# Sistema de Gestão para Biblioteca Municipal

## Descrição

Este projeto tem como objetivo o desenvolvimento de um sistema para gerenciar livros de uma biblioteca municipal. Ele permite o cadastro, consulta, edição e exclusão de livros, autores e categorias, facilitando a organização do acervo e o atendimento aos usuários.

## Tecnologias Utilizadas

- Back-end: Java 17, Spring Boot

- Front-end: HTML, CSS, JavaScript

- Banco de Dados: MySQL

- Ferramentas de Apoio: GitHub, Postman

## Estrutura do Projeto

```
gestaobibliotecamunicipal/
├── backend/
│   └── src/main/java/com/biblioteca/...
├── frontend/
│   ├── index.html
│   ├── script.js
│   └── style.css
├── database/
│   └── banco.sql
├── README.md
```

## Funcionalidades

- Cadastro de livros, autores e categorias

- Listagem de livros por título, autor ou categoria

- Atualização e exclusão de registros

- Integração com banco de dados relacional

- Interface web funcional e responsiva

- Testes com Postman

## Publicação do Front-end

O front-end está publicado e acessível via GitHub Pages:

***https://caursnn.github.io/gestaobibliotecamunicipal***

## Configuração e Execução Local

1. Requisitos

- Java 17
- MySQL
- Maven
- Navegador web
- Git (opcional)

2. Configurar o banco de dados

Crie o banco e execute o script banco.sql:

```
CREATE DATABASE biblioteca;
-- Demais comandos estão no arquivo database/banco.sql
```

3. Rodar o back-end

Configure o arquivo application.properties:

```
spring.datasource.url=jdbc:mysql://localhost:3306/biblioteca
spring.datasource.username=root
spring.datasource.password=sua_senha
spring.jpa.hibernate.ddl-auto=update
```

