Gerenciador de Downloads do YouTube

Aplicação cliente-servidor para Download de Mídia

Guilherme S. Sampaio (Backend & DevOps)

Pedro Mota (Frontend & Testes)

Novembro/Dezembro 2025

O que é o Sistema?

- ✓ Aplicação web para download de vídeos e músicas do YouTube.
- ✓ Interface moderna e responsiva para fácil utilização.
- ✓ Arquivos nos formatos MP3 (áudio) e MP4 (vídeo).
- ✓ Download individual ou em lote (comprimido em arquivo ZIP).
- ✓ Feedback em tempo real do progresso via WebSocket, que também é usado para a limpeza de arquivos temporários no backend.

Principais Funcionalidades

Pesquisa Integrada

Busca de vídeos diretamente pela API do YouTube.

Download de Áudio

Baixe no formato MP3.

Download de Vídeo

Baixe em MP4

Download em Lote

Baixa múltiplos arquivos em um único ZIP.

Tempo Real

Comunicação via WebSocket para feedback de progresso e limpeza de arquivos temporários.

Responsivo

Interface adaptável (mobile-friendly).

Modelo Cliente-Servidor

FRONTEND (Cliente)

Tecnologia: React + Vite

Porta: 5173

Comunicação: HTTP / WebSocket

BACKEND (Servidor)

Tecnologia: FastAPI (Python)

Porta: 8000

Dependências: pytubefix, MoviePy, YouTube API

Socket x Websocket

Socket

- Ponto final de comunicação (IP + Porta) utilizado por diversos protocolos (incluindo HTTP).
- Comunicação tipicamente de "Requisição-Resposta".

Websocket

- Canal de comunicação **persistente** e **bidirecional (full-duplex)**.
- Permite que o servidor envie dados ao cliente a qualquer momento.
- Ideal para feedback e notificações em tempo real.

Backend – FastAPI e Tecnologias

- **FastAPI:** Framework web assíncrono moderno de alta performance.
- **pytubefix:** Biblioteca para extração de mídia do YouTube.
- **MoviePy:** Processamento e conversão de áudio/vídeo.
- **Pydantic:** Validação de dados robusta e intuitiva.
- **WebSockets:** Comunicação bidirecional para feedback em tempo real.
- **Uvicorn:** Servidor ASGI de alta performance para FastAPI.

