IFSP - Instituto Federal de Educação, Ciência e Tecnologia de São Paulo
BANCO DE DADOS II

Lucas Gabriel (JC3033091), Andrey Rodrigues(JC3033872), Guilherme Frazão(JC3033635) e João Pedro de Andrade(JC303366X)

Sistema de Gestão de Biblioteca

Jacareí - SP
2025

Sumário

1. Divisão do trabalho

Lucas Gabriel: Implementação das rotas, regras de negócio e elaboração do relatório;
Andrey Rodrigues: Implementação dos ORM e implementação dos relacionamentos entre entidades no lado do servidor;
Guilherme Frazão: Modelagem dos diagramas e elaboração do relatório;
João Pedro de Andrade: Desenvolvimento da interface web e aplicação das rotas no lado do cliente.

2. Tecnologias utilizadas

As tecnologias foram escolhidas com base no conhecimento e experiência com elas dos alunos envolvidos, para que fosse construído o melhor sistema possível no tempo hábil.

2.1. Backend:

O backend é responsável por toda a lógica de negócio, interações com a base de dados e por fornecer uma API para o frontend.

- **Python:**
  - Linguagem de programação principal do projeto. Foi escolhida pela sua sintaxe simples, ecossistema robusto e vasta quantidade de bibliotecas, o que acelera o desenvolvimento.

- **Flask:**
  - Microframework web para Python. A sua natureza minimalista e modular foi ideal para este projeto, permitindo construir uma aplicação leve e rápida, focada principalmente em fornecer uma API RESTful, sem incluir funcionalidades desnecessárias.

- **SQLAlchemy:**
  - Biblioteca de mapeamento objeto-relacional (ORM) e toolkit SQL. Foi uma escolha crucial por duas razões:

    1. ORM: Permite interagir com a base de dados utilizando objetos Python em vez de escrever SQL manualmente, o que simplifica o código, aumenta a produtividade e ajuda a prevenir erros comuns. A sua capacidade de gerir relações complexas, como a herança polimórfica dos usuários, foi fundamental.

2. Core (Toolkit SQL): Como visto na função de filtro, o SQLAlchemy também nos dá a flexibilidade de descer para um nível mais baixo e construir ou executar queries SQL puras (text()), oferecendo controle total quando necessário.

2.2. Frontend

O frontend é responsável pela interface com o administrador, permitindo a gestão de livros, autores e empréstimos de forma dinâmica.

- **HTML5:**
  - **Justificativa:** Linguagem de marcação padrão para estruturar o conteúdo das páginas web, definindo elementos como formulários, tabelas e listas.
- **CSS3:**
  - **Justificativa:** Utilizado para a estilização visual dos componentes. A abordagem de CSS puro foi adotada para manter o código leve e ter controle total sobre a aparência da aplicação.
- **JavaScript (Vanilla):**
  - **Justificativa:** Linguagem de programação executada no navegador. É o motor por trás da interatividade da aplicação. Optou-se por JavaScript (vanilla) para evitar dependências de frameworks pesados (como React ou Angular), tornando as páginas mais rápidas e o código mais focado nas necessidades específicas do projeto.
- **Fetch API:**
  - **Justificativa:** Interface moderna do JavaScript para realizar requisições de rede (AJAX). É utilizada extensivamente para a comunicação assíncrona entre o frontend e a API do backend, permitindo que as páginas sejam atualizadas dinamicamente (carregar listas, adicionar/editar/deletar itens) sem a necessidade de recarregar a página inteira.

2.3. Base de Dados

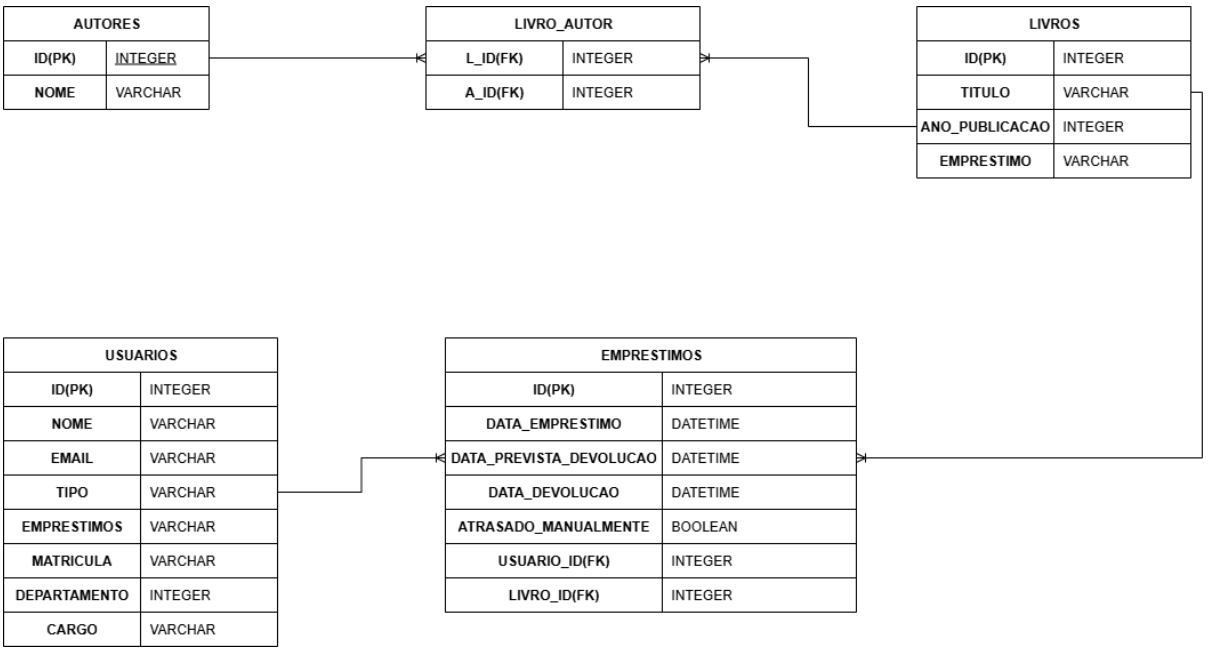
- **Modelo Relacional:**
  - **Justificativa:** A natureza dos dados do sistema (livros, autores, usuários com relações claras entre si) adapta-se perfeitamente a uma base de dados relacional. Este modelo garante a integridade e a consistência dos dados através de chaves primárias, chaves estrangeiras e transações. A escolha específica de um SGBD como o SQLite é ideal para desenvolvimento pela sua simplicidade (um único ficheiro, sem necessidade de servidor), podendo ser facilmente trocado por um sistema mais robusto como o PostgreSQL em produção.