No terminal:
cd backend
./mvnw spring-boot:run

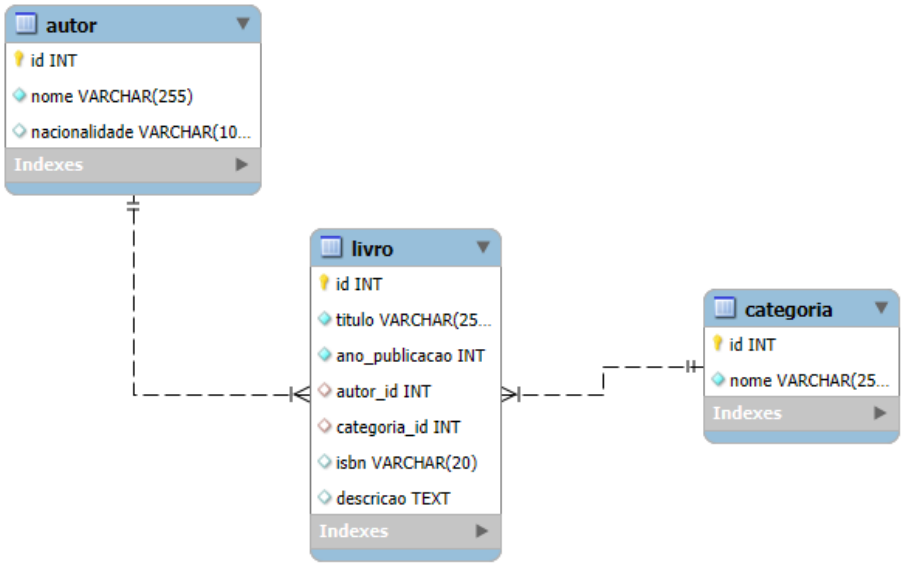
4. Rodar o front-end

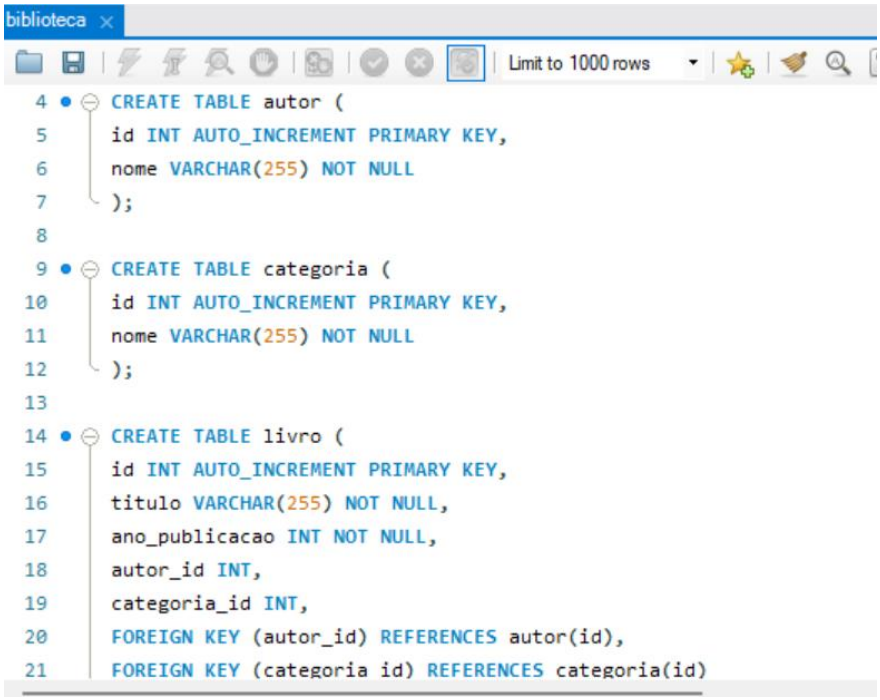
Abra o arquivo index.html no navegador ou acesse a URL do GitHub Pages.

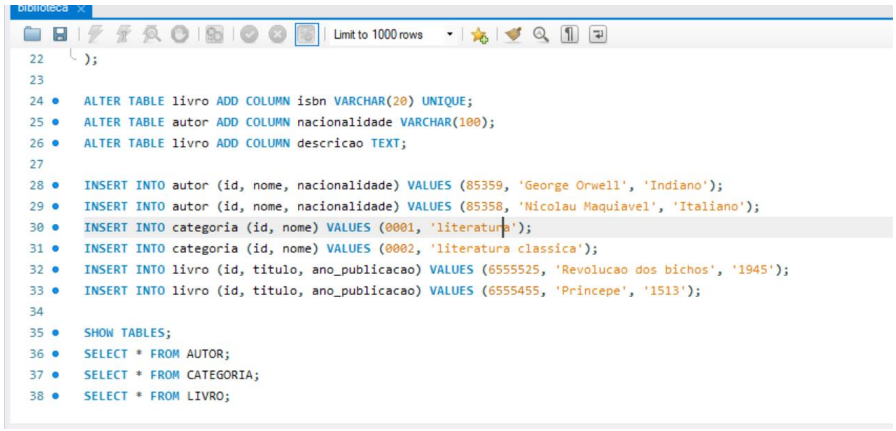
## Justificativa Técnica

A modelagem do sistema segue os princípios da Programação Orientada a Objetos, com as classes Livro, Autor e Categoria. O relacionamento entre elas foi definido de forma a refletir a estrutura real de uma biblioteca. O uso de Spring Boot facilita a criação de APIs REST e a integração com o banco de dados.

## Diagrama de Classes e Script SQL






## Levantamento de Soluções

As principais causas da ineficiência em bibliotecas incluem a ausência de sistemas digitais e o excesso de processos manuais. Os mais afetados são bibliotecários, usuários e a administração. A implementação de um sistema de gestão resolve esses problemas, mas enfrenta barreiras como resistência à mudança e custos. O sistema proposto aplica conceitos de POO e, a longo prazo, sua ausência compromete a eficiência, o financiamento e a promoção da leitura.

