



Projeto prático de Banco de Dados

Grupo: 1

Autores:

- Gustavo Barbosa de Almeida - 202037589
- Ana Beatriz - 180012428
- Lucas da Silva - 180125699
- Hideki Tomiyama - 190014351
- Thiago Silva Ribeiro - 202037702

CRedit (Contributor Roles Taxonomy):

- **Gustavo** configuração do docker, configuração do backend e documentação dos mesmos, configuração e instalação do frontend, autenticação e gerenciamento de rotas e permissões.
- **Lucas** criação do repositório, instalação do npm e do nodejs, documentação do mesmo e documentação final.
- **Ana Beatriz** instalação do postgresql e configuração do mesmo.
- **Thiago** criação do modelo de banco de dados e documentação do mesmo, criação dos CRUDS e teste de rotas.
- **Hideki** criação dos scripts SQL e documentação dos mesmos.

Data da Versão Atual: 30/11/2023

Sistema de Gerenciamento de Materiais para um Laboratório Didático

Descrição

Para auxiliar os estudantes e professores, o seu grupo ficou encarregado de elaborar um sistema de informação para gerenciar livros de ensino e materiais didáticos em um laboratório. O sistema será projetado para organizar e disponibilizar esses recursos para empréstimo através de um sistema computacional.

O foco principal desta especificação é a definição do banco de dados que será utilizado para armazenar informações sobre os livros e materiais. O sistema deve ter diferentes níveis de acesso para os usuários (por exemplo, administradores do sistema computacional, membros do laboratório e estudantes em geral), de maneira que todos os usuários possam pesquisar os livros e materiais, mas apenas membros do laboratório possam pegar os materiais emprestados.

1. Documentação Técnica

Tecnologias Utilizadas

NestJS

- O NestJS é um framework de desenvolvimento back-end para Node.js que utiliza TypeScript e segue o padrão arquitetural do Angular. Ele oferece uma estrutura robusta e modular para criar aplicativos escaláveis e bem organizados.

Next.js

- O Next.js é um framework de desenvolvimento front-end para React que simplifica a construção de aplicativos React universais. Ele oferece recursos como renderização do lado do servidor, roteamento simples e pré-renderização, tornando-o adequado para aplicativos da web modernos.

Knex.js

- O Knex.js é um construtor de consultas SQL para Node.js. Ele facilita a interação com bancos de dados relacionais, permitindo a criação de consultas de forma programática e intuitiva. É uma escolha popular para lidar com operações de banco de dados em aplicativos Node.js.

Node.js

- O Node.js é um ambiente de tempo de execução JavaScript que permite que você execute código JavaScript do lado do servidor. Ele é amplamente usado para construir aplicativos de servidor escaláveis e em tempo real, graças ao seu modelo de E/S não bloqueante.

PostgreSQL

- O PostgreSQL é um sistema de gerenciamento de banco de dados relacional de código aberto. Ele é conhecido por sua confiabilidade, recursos avançados e extensibilidade. O PostgreSQL é uma escolha popular para aplicativos que requerem um banco de dados robusto e escalável.

Sistema operacional

O sistema operacional utilizado pela maioria da equipe será linux.

1.1 Documentação de Configuração de Ambiente e Tecnologias

Esta documentação descreve os passos necessários para configurar o ambiente de desenvolvimento e lista as tecnologias utilizadas neste projeto.

Configuração de Ambiente

1.1.1. Instalação do NVM (Node Version Manager) e Node.js (Linux)

Antes de começar, é importante garantir que o sistema esteja atualizado.

```
sudo apt update  
sudo apt upgrade
```

1.1.2. Instalação do NVM

Você pode escolher entre dois métodos para instalar o NVM: usando curl ou wget. Escolha um dos seguintes comandos:

```
## Usando curl  
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.5/install.sh |  
bash
```

```
## Ou usando wget  
wget -qO- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.5/install.sh |  
bash
```

Após a instalação, feche e reabra o terminal. Para verificar a instalação do NVM:

```
nvm --version
```

1.1.3. Instalação do Node.js

Com o NVM instalado, você pode instalar o Node.js. Recomendamos a instalação da versão LTS mais recente:

```
nvm install --lts
```

Para verificar a versão do Node.js:

```
node --version
```

1.1.4. Instalação do PostgreSQL (Sistema de Gerenciamento de Banco de Dados)

Para instalar o PostgreSQL no Linux, execute o seguinte comando:

```
sudo apt install postgresql postgresql-contrib libpq-dev
```

Para verificar a instalação do PostgreSQL:

```
pg_config --version
```

1.2. Guia de Uso do Docker com PostgreSQL (Opcional)

Se preferir usar o Docker com o PostgreSQL, siga as instruções em Guia de Uso do Docker com PostgreSQL para configuração e uso do contêiner PostgreSQL.

Pré-requisitos

- [Docker](#) instalado em seu sistema.
- [Docker Compose](#) (geralmente incluído com a instalação do Docker).

Configuração do Docker Compose

No diretório do projeto, verifique se existe um arquivo docker-compose.yml. Este arquivo contém as configurações necessárias para criar o contêiner PostgreSQL.

Iniciar o Banco de Dados PostgreSQL

Abra um terminal e navegue até o diretório do projeto onde está o arquivo docker-compose.yml.

- *Para iniciar o contêiner PostgreSQL, execute o seguinte comando:*

```
docker-compose up -d
```

Isso criará e iniciará o contêiner PostgreSQL em segundo plano (-d). Aguarde até que o contêiner esteja em execução.

- *Você pode verificar o status do contêiner com o seguinte comando:*

```
docker ps
```

Certifique-se de que o contêiner PostgreSQL esteja listado na saída.

Conectar-se ao Banco de Dados PostgreSQL

Para se conectar ao banco de dados PostgreSQL a partir do terminal, use o seguinte comando:

```
psql -h localhost -U postgres -d db
```

- `-h localhost`: Especifica o host onde o PostgreSQL está sendo executado (local).
- `-U postgres`: Especifica o nome de usuário (geralmente é "postgres" por padrão).
- `-d db`: Especifica o nome do banco de dados ao qual você deseja se conectar.
- Será solicitada a senha do usuário "postgres". Insira a senha configurada no arquivo docker-compose.yml (por padrão, é "postgres").

Você estará conectado ao banco de dados PostgreSQL e poderá executar comandos SQL.

Encerrar o Contêiner

Quando você terminar de trabalhar com o banco de dados, você pode parar e remover o contêiner PostgreSQL usando o seguinte comando:

```
docker-compose down
```

Isso desligará e removerá o contêiner PostgreSQL. Certifique-se de que nenhum dado importante seja perdido antes de executar este comando.

1.3. Rodando o projeto

1.3.1. Frontend com Next.js

Para executar o frontend do projeto com Next.js, siga os passos abaixo:

Instale as dependências:

```
npm install
```

Inicie o servidor de desenvolvimento:

```
npm run dev
```

1.3.2. Backend com NestJS

Para executar o backend do projeto com NestJS, siga os passos abaixo:

Instale as dependências:

```
npm install
```

Inicie a aplicação no modo de desenvolvimento:

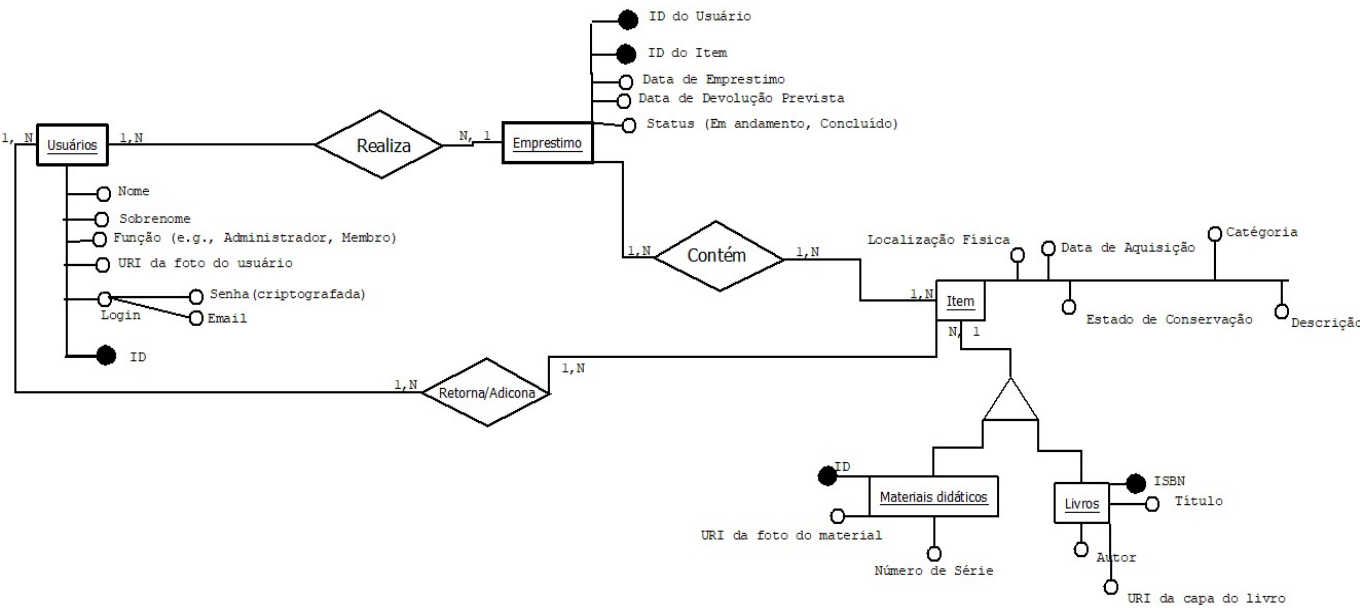
```
npm run start
```

2. Arquitetura do Sistema (Modelo de Banco de Dados)

Um Modelo de Banco de Dados é essencial em projetos, definindo a estrutura e organização dos dados, garantindo eficiência, integridade e escalabilidade. É a base para a gestão de informações eficaz.

