Desenvolver uma aplicação backend completa utilizando Flask e FastAPI, aplicando os conceitos de HTTP, CRUD, e conexão com banco de dados relacional, simulando um ambiente real de desenvolvimento. Temas Avaliados:
• Protocolo HTTP (métodos, códigos de status e headers)
• Flask: estrutura de rotas, templates e requisições
• Banco de Dados (SQLite ou MySQL): conexão, criação de tabelas, CRUD
• FastAPI: criação de endpoints, validação de dados com Pydantic, documentação automática

Você foi contratado para desenvolver um sistema simples de cadastro de livros para uma biblioteca universitária. O sistema deve permitir que usuários possam cadastrar, listar, atualizar e excluir livros via API REST, e consultar as informações via navegador.

Requisitos Técnicos
Parte 1 – Banco de Dados
Crie um banco de dados chamado biblioteca.db (SQLite) ou biblioteca
(MySQL).
Tabela: livros
id (chave primária, autoincremento)
titulo (texto)
autor (texto)
ano_publicacao (inteiro)
disponivel (booleano)

Requisitos Técnicos
Parte 1 – Aplicação Flask
Crie um projeto Flask chamado app_flask.py que:
Tenha uma rota / que exiba uma página HTML com a lista de livros cadastrados.
Possua formulário (HTML + método POST) para adicionar novos livros.
Utilize templates Jinja2 e métodos HTTP (GET e POST).
Conecte-se ao banco de dados para realizar o insert e listagem.

Requisitos Técnicos
Parte 3 – API FastAPI
Crie um segundo arquivo api_fast.py que:
Implemente os endpoints REST para a mesma tabela livros:
GET /livros → lista todos os livros
GET /livros/{id} → obtém um livro específico
POST /livros → adiciona novo livro
PUT /livros/{id} → atualiza um livro existente
DELETE /livros/{id} → exclui um livro
Utilize Pydantic para validação dos dados.
Retorne JSON e use códigos de status HTTP adequados (201, 200, 404, 400).
Teste a API no Swagger (/docs).

Requisitos Técnicos
Parte 4 – Integração e Teste
Execute o Flask na porta 5000 e o FastAPI na porta 8000.
Teste as operações CRUD pelo navegador (Flask) e pela API (FastAPI).
Opcional: use requests para que o Flask consuma os endpoints da API FastAPI.

Entrega
Código-fonte (.py), banco de dados (.db ou .sql), e instruções de

execução (README.md).
O código deve conter comentários explicativos.

Vou criar um sistema completo de cadastro de livros para biblioteca universitária. Aqui está a implementação:

Parte 1 - Banco de Dados (SQLite)

database.py

```python
import sqlite3
from contextlib import contextmanager

class Database:
    def __init__(self, db_name='biblioteca.db'):
        self.db_name = db_name
        self.init_db()

    @contextmanager
    def get_connection(self):
        """Context manager para gerenciar conexões com o banco"""
        conn = sqlite3.connect(self.db_name)
        conn.row_factory = sqlite3.Row  # Para acessar colunas por nome
        try:
            yield conn
            conn.commit()
        except Exception as e:
            conn.rollback()
            raise e
        finally:
            conn.close()

    def init_db(self):
        """Inicializa o banco de dados criando a tabela se não existir"""
        with self.get_connection() as conn:
            conn.execute('''
                CREATE TABLE IF NOT EXISTS livros (
                    id INTEGER PRIMARY KEY AUTOINCREMENT,
                    titulo TEXT NOT NULL,
                    autor TEXT NOT NULL,
                    ano_publicacao INTEGER NOT NULL,
                    disponivel BOOLEAN DEFAULT TRUE
                )
            ''')

    def criar_livro(self, titulo, autor, ano_publicacao, disponivel=True):
        """Cria um novo livro no banco"""
        with self.get_connection() as conn:
            cursor = conn.execute(
                'INSERT INTO livros (titulo, autor, ano_publicacao, disponivel) VAL
UES (?, ?, ?, ?)',
                (titulo, autor, ano_publicacao, disponivel)
            )
            return cursor.lastrowid

    def listar_livros(self):
        """Lista todos os livros do banco"""
        with self.get_connection() as conn:
            cursor = conn.execute('SELECT * FROM livros ORDER BY id DESC')
            return [dict(row) for row in cursor.fetchall()]

    def buscar_livro_por_id(self, livro_id):
        """Busca um livro específico pelo ID"""
        with self.get_connection() as conn:
            cursor = conn.execute('SELECT * FROM livros WHERE id = ?', (livro_id,))
            row = cursor.fetchone()
            return dict(row) if row else None
```

```
    def atualizar_livro(self, livro_id, titulo, autor, ano_publicacao, disponivel):
        """Atualiza um livro existente"""
        with self.get_connection() as conn:
            conn.execute(
                'UPDATE livros SET titulo = ?, autor = ?, ano_publicacao = ?, dispo
nivel = ? WHERE id = ?',
                (titulo, autor, ano_publicacao, disponivel, livro_id)
            )
            return conn.total_changes > 0

    def excluir_livro(self, livro_id):
        """Exclui um livro do banco"""
        with self.get_connection() as conn:
            conn.execute('DELETE FROM livros WHERE id = ?', (livro_id,))
            return conn.total_changes > 0

# Instância global do banco de dados
db = Database()
```

