**Documentação da API de Tarefas**

**Projeto:** API REST para gerenciamento de tarefas, desenvolvida com Flask e SQLite.  
**Objetivo:** Criar uma API funcional para criar, listar, buscar, atualizar e deletar tarefas, ideal para portfólio e projetos freelance (R$ 700–1.500 ou US$ 150–300 por API).  
**Autor:** Cristiano  
**Data:** 26/06/2025  
**Status:** Passo 1 concluído (ambiente configurado, API básica rodando em http://localhost:5001/).

**Visão Geral**

A API de Tarefas permite gerenciar tarefas com operações CRUD (Create, Read, Update, Delete). Desenvolvida com Python 3.10.4, Flask, e SQLite, ela é leve, fácil de configurar e ideal para deploy em plataformas como Vercel. O projeto é parte de um portfólio para atrair freelas (meta: R$ 2.000–4.000/mês) e será hospedado no GitHub e integrado ao LinkedIn.

**Passo 1: Configurar o ambiente (Concluído, 16:52–19:14, 26/06/2025)**

**Objetivo**

Configurar o ambiente de desenvolvimento com Python 3.10.4, Flask, SQLite, e rodar uma API básica que responde em http://localhost:5001/.

**Pré-requisitos**

* Python 3.10.4 (verificado com python --version).
* pip associado ao Python 3.10.4 (verificado com pip --version).
* Editor: VS Code.
* Postman (opcional, para testes no Passo 3).
* Conta no GitHub (para Passo 4).

**Ações realizadas**

1. **Criação da pasta do projeto:**
2. mkdir C:\Users\crisf\Desktop\Projetos\todo-api
3. cd C:\Users\crisf\Desktop\Projetos\todo-api
4. **Configuração do ambiente virtual:**
5. python -m venv venv
6. venv\Scripts\activate
   * Confirmado (venv) no PowerShell.
7. **Instalação de dependências:**
8. pip install flask flask-sqlalchemy
9. pip freeze > requirements.txt
   * Dependências: Flask, Flask-SQLAlchemy.
10. **Criação do app.py:**
    * Arquivo criado em C:\Users\crisf\Desktop\Projetos\todo-api\app.py.
    * Código inicial:
    * from flask import Flask, request, jsonify
    * from flask\_sqlalchemy import SQLAlchemy
    * app = Flask(\_\_name\_\_)
    * app.config['SQLALCHEMY\_DATABASE\_URI'] = 'sqlite:///tarefas.db'
    * app.config['SQLALCHEMY\_TRACK\_MODIFICATIONS'] = False
    * db = SQLAlchemy(app)
    * class Tarefa(db.Model):
    * id = db.Column(db.Integer, primary\_key=True)
    * titulo = db.Column(db.String(100), nullable=False)
    * descricao = db.Column(db.String(200))
    * concluida = db.Column(db.Boolean, default=False)
    * @app.route('/')
    * def index():
    * return jsonify({'message': 'API de Tarefas'})
    * if \_\_name\_\_ == '\_\_main\_\_':
    * with app.app\_context():
    * db.create\_all()
    * app.run(debug=True, port=5001)
11. **Resolução de erros:**
    * **Erro 1: Arquivo não encontrado ([Errno 2] No such file or directory):**
      + Problema: app.py foi criado em C:\Users\crisf\Desktop\Projetos\todo-api\venv.
      + Solução: Movido para C:\Users\crisf\Desktop\Projetos\todo-api com:
      + move C:\Users\crisf\Desktop\Projetos\todo-api\venv\app.py C:\Users\crisf\Desktop\Projetos\todo-api\app.py
    * **Erro 2: IndentationError:**
      + Problema: Indentação extra na linha from flask\_sqlalchemy import SQLAlchemy.
      + Solução: Removida indentação com VS Code (Ctrl+A, Shift+Tab) e recriado app.py.
    * **Erro 3: NameError (jsonity):**
      + Problema: Uso de jsonity em vez de jsonify.
      + Solução: Corrigido para jsonify no import e na função index.
12. **Teste da API:**
    * Executado:
    * python app.py
    * Acessado http://localhost:5001/ no navegador, retornou:
    * {"message": "API de Tarefas"}
    * Confirmado banco SQLite (tarefas.db) em C:\Users\crisf\Desktop\Projetos\todo-api.