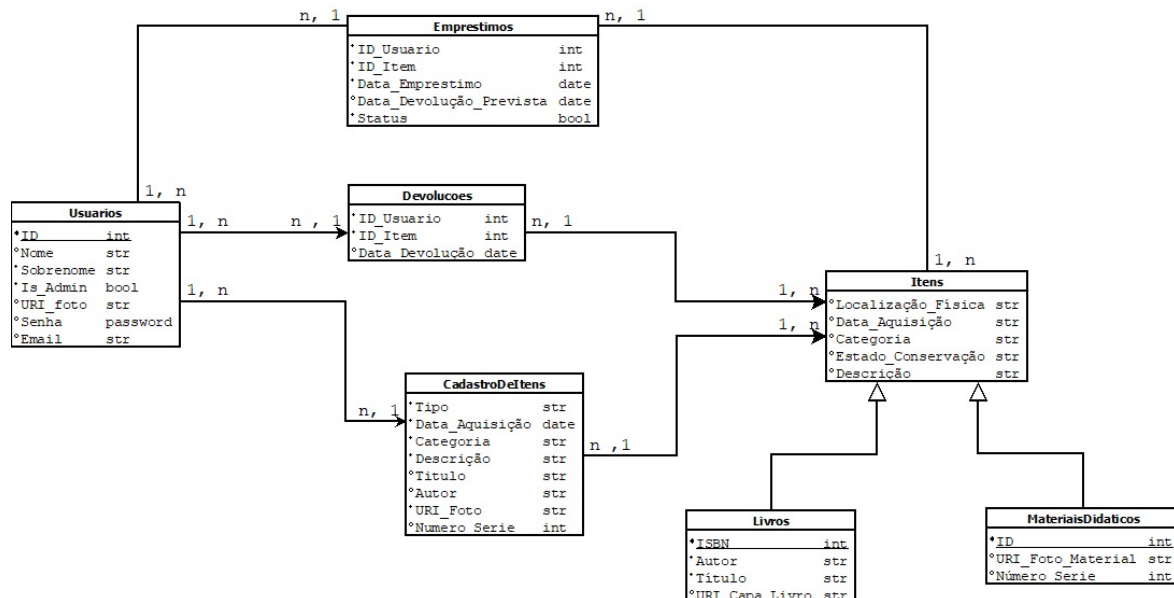
2.1. Diagramas de entidade-relacionamento (DERs)

Diagramas de Entidade-Relacionamento (DERs) são representações visuais que descrevem a estrutura de um banco de dados, mostrando entidades, atributos e relacionamentos entre eles.



2.2. Diagrama do Modelo Lógico (Relacional)

Um Diagrama do Modelo Lógico Relacional é uma representação visual que descreve as tabelas de um banco de dados relacional, seus campos, chaves primárias e chaves estrangeiras. Tabelas representam entidades, campos representam atributos, chaves primárias garantem unicidade e identificação única de registros, e chaves estrangeiras estabelecem relações entre tabelas. Sua importância reside na definição clara da estrutura do banco de dados, permitindo o armazenamento eficiente e a recuperação de informações, garantindo integridade de dados e facilitando o desenvolvimento de consultas e relatórios. Além disso, o modelo lógico serve como guia para a implementação física do banco de dados.



2.3. Scripts SQL

2.3.1. Criação de tabelas

2.3.1.1. Tabela Usuários

```

CREATE TABLE "Usuarios" (
    "id" int NOT NULL,
    "nome" varchar(255) NOT NULL,
    "sobrenome" varchar(255) NOT NULL,
    "is_admin" boolean NOT NULL,
    "uri_foto" varchar(255) NOT NULL,
    "senha" varchar(64) NOT NULL,
    "email" varchar(255) NOT NULL,
    CONSTRAINT "pk_usuarios" PRIMARY KEY (
        "id"
    ),
    CONSTRAINT "uc_usuarios_nome" UNIQUE (
        "nome"
    )
);
  
```

2.3.1.2. Tabela Empréstimos

```

CREATE TABLE "Emprestimos" (
    "id_usuario" int NOT NULL,
    "id_item" int NOT NULL,
    "data_emprestimo" date NOT NULL,
    "data_devolucao_prevista" date NOT NULL,
    "status" boolean NOT NULL
);
  
```

2.3.1.3. Tabela Devoluções

```
CREATE TABLE "Devolucoes" (  
  "id_usuario" int NOT NULL,  
  "id_item" int NOT NULL  
);
```

2.3.1.4. Tabela CadastroDeItens

```
CREATE TABLE "CadastroDeItens" (  
  "id_item" int NOT NULL,  
  "tipo" varchar(255) NOT NULL,  
  "data_aquisicao" date NOT NULL,  
  "categoria" varchar(255) NOT NULL,  
  "descricao" varchar(255) NOT NULL,  
  "titulo" varchar(255) NOT NULL,  
  "autor" varchar(255) NOT NULL,  
  "uri_foto" varchar(255) NOT NULL,  
  "numero_serie" int NOT NULL  
);
```

2.3.1.5. Tabela Itens

```
CREATE TABLE "Itens" (  
  "id" int NOT NULL,  
  "id_material" int NOT NULL,  
  "id_isbn" int NOT NULL,  
  "localizacao_fisica" varchar(255) NOT NULL,  
  "data_aquisicao" date NOT NULL,  
  "categoria" varchar(255) NOT NULL,  
  "estado_conservacao" varchar(255) NOT NULL,  
  "descricao" varchar(255) NOT NULL,  
  CONSTRAINT "pk_itens" PRIMARY KEY (  
    "id"  
  )  
);
```

2.3.1.6. Tabela Livros

```
CREATE TABLE "Livros" (  
  "ISBN" int NOT NULL,  
  "autor" varchar(255) NOT NULL,  
  "titulo" varchar(255) NOT NULL,  
  "uri_capa_livro" varchar(255) NOT NULL,
```



```
CONSTRAINT "pk_livros" PRIMARY KEY (  
    "ISBN"  
)  
);
```

2.3.1.7. Tabela MateriaisDidáticos

```
CREATE TABLE "MateriaisDidaticos" (  
    "id" int NOT NULL,  
    "uri_foto_material" varchar(255) NOT NULL,  
    "numero_serie" int NOT NULL,  
    CONSTRAINT "pk_materiaisDidaticos" PRIMARY KEY (  
        "id"  
    )  
);
```

2.4. Chaves estrangeiras

```
ALTER TABLE "Emprestimos" ADD CONSTRAINT "fk_emprestimos_id_usuario"  
FOREIGN KEY("id_usuario")  
REFERENCES "Usuarios" ("id");  
  
ALTER TABLE "Emprestimos" ADD CONSTRAINT "fk_emprestimos_id_item" FOREIGN  
KEY("id_item")  
REFERENCES "Itens" ("id");  
  
ALTER TABLE "Devolucoes" ADD CONSTRAINT "fk_devolucoes_id_usuario" FOREIGN  
KEY("id_usuario")  
REFERENCES "Usuarios" ("id");  
  
ALTER TABLE "Devolucoes" ADD CONSTRAINT "fk_devolucoes_id_item" FOREIGN  
KEY("id_item")  
REFERENCES "Itens" ("id");  
  
ALTER TABLE "CadastroDeItens" ADD CONSTRAINT "uc_cadastroDeItens_id_item"  
UNIQUE ("id_item");  
  
ALTER TABLE "Itens" ADD CONSTRAINT "fk_itens_id" FOREIGN KEY("id")  
REFERENCES "CadastroDeItens" ("id_item");  
  
ALTER TABLE "Itens" ADD CONSTRAINT "fk_itens_id_material" FOREIGN  
KEY("id_material")  
REFERENCES "MateriaisDidaticos" ("id");  
  
ALTER TABLE "Itens" ADD CONSTRAINT "fk_itens_id_isbn" FOREIGN  
KEY("id_isbn")  
REFERENCES "Livros" ("ISBN");
```

2.5. Inserção de Dados nas tabelas

2.5.1. Tabela Usuários

```
-- Inserção de dados adicionais na tabela Usuarios
INSERT INTO Usuarios (id, nome, sobrenome, is_admin, uri_foto, senha,
email)
VALUES
    (4, 'Usuario3', 'Sobrenome3', false, 'http://urifotousuario3.com',
'hashed_password_usuario3', 'usuario3@example.com'),
    (5, 'Usuario4', 'Sobrenome4', false, 'http://urifotousuario4.com',
'hashed_password_usuario4', 'usuario4@example.com'),
    (6, 'Usuario5', 'Sobrenome5', false, 'http://urifotousuario5.com',
'hashed_password_usuario5', 'usuario5@example.com');
```