Backend em Funcionamento

Pontos da Demonstração

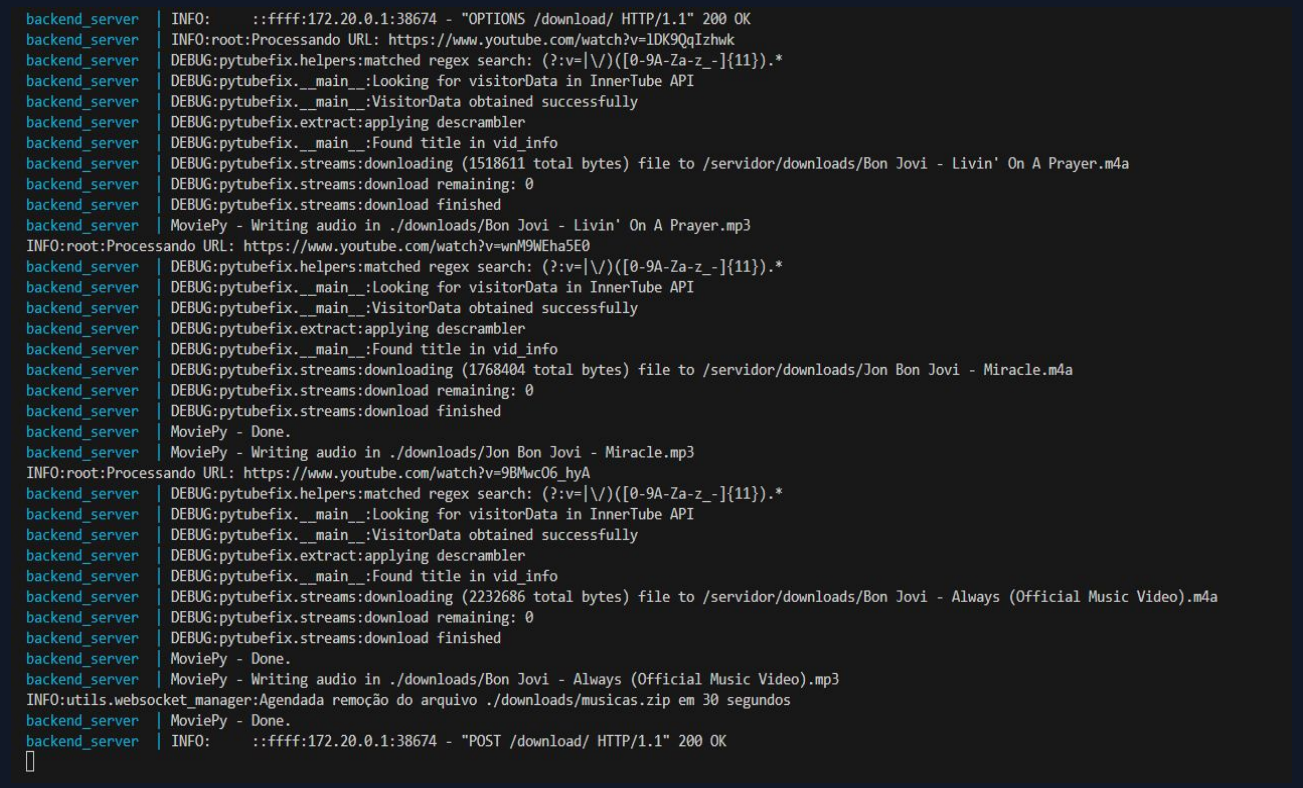
Containers rodando

Portas abertas (8000 para API, WebSocket)

Documentação automática em `/docs`

Exemplo de requisição POST para `/download/`

Logs de download em tempo real



Comunicação em Tempo Real (WebSocket)

Fluxo WebSocket

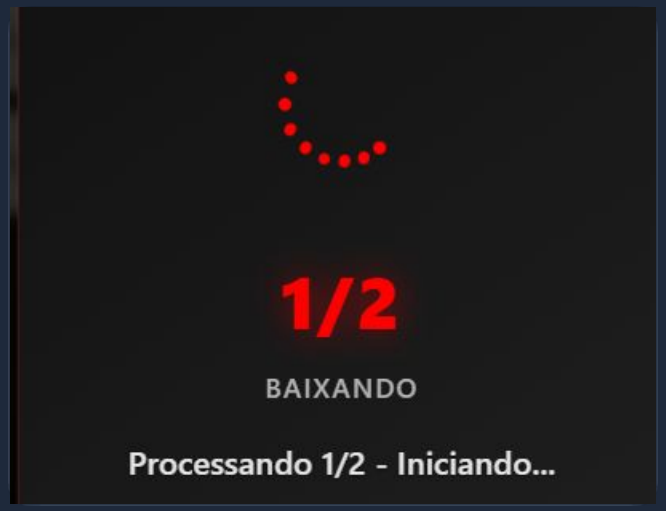
- **Conexão:** Cliente conecta em `ws://localhost:8000/ws`.
- **Progresso:** Backend notifica o progresso do download (ex: `download_progress`).
- **Interface:** Frontend atualiza a UI em tempo real com as mensagens.
- **Conclusão:** Notificação de `download_complete` e limpeza (`file_cleaned`).

Frontend – React e Interface Moderna

- **React 19:** Biblioteca moderna para construção de interfaces.
- **Vite:** Build tool de alta velocidade com hot reload instantâneo.
- **Bootstrap 5:** Framework CSS para design responsivo e componentes.
- **Custom Hooks:** Gerenciamento de estado e lógica (ex: `useDownloadManager.js`).
- **WebSocket Client:** Integração nativa para comunicação em tempo real.

Interface do Usuário

Tela Inicial e Pesquisa



Download em Progresso (Feedback)

Interface Mobile (Responsivo)

Frontend em Funcionamento

Fluxo de Demonstração

1. **Pesquisar:** Utilizar o campo de busca para encontrar um vídeo.
2. **Selecionar:** Adicionar múltiplos vídeos à lista de download.
3. **Escolher:** Definir o tipo de download (Áudio/MP3 ou Vídeo/MP4).
4. **Baixar:** Iniciar o processo de download (individual ou em lote).
5. **Acompanhar:** Visualizar o progresso em tempo real na interface.
6. **Receber:** Download automático do arquivo final (MP4 ou ZIP).