**Dicas**

* Configure o VS Code: **Editor: Insert Spaces** = true, **Tab Size** = 4, **Render Whitespace** = all.
* Se a porta 5001 estiver ocupada, use port=5002 em app.py.
* Mantenha requirements.txt atualizado para deploy.

**Passo 2: Implementar endpoints CRUD (Planejado, 19:14–20:44, 26/06/2025)**

**Objetivo**

Adicionar endpoints para criar, listar, buscar, atualizar e deletar tarefas.

**Ações**

1. **Atualizar app.py:**
   * Substituir o código em C:\Users\crisf\Desktop\Projetos\todo-api\app.py por:
   * from flask import Flask, request, jsonify
   * from flask\_sqlalchemy import SQLAlchemy
   * app = Flask(\_\_name\_\_)
   * app.config['SQLALCHEMY\_DATABASE\_URI'] = 'sqlite:///tarefas.db'
   * app.config['SQLALCHEMY\_TRACK\_MODIFICATIONS'] = False
   * db = SQLAlchemy(app)
   * class Tarefa(db.Model):
   * id = db.Column(db.Integer, primary\_key=True)
   * titulo = db.Column(db.String(100), nullable=False)
   * descricao = db.Column(db.String(200))
   * concluida = db.Column(db.Boolean, default=False)
   * @app.route('/')
   * def index():
   * return jsonify({'message': 'API de Tarefas'})
   * @app.route('/tarefas', methods=['GET'])
   * def get\_tarefas():
   * tarefas = Tarefa.query.all()
   * return jsonify([{'id': t.id, 'titulo': t.titulo, 'descricao': t.descricao, 'concluida': t.concluida} for t in tarefas])
   * @app.route('/tarefas', methods=['POST'])
   * def create\_tarefa():
   * data = request.get\_json()
   * if not data or 'titulo' not in data:
   * return jsonify({'error': 'Título é obrigatório'}), 400
   * tarefa = Tarefa(titulo=data['titulo'], descricao=data.get('descricao'), concluida=data.get('concluida', False))
   * db.session.add(tarefa)
   * db.session.commit()
   * return jsonify({'id': tarefa.id, 'titulo': tarefa.titulo, 'descricao': tarefa.descricao, 'concluida': tarefa.concluida}), 201
   * @app.route('/tarefas/<int:id>', methods=['GET'])
   * def get\_tarefa(id):
   * tarefa = Tarefa.query.get\_or\_404(id)
   * return jsonify({'id': tarefa.id, 'titulo': tarefa.titulo, 'descricao': tarefa.descricao, 'concluida': tarefa.concluida})
   * @app.route('/tarefas/<int:id>', methods=['PUT'])
   * def update\_tarefa(id):
   * tarefa = Tarefa.query.get\_or\_404(id)
   * data = request.get\_json()
   * if not data:
   * return jsonify({'error': 'Dados não fornecidos'}), 400
   * tarefa.titulo = data.get('titulo', tarefa.titulo)
   * tarefa.descricao = data.get('descricao', tarefa.descricao)
   * tarefa.concluida = data.get('concluida', tarefa.concluida)
   * db.session.commit()
   * return jsonify({'id': tarefa.id, 'titulo': tarefa.titulo, 'descricao': tarefa.descricao, 'concluida': tarefa.concluida})
   * @app.route('/tarefas/<int:id>', methods=['DELETE'])
   * def delete\_tarefa(id):
   * tarefa = Tarefa.query.get\_or\_404(id)
   * db.session.delete(tarefa)
   * db.session.commit()
   * return jsonify({'message': 'Tarefa deletada'})
   * if \_\_name\_\_ == '\_\_main\_\_':
   * with app.app\_context():
   * db.create\_all()
   * app.run(debug=True, port=5001)
   * **Endpoints:**
     + GET /tarefas: Lista todas as tarefas.
     + POST /tarefas: Cria uma tarefa (JSON: {"titulo": "", "descricao": "", "concluida": false}).
     + GET /tarefas/<id>: Busca uma tarefa por ID.
     + PUT /tarefas/<id>: Atualiza uma tarefa.
     + DELETE /tarefas/<id>: Deleta uma tarefa.
2. **Teste básico:**
   * Executar:
   * python app.py
   * Acessar http://localhost:5001/tarefas com:
   * curl http://localhost:5001/tarefas
     + Deve retornar [] (lista vazia).
3. **Dicas:**
   * Verifique indentação com **Render Whitespace** = all no VS Code.
   * Se a porta 5001 estiver ocupada, use port=5002.