2.5.2. Tabela Empréstimos

```
-- Inserção de dados adicionais na tabela Emprestimos
INSERT INTO Emprestimos (id_usuario, id_item, data_emprestimo,
data_devolucao_prevista, status)
VALUES
    (4, 3, '2023-04-04', '2023-05-04', true),
    (5, 1, '2023-04-05', '2023-05-05', false),
    (6, 2, '2023-04-06', '2023-05-06', true);
```

2.5.3. Tabela CadastroDeItens

```
-- Inserção de dados adicionais na tabela CadastroDeItens
INSERT INTO CadastroDeItens (id_item, tipo, data_aquisicao, categoria,
descricao, titulo, autor, uri_foto, numero_serie)
VALUES
    (4, 'Livro', '2023-04-01', 'Suspense', 'Descrição do Livro 3', 'Livro
3', 'Autor 3', 'http://urifotolivro3.com', null),
    (5, 'Material Didático', '2023-04-02', 'Ciência', 'Descrição do
Material 2', null, null, 'http://urifotomaterial2.com', 345678),
    (6, 'Livro', '2023-04-03', 'Romance', 'Descrição do Livro 4', 'Livro
4', 'Autor 4', 'http://urifotolivro4.com', null);
```

2.5.4. Tabela Itens

```
-- Inserção de dados adicionais na tabela Itens
INSERT INTO Itens (id, id_material, id_isbn, localizacao_fisica,
data_aquisicao, categoria, estado_conservacao, descricao)
VALUES
    (4, null, 4, 'Estante D', '2023-04-01', 'Suspense', 'Bom', 'Descrição
```

```
do Livro 3'),
  (5, 4, null, 'Armário E', '2023-04-02', 'Ciência', 'Ótimo', 'Descrição
do Material 2'),
  (6, null, 5, 'Estante F', '2023-04-03', 'Romance', 'Regular',
'Descrição do Livro 4');
```

2.5.5. Tabela Livros

```
-- Inserção de dados adicionais na tabela Livros
INSERT INTO Livros (ISBN, autor, titulo, uri_capa_livro)
VALUES
  (3, 'Autor 3', 'Livro 3', 'http://uricapalivro3.com'),
  (4, 'Autor 4', 'Livro 4', 'http://uricapalivro4.com');
  (5, 'Autor 5', 'Livro 5', 'http://uricapalivro5.com');
```

2.5.6. Tabela MateriaisDidáticos

```
-- Inserção de dados adicionais na tabela MateriaisDidaticos
INSERT INTO MateriaisDidaticos (id, uri_foto_material, numeroSerie)
VALUES
  (3, 'http://urifotomaterial3.com', 567890),
  (4, 'http://urifotomaterial4.com', 123456);
  (5, 'http://urifotomaterial5.com', 234567);
```

Estes scripts SQL fornecem uma visão geral das tabelas criadas, suas relações e exemplos de inserções de dados.

2.6. Criação da camada de persistência.

As tecnologias utilizadas foram **Nestjs** js que é um framework de nodejs e **Knexjs** que é um query builder para SQL, dito isso, os CRUDS foram criados com SQL puro, tanto migrations quanto as seeds também, o knexjs possui um método `knex.raw` que permite colocar queries em SQL puro dentro de um objeto javascript então assim foi feito os CRUDS, Migrations e as Seeds.

2.6.1. Exemplos:

Assim foi feito a migração das tabelas de usuários

```
1 import { Knex } from 'knex';
2
3 export async function up(knex: Knex): Promise<void> {
4   await knex.raw(`
5     DROP TYPE IF EXISTS role;
6     CREATE TYPE role AS ENUM ('admin', 'estudante', 'laboratorio');
7   `);
8 }
```

```

8
9 CREATE TABLE IF NOT EXISTS Usuarios(
10     "id" SERIAL PRIMARY KEY NOT NULL,
11     "nome" varchar(255) NOT NULL,
12     "sobrenome" varchar(255) NOT NULL,
13     "role" role NOT NULL,
14     "uri_foto" bytea NOT NULL,
15     "senha" varchar(64) NOT NULL,
16     "email" varchar(255) NOT NULL,
17     CONSTRAINT "uc_usuarios_nome" UNIQUE (
18         "nome"
19     )
20 );
21
22 CREATE TABLE IF NOT EXISTS Categorias (
23     "id_categoria" SERIAL PRIMARY KEY,
24     "nome" varchar(255) NOT NULL
25 );
26
27 CREATE TABLE IF NOT EXISTS Emprestimos(
28     "id_usuario" int NOT NULL,
29     "id_item" int NOT NULL,
30     "data_emprestimo" date NOT NULL,
31     "data_devolucao_prevista" date NOT NULL,
32     "status" boolean NOT NULL
33 );
34
35 CREATE TABLE IF NOT EXISTS Devolucoes(
36     "id_usuario" int NOT NULL,
37     "id_item" int NOT NULL
38 );
39
40 CREATE TABLE IF NOT EXISTS CadastroDeItens (
41     "id_item" SERIAL PRIMARY KEY,
42     "tipo" varchar(255) NOT NULL,
43     "data_aquisicao" date NOT NULL,
44     "id_categoria" int NOT NULL,
45     "descricao" varchar(255) NOT NULL,
46     "uri_foto" bytea NOT NULL,
47     CONSTRAINT "fk_cadastro_itens_categoria" FOREIGN KEY("id_categoria")
48     REFERENCES Categorias ("id_categoria")
49 );
50
51 CREATE TABLE IF NOT EXISTS Livros (
52     "id_item" int PRIMARY KEY NOT NULL,
53     "isbn" SERIAL PRIMARY KEY NOT NULL,
54     "autor" varchar(255) NOT NULL,
55     "titulo" varchar(255) NOT NULL,
56     "uri_capa_livro" bytea ,
57     "localizacao_fisica" varchar(255) NOT NULL,
58     "estado_conservacao" varchar(255) NOT NULL,
59 );
60
61 CREATE TABLE IF NOT EXISTS MateriaisDidaticos (
62     "id" SERIAL PRIMARY KEY NOT NULL,
63     "id_item" varchar(255) NOT NULL,
64     "uri_foto_material" bytea ,
65     "numero_serie" int NOT NULL,
66     "localizacao_fisica" varchar(255) NOT NULL,
67     "data_aquisicao" date ,
68 );
69
70 ALTER TABLE Emprestimos ADD CONSTRAINT "fk_emprestimos_id_usuario" FOREIGN KEY("id_usuario")
71 REFERENCES Usuarios ("id");
72
73 ALTER TABLE Devolucoes ADD CONSTRAINT "fk_devolucoes_id_usuario" FOREIGN KEY("id_usuario")
74 REFERENCES Usuarios ("id");
75
76 ALTER TABLE CadastroDeItens ADD CONSTRAINT "fk_cadastro_itens_categoria" FOREIGN KEY("id_categoria")
77 REFERENCES Categorias ("id_categoria");
78
79 ALTER TABLE Livros ADD CONSTRAINT "uc_livros_isbn" UNIQUE ("isbn");
80 `);
81 }
82
83 export async function down(knex: Knex): Promise<void> {
84     await knex.raw(
85         `
86         DROP TABLE IF EXISTS Usuarios CASCADE;
87         DROP TABLE IF EXISTS Emprestimos CASCADE;
88         DROP TABLE IF EXISTS Devolucoes CASCADE;
89         DROP TABLE IF EXISTS Livros CASCADE;
90         DROP TABLE IF EXISTS MateriaisDidaticos CASCADE;

```

```
90 DROP TABLE IF EXISTS MateriaisDidaticos CASCADE;  
91 `;  
92 );  
93 }  
94
```

E assim foi feito a criação da seed de usuarios

```
1 import { Knex } from 'knex';  
2  
3 export async function seed(knex: Knex): Promise<void> {  
4   // Deletes ALL existing entries  
5   await knex.raw(`DELETE FROM Usuarios CASCADE`)  
6  
7   // Inserts seed entries  
8   await knex.raw(  
9     `  
10     INSERT INTO Usuarios (nome, sobrenome, email, uri_foto, senha, role) VALUES  
11     ('Veigh', 'Faz dinheiro', 'fino@email.com', 'foto', 'senha','admin'),  
12     ('Caio', 'blaque', 'segredo@email.com', 'foto', '123', 'estudante'),  
13     ('Matue', 'trinta', 'trinta@email.com', 'foto', 'luz', 'laboratorio');  
14     `;  
15   )  
16 }
```