Parte 2 - Aplicação Flask

app_flask.py

```
python

from flask import Flask, render_template, request, redirect, url_for, flash, jsonif
y
from database import db
import requests  # Para integração opcional com FastAPI

app = Flask(__name__)
app.secret_key = 'chave_secreta_biblioteca'  # Necessário para flash messages

# Configuração para integração com FastAPI (opcional)
FASTAPI_URL = "http://localhost:8000"

@app.route('/')
def index():
    """
    Rota principal que exibe a lista de livros cadastrados
    Método HTTP: GET
    """
    try:
        # Busca todos os livros no banco de dados
        livros = db.listar_livros()
        return render_template('index.html', livros=livros)
    except Exception as e:
        flash(f'Erro ao carregar livros: {str(e)}', 'error')
        return render_template('index.html', livros=[])

@app.route('/adicionar', methods=['GET', 'POST'])
def adicionar_livro():
    """
    Rota para adicionar novos livros
    Métodos HTTP: GET (exibe formulário) e POST (processa dados)
    """
    if request.method == 'POST':
        try:
            # Obtém dados do formulário
            titulo = request.form.get('titulo', '').strip()
            autor = request.form.get('autor', '').strip()
            ano_publicacao = request.form.get('ano_publicacao', '').strip()

            # Validação básica dos dados
            if not titulo or not autor or not ano_publicacao:
                flash('Todos os campos são obrigatórios!', 'error')
                return redirect(url_for('adicionar_livro'))
```

```
            # Converte ano para inteiro
            try:
                ano_publicacao = int(ano_publicacao)
            except ValueError:
                flash('Ano de publicação deve ser um número válido!', 'error')
                return redirect(url_for('adicionar_livro'))

            # Insere no banco de dados
            db.criar_livro(titulo, autor, ano_publicacao, True)

            flash('Livro cadastrado com sucesso!', 'success')
            return redirect(url_for('index'))

        except Exception as e:
            flash(f'Erro ao cadastrar livro: {str(e)}', 'error')
            return redirect(url_for('adicionar_livro'))

    # GET: Exibe o formulário
    return render_template('adicionar.html')

@app.route('/editar/<int:livro_id>', methods=['GET', 'POST'])
def editar_livro(livro_id):
    """
    Rota para editar livros existentes
    """
    livro = db.buscar_livro_por_id(livro_id)
    if not livro:
        flash('Livro não encontrado!', 'error')
        return redirect(url_for('index'))

    if request.method == 'POST':
        try:
            titulo = request.form.get('titulo', '').strip()
            autor = request.form.get('autor', '').strip()
            ano_publicacao = request.form.get('ano_publicacao', '').strip()
            disponivel = 'disponivel' in request.form

            if not titulo or not autor or not ano_publicacao:
                flash('Todos os campos são obrigatórios!', 'error')
                return redirect(url_for('editar_livro', livro_id=livro_id))

            try:
                ano_publicacao = int(ano_publicacao)
            except ValueError:
                flash('Ano de publicação deve ser um número válido!', 'error')
                return redirect(url_for('editar_livro', livro_id=livro_id))

            db.atualizar_livro(livro_id, titulo, autor, ano_publicacao, disponivel)
            flash('Livro atualizado com sucesso!', 'success')
            return redirect(url_for('index'))

        except Exception as e:
            flash(f'Erro ao atualizar livro: {str(e)}', 'error')
            return redirect(url_for('editar_livro', livro_id=livro_id))

    return render_template('editar.html', livro=livro)

@app.route('/excluir/<int:livro_id>')
def excluir_livro(livro_id):
    """
    Rota para excluir livros
    Método HTTP: GET
    """
    try:
        if db.excluir_livro(livro_id):
            flash('Livro excluído com sucesso!', 'success')
        else:
            flash('Livro não encontrado!', 'error')
    except Exception as e:
        flash(f'Erro ao excluir livro: {str(e)}', 'error')
```

```
    return redirect(url_for('index'))

# Integração opcional com FastAPI
@app.route('/api/livros')
def listar_livros_api():
    """
    Endpoint que consome a API FastAPI (opcional)
    Demonstra integração entre Flask e FastAPI
    """
    try:
        response = requests.get(f"{FASTAPI_URL}/livros")
        if response.status_code == 200:
            return jsonify(response.json())
        else:
            return jsonify({"error": "Erro ao buscar livros na API"}), 500
    except requests.exceptions.RequestException:
        return jsonify({"error": "FastAPI não está disponível"}), 503

if __name__ == '__main__':
    app.run(debug=True, port=5000)
```