**Fontes de Pesquisa Primária**
**1-Documentação oficial do Spring Boot**
Link: https://spring.io/projects/spring-boot
**2- Criando um CRUD com Java Spring Boot e MySQL (Vídeo aula no youtube)**
Link: https://youtu.be/TnI4YnB6E54?si=5GL9adptnAIcYK9G
3- **Postman Learning Center para aprender a testas as APIs do jeito certo**
Link: https://learning.postman.com/
4- **Documentação oficial do MySQL**
link: https://dev.mysql.com/doc/

## LEVANTAMENTO DE SOLUÇÕES

1. Quais são as principais causas da ineficiência na gestão da biblioteca?

As principais causas são o uso de processos manuais, ausência de um sistema informatizado, dificuldade na catalogação e localização de livros, registros inconsistentes e falta de relatórios gerenciais. Esses fatores comprometem a organização do acervo, aumentam o tempo de atendimento e dificultam a tomada de decisões.

### 2. Quem são os principais afetados por essa ineficiência e como eles são impactados?

Bibliotecários, usuários, administração pública e, em alguns casos, professores e alunos são os mais impactados. A ineficiência gera sobrecarga de trabalho, dificulta o acesso aos livros, compromete o controle do acervo e impede decisões estratégicas, afetando diretamente a qualidade do serviço prestado.

### 3. Quais são as possíveis soluções para otimizar a gestão da biblioteca e quais são os prós e contras de cada uma?

Uma solução eficaz é implantar um sistema digital, que organiza o acervo e agiliza o atendimento.
**Prós:** maior eficiência, melhor controle e acesso rápido às informações.
**Contras:** custo inicial, necessidade de manutenção e treinamento.

Outra solução é capacitar os funcionários para usar o sistema.
**Prós:** melhora o uso das ferramentas e reduz erros.
**Contras:** demanda tempo e pode haver resistência à mudança.

Por fim, digitalizar o acervo com etiquetas ou códigos facilita a automação.
**Prós:** controle mais preciso e menos perdas.
**Contras:** exige investimento em equipamentos e tempo para implementar.

### 4. Quais são as barreiras ou desafios que podem surgir ao implementar um novo sistema de gerenciamento?

Os principais desafios são a resistência dos funcionários à mudança, os custos de implantação, a necessidade de treinamento adequado, a migração cuidadosa dos dados antigos para o novo sistema e a manutenção contínua para garantir o bom funcionamento.

### 5. Como a proposta de um novo sistema de gerenciamento se relaciona com conceitos de POO?

O sistema usa conceitos de POO como encapsulamento, protegendo os dados com atributos privados e métodos públicos; abstração, representando entidades reais de forma simplificada; herança, para criar classes base e especializadas; e polimorfismo, para tratar diferentes tipos de obras de forma uniforme. Isso torna o código organizado, reutilizável e fácil de manter.

### 6. Quais seriam as implicações a longo prazo se as ineficiências na gestão da biblioteca não forem resolvidas?

A longo prazo, a biblioteca pode perder usuários devido ao atendimento lento, perder controle do acervo com extravios, enfrentar redução de financiamento público, prejudicar o incentivo à leitura na comunidade e sofrer desvalorização enquanto serviço público.

-Anexar O SCRIPT e a foto do banco my sql que tenho salvo.

**Explicação**:

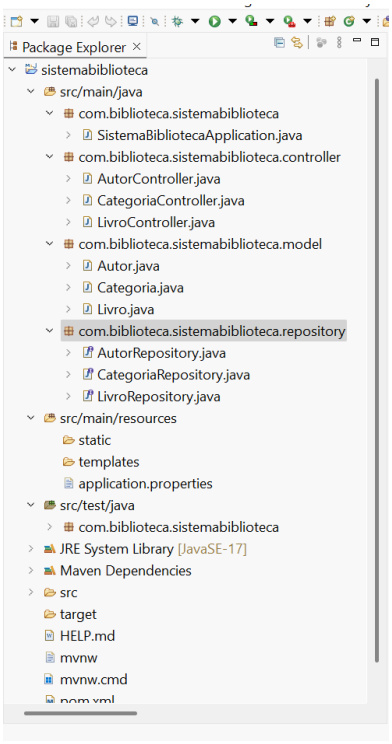
O sistema foi modelado com três classes principais: Livro, Autor e Categoria, que representam as entidades essenciais para a gestão do acervo da biblioteca. A classe Livro é o foco do sistema, representando cada obra disponível para empréstimo, com atributos básicos como título e ano de publicação para facilitar sua identificação e catalogação. A classe Autor foi separada para evitar redundância, permitindo que um mesmo autor esteja associado a vários livros, o que facilita a manutenção dos dados e as consultas. Já a classe Categoria organiza os livros em grupos temáticos, auxiliando na busca e classificação das obras conforme gêneros ou áreas de interesse. Os relacionamentos entre as classes refletem a estrutura real da biblioteca, onde cada Livro está associado a um único Autor, garantindo a integridade da informação e facilitando a navegação entre os dados. Além disso, cada Livro também está vinculado a uma única Categoria, o que possibilita organizar o acervo em setores temáticos e facilitar sua localização. Essa estrutura orientada a objetos proporciona uma representação clara e eficiente dos dados, alinhada aos princípios da programação orientada a objetos, como encapsulamento e composição, tornando o sistema modular, reutilizável e fácil de manter.