2.6.2. CRUD de usuários

No controller de usuários que é onde gerenciamos as rotas e recebemos as requisições para darmos alguma resposta pro servidor chamando os métodos criados na service de usuários possui essa estrutura:

```
1 import { Controller, Get, Post, Body, Param, Delete, Patch } from '@nestjs/common';
2 import { UsuariosService } from '../usuarios.service';
3 import { CreateUsuarioDto } from '../dto/create-usuario.dto';
4 import { UpdateUsuarioDto } from '../dto/update-usuario.dto';
5
6 @Controller('usuarios')
7 export class UsuariosController {
8   constructor(private readonly usuariosService: UsuariosService) {}
9
10  @Post('cadastro')
11  create(@Body() createUsuarioDto: CreateUsuarioDto) {
12    return this.usuariosService.create(createUsuarioDto);
13  }
14
15  @Get()
16  findAll(){
17    return this.usuariosService.findAll();
18  }
19
20  @Get('/:id')
21  findOne(@Param('id') id: string) {
22    return this.usuariosService.findOne(+id);
23  }
24
25  @Patch('/:id')
26  update(@Param('id') id: string, @Body() updateUsuarioDto: UpdateUsuarioDto) {
27    return this.usuariosService.update(+id, updateUsuarioDto);
28  }
29
30  @Delete('/:id')
31  remove(@Param('id') id: string) {
32    return this.usuariosService.remove(+id);
33  }
34
35 }
```

2.6.3. Método Create

Aqui está um exemplo do método create de usuários

```
1 async create(createUsuarioDto: CreateUsuarioDto, imagemPerfil?: Express.Multer.File) {
2   try {
3     let createdUser = null;
4
5     // Start a transaction
6     await this.knex.transaction(async (trx) => {
7       if (!createUsuarioDto.email || !createUsuarioDto.nome || !createUsuarioDto.sobrenome || !createUsuarioDto.senha || !createUsuarioDto.role || !createUsuarioDto.uri_foto) {
8         throw new BadRequestException('Todos os campos são obrigatórios');
9       }
10
11       const hashedPassword = await bcrypt.hash(createUsuarioDto.senha, 10);
12
13       let usuario = {
14         ...createUsuarioDto,
15         senha: hashedPassword,
16       };
17
18       if (imagemPerfil) {
19         const caminhoDestino = path.join(__dirname, '../..../uploads', imagemPerfil.originalname);
20
21         fs.writeFileSync(caminhoDestino, imagemPerfil.buffer);
22
23         const imagemBuffer = fs.readFileSync(caminhoDestino);
24         usuario = { ...usuario, uri_foto: imagemBuffer };
25
26         fs.unlinkSync(caminhoDestino);
27       }
28
29       // Insert the user within the transaction
30       const result = await trx.raw(
31         `
32         INSERT INTO Usuarios (email, nome, senha, sobrenome, uri_foto, role)
33         VALUES (?, ?, ?, ?, ?, ?) RETURNING *
34         `,
35         [usuario.email, usuario.nome, usuario.senha, usuario.sobrenome, usuario.uri_foto, usuario.role]
36       );
37
38       createdUser = result.rows[0];
39     });
40
41     if (createdUser) {
42       return {
43         usuario: {
44           ...createdUser,
45           senha: undefined,
46         },
47       };
48     }
49
50     return null;
51   } catch (error) {
52     console.error('Error during user creation:', error);
53     throw new InternalServerErrorException('Falha ao criar usuário');
54   }
55 }
```

Com isso temos uma ideia de como está sendo feito a camada de persistência do sistema, para ter uma visão completa só acessar nosso repositório no [Github](#)

3. Descrição de Funcionalidades

3.1 Registro de Usuário

Cada usuário que desejar usar a plataforma consegue fazer um novo cadastro como estudante.

3.2 Cadastro de Livros e Materiais

Cada usuário administrador e chefe de laboratório consegue registrar um novo livro e/ou material.

3.3 Catalogação

Cada item possui uma catalogação detalhada sobre si, contendo título, autor, ISBN (para livros), descrição, categoria, número de série (para materiais), data de aquisição e estado de conservação.

3.4 Armazenamento Físico

Cada item possui um número indicando sua estante e prateleira fisicamente.

3.5 Empréstimo e Devolução

Cada usuário consegue fazer empréstimos e devoluções ao sistema.

3.6 Pesquisa e Consulta

Cada usuário consegue consultar os livros e materiais disponíveis e locados.

3.7 Controle de Acesso

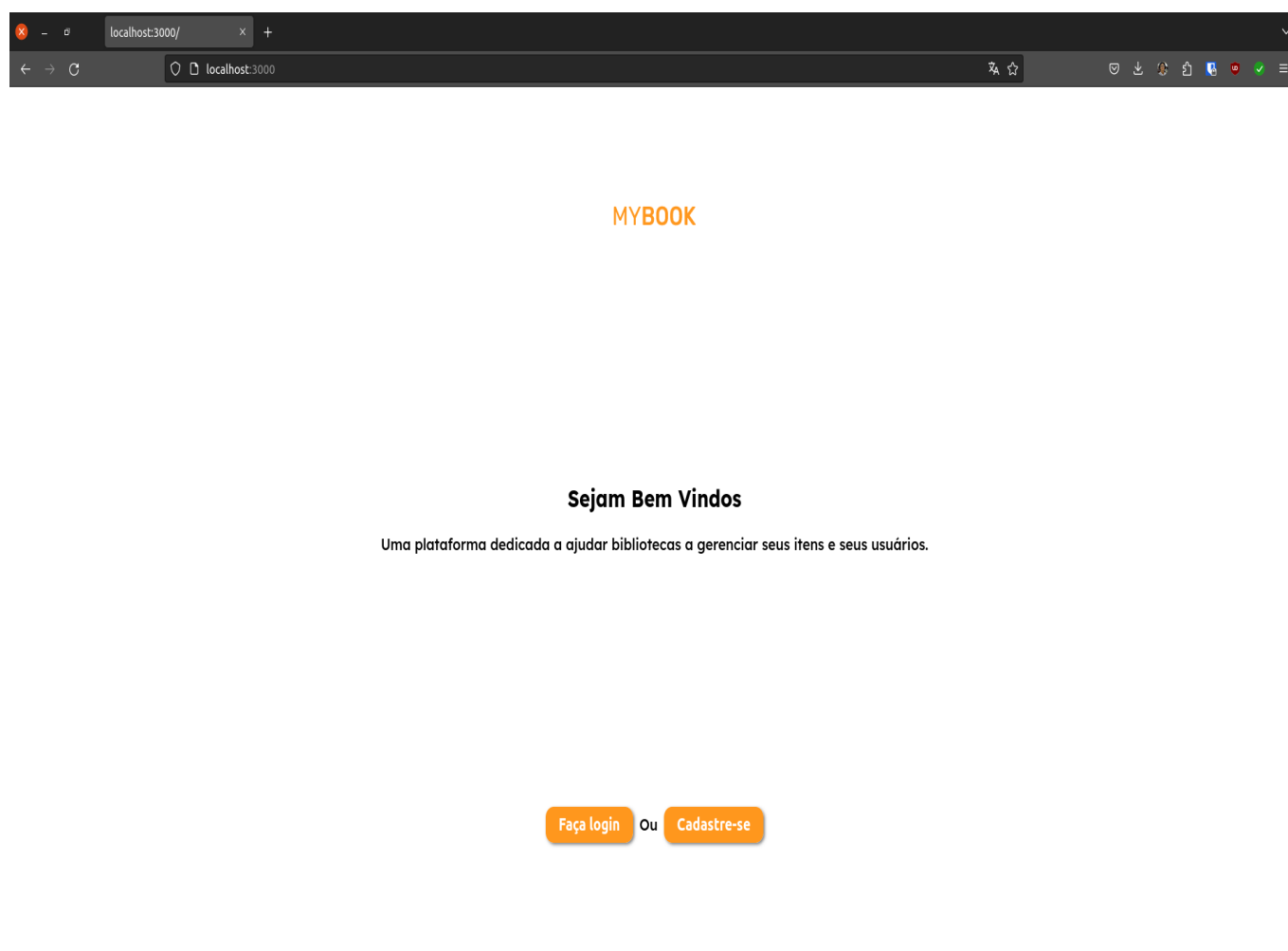
Cada usuário só tem acesso as funcionalidades do próprio tipo de acesso, como admin, chefe de laboratório e estudantes (comum).