Templates Flask

templates/base.html

```
html

<!DOCTYPE html>
<html lang="pt-BR">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>{% block title %}Biblioteca Universitária{% endblock %}</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.mi
n.css" rel="stylesheet">
    <style>
        .book-card {
            transition: transform 0.2s;
        }
        .book-card:hover {
            transform: translateY(-5px);
        }
        .available {
            color: green;
        }
        .unavailable {
            color: red;
        }
    </style>
</head>
<body>
    <nav class="navbar navbar-expand-lg navbar-dark bg-primary">
        <div class="container">
            <a class="navbar-brand" href="{{ url_for('index') }}">📚 Biblioteca Uni
versitária</a>
        </div>
    </nav>

    <div class="container mt-4">
        <!-- Flash Messages -->
        {% with messages = get_flashed_messages(with_categories=true) %}
            {% if messages %}
                {% for category, message in messages %}
                    <div class="alert alert-{{ 'danger' if category == 'error' else
'success' }} alert-dismissible fade show" role="alert">
                        {{ message }}
                        <button type="button" class="btn-close" data-bs-dismiss="al
ert"></button>
                    </div>
```

```
                {% endfor %}
            {% endif %}
        {% endwith %}

        {% block content %}{% endblock %}
    </div>

    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"></script>
</body>
</html>
```