### 6. Quais seriam as implicações a longo prazo se as ineficiências na gestão da biblioteca não forem resolvidas?

A longo prazo, a biblioteca pode perder usuários devido ao atendimento lento, perder controle do acervo com extravios, enfrentar redução de financiamento público, prejudicar o incentivo à leitura na comunidade e sofrer desvalorização enquanto serviço público.

Sistema de gestão para biblioteca municipal

-Como está no eclipse a estrutura:




Detalhadamente os códigos da estrutura em cada interface, class, package... :

No package com.biblioteca.sistemabiblioteca

-Autor.java:

```java
package com.biblioteca.sistemabiblioteca.model;

import jakarta.persistence.*;

@Entity
public class Autor {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nome;

    public Autor() {}

    public Autor(String nome) {
        this.nome = nome;
    }

    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }
    public String getNome() { return nome; }
    public void setNome(String nome) { this.nome = nome; }
}
```

-Livro.java:

application.... Livro.java Autor.java Categoria.java LivroRep

```java
package com.biblioteca.sistemabiblioteca.model;

import jakarta.persistence.*;

@Entity
public class Livro {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String titulo;

    @ManyToOne
    private Autor autor;

    @ManyToOne
    private Categoria categoria;

    public Livro() {}

    public Livro(String titulo, Autor autor, Categoria categoria) {
        this.titulo = titulo;
        this.autor = autor;
        this.categoria = categoria;
    }

    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }
    public String getTitulo() { return titulo; }
    public void setTitulo(String titulo) { this.titulo = titulo; }
    public Autor getAutor() { return autor; }
    public void setAutor(Autor autor) { this.autor = autor; }
    public Categoria getCategoria() { return categoria; }
    public void setCategoria(Categoria categoria) { this.categoria = categoria; }
}
```

```java
    public Categoria getCategoria() { return categoria; }
    public void setCategoria(Categoria categoria) { this.categoria = categoria; }
}
```

-SistemaBibliotecaApplication.java:

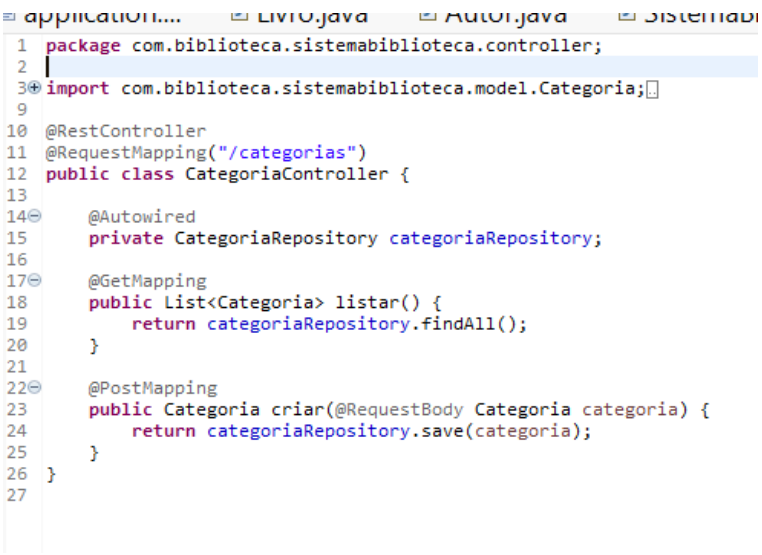
```java
package com.biblioteca.sistemabiblioteca;

import org.springframework.boot.SpringApplication;

@SpringBootApplication
public class SistemaBibliotecaApplication {
    public static void main(String[] args) {
        SpringApplication.run(SistemaBibliotecaApplication.class, args);
    }
}
```