2.4. Arquitetura e Padrões

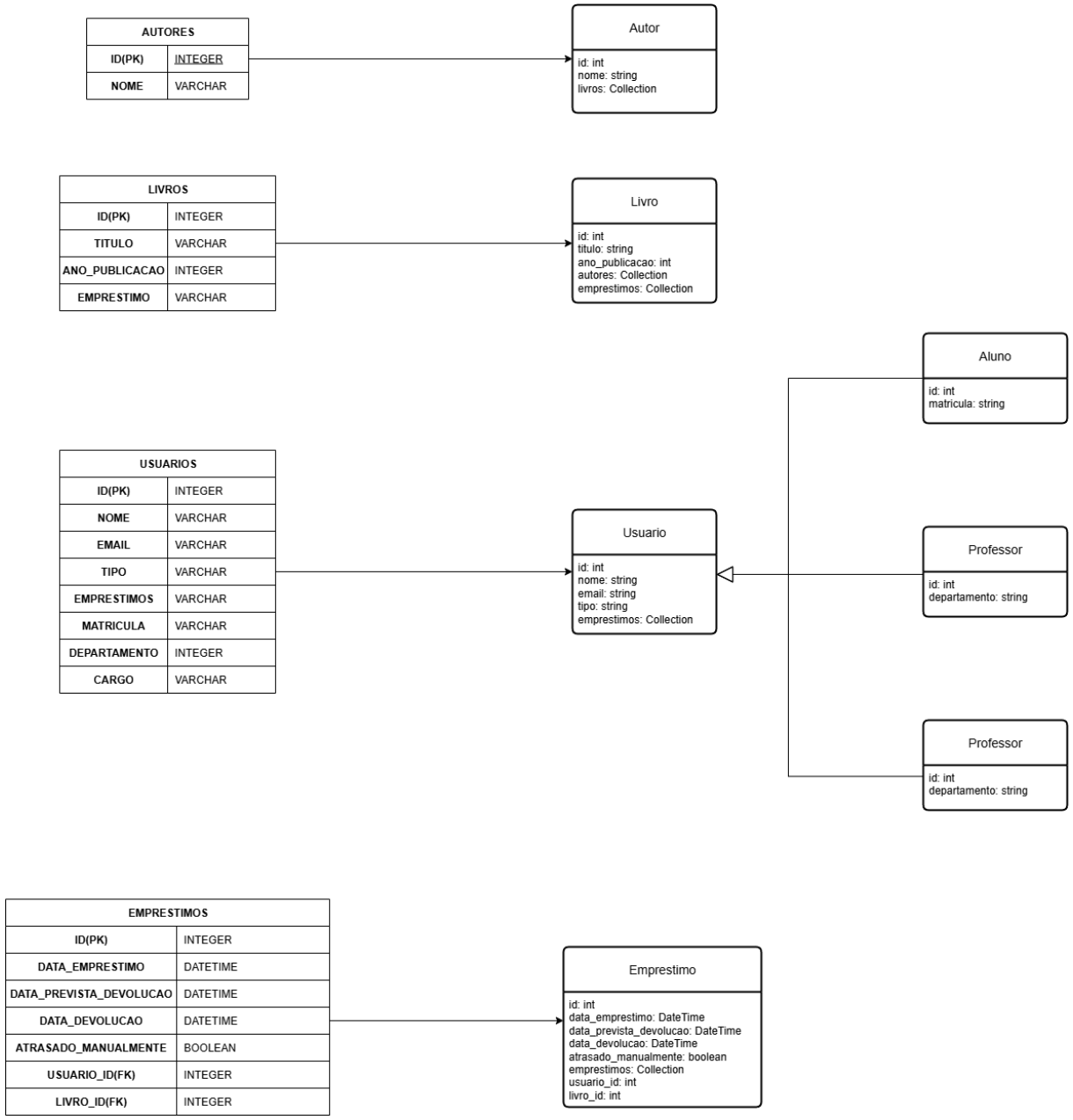
- **API RESTful:**
    - **Justificativa:** A comunicação entre o backend e o frontend segue os princípios REST (Representational State Transfer). Isto significa que utilizamos endpoints baseados em recursos (ex: `/livros`, `/usuarios`) e os métodos HTTP (GET, POST, PUT, DELETE) para as operações. Esta abordagem desacopla o frontend do backend, permitindo que possam ser desenvolvidos e mantidos de forma independente.
- **Blueprints (Flask):**
    - **Justificativa:** Para organizar o código, a aplicação foi dividida em "Blueprints". Um blueprint é um conjunto de rotas que pode ser registrado na aplicação principal. Utilizamos esta abordagem para separar as rotas da API das rotas que renderizam as páginas do painel de administração, tornando o projeto mais fácil de navegar e manter.