4. Manual de Usuário

Conteúdo

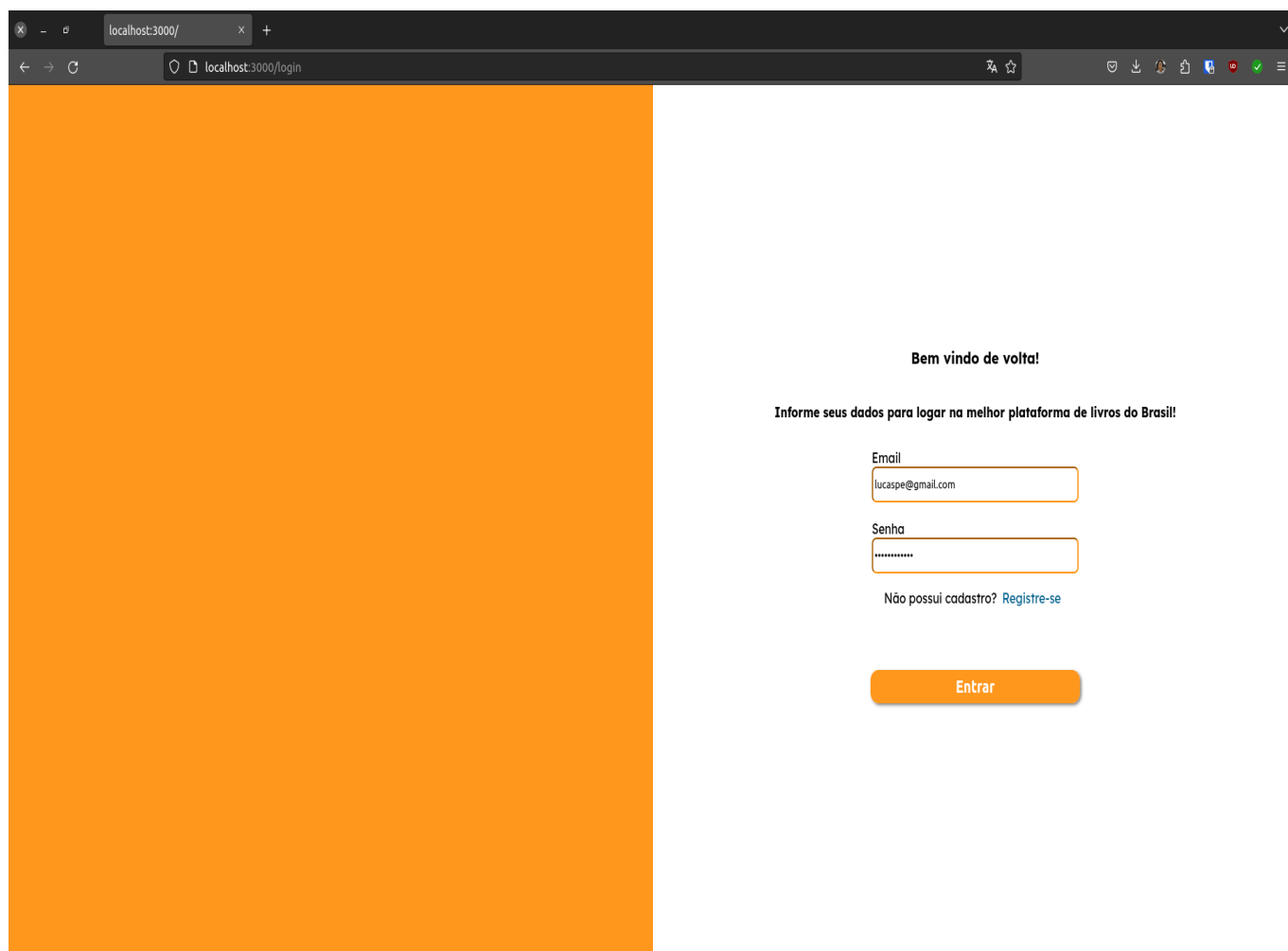
1. [Login](#)
2. [Registrar](#)
3. [Visualizar Materiais](#)

4.1 Barra



- Para fazer login, clique em 'Faça login'.
- Para fazer um cadastro, clique em 'cadastre-se'.

4.2. Login



localhost:3000/

localhost:3000/login

Bem vindo de volta!

Informe seus dados para logar na melhor plataforma de livros do Brasil!

Email
lucaspe@gmail.com

Senha

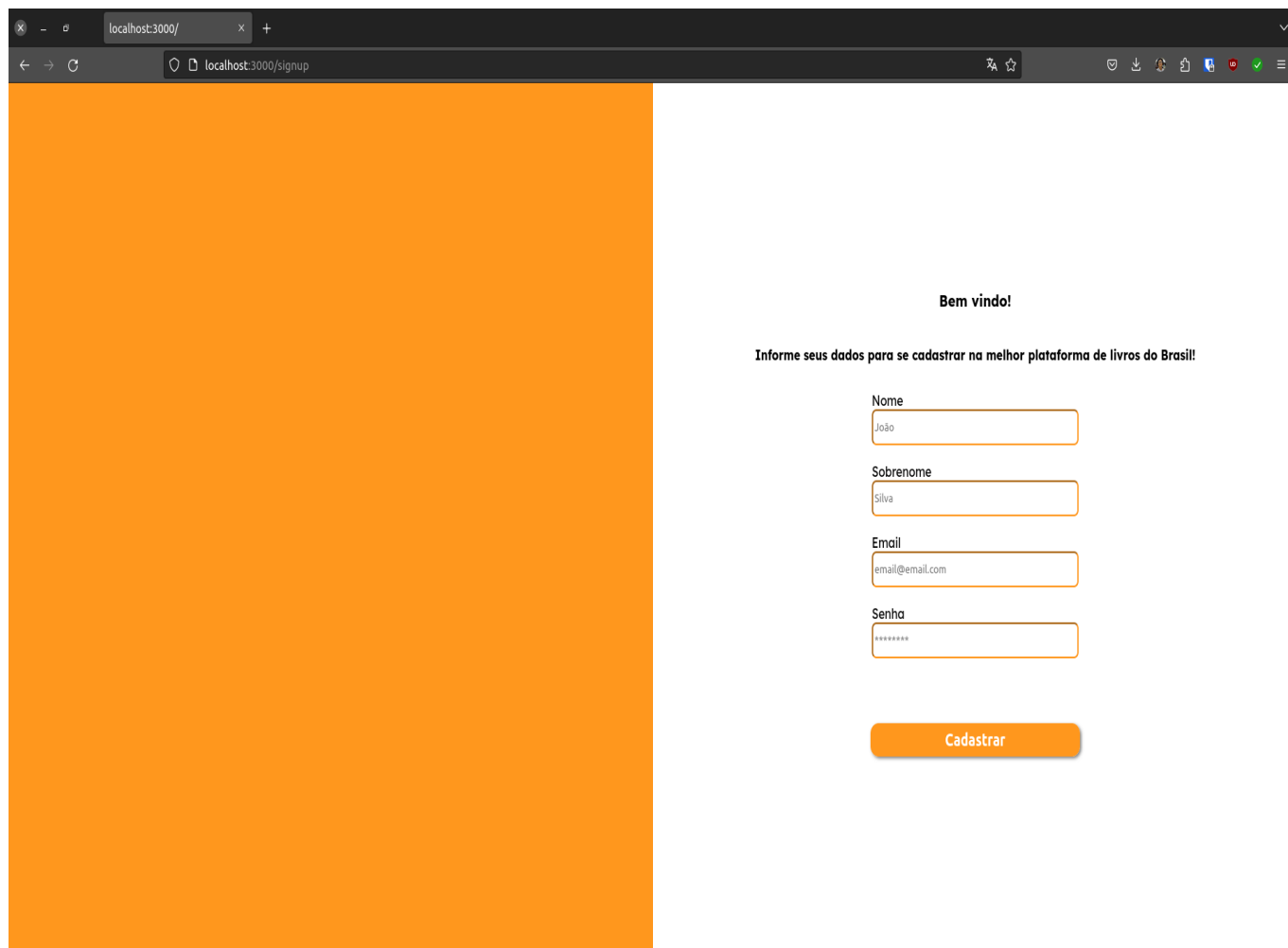
Não possui cadastro? [Registre-se](#)

Entrar

Para acessar o sistema, siga os passos abaixo:

- Abra o navegador e vá para a página de login em <http://localhost:3000/login>.
- Digite seu email de usuário e senha nos campos fornecidos.