Comunicação Entre Componentes

Fluxo de Dados

- **1. Frontend → Backend:** Requisição `POST /download/` com as URLs.
- **2. Backend → YouTube:** `pytubefix` busca as mídias no YouTube.
- **3. Backend (Interno):** `MoviePy` realiza a conversão para MP3/MP4.
- **4. Backend → Frontend:** Mensagens WebSocket enviam o progresso (ex: "Convertendo 2/3").
- **5. Backend → Frontend:** Resposta HTTP finaliza com o arquivo/ZIP para download.

Diagrama de Fluxo

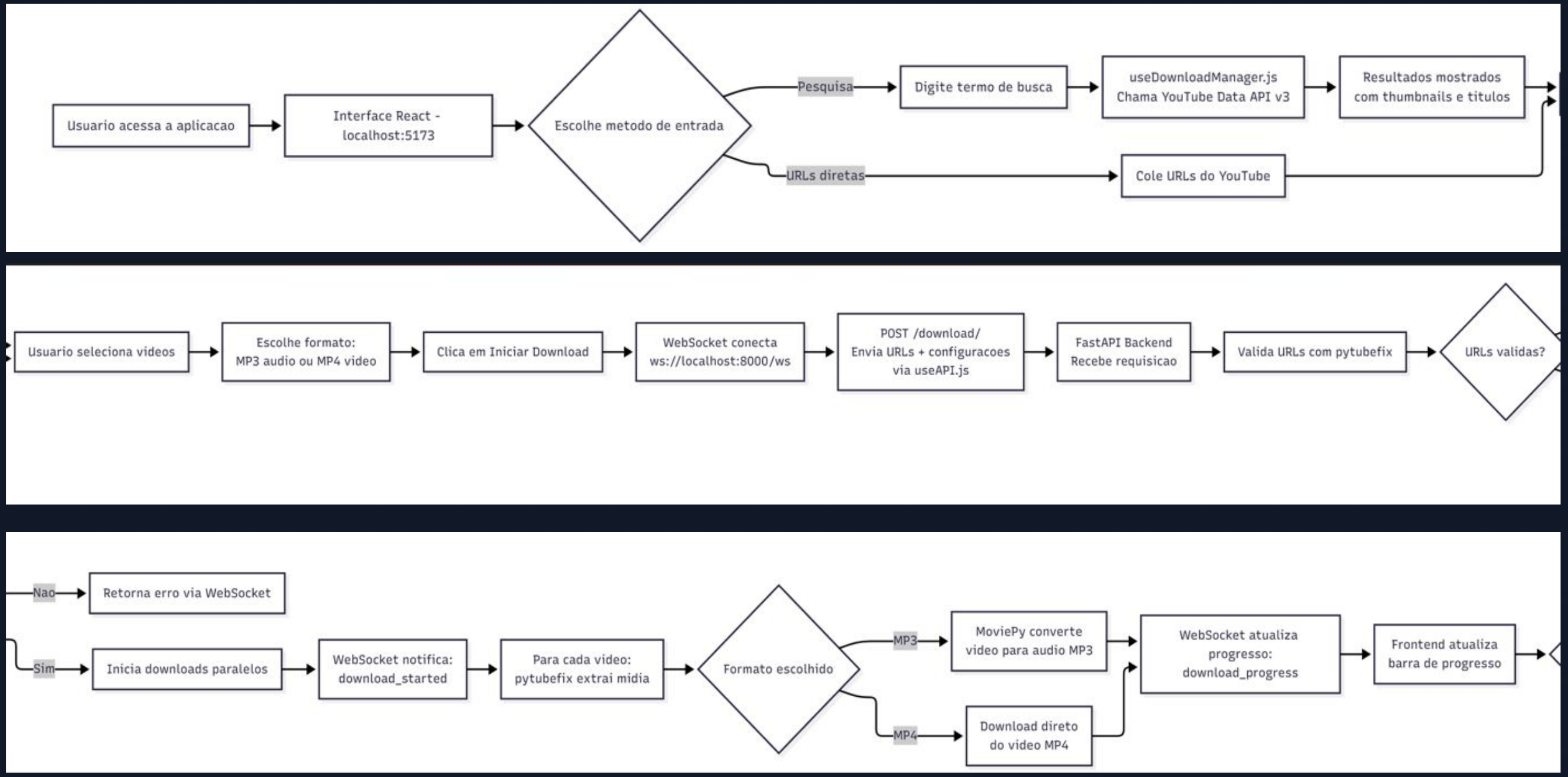
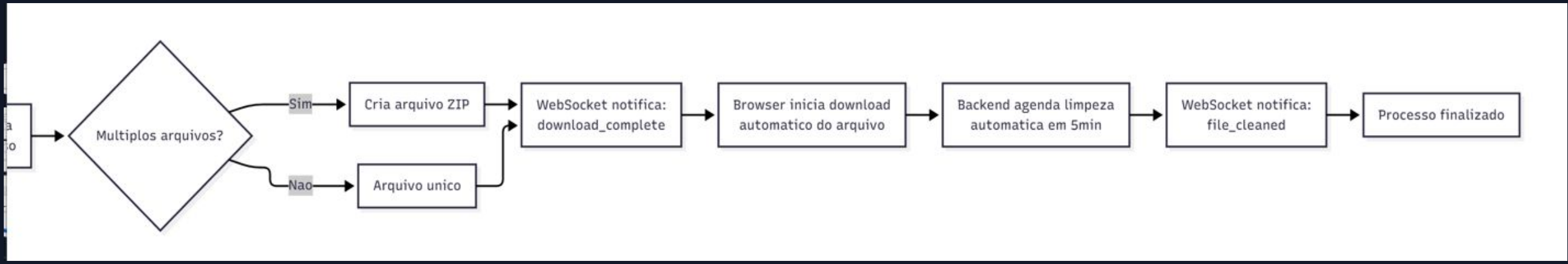