3. Modelagem Conceitual e Relacional

3.1. Diagrama Entidade-Relacionamento



3.2. Conversão para o Modelo Relacional



3.3. Justificar escolhas de cardinalidade e integridade

- **Livro - Autor:**
  - Cardinalidade:
    - Um livro pode ter vários autores e um autor pode ter vários livros
    - Esse relacionamento N:N é resolvido pela tabela livro_autor
  - Integridade
    - A tabela livro_autor possui chave primária composta (livro_id, autor_id). Fazer com que a chave primária seja desta maneira evita um autor de ser vinculado ao mesmo livro duas vezes

- **Usuário - Empréstimo:**
  - Cardinalidade:
    - Um usuário pode ter vários empréstimos ao mesmo tempo e cada empréstimo está vinculado a apenas um usuário
  - Integridade:
    - A Foreign Key usuario_id da tabela empréstimo garante que não exista empréstimo sem usuário associado

- **Livro - Empréstimo**
  - Cardinalidade:
    - Um livro pode ser emprestado várias vezes em momentos diferentes e cada empréstimo se refere a apenas um exemplar de livro
  - Integridade:
    - De maneira semelhante à relação entre usuário e empréstimo a Foreign Key livro_id da tabela empréstimo garante que não irá existir um empréstimo sem livro associado

4. Demonstração do Sistema

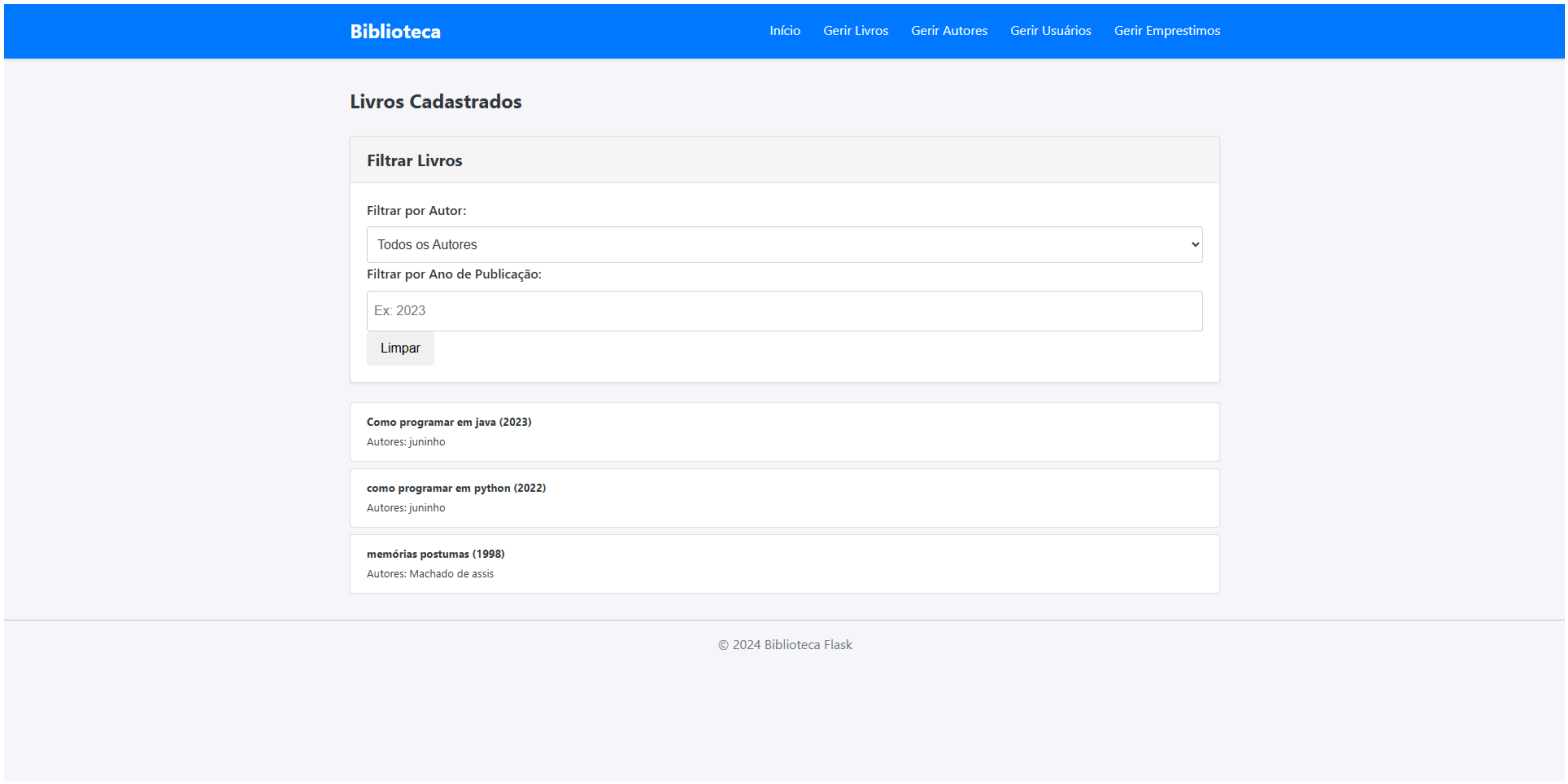


Figura 3 - Tela inicial

Na tela inicial (Figura 3), o usuário poderá ver todos os livros cadastrados no sistema, além de seu ano de publicação e autor, e também poderá filtrá-los por seus metadados (também

autores e ano de publicação). Fora isso, nesta e em todas as outras páginas, há uma barra de navegação (barra azul no topo da página), que permite ao usuário navegar entre as diferentes páginas do sistema.