templates/index.html

```
html

{% extends "base.html" %}

{% block title %}Lista de Livros - Biblioteca Universitária{% endblock %}

{% block content %}
<div class="d-flex justify-content-between align-items-center mb-4">
    <h1>Acervo de Livros</h1>
    <a href="{{ url_for('adicionar_livro') }}" class="btn btn-success">+ Adicionar Livro</a>
</div>

{% if livros %}
<div class="row">
    {% for livro in livros %}
    <div class="col-md-6 col-lg-4 mb-4">
        <div class="card book-card h-100">
            <div class="card-body">
                <h5 class="card-title">{{ livro.titulo }}</h5>
                <h6 class="card-subtitle mb-2 text-muted">{{ livro.autor }}</h6>
                <p class="card-text">
                    <strong>Ano:</strong> {{ livro.ano_publicacao }}<br>
                    <strong>Status:</strong>
                    <span class="{{ 'available' if livro.disponivel else 'unavailable' }}">
                        {{ 'Disponível' if livro.disponivel else 'Indisponível' }}
                    </span>
                </p>
            </div>
            <div class="card-footer">
                <div class="btn-group w-100">
                    <a href="{{ url_for('editar_livro', livro_id=livro.id) }}" class="btn btn-outline-primary btn-sm">Editar</a>
                    <a href="{{ url_for('excluir_livro', livro_id=livro.id) }}" class="btn btn-outline-danger btn-sm"
                       onclick="return confirm('Tem certeza que deseja excluir este livro?')">Excluir</a>
                </div>
            </div>
        </div>
    </div>
    {% endfor %}
</div>
{% else %}
<div class="alert alert-info text-center">
    <h4>Nenhum livro cadastrado</h4>
    <p>Comece adicionando o primeiro livro ao acervo!</p>
    <a href="{{ url_for('adicionar_livro') }}" class="btn btn-primary">Adicionar Primeiro Livro</a>
</div>
{% endif %}
{% endblock %}
```

templates/adicionar.html

```
html

{% extends "base.html" %}

{% block title %}Adicionar Livro - Biblioteca Universitária{% endblock %}

{% block content %}
<div class="row justify-content-center">
    <div class="col-md-8">
        <div class="card">
            <div class="card-header">
                <h2 class="card-title mb-0">Adicionar Novo Livro</h2>
            </div>
            <div class="card-body">
                <form method="POST">
                    <div class="mb-3">
                        <label for="titulo" class="form-label">Título do Livro *</l
abel>
                        <input type="text" class="form-control" id="titulo" name="t
itulo" required
                               placeholder="Digite o título do livro">
                    </div>

                    <div class="mb-3">
                        <label for="autor" class="form-label">Autor *</label>
                        <input type="text" class="form-control" id="autor" name="au
tor" required
                               placeholder="Digite o nome do autor">
                    </div>

                    <div class="mb-3">
                        <label for="ano_publicacao" class="form-label">Ano de Publi
cação *</label>
                        <input type="number" class="form-control" id="ano_publicaca
o" name="ano_publicacao" required
                               min="1000" max="2024" placeholder="Ex: 2020">
                    </div>

                    <div class="d-grid gap-2 d-md-flex justify-content-md-end">
                        <a href="{{ url_for('index') }}" class="btn btn-secondary m
e-md-2">Cancelar</a>
                        <button type="submit" class="btn btn-primary">Cadastrar Liv
ro</button>
                    </div>
                </form>
            </div>
        </div>
    </div>
</div>
{% endblock %}
```

templates/editar.html

```
html

{% extends "base.html" %}

{% block title %}Editar Livro - Biblioteca Universitária{% endblock %}

{% block content %}
<div class="row justify-content-center">
    <div class="col-md-8">
        <div class="card">
            <div class="card-header">
                <h2 class="card-title mb-0">Editar Livro</h2>
            </div>
            <div class="card-body">
                <form method="POST">
                    <div class="mb-3">
                        <label for="titulo" class="form-label">Título do Livro *</l
```

```
abel>
                        <input type="text" class="form-control" id="titulo" name="t
itulo"
                               value="{{ livro.titulo }}" required>
                    </div>

                    <div class="mb-3">
                        <label for="autor" class="form-label">Autor *</label>
                        <input type="text" class="form-control" id="autor" name="au
tor"
                               value="{{ livro.autor }}" required>
                    </div>

                    <div class="mb-3">
                        <label for="ano_publicacao" class="form-label">Ano de Publi
cação *</label>
                        <input type="number" class="form-control" id="ano_publicaca
o" name="ano_publicacao"
                               value="{{ livro.ano_publicacao }}" required min="100
0" max="2024">
                    </div>

                    <div class="mb-3 form-check">
                        <input type="checkbox" class="form-check-input" id="disponi
vel" name="disponivel"
                               {{ 'checked' if livro.disponivel }}>
                        <label class="form-check-label" for="disponivel">Disponível
para empréstimo</label>
                    </div>

                    <div class="d-grid gap-2 d-md-flex justify-content-md-end">
                        <a href="{{ url_for('index') }}" class="btn btn-secondary m
e-md-2">Cancelar</a>
                        <button type="submit" class="btn btn-primary">Atualizar Liv
ro</button>
                    </div>
                </form>
            </div>
        </div>
    </div>
</div>
{% endblock %}
```