**Passo 3: Testar com Postman (Planejado, 20:44–21:14, 26/06/2025)**

**Objetivo**

Testar todos os endpoints CRUD com Postman ou cURL.

**Ações**

1. **Instalar Postman (se necessário):**
   * Baixar em [www.postman.com/downloads/](https://www.postman.com/downloads/).
2. **Testar endpoints:**
   * **GET /tarefas:**
     + Requisição: GET http://localhost:5001/tarefas.
     + Esperado: [] (lista vazia).
   * **POST /tarefas:**
     + Requisição: POST http://localhost:5001/tarefas.
     + Corpo (raw, JSON):
     + {
     + "titulo": "Estudar Flask",
     + "descricao": "Aprender a criar APIs REST",
     + "concluida": false
     + }
     + Esperado: Status 201, JSON com id (ex.: 1), titulo, descricao, concluida.
   * **GET /tarefas/1:**
     + Requisição: GET http://localhost:5001/tarefas/1.
     + Esperado: JSON da tarefa criada.
   * **PUT /tarefas/1:**
     + Requisição: PUT http://localhost:5001/tarefas/1.
     + Corpo (raw, JSON):
     + {
     + "titulo": "Estudar Flask e SQLAlchemy",
     + "concluida": true
     + }
     + Esperado: JSON da tarefa atualizada.
   * **DELETE /tarefas/1:**
     + Requisição: DELETE http://localhost:5001/tarefas/1.
     + Esperado: {"message": "Tarefa deletada"}.
3. **Com cURL (alternativa):**
4. curl http://localhost:5001/tarefas
5. curl -X POST http://localhost:5001/tarefas -H "Content-Type: application/json" -d "{\"titulo\":\"Estudar Flask\",\"descricao\":\"Aprender APIs\",\"concluida\":false}"
6. curl http://localhost:5001/tarefas/1
7. curl -X PUT http://localhost:5001/tarefas/1 -H "Content-Type: application/json" -d "{\"titulo\":\"Estudar Flask e SQLAlchemy\",\"concluida\":true}"
8. curl -X DELETE http://localhost:5001/tarefas/1
9. **Dicas:**
   * Salve requisições no Postman em uma coleção “API de Tarefas”.
   * Verifique erros no console do python app.py.

**Passo 4: Publicar no GitHub (Planejado, 21:14–21:44, 26/06/2025)**

**Objetivo**

Publicar o projeto no GitHub para portfólio.

**Ações**

1. **Inicializar repositório:**
2. cd C:\Users\crisf\Desktop\Projetos\todo-api
3. git init
4. git add .
5. git commit -m "API de Tarefas inicial com Flask e SQLite"
6. **Criar .gitignore:**
   * Criar C:\Users\crisf\Desktop\Projetos\todo-api\.gitignore:
   * venv/
   * \*.db
   * \_\_pycache\_\_/
   * \*.pyc
7. **Publicar no GitHub:**
   * Criar repositório no GitHub (ex.: todo-api).
   * Conectar e enviar:
   * git remote add origin https://github.com/seu-usuario/todo-api.git
   * git push -u origin main
8. **Adicionar README.md:**
   * Criar README.md com:
   * # API de Tarefas
   * API REST para gerenciamento de tarefas com Flask e SQLite.
   * ## Endpoints
   * - GET /tarefas: Lista todas as tarefas.
   * - POST /tarefas: Cria uma tarefa.
   * - GET /tarefas/<id>: Busca uma tarefa.
   * - PUT /tarefas/<id>: Atualiza uma tarefa.
   * - DELETE /tarefas/<id>: Deleta uma tarefa.
   * ## Como rodar
   * 1. Clone o repositório: `git clone https://github.com/seu-usuario/todo-api.git`
   * 2. Crie ambiente virtual: `python -m venv venv`
   * 3. Ative: `venv\Scripts\activate`
   * 4. Instale dependências: `pip install -r requirements.txt`
   * 5. Execute: `python app.py`
   * 6. Acesse: `http://localhost:5001/`

**Passo 5: Deploy no Vercel (Planejado, 21:44–22:44, 26/06/2025)**

**Objetivo**

Hospedar a API no Vercel para acesso online.