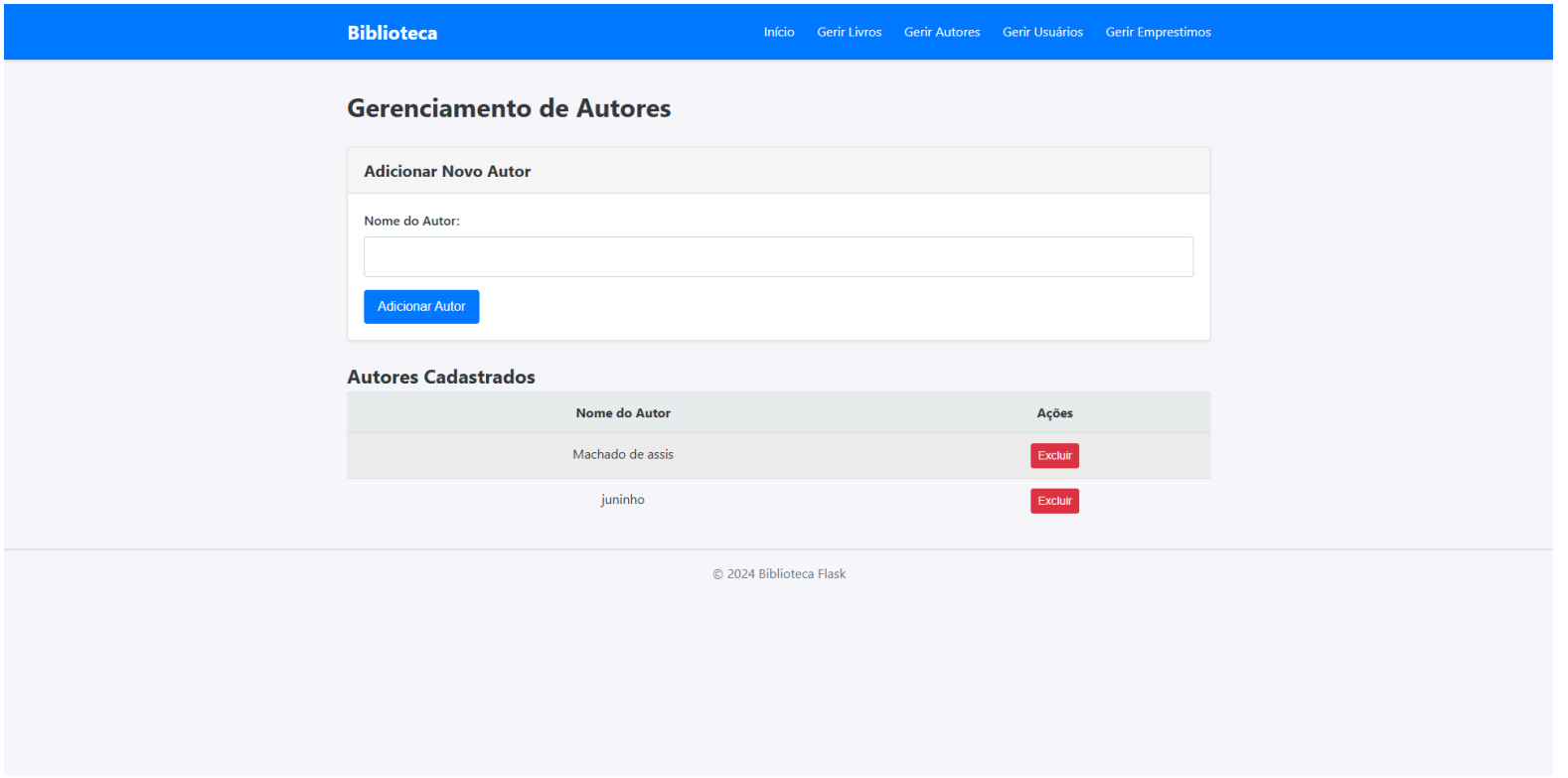


Figura 4 - Página de gerenciamento dos autores

Na página de gerenciamento dos autores (Figura 4), o usuário poderá adicionar novos autores, além de visualizar e excluir os já existentes.

