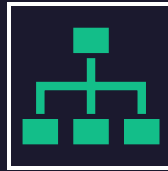
# Organização no Visual Studio Code



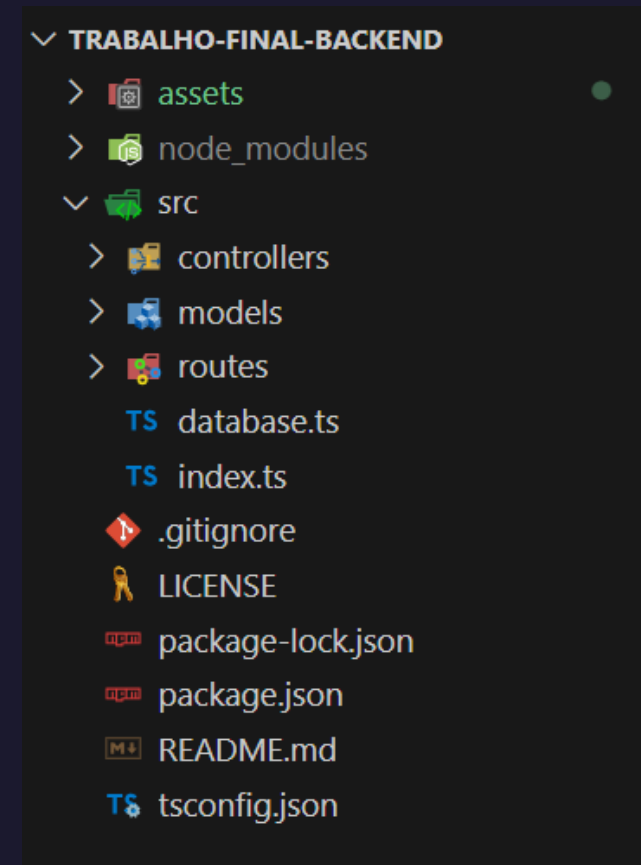
Depois de terem sido instaladas as bibliotecas prosseguimos com organização das pastas e arquivos.



As pastas 'node\_modules' e 'src' (ou seja, source) estarão em conjunto dos arquivos 'gitignore', 'tsconfig.json', 'package.json' e 'package-lock.json' na pasta 'trabalho-final-backend'.



As pastas 'controllers', 'models' e 'routes' estarão em conjunto dos arquivos 'index.ts' e 'database.ts' dentro da pasta src (pasta citada anteriormente).



# Pastas e Arquivos

// Explicação da Função de Cada Pasta e Arquivo (Parte do Luiz)

//

//

//

//

//

//





```
1  import { Router } from "express";
2  import { CadastrosController } from "../controllers/cadastrosController";
3
4  const router = Router();
5  router.get('/cadastros', CadastrosController.getAll);
6  router.get('/cadastros/:id', CadastrosController.getById);
7  router.post('/cadastros', CadastrosController.create);
8  router.put('/cadastros/:id', CadastrosController.update);
9  router.delete('/cadastros/:id', CadastrosController.delete);
10
11 export default router;
```

```

1 import { Request, Response } from 'express';
2 import { Cadastro } from '../models/cadastro';
3
4 export class CadastrosController{
5   static async getAll(req: Request, res: Response){
6     try {
7       const cadastros = await Cadastro.getAll();
8       res.status(200).json(cadastros);
9     } catch(error) {
10      res.status(500).json({'message': error});
11    }
12  }
13  static async getById(req: Request, res: Response){
14    try {
15      const { id } = req.params;
16      const cadastro = await Cadastro.getById(parseInt(id, 10));
17      res.status(200).json(cadastro);
18    } catch(error) {
19      res.status(404).json({'message': error});
20    }
21  }
22  static async create(req: Request, res: Response){
23    try {
24      const { observacoes, idCliente, idLivro, dataCadastro } = req.body;
25      const resultado = await Cadastro.create(observacoes, idCliente, idLivro, dataCadastro);
26      res.status(201).json(resultado);
27    } catch(error) {
28      res.status(500).json({'message': error});
29    }
30  }
31  static async update(req: Request, res: Response){
32    try {
33      const { id } = req.params;
34      const { observacoes, idCliente, idLivro, dataCadastro } = req.body;
35      const resultado = await Cadastro.update(parseInt(id, 10), observacoes, idCliente, idLivro, dataCadastro);
36      res.status(200).json(resultado);
37    } catch(error) {
38      res.status(500).json({'message': error});
39    }
40  }
41  static async delete(req: Request, res: Response){
42    try {
43      const { id } = req.params;
44      const resultado = await Cadastro.delete(parseInt(id, 10));
45      res.status(200).json(resultado);
46    } catch(error) {
47      res.status(500).json({'message': error});
48    }
49  }
50 }