Parte 3 - API FastAPI

api_fast.py

```python
from fastapi import FastAPI, HTTPException, status
from pydantic import BaseModel, Field, validator
from typing import List, Optional
from database import db
import uvicorn

# Inicializa a aplicação FastAPI
app = FastAPI(
    title="API Biblioteca Universitária",
    description="API REST para gerenciamento de livros da biblioteca universitári
a",
    version="1.0.0"
)

# Modelo Pydantic para validação de dados
class LivroBase(BaseModel):
    titulo: str = Field(..., min_length=1, max_length=200, description="Título do l
ivro")
    autor: str = Field(..., min_length=1, max_length=100, description="Nome do auto
r")
```

```
    ano_publicacao: int = Field(..., ge=1000, le=2024, description="Ano de publicaç
ão")
    disponivel: bool = Field(default=True, description="Disponibilidade do livro")

    @validator('titulo', 'autor')
    def campos_nao_vazios(cls, v):
        """Valida se os campos não estão vazios após strip"""
        if not v.strip():
            raise ValueError('Campo não pode ser vazio')
        return v.strip()

class LivroCreate(LivroBase):
    pass

class LivroUpdate(BaseModel):
    titulo: Optional[str] = Field(None, min_length=1, max_length=200)
    autor: Optional[str] = Field(None, min_length=1, max_length=100)
    ano_publicacao: Optional[int] = Field(None, ge=1000, le=2024)
    disponivel: Optional[bool] = None

    @validator('titulo', 'autor')
    def campos_nao_vazios(cls, v):
        if v is not None and not v.strip():
            raise ValueError('Campo não pode ser vazio')
        return v.strip() if v else v

class LivroResponse(LivroBase):
    id: int

    class Config:
        orm_mode = True

@app.get("/")
async def root():
    """
    Endpoint raiz da API
    """
    return {
        "message": "Bem-vindo à API da Biblioteca Universitária",
        "docs": "/docs",
        "redoc": "/redoc"
    }

@app.get("/livros", response_model=List[LivroResponse])
async def listar_livros():
    """
    Lista todos os livros cadastrados
    - **Método HTTP**: GET
    - **Código de Sucesso**: 200 OK
    """
    try:
        livros = db.listar_livros()
        return livros
    except Exception as e:
        raise HTTPException(
            status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
            detail=f"Erro interno do servidor: {str(e)}"
        )

@app.get("/livros/{livro_id}", response_model=LivroResponse)
async def buscar_livro(livro_id: int):
    """
    Busca um livro específico pelo ID
    - **Método HTTP**: GET
    - **Parâmetro Path**: livro_id (ID do livro)
    - **Código de Sucesso**: 200 OK
    - **Código de Erro**: 404 Not Found (livro não encontrado)
    """
    livro = db.buscar_livro_por_id(livro_id)
    if not livro:
        raise HTTPException(
```

```
            status_code=status.HTTP_404_NOT_FOUND,
            detail=f"Livro com ID {livro_id} não encontrado"
        )
    return livro

@app.post("/livros", response_model=LivroResponse, status_code=status.HTTP_201_CREA
TED)
async def criar_livro(livro: LivroCreate):
    """
    Cria um novo livro no sistema
    - **Método HTTP**: POST
    - **Body**: Dados do livro (JSON)
    - **Código de Sucesso**: 201 Created
    - **Código de Erro**: 400 Bad Request (dados inválidos)
    """
    try:
        # Validação adicional
        if not livro.titulo.strip():
            raise HTTPException(
                status_code=status.HTTP_400_BAD_REQUEST,
                detail="Título do livro é obrigatório"
            )

        # Insere no banco de dados
        livro_id = db.criar_livro(
            titulo=livro.titulo,
            autor=livro.autor,
            ano_publicacao=livro.ano_publicacao,
            disponivel=livro.disponivel
        )