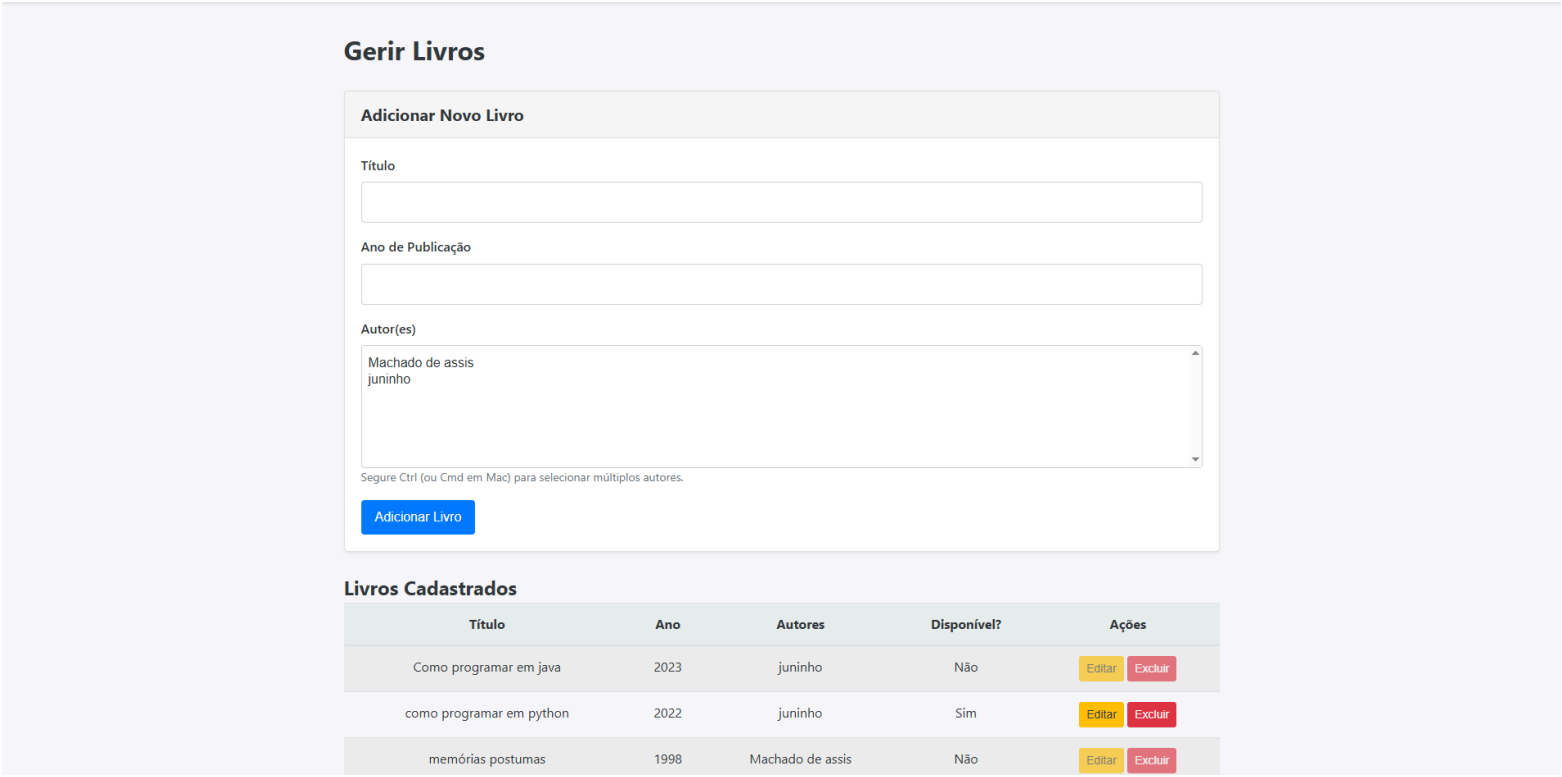


Figura 5 - Página de gerenciamento dos livros

Na página de gerenciamento dos livros (Figura 5), o usuário poderá adicionar novos livros, inserindo seu nome, ano de publicação e autores (podendo adicionar mais de um) , além de visualizar, editar e excluir os já existentes.