```

```

1 static async getAll(req: Request, res: Response){
2   try {
3     const cadastros = await Cadastro.getAll();
4     res.status(200).json(cadastros);
5   } catch(error) {
6     res.status(500).json({'message': error});
7   }
8 }
9 static async getById(req: Request, res: Response){
10  try {
11    const { id } = req.params;
12    const cadastro = await Cadastro.getById(parseInt(id, 10));
13    res.status(200).json(cadastro);
14  } catch(error) {
15    res.status(404).json({'message': error});
16  }
17 }

```



```
1 import { db } from "../database";
2
3 export class Cadastro{
4   private _idCadastro: number;
5   private _observacoes: string;
6   private _idCliente: number;
7   private _idLivro: number;
8   private _dataCadastro: string;
9
10  constructor(idCadastro: number, observacoes: string, idCliente: number, idLivro: number, dataCadastro: string){
11    this._idCadastro = idCadastro;
12    this._observacoes = observacoes;
13    this._idCliente = idCliente;
14    this._idLivro = idLivro;
15    this._dataCadastro = dataCadastro;
16  }
17
18  static async getAll(){
19    try {
20      const [rows] = await db.query('SELECT * FROM Cadastros');
21      return rows;
22    } catch {
23      throw new Error('Erro ao buscar cadastros no banco de dados.');
```

```
24    }
25  }
26  static async getById(idCadastro: number){
27    try {
28      const [rows] = await db.query('SELECT * FROM Cadastros WHERE id_cadastro = ?', [idCadastro]);
29      return rows;
30    } catch {
31      throw new Error('Erro ao buscar cadastro no banco de dados.');
```

```
32    }
33  }
34  static async create(observacoes: string, idCliente: number, idLivro: number, dataCadastro: string){
35    try {
36      const [rows] = await db.query('INSERT INTO Cadastros (observacoes, id_cliente, id_livro, data_cadastro) VALUES (?, ?, ?, ?)', [observacoes, idCliente, idLivro, dataCadastro]);
37      return rows;
38    } catch {
39      throw new Error('Erro ao inserir cadastro no banco de dados.');
```

```
40    }
41  }
42  static async update(idCadastro: number, observacoes: string, idCliente: number, idLivro: number, dataCadastro: string){
43    try {
44      const [rows] = await db.query('UPDATE Cadastros SET observacoes = ?, id_cliente = ?, id_livro = ?, data_cadastro = ? WHERE id_cadastro = ?', [observacoes, idCliente, idLivro, dataCadastro, idCadastro]);
45      return rows;
46    } catch {
47      throw new Error('Erro ao atualizar cadastro no banco de dados.');
```

```
48    }
49  }
50  static async delete(idCadastro: number){
51    try {
52      const [rows] = await db.query('DELETE FROM Cadastros WHERE id_cadastro = ?', [idCadastro]);
53      return rows;
54    } catch {
55      throw new Error('Erro ao deletar cadastro no banco de dados.');
```

```
56    }
57  }
58 }
```

# Banco de Dados

// Explicação das Linhas do MySQL Workbench (Texto do Luiz) + Imagem

//

//

//

//

//


//



# Linhas de Código

Por exemplo, comentaremos um pouco sobre este bloco de código do arquivo 'cadastroController.ts'.

Nele podemos observar uma função estática e assíncrona com os parâmetros req e res.



```
1 static async update(req: Request, res: Response){
2   try {
3     const { id } = req.params;
4     const { observacoes, idCliente, idLivro, dataCadastro } = req.body;
5     const resultado = await Cadastro.update(parseInt(id, 10), observacoes, idCliente, idLivro, dataCadastro);
6     res.status(200).json(resultado);
7   } catch(error) {
8     res.status(500).json({'message': error});
9   }
10 }
```

Uma função estática permite que a função seja chamada mesmo dentro de uma classe e a característica assíncrona impede que o sistema pare após as consultas, que requerem um determinado tempo para serem concluídas pelo banco de dados. Também temos um try/catch para verificar se há algum erro e notificar com o código HTTP e a mensagem de correspondente. Por fim, com as variáveis constantes conseguimos armazenar os dados de requisição e usamos eles dentro dos parênteses para chamando a função update do cadastro (model).

No exemplo mostrado, exibimos em formato json o resultado do update e através do res o código HTTP de sucesso ('OK'), sendo que o parseInt dentro dos parênteses converte o resultado da busca em um inteiro.

# Requisições

// Exemplos de Requisições (Texto do Luiz) + Imagem

//

//

//

//

//

//





# Referências

## Google

- W3Schools (Site)
- SQL Server Tutorial (Site)

## Youtube

- Curso em Vídeo (Canal)
- Felipe Rocha (Canal)
- Dev Aprender (Canal)





Obrigado por sua atenção!