        # Retorna o livro criado
        livro_criado = db.buscar_livro_por_id(livro_id)
        return livro_criado

    except HTTPException:
        raise
    except Exception as e:
        raise HTTPException(
            status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
            detail=f"Erro ao criar livro: {str(e)}"
        )

@app.put("/livros/{livro_id}", response_model=LivroResponse)
async def atualizar_livro(livro_id: int, livro_update: LivroUpdate):
    """
    Atualiza um livro existente
    - **Método HTTP**: PUT
    - **Parâmetro Path**: livro_id (ID do livro)
    - **Body**: Dados atualizados do livro (JSON)
    - **Código de Sucesso**: 200 OK
    - **Código de Erro**: 404 Not Found (livro não encontrado)
    """
    # Verifica se o livro existe
    livro_existente = db.buscar_livro_por_id(livro_id)
    if not livro_existente:
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND,
            detail=f"Livro com ID {livro_id} não encontrado"
        )

    try:
        # Prepara os dados para atualização
        dados_atualizados = {
            'titulo': livro_update.titulo or livro_existente['titulo'],
            'autor': livro_update.autor or livro_existente['autor'],
            'ano_publicacao': livro_update.ano_publicacao or livro_existente['ano_p
ublicacao'],
            'disponivel': livro_update.disponivel if livro_update.disponivel is not
None else livro_existente['disponivel']
        }
```

```
            # Atualiza no banco de dados
            sucesso = db.atualizar_livro(
                livro_id=livro_id,
                titulo=dados_atualizados['titulo'],
                autor=dados_atualizados['autor'],
                ano_publicacao=dados_atualizados['ano_publicacao'],
                disponivel=dados_atualizados['disponivel']
            )

            if sucesso:
                livro_atualizado = db.buscar_livro_por_id(livro_id)
                return livro_atualizado
            else:
                raise HTTPException(
                    status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
                    detail="Erro ao atualizar livro"
                )

        except Exception as e:
            raise HTTPException(
                status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
                detail=f"Erro ao atualizar livro: {str(e)}"
            )

    @app.delete("/livros/{livro_id}", status_code=status.HTTP_204_NO_CONTENT)
    async def excluir_livro(livro_id: int):
        """
        Exclui um livro do sistema
        - **Método HTTP**: DELETE
        - **Parâmetro Path**: livro_id (ID do livro)
        - **Código de Sucesso**: 204 No Content
        - **Código de Erro**: 404 Not Found (livro não encontrado)
        """
        # Verifica se o livro existe
        livro = db.buscar_livro_por_id(livro_id)
        if not livro:
            raise HTTPException(
                status_code=status.HTTP_404_NOT_FOUND,
                detail=f"Livro com ID {livro_id} não encontrado"
            )

        try:
            sucesso = db.excluir_livro(livro_id)
            if not sucesso:
                raise HTTPException(
                    status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
                    detail="Erro ao excluir livro"
                )
        except HTTPException:
            raise
        except Exception as e:
            raise HTTPException(
                status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
                detail=f"Erro ao excluir livro: {str(e)}"
            )

    if __name__ == "__main__":
        uvicorn.run(
            "api_fast:app",
            host="0.0.0.0",
            port=8000,
            reload=True  # Auto-reload para desenvolvimento
        )
```