-Package com.biblioteca.sistemabiblioteca.controller:

-AutorController.java:

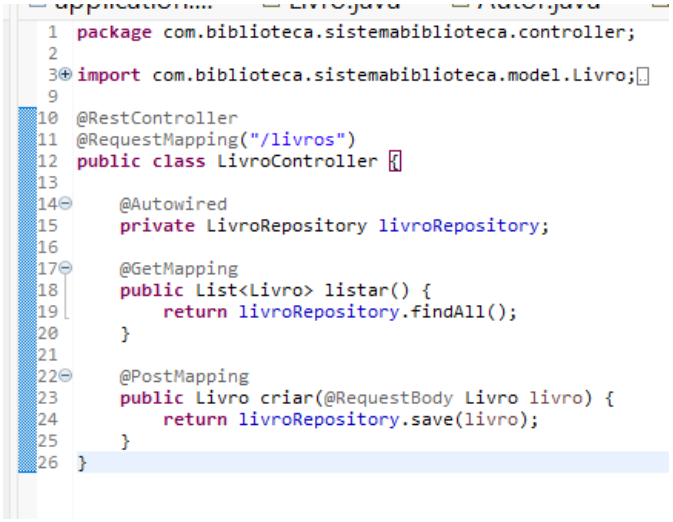
```java
package com.biblioteca.sistemabiblioteca.controller;

import com.biblioteca.sistemabiblioteca.model.Autor;

@RestController
@RequestMapping("/autores")
public class AutorController {

    @Autowired
    private AutorRepository autorRepository;

    @GetMapping
    public List<Autor> listar() {
        return autorRepository.findAll();
    }

    @PostMapping
    public Autor criar(@RequestBody Autor autor) {
        return autorRepository.save(autor);
    }
}
```

-CategoriaController.java:



```java
package com.biblioteca.sistemabiblioteca.controller;

import com.biblioteca.sistemabiblioteca.model.Categoria;

@RestController
@RequestMapping("/categorias")
public class CategoriaController {

    @Autowired
    private CategoriaRepository categoriaRepository;

    @GetMapping
    public List<Categoria> listar() {
        return categoriaRepository.findAll();
    }

    @PostMapping
    public Categoria criar(@RequestBody Categoria categoria) {
        return categoriaRepository.save(categoria);
    }
}
```

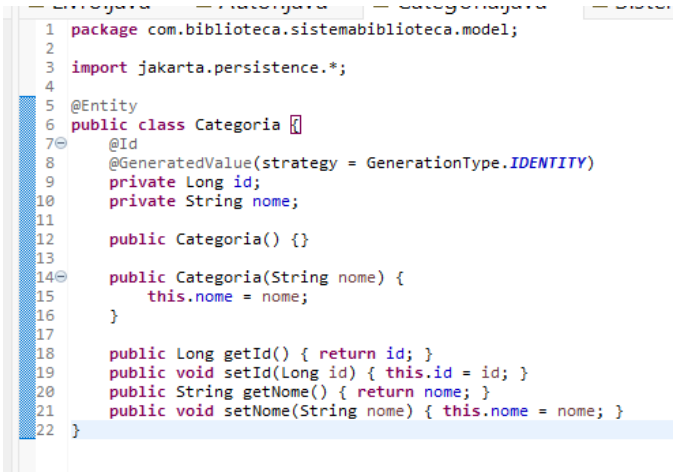
-LivroController.java:



```java
package com.biblioteca.sistemabiblioteca.controller;

import com.biblioteca.sistemabiblioteca.model.Livro;

@RestController
@RequestMapping("/livros")
public class LivroController {

    @Autowired
    private LivroRepository livroRepository;

    @GetMapping
    public List<Livro> listar() {
        return livroRepository.findAll();
    }

    @PostMapping
    public Livro criar(@RequestBody Livro livro) {
        return livroRepository.save(livro);
    }
}
```