4.3. Registrar

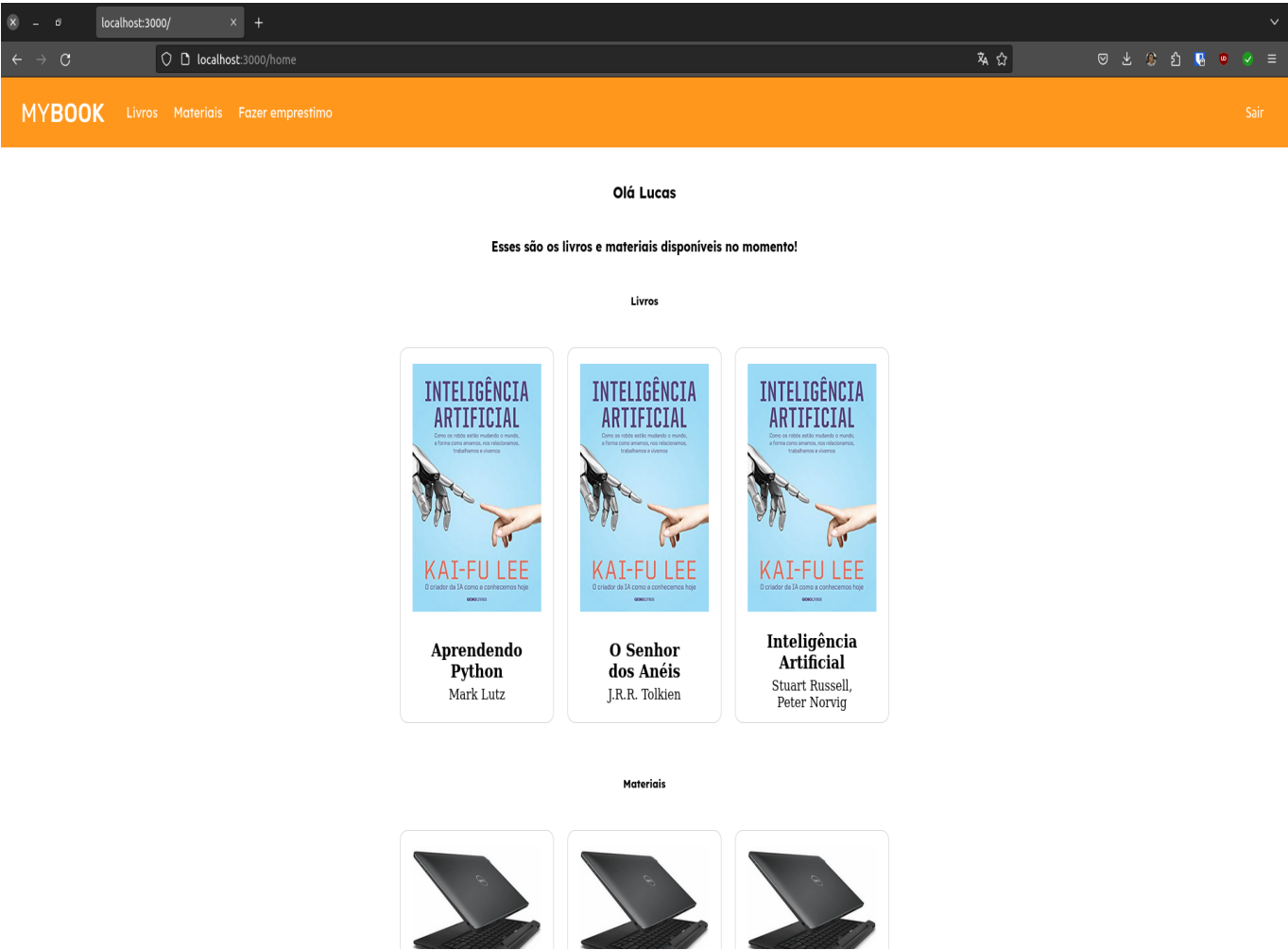


The screenshot shows a web browser window with the address bar displaying 'localhost:3000/signup'. The page has a large orange rectangular area on the left side. On the right side, there is a form titled 'Bem vindo!' (Welcome!) with the subtitle 'Informe seus dados para se cadastrar na melhor plataforma de livros do Brasil!' (Provide your data to register on the best book platform in Brazil!). The form contains four input fields: 'Nome' (Name) with the value 'João', 'Sobrenome' (Surname) with the value 'Silva', 'Email' with the value 'email@email.com', and 'Senha' (Password) with a masked value '*****'. Below the fields is an orange button labeled 'Cadastrar' (Register).

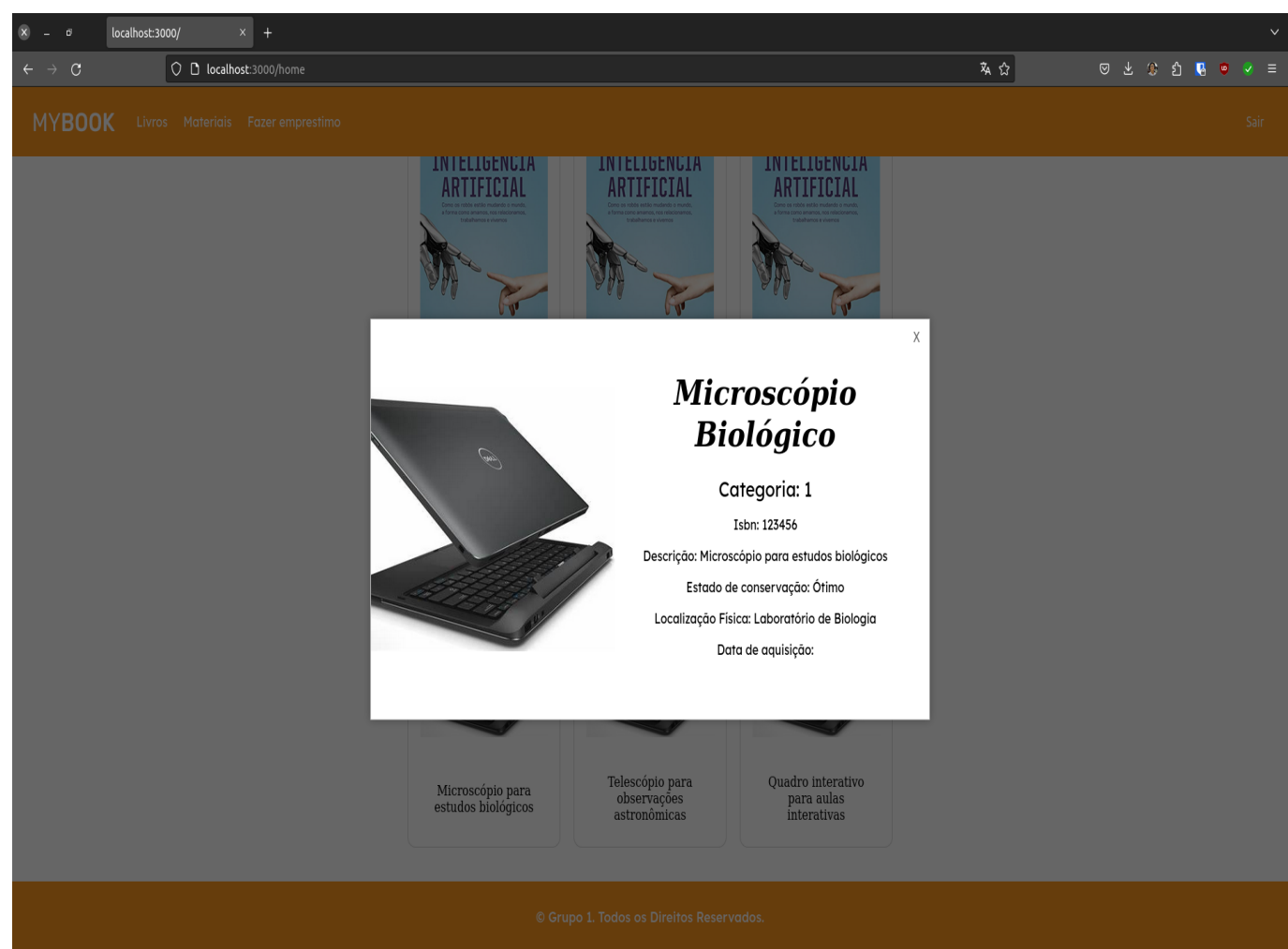
Para se registrar no sistema, siga os passos abaixo:

- Abra o navegador e vá para a página de login em <http://localhost:3000/signup>.
- Digite seu nome, sobrenome, email de usuário e senha.

4.4. Página inicial



- Para visualizar a localização física, clique em um material.



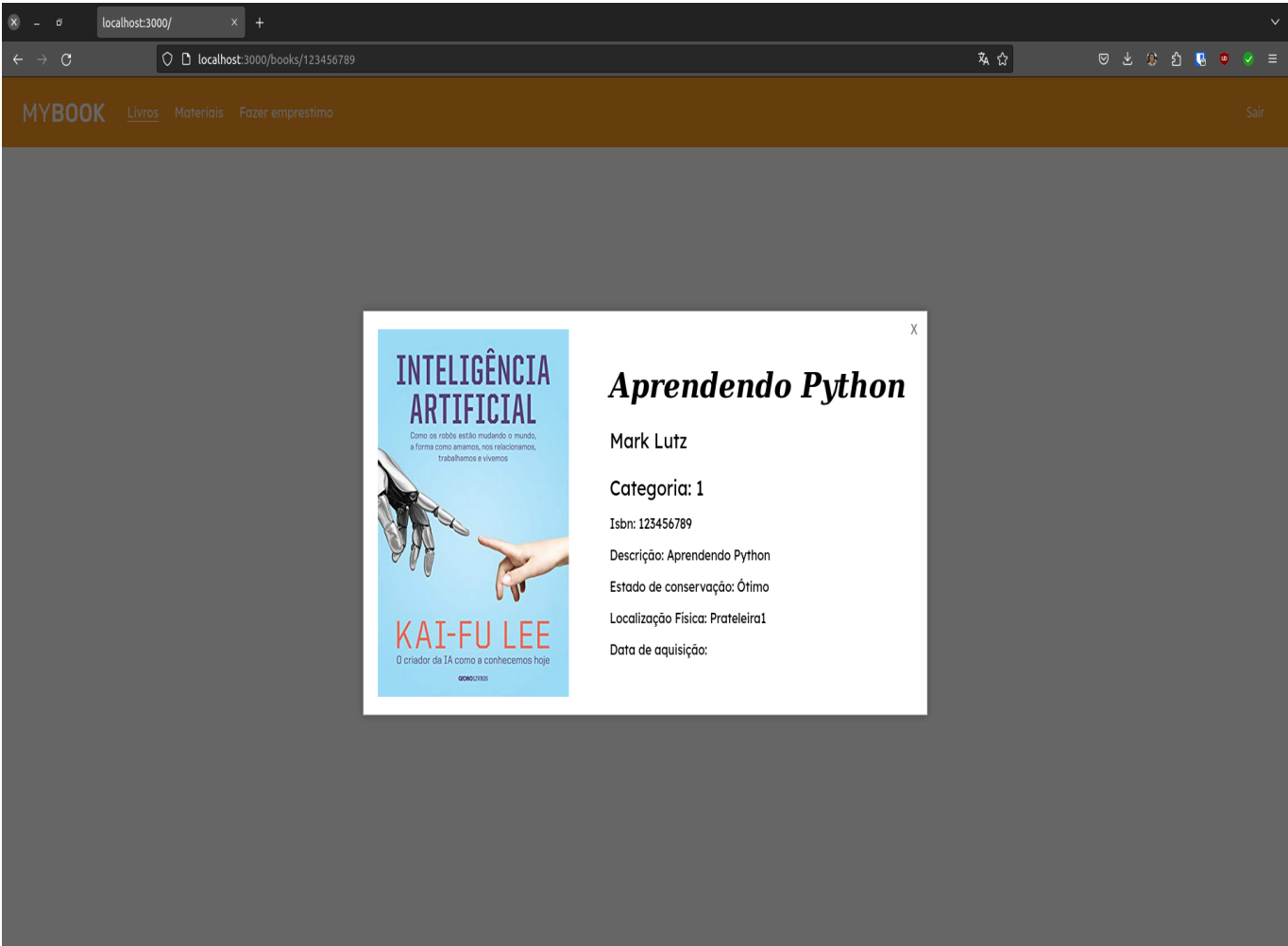
4.5. Livros



Livros disponíveis!