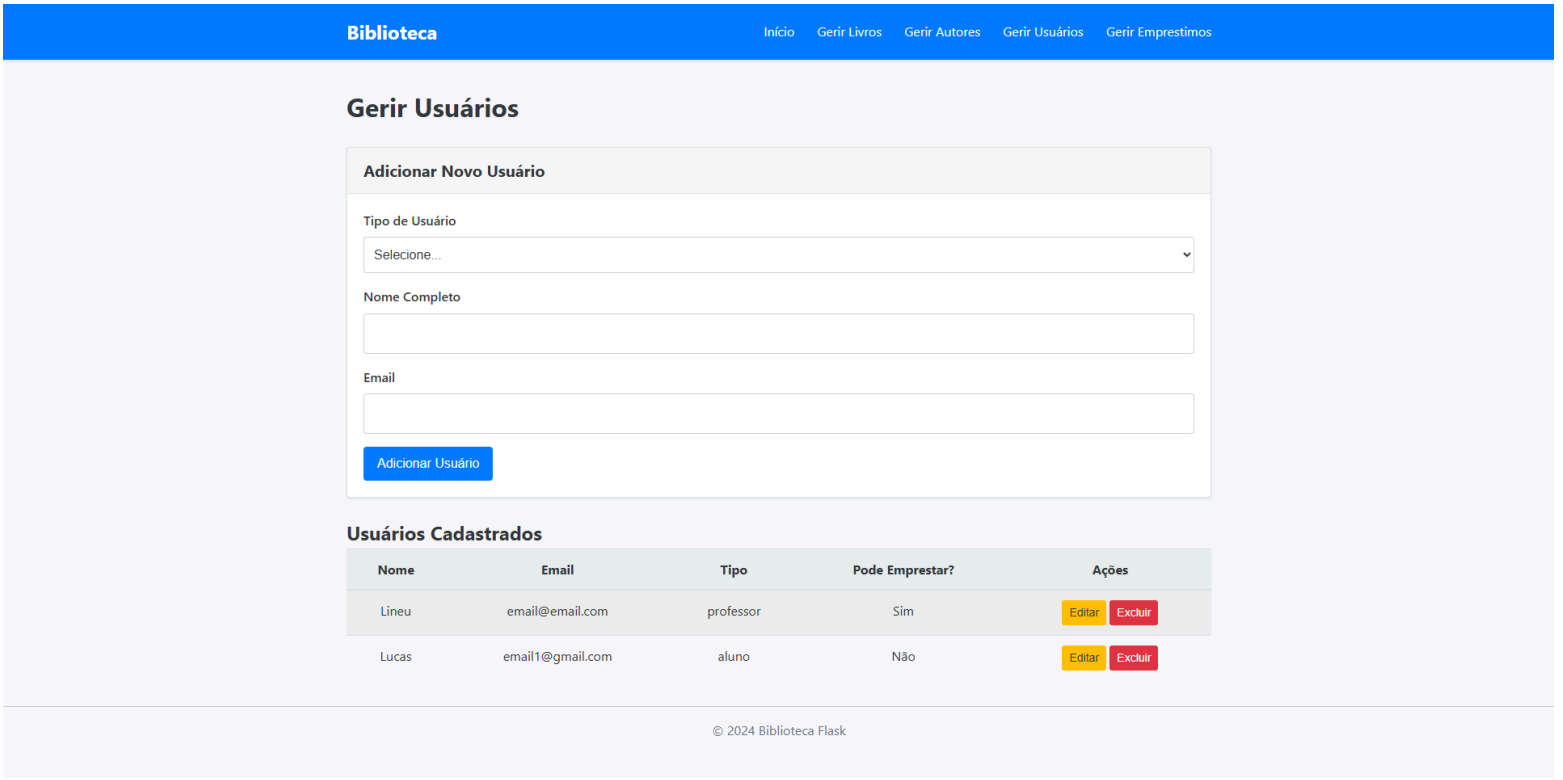


Figura 6 - Página de gerenciamento dos usuários

Na página de gerenciamento dos usuários (Figura 6), o usuário poderá adicionar novos usuários, inserindo seu nome, email, tipo de usuário (aluno, funcionário ou professor) e em cada tipo de usuário, uma informação extra será solicitado, sendo essas: matrícula, cargo e departamento, respectivamente; além de poder visualizar, editar e excluir os usuários já existentes.

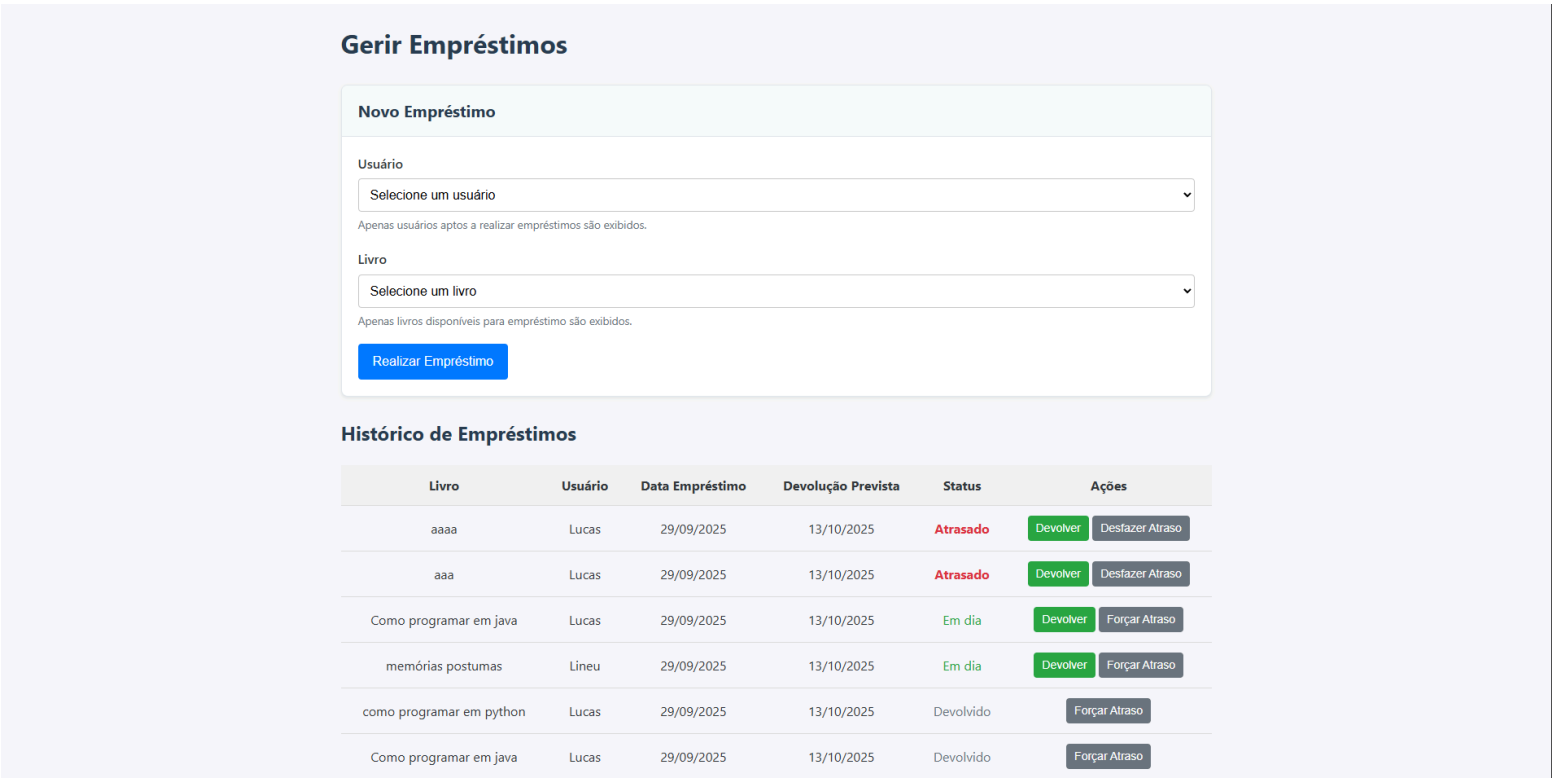


Figura 7 - Página de gerenciamento de empréstimos

Na página de gerenciamento de empréstimos (Figura 7), o usuário poderá registrar empréstimos, escolhendo o usuário (apenas usuário existentes e que podem fazer um empréstimo serão exibidos) e o livro (apenas livros disponíveis serão exibidos). Além de poder acompanhar os empréstimos já registrados, registrar uma devolução ou forçar uma situação de atraso (opção feita para testes e demonstração da situação de atraso).