**Ações**

1. **Configurar para deploy:**
   * Criar vercel.json em C:\Users\crisf\Desktop\Projetos\todo-api:
   * {
   * "version": 2,
   * "builds": [
   * {
   * "src": "app.py",
   * "use": "@vercel/python"
   * }
   * ],
   * "routes": [
   * {
   * "src": "/(.\*)",
   * "dest": "app.py"
   * }
   * ]
   * }
   * Adicionar requirements.txt (já criado).
2. **Instalar Vercel CLI:**
3. npm install -g vercel
4. **Deploy:**
5. vercel
   * Seguir prompts (login, configuração).
   * Obter URL (ex.: https://todo-api.vercel.app).
6. **Testar:**
   * Acessar https://todo-api.vercel.app/ e https://todo-api.vercel.app/tarefas.

**Passo 6: Integrar ao portfólio (Planejado, 22:44–23:14, 26/06/2025)**

**Objetivo**

Adicionar a API ao portfólio e propostas de freelancing.

**Ações**

1. **Atualizar LinkedIn:**
   * Adicionar projeto: “API de Tarefas com Flask e SQLite, hospedada no Vercel: [link]. Gerencia tarefas com CRUD.”
   * Compartilhar post: “Concluí uma API REST com Flask! [link GitHub]. Busco projetos freelance (R$ 700–1.500 ou US$ 150–300).”
2. **Propostas de freelancing:**
   * **Workana/99Freelas:** “Desenvolvi uma API de tarefas com Flask: [link GitHub]. Posso criar APIs REST por R$ 800, entrega em 4 dias.”
   * **Upwork:** “Built a task management API with Flask and SQLite: [link GitHub]. Available for API projects at US$ 150, 4-day delivery.”
3. **Planilha de controle:**
   * Criar planilha (Excel/Google Sheets): Cliente, Valor, Data, Status.
   * Ex.: “API de Tarefas, R$ 800, 30/06/2025, Pago via Pix.”

**Precificação e Freelas sem MEI**

* **Valor:** R$ 700–1.500 (Brasil, R$ 70–100/hora) ou US$ 150–300 (Upwork, US$ 15–25/hora). Ex.: API de Tarefas (7 horas) = R$ 700–1.000 ou US$ 150–200.
* **Brasil (Workana, 99Freelas):**
  + Contratos simples: “Serviço: API REST, R$ 800, entrega: 30/06, Pix.”
  + Registrar tudo (planilha) para declarar no IR 2026. Ex.: R$ 2.000/mês = ~R$ 300 IRPF (15%, acima de R$ 2.112).
* **Upwork:**
  + Cadastrar-se (1–2 dias para aprovação). Proposta: “API de tarefas com Flask: [link GitHub]. US$ 150, entrega em 4 dias.”
  + Receber via Wise (CPF), declarar via Carnê-Leão (IRPF 15–27,5%). Ex.: US$ 200 (R$ 1.000) = ~R$ 150 IRPF + R$ 3,80 IOF.
* **Meta:** 1–2 projetos (ex.: R$ 1.000 + US$ 200 = R$ 2.000) para cobrir despesas, usando seguro-desemprego (R$ 1.518–2.424).
* **Contador:** Enviar WhatsApp: “Quero freelar sem MEI por 1–2 meses (R$ 2.000–4.000). Como declarar via Carnê-Leão? INSS? MEI quando? Call amanhã?”.

**Próximos Passos**

1. **Passo 2 (19:14–20:44):**
   * Atualizar app.py com endpoints CRUD.
   * Testar http://localhost:5001/tarefas (deve retornar []).
2. **Passo 3 (20:44–21:14):** Testar endpoints com Postman/cURL.
3. **Passo 4 (opcional hoje):** Publicar no GitHub.
4. **Passo 5–6 (sugerido para 27/06):** Deploy no Vercel e integrar ao portfólio.
5. **Próximo projeto:** Script de Automação de Relatórios (2–3h).