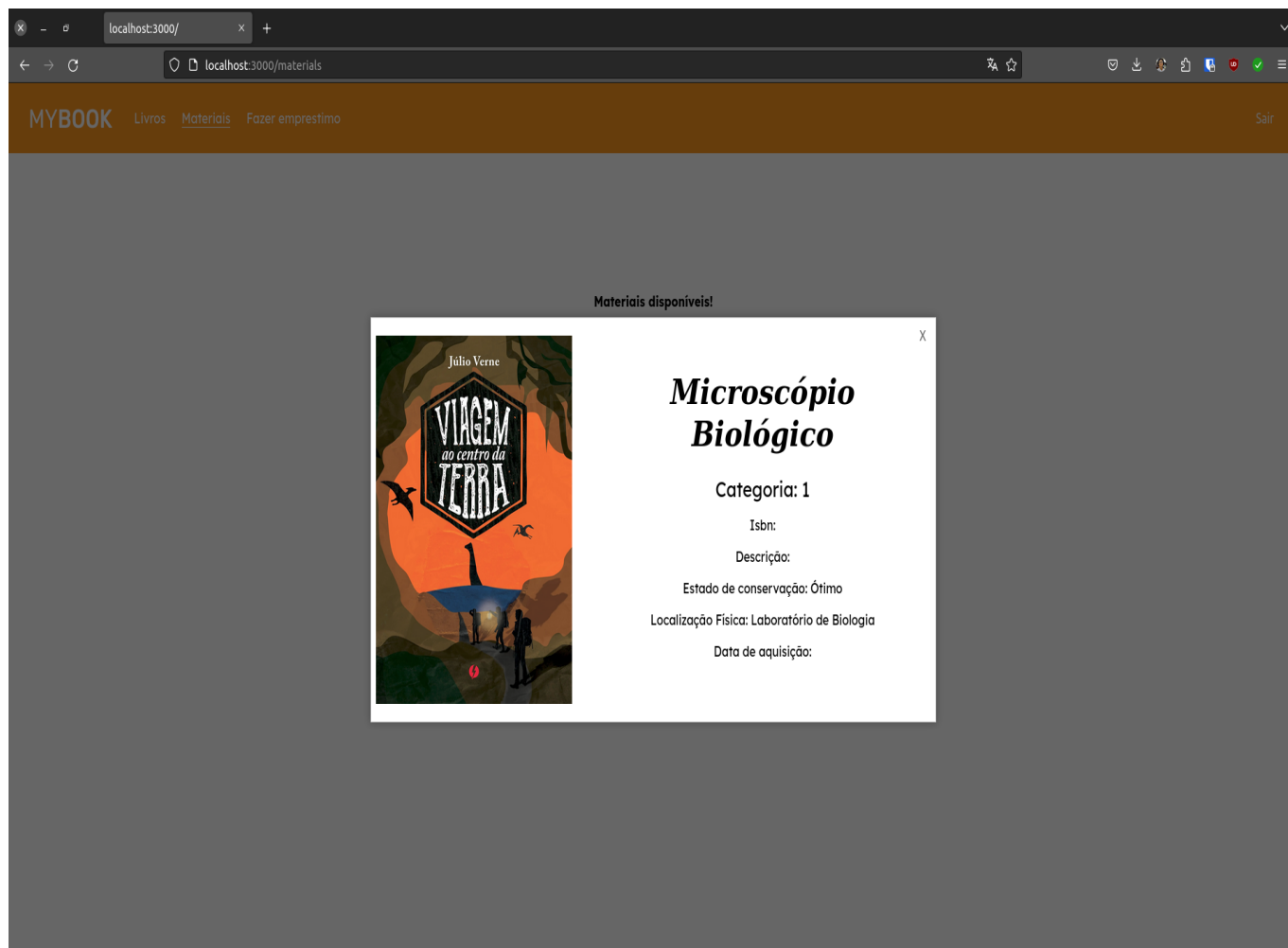
- Para saber detalhes de um livro clique no modelo.