- package com.biblioteca.sistemabiblioteca.model:

-Categoria.java:



```java
package com.biblioteca.sistemabiblioteca.model;

import jakarta.persistence.*;

@Entity
public class Categoria {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nome;

    public Categoria() {}

    public Categoria(String nome) {
        this.nome = nome;
    }

    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }
    public String getNome() { return nome; }
    public void setNome(String nome) { this.nome = nome; }
}
```

-package com.biblioteca.sistemabiblioteca.repository:

-AutorRepository.java:

```java
package com.biblioteca.sistemabiblioteca.repository;

import com.biblioteca.sistemabiblioteca.model.Autor;

public interface AutorRepository extends JpaRepository<Autor, Long> {}
```

---

-CategoriaRepository.java:

```java
package com.biblioteca.sistemabiblioteca.repository;

import com.biblioteca.sistemabiblioteca.model.Categoria;

public interface CategoriaRepository extends JpaRepository<Categoria, Long> {}
```

-LivroRepository.java:

```java
package com.biblioteca.sistemabiblioteca.repository;

import com.biblioteca.sistemabiblioteca.model.Livro;

public interface LivroRepository extends JpaRepository<Livro, Long> {}
```

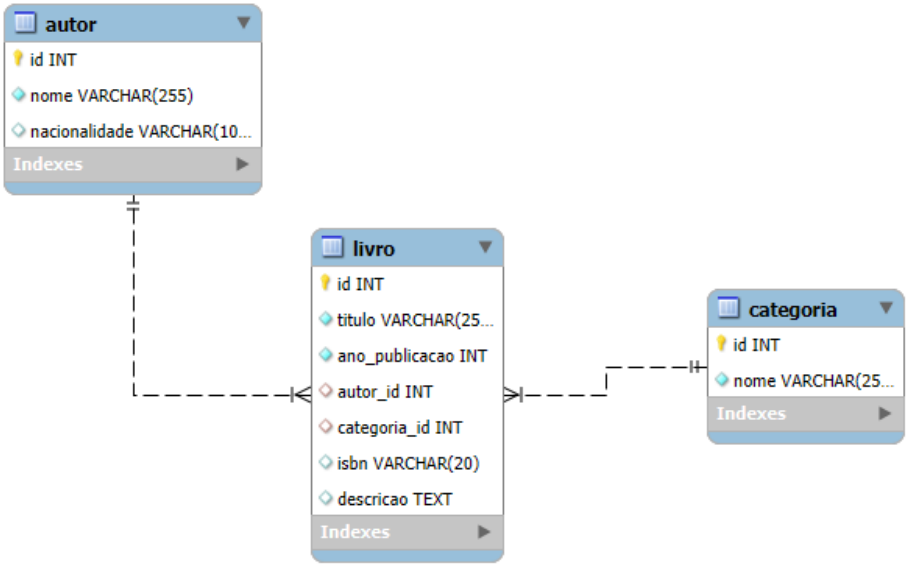
-pom.xml

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.2.5</version>
        <relativePath/>
    </parent>

    <groupId>com.biblioteca</groupId>
    <artifactId>sistemabiblioteca</artifactId>
    <version>1.0.0</version>
    <packaging>jar</packaging>
    <name>sistemabiblioteca</name>