Diagrama de Fluxo



Containerização e IPv6

Containers

Backend: Imagem `python:3.12-slim` + FastAPI e dependências.

Frontend: Imagem `node:20-alpine` + Vite (servidor de desenvolvimento).

Hot Reload: Configurado em ambos os containers para desenvolvimento otimizado.

Suporte a IPv6

Configuração Dual-stack no Docker Compose.

Backend (Uvicorn) configurado para escutar em `::` (IPv6) e `0.0.0.0` (IPv4).

```
#ipv4
docker compose up --build
#ipv6
docker compose -f
./Docker-compose-ipv6.yaml up –build
```

Diagrama de Arquitetura Final

Frontend (React App)

Vite, Bootstrap, WebSocket Client.

(Container Docker: Node:20)

Backend (FastAPI)

pytubefix, MoviePy, WebSocket Server.

(Container Docker: Python:3.12)

Serviços Externos

YouTube API (para busca)

Servidores do YouTube (para mídia)

Downloads (MP3, MP4, ZIP)

Qualidade e Profissionalismo

Pontos de Destaque

✅ Código bem organizado e comentado

✅ Interface moderna e profissional

✅ README.md completo e detalhado

✅ Hot reload funcionando perfeitamente

✅ WebSocket implementado corretamente

✅ Suporte IPv6 configurado e funcional

✅ Containerização completa (Docker)

✅ Arquitetura escalável

Suporte ipv4/ipv6

O projeto oferece um **Deployment Flexível** com opções para:

- **IPv4 (padrão):**
  docker compose up --build
- **IPv6 (dual-stack):**
  docker compose -f
  ./Docker-compose-ipv6.yaml up

(A escolha é baseada no ambiente de deploy.)

Benefícios:

- Compatibilidade universal
- Performance otimizada em redes modernas
- Pronto para infraestruturas IPv6
- Zero impacto na funcionalidade

Conclusão e Próximos Passos

O que foi alcançado

Sistema fullstack completo e funcional.

Interface moderna, intuitiva e responsiva.

Comunicação em tempo real com WebSocket.

Suporte a múltiplos formatos (MP3, MP4, ZIP).

Containerização e suporte a IPv6.

Possíveis Melhorias Futuras

Autenticação de usuários.

Histórico de downloads por usuário.

Playlists personalizadas.

Download de playlists inteiras do YouTube.

Criação de uma API pública.

Obrigado!