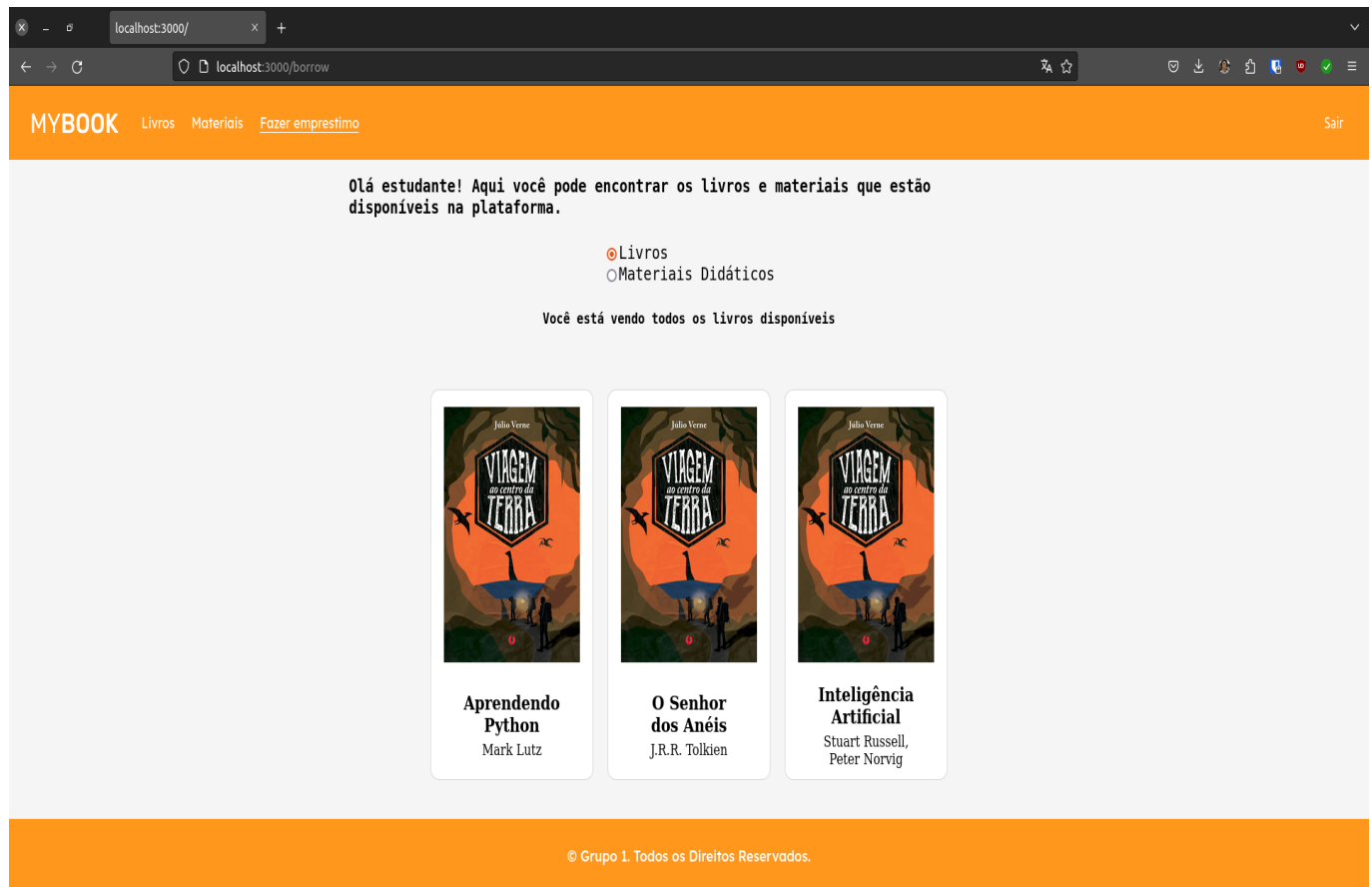
4.6. Materiais



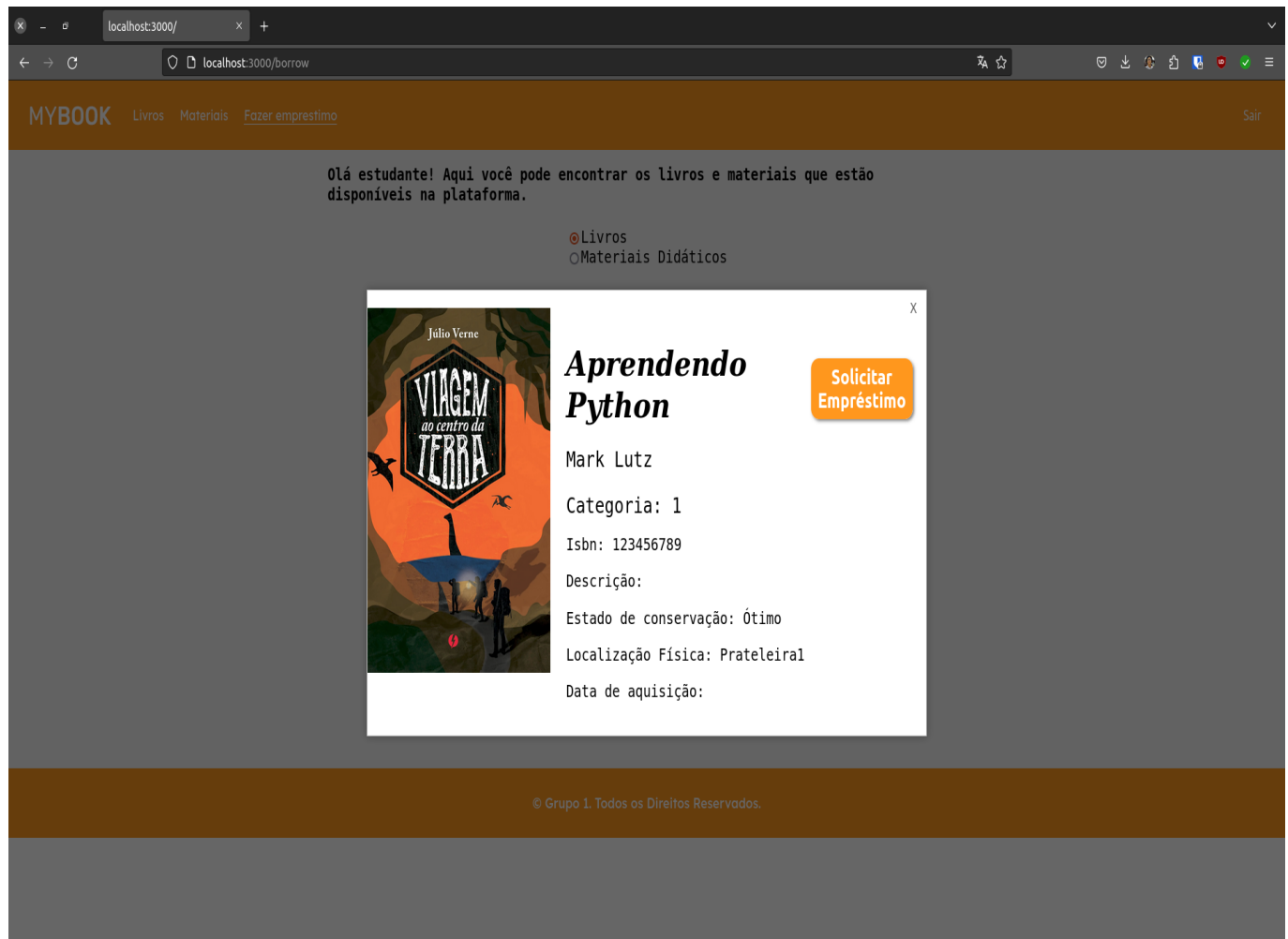
- Para saber detalhes de um material clique no modelo.



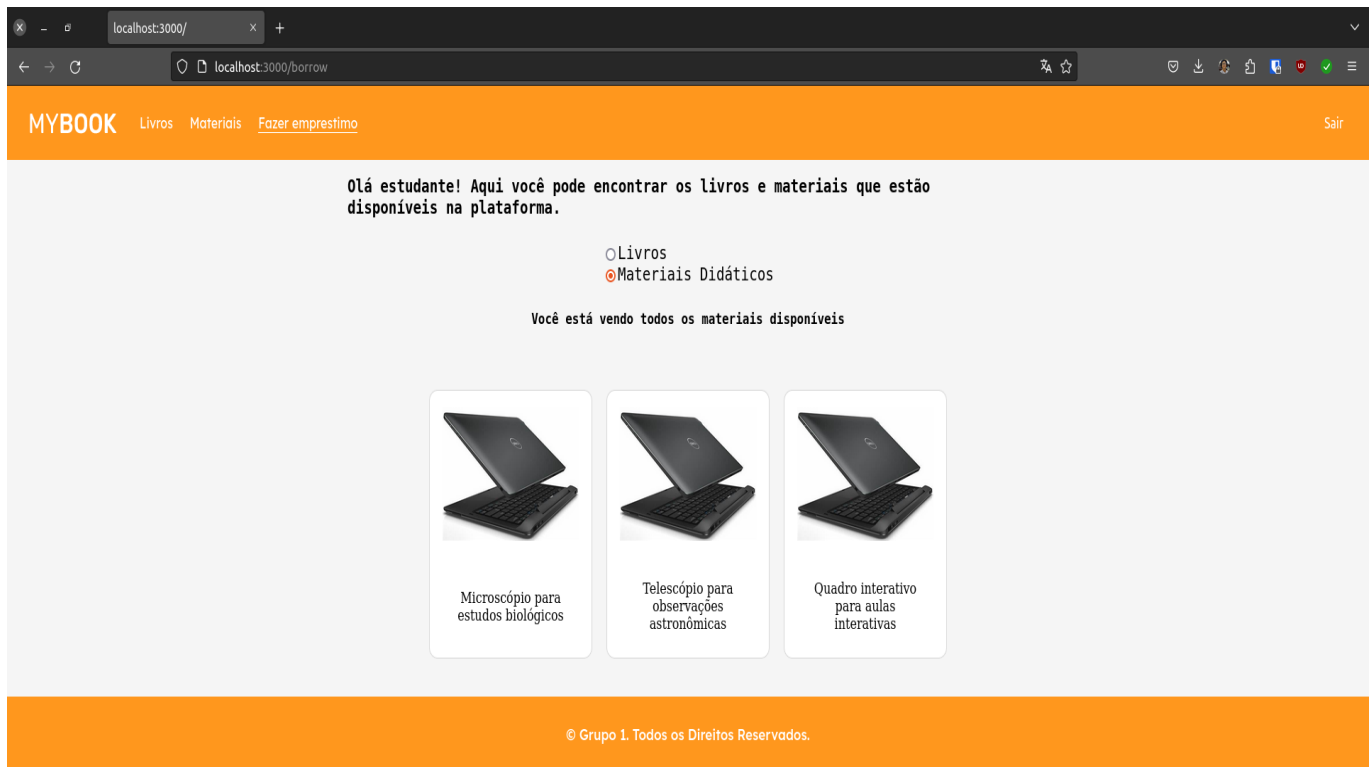
4.7. Fazer empréstimo



- Selecione um tipo de empréstimo que deseja fazer para mostrar somente esses itens.
- No exemplo acima temos **Livros** selecionado.
- Para visualizar os detalhes do livro e solicitar um empréstimo, clique no item desejado.



- Para solicitar empréstimo clique em 'Solicitar Empréstimo'.



- Selecione um tipo de empréstimo que deseja fazer para mostrar somente esses itens.
- No exemplo acima temos **Materiais Didáticos** selecionado.
- Para visualizar os detalhes do material e solicitar um empréstimo, clique no item desejado.



- Para solicitar empréstimo clique em 'Solicitar Empréstimo'.

5. Relatório de Implantação

O sistema pode ser muito útil em ambientes reais como escolas e bibliotecas, privadas ou públicas, visando automatizar a rotina de empréstimo/devolução de livros e demais materiais que possuem, como laptops, tablets, etc. Além do mais o sistema pode funcionar muito bem como cabines de consulta, sendo microcomputadores com pouco recursos de hardware e com custo menor, o que o torna mais versátil e possível de aplicar em vários ambientes de mundo real.

6. Relatório de Correções

Relatório de Mudança em Tabela de Banco de Dados

Mudança Realizada: Foi adicionada uma nova tabela chamada Roles ao banco de dados.

Detalhes da Mudança:

- **Nova Tabela:**

```
CREATE TABLE IF NOT EXISTS Roles (  
  "id" SERIAL PRIMARY KEY,
```

```
    "nome" varchar(255) NOT NULL,  
    CONSTRAINT "uc_roles_nome" UNIQUE ("nome")  
);
```

Essa mudança visa introduzir uma tabela para armazenar papéis (Roles) no sistema. A tabela possui um identificador único automático (id) e um campo para o nome do papel, garantindo que não haja duplicatas através de uma restrição de chave única.

Para qualquer dúvida ou esclarecimento adicional, favor entrar em contato.