5. Arquitetura de Rotas

As rotas do sistema foram estruturadas em dois tipos de Blueprints distintos para garantir uma separação clara de responsabilidades, seguindo as melhores práticas de desenvolvimento com Flask. O primeiro blueprint, admin_bp, é responsável por renderizar e servir as páginas HTML do painel de administração, que compõem a interface visual do sistema. O segundo, em que cada um recebe o nome da entidade em questão, definem uma

API RESTful completa, responsável por toda a manipulação de dados (criação, leitura, atualização e exclusão). Esta abordagem desacoplada permite que o frontend (JavaScript) interaja com o backend de forma assíncrona, proporcionando uma experiência de usuário dinâmica e responsiva, sem a necessidade de recarregar as páginas a cada ação.

5.1. Rotas do Painel de Administração (admin_bp)

Estas rotas são responsáveis por renderizar as páginas HTML que o administrador utiliza para gerir o sistema.

Método	Endpoint	Descrição
GET	/	Renderiza a página inicial da biblioteca, que exibe a lista de livros com funcionalidades de filtro.
GET	/admin/usuarios	Renderiza a página de gestão de usuários (leitores), com formulário para adição e uma lista de usuários existentes.
GET	/admin/livros	Renderiza a página de gestão de livros, com formulário para adição e uma lista de livros existentes.
GET	/admin/autores	Renderiza a página de gestão de autores, com formulário para adição e uma lista de autores existentes.
GET	/admin/emprestimos	Renderiza a página de gestão de empréstimos, com formulário para novos empréstimos e o histórico completo.

Tabela 1 - Tabela de rotas de renderização

5.2 Rotas da API

Estes endpoints fornecem a interface de dados no formato JSON, utilizada pelo JavaScript das páginas de administração.

Entidade: Usuários (usuarios_bp)

Método(s)	Endpoint	Descrição
GET, POST	/usuarios	GET: Lista todos os usuários. POST: Cria um novo usuário (aluno, professor ou funcionário).

GET, PUT, DELETE	/usuarios/<id>	GET: Obtém um usuário específico. PUT: Atualiza um usuário. DELETE: Remove um usuário.

Tabela 2 - Tabela de rotas de api dos usuários

Entidade: Livros (livros_bp)

Método(s)	Endpoint	Descrição
GET, POST	/livros	GET: Lista todos os livros, permitindo filtros por autor e ano na URL. POST: Cria um novo livro.
GET, PUT, DELETE	/livros/<id>	GET: Obtém um livro específico. PUT: Atualiza um livro. DELETE: Remove um livro e seus empréstimos associados.

Tabela 3 - Tabela de rotas de api dos livros

Entidade: Autores (autores_bp)

Método(s)	Endpoint	Descrição
GET, POST	/autores	GET: Lista todos os autores. POST: Cria um novo autor.
GET, DELETE	/autores/<id>	GET: Obtém um autor específico. DELETE: Remove um autor, se ele não estiver associado a nenhum livro.

Tabela 4 - Tabela de rotas de api dos autores

Entidade: Empréstimos (emprestimos_bp)

Método(s)	Endpoint	Descrição
GET, POST	/emprestimos	GET: Lista todos os autores. POST: Cria um novo autor.

GET, PUT	/emprestimos/<id>	GET: Obtém um empréstimo específico. PUT: Atualiza um empréstimo (para devolução ou para forçar/desfazer o estado de atraso).

Tabela 5 - Tabela de rotas de api dos empréstimos

6. Regras de Negócio Implementadas

Para garantir a integridade dos dados e o funcionamento correto da biblioteca, diversas regras de negócio foram implementadas diretamente no backend da aplicação. Estas regras são validadas em tempo real através da API antes de qualquer operação de escrita na base de dados, assegurando que o sistema permaneça sempre num estado consistente e válido.

6.1. Limite de Empréstimos por Tipo de Usuário

Cada tipo de usuário possui um limite máximo de livros que pode emprestar simultaneamente. Esta regra é implementada através de uma propriedade polimórfica nos modelos.

Alunos: 3 livros.

Funcionários: 5 livros.

Professores: 10 livros.

6.2. Bloqueio de Empréstimo por Atraso

Um usuário não pode realizar um novo empréstimo se possuir qualquer livro com a devolução em atraso. O sistema verifica esta condição antes de autorizar a criação de um novo registo de empréstimo.

6.3. Disponibilidade de Livros

Um livro que já se encontra emprestado (ou seja, tem um registo de empréstimo ativo sem data de devolução) não pode ser emprestado a outro usuário. A API valida a disponibilidade do livro antes de confirmar a operação.

6.4. Integridade na Exclusão de Entidades

Para manter a consistência referencial da base de dados, foram implementadas duas regras de exclusão:

Autores: Um autor não pode ser excluído se estiver associado a um ou mais livros no sistema.

Usuários: Um usuário (leitor) não pode ser excluído se possuir empréstimos ativos (livros ainda não devolvidos).

7. Demonstração de Consultas Avançadas

Para além das operações CRUD (Criar, Ler, Atualizar, Deletar) básicas, o sistema implementa uma funcionalidade de consulta avançada na sua página inicial, permitindo ao administrador filtrar a lista de livros de forma dinâmica por autor e por ano de publicação. Esta funcionalidade foi desenhada para ser eficiente e segura, delegando o processamento da filtragem diretamente para a base de dados.