Parte 4 - Instruções de Execução

README.md

````
markdown

# Sistema de Cadastro de Livros - Biblioteca Universitária

Sistema completo desenvolvido com Flask (interface web) e FastAPI (API REST) para g
erenciamento de livros de uma biblioteca universitária.

## 📋 Requisitos

- Python 3.8+
- Dependências listadas no `requirements.txt`

## 🚀 Instalação e Execução

### 1. Clone o repositório ou baixe os arquivos

### 2. Instale as dependências
```bash
pip install -r requirements.txt
````

3. Execute a aplicação Flask (Porta 5000)

```
bash

python app_flask.py
```

4. Execute a API FastAPI (Porta 8000)

Em outro terminal:

```
bash

python api_fast.py
```

5. Acesse as aplicações

- **Flask (Interface Web)**: http://localhost:5000
- **FastAPI (Documentação)**: http://localhost:8000/docs
- **FastAPI (Redoc)**: http://localhost:8000/redoc

🗄️ Estrutura do Banco de Dados

O sistema utiliza SQLite com a tabela `livros`:

- `id` (INTEGER, PRIMARY KEY, AUTOINCREMENT)
- `titulo` (TEXT, NOT NULL)
- `autor` (TEXT, NOT NULL)
- `ano_publicacao` (INTEGER, NOT NULL)
- `disponivel` (BOOLEAN, DEFAULT TRUE)

🔧 Funcionalidades

Flask (Interface Web)

- ✅ Listagem de livros
- ✅ Cadastro de novos livros
- ✅ Edição de livros existentes
- ✅ Exclusão de livros
```

- ✅ Interface responsiva com Bootstrap

FastAPI (API REST)

- ✅ `GET /livros` - Lista todos os livros
- ✅ `GET /livros/{id}` - Busca livro por ID
- ✅ `POST /livros` - Cria novo livro
- ✅ `PUT /livros/{id}` - Atualiza livro existente
- ✅ `DELETE /livros/{id}` - Exclui livro
- ✅ Validação de dados com Pydantic
- ✅ Documentação automática (Swagger/Redoc)

🧪 Testes

Teste a interface web:

1. Acesse http://localhost:5000
2. Adicione, edite e exclua livros

Teste a API:

1. Acesse http://localhost:8000/docs
2. Use a interface Swagger para testar os endpoints
3. Ou use ferramentas como curl, Postman, ou Insomnia

Exemplo de requisições curl:

Listar livros:

```bash
curl -X GET "http://localhost:8000/livros"
```

Criar livro:

```bash
curl -X POST "http://localhost:8000/livros" \
     -H "Content-Type: application/json" \
     -d '{"titulo":"Dom Casmurro","autor":"Machado de Assis","ano_publicacao":189
9,"disponivel":true}'
```

Atualizar livro:

```bash
curl -X PUT "http://localhost:8000/livros/1" \
     -H "Content-Type: application/json" \
     -d '{"titulo":"Dom Casmurro - Edição Especial","disponivel":false}'
```

Excluir livro:

```bash
curl -X DELETE "http://localhost:8000/livros/1"
```

📁 Estrutura de Arquivos

```
text

biblioteca/
├── app_flask.py          # Aplicação Flask
├── api_fast.py           # API FastAPI
├── database.py           # Gerenciamento do banco de dados
├── requirements.txt      # Dependências do projeto
├── biblioteca.db         # Banco de dados SQLite (criado automaticamente)
└── templates/            # Templates HTML do Flask
    ├── base.html
    ├── index.html
    ├── adicionar.html
    └── editar.html
```

🔄 Códigos de Status HTTP Utilizados

- `200 OK` - Requisição bem-sucedida
- `201 Created` - Recurso criado com sucesso
- `204 No Content` - Exclusão bem-sucedida
- `400 Bad Request` - Dados inválidos
- `404 Not Found` - Recurso não encontrado
- `500 Internal Server Error` - Erro interno do servidor

🛠️ Tecnologias Utilizadas

- **Backend**: Flask, FastAPI
- **Banco de Dados**: SQLite
- **Validação**: Pydantic
- **Frontend**: HTML, Bootstrap, Jinja2
- **Documentação**: Swagger UI, ReDoc

```
text


**requirements.txt**
```txt
flask==2.3.3
fastapi==0.104.1
uvicorn==0.24.0
pydantic==2.5.0
requests==2.31.0
```

Como executar:

1. **Salve todos os arquivos** em uma pasta
2. **Instale as dependências**: `pip install -r requirements.txt`
3. **Execute o Flask**: `python app_flask.py`
4. **Execute o FastAPI** (em outro terminal): `python api_fast.py`
5. **Acesse**:
```