    <properties>
        <java.version>17</java.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
            <groupId>com.mysql</groupId>
            <artifactId>mysql-connector-j</artifactId>
        </dependency>
```

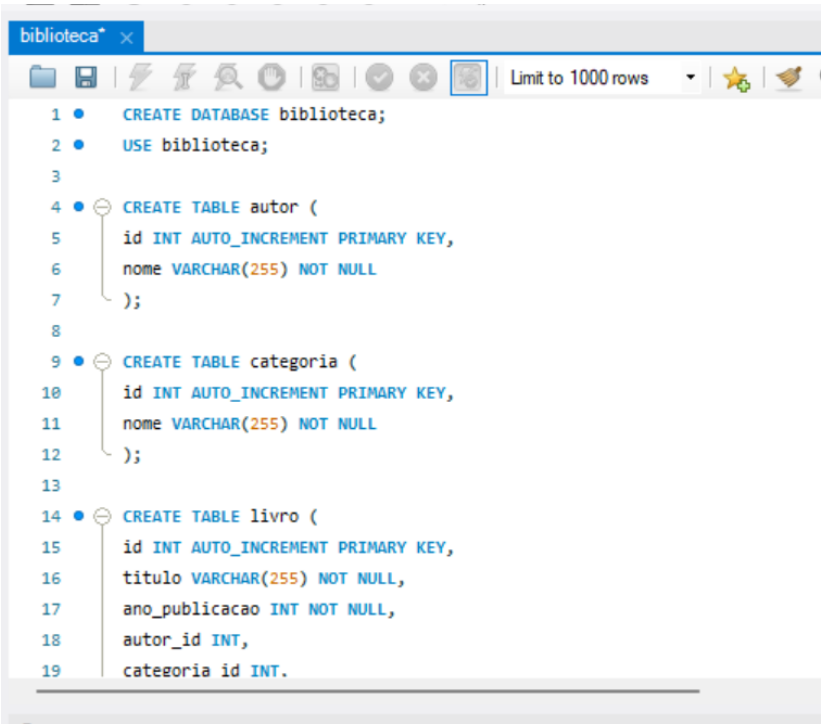
Categoria.java | LivroReposi... | AutorReposi... | Categ

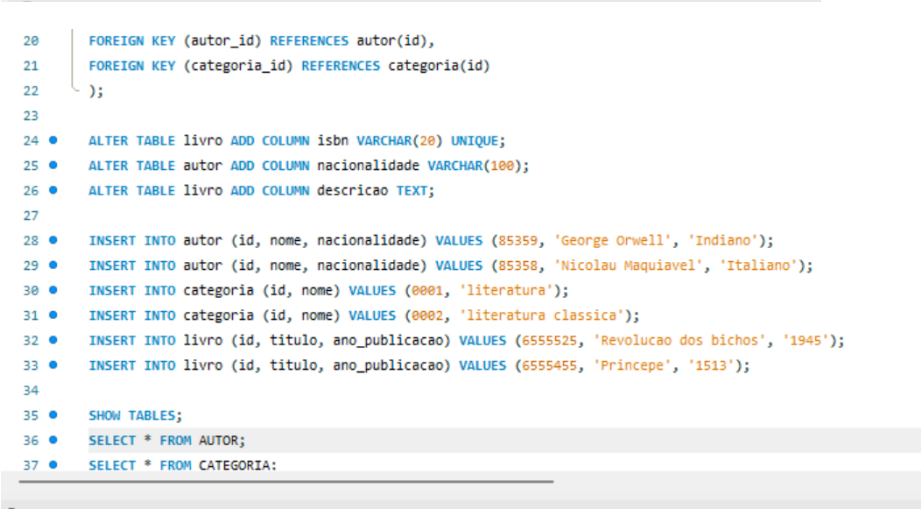
```
30        <dependency>
31            <groupId>org.springframework.boot</groupId>
32            <artifactId>spring-boot-starter-web</artifactId>
33        </dependency>
34
35        <dependency>
36            <groupId>com.mysql</groupId>
37            <artifactId>mysql-connector-j</artifactId>
38        </dependency>
39
40        <dependency>
41            <groupId>org.projectlombok</groupId>
42            <artifactId>lombok</artifactId>
43            <optional>true</optional>
44        </dependency>
45
46        <dependency>
47            <groupId>com.h2database</groupId>
48            <artifactId>h2</artifactId>
49            <scope>runtime</scope>
50        </dependency>
51
52        <dependency>
53            <groupId>org.springframework.boot</groupId>
54            <artifactId>spring-boot-starter-test</artifactId>
55            <scope>test</scope>
56        </dependency>
57    </dependencies>
58
59    <build>
60        <plugins>
61            <plugin>
62                <groupId>org.springframework.boot</groupId>
63                <artifactId>spring-boot-maven-plugin</artifactId>
64            </plugin>
65        </plugins>
66    </build>
67
68 </project>
69
```

-Como está o diagrama de classe:



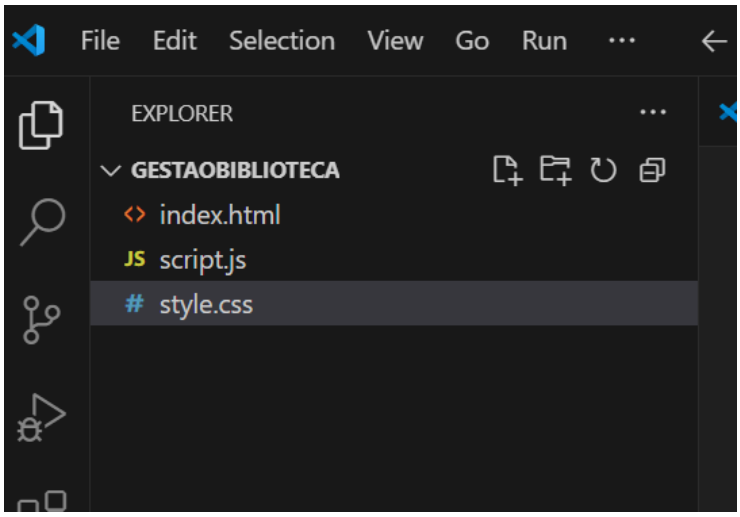

-Como está os scripts no mysql:




-Como está no visual studio code o código do front-end:

-Estrutura:




Código:

-index.html:

```
index.html > ...
1   <!DOCTYPE html>
2   <html lang="pt-br">
3   <head>
4       <meta charset="UTF-8">
5     <title>Gestao Livros Biblioteca Municipal LOGIN</title>
6     <link rel="stylesheet" href="style.css">
7   </head>
8   <body>
9       <div class="login-container">
10      <h2>LOGIN SISTEMA BIBLIOTECA</h2>
11      <div class="form-group">
12        <label for="usuario">Usuário:</label><br>
13        <input type="text" id="usuario" name="usuario">
14      </div>
15      <div class="form-group">
16        <label for="senha">Senha:</label><br>
17        <input type="password" id="senha" name="senha">
18      </div>
19      <button onclick="fazerLogin()">Acessar</button>
20      </div>
21      <script src="script.js"></script>
22  </body>
23  </html>
24
```

-script.js:

```
JS script.js > fazerLogin
1   function fazerLogin() {
2     const usuario = document.getElementById('usuario').value;
3     const senha = document.getElementById('senha').value;
4
5     if (usuario === "admin" && senha === "123") {
6       alert("Login bem-sucedido!");
7     } else {
8       alert("Usuário ou senha incorretos!");
9     }
10  }
11
```

-style.css:

```
# style.css > .login-container h2
1   body {
2     margin: 0;
3     padding: 0;
4     font-family: Arial, sans-serif;
5     background-color: #487bcb;
6     display: flex;
7     justify-content: center;
8     align-items: center;
9     height: 100vh;
10  }
11
12  .login-container {
13    text-align: center;
14    color: white;
15  }
16
17  .login-container h2 {
18    margin-bottom: 20px;
19  }
20
21  .form-group {
22    margin-bottom: 15px;
23  }
24
25  input[type="text"],
26  input[type="password"] {
27    width: 200px;
28    padding: 8px;
29    margin-top: 5px;
30    border: none;
31    border-radius: 4px;
32  }
33
34  button {
35    padding: 8px 20px;
36    background-color: #f2f2f2;
37    border: none;
38    border-radius: 4px;
39    cursor: pointer;
40  }
41
42  button:hover {
43    background-color: #e0e0e0;
44  }
```

-ainda baixei para usar o postman.