7.1. Funcionamento da Filtragem

A filtragem opera através da interação entre o frontend e um endpoint de API parametrizado:

Frontend (JavaScript): Na página inicial, index.html, um script JavaScript monitora os campos de filtro. Sempre que o administrador seleciona um autor ou digita um ano, o script constrói dinamicamente uma URL de requisição para a API, anexando os filtros como parâmetros de consulta (query parameters).

Backend (API): A rota GET /livros foi projetada para receber e interpretar estes parâmetros. Internamente, uma função auxiliar constrói uma query SQL pura (utilizando `text()` do SQLAlchemy) que incorpora as cláusulas `JOIN` e `WHERE`necessárias com base nos filtros fornecidos. Para garantir a segurança contra ataques de injeção de SQL, os valores dos filtros são passados como "named parameters", sendo higienizados pela biblioteca antes da execução.

Resultado: A base de dados executa a query otimizada e retorna apenas os livros que correspondem aos critérios. O backend então serializa estes resultados para JSON e envia-os de volta para o frontend, que atualiza a lista de livros na página sem a necessidade de um recarregamento completo.

7.2. Exemplo Prático

Vamos considerar um cenário em que o administrador deseja ver todos os livros do autor com ID = 5, que foram publicados no ano de 2021.

- **URL da Requisição (gerada pelo frontend):**

```
GET http://localhost:5000/livros?autor_id=5&ano=2021
```

- Query SQL (construída e executada pelo backend):

```
SELECT livros.*
FROM livros JOIN livro_autor ON livros.id = livro_autor.livro_id
WHERE livro_autor.autor_id = :autor_id AND livros.ano_publicacao = :ano
ORDER BY livros.titulo
```

(com os parâmetros `:autor_id = 5` e `:ano = 2021`)

Esta abordagem garante que apenas os dados relevantes são transferidos pela rede e que a lógica de filtragem complexa é executada de forma eficiente pelo sistema de gestão da base de dados.

8. Demonstração de Persistência e Transação

A fiabilidade de qualquer sistema de gestão depende diretamente da sua capacidade de armazenar dados de forma segura e consistente. No contexto deste sistema, estes dois conceitos são fundamentais: a persistência garante que os dados não se perdem quando a aplicação é encerrada, e o controle de transações assegura que as operações na base de dados são realizadas de forma atómica e segura, prevenindo a corrupção dos dados.

8.1. Persistência de Dados via ORM

A persistência de dados no sistema é gerida pelo SQLAlchemy através do seu Mapeamento Objeto-Relacional (ORM). Esta camada de abstração permite que os dados sejam tratados como objetos Python dentro da aplicação, enquanto o ORM se encarrega de traduzir estes objetos em linhas numa tabela da base de dados relacional (SQLite). Por exemplo, um objeto da classe Livro em Python corresponde a um registo na tabela livros. Quando este objeto é salvo, o seu estado é "persistido" no disco, garantindo que a informação do livro sobreviva para além da execução do programa.

8.2. Controle de Transações (Commit & Rollback)

Uma transação é uma sequência de uma ou mais operações na base de dados que são tratadas como uma única unidade de trabalho. O princípio fundamental de uma transação é a atomicidade: ou todas as operações são concluídas com sucesso (um commit), ou, se ocorrer qualquer erro, nenhuma das operações é aplicada e a base de dados reverte para o seu estado anterior (um rollback). Esta abordagem é crucial para manter a integridade dos dados.

8.3. Demonstração de Transação no Sistema

O controle de transações é aplicado em todas as rotas da API que modificam dados (`POST`, `PUT`, `DELETE`). O exemplo abaixo, retirado da rota de criação de um novo livro, demonstra perfeitamente este conceito.

```
# Trecho da rota POST /livros

@api_bp.route('/livros', methods=['POST'])
def livros_collection():
    data = request.get_json()
    # ... (validação dos dados recebidos) ...

    # Início do bloco de transação
    try:
        # 1. Busca os autores (operação de leitura dentro da transação)
        autores =
Autor.query.filter(Autor.id.in_(data['autores_ids'])).all()

        # 2. Cria o objeto Livro em memória
        novo_livro = Livro(
            titulo=data['titulo'],
            ano_publicacao=data['ano_publicacao'],
            autores=autores
        )

        # 3. Adiciona o novo livro à sessão (prepara para salvar)
        db.session.add(novo_livro)

        # 4. COMMIT: Tenta salvar todas as alterações na base de dados.
        #    Isto inclui inserir o novo livro na tabela 'livros' e criar
        #    as associações na tabela 'livro_autor'.
        #    Se tudo correr bem, a transação é concluída com sucesso.
        db.session.commit()

        return jsonify(serialize_livro(novo_livro)), 201

    except Exception as e:
        # 5. ROLLBACK: Se qualquer erro ocorrer durante o commit
        #    (ex: um autor_id não existe, violando uma foreign key),
        #    o rollback é acionado. Todas as alterações feitas
        #    dentro do bloco 'try' são desfeitas.
        db.session.rollback()

        # Retorna uma mensagem de erro, garantindo que a base de dados
        # permaneceu no seu estado consistente anterior.
```

```
        return jsonify({'erro': str(e)}), 500
```

Neste fluxo, a criação de um livro e a sua associação com autores são tratadas como uma única operação indivisível. O uso do padrão `try/except` com `commit` e `rollback` garante que a base de dados nunca ficará num estado inconsistente, como, por exemplo, ter um livro registado sem os seus autores devidamente associados.