**Guia Definitivo: Down-Dot-You-Be**

Este documento é o registro completo e o manual de operações para o projeto Down-Dot-You-Be, uma aplicação web desenvolvida em Python com o framework Flask. Ele detalha a arquitetura, funcionalidades e o processo de desenvolvimento, servindo como uma base de conhecimento para manutenção, futuras expansões e como um exemplo de projeto para portfólios de desenvolvimento.

**Sumário**

1. Visão Geral do Projeto
   * 1.1. Descrição do Projeto
   * 1.2. Funcionalidades Implementadas
   * 1.3. Prompt para Replicação por IA
2. Tecnologias e Ferramentas
   * 2.1. Glossário de Tecnologias
   * 2.2. Ícones e Recursos Visuais
3. Arquitetura e Fluxo de Trabalho
   * 3.1. Estrutura de Diretórios e Arquivos
   * 3.2. Fluxo de Operação da Aplicação
   * 3.3. Lógica da Barra de Progresso
4. Guia de Instalação e Execução
   * 4.1. Pré-requisitos
   * 4.2. Passos para Instalação
   * 4.3. Executando a Aplicação
5. Base de Conhecimento do Projeto
   * 5.1. Refinamento das Solicitações Chave
   * 5.2. Código Fonte Completo

**1. Visão Geral do Projeto**

**1.1. Descrição do Projeto**

Down-Dot-You-Be é uma aplicação web que permite aos usuários baixar vídeos, áudios e playlists completas do YouTube de forma organizada e eficiente. A principal característica do projeto é seu sistema de fila de downloads, que permite ao usuário adicionar múltiplos itens e monitorar o progresso de cada um individualmente através de uma interface limpa e reativa. A aplicação foi construída com Python e Flask no backend, e HTML com Tailwind CSS e JavaScript no frontend.

**1.2. Funcionalidades Implementadas**

* Fila de Downloads: Capacidade de adicionar até 10 itens (vídeos ou playlists) em uma fila.
* Detecção de Conteúdo: O sistema identifica automaticamente se uma URL é de um vídeo único ou de uma playlist.
* Downloads em Alta Qualidade: Utiliza yt-dlp para baixar vídeos na melhor qualidade disponível (unindo fluxos de vídeo e áudio) e áudios no formato MP3.
* Barra de Progresso por Item: Cada item na fila possui sua própria barra de progresso, que é atualizada em tempo real, mostrando o status (Pendente, Baixando, Processando, Concluído, Erro).
* Arquitetura Assíncrona: O processo de download é executado em uma *thread* separada no backend, permitindo que a interface do usuário permaneça responsiva.
* Logging de Operações: Todas as operações de download, bem-sucedidas ou com falha, são registradas em um arquivo master\_log.csv para auditoria.
* Organização de Arquivos: Os arquivos baixados são salvos automaticamente em pastas separadas (/downloads/videos e /downloads/audios).

**1.3. Prompt para Replicação por IA**

Crie uma aplicação web completa em Python usando o framework Flask, chamada "Down-Dot-You-Be". A aplicação deve permitir que os usuários baixem vídeos e áudios do YouTube através de uma interface com fila.

Requisitos Funcionais:

1. Interface Principal: Uma página web única (index.html) com:
   * Um formulário para inserir uma URL do YouTube e selecionar o tipo de download (vídeo ou áudio).
   * Um botão "Adicionar à Fila" que, sem recarregar a página, consulta as informações da URL e adiciona o item a uma tabela na interface.
   * Uma tabela que exibe a fila de downloads. As colunas devem ser: "Ordem", "Título", "Tamanho", "Tag" (ícone para vídeo, áudio, playlist de vídeo, playlist de áudio) e "Progresso".
   * A fila deve ter um limite de 10 itens.
2. Sistema de Fila e Progresso:
   * A fila de downloads deve ser gerenciada na sessão do usuário. Cada item adicionado deve receber um ID único (uuid).
   * Ao clicar em "Iniciar Downloads", o backend deve iniciar o processo em uma thread de trabalho separada para não bloquear a aplicação.
   * O frontend deve usar JavaScript para consultar uma rota /progress no backend a cada segundo.
   * A rota /progress deve retornar um JSON com o status de cada item da fila (ex: {"id": "...", "status": "downloading", "percentage": 45}).
   * O JavaScript deve atualizar a barra de progresso de cada item na tabela em tempo real, mudando a cor e o texto de acordo com o status (pending, downloading, merging, completed, error).
3. Lógica de Backend (Flask):
   * Utilize a biblioteca yt-dlp para toda a interação com o YouTube.
   * Crie uma rota /add que usa yt-dlp para extrair as informações do vídeo/playlist (título, tamanho, etc.) sem fazer o download, e adiciona esses dados à fila na sessão.
   * Crie uma rota /download\_queue que inicia a thread de download.
   * A thread de download deve iterar sobre os itens da fila e usar o progress\_hooks do yt-dlp para atualizar uma estrutura de dados global que armazena o progresso.
   * O download de vídeo deve buscar o melhor formato de vídeo e áudio e uni-los ('bestvideo+bestaudio/best'). O download de áudio deve extrair o melhor áudio e convertê-lo para MP3.
4. Estrutura de Arquivos: O projeto deve seguir uma estrutura organizada com pastas templates, static/images, downloads e logs. O arquivo .gitignore deve ser configurado para ignorar o ambiente virtual, a pasta downloads e os arquivos de log.

**2. Tecnologias e Ferramentas**

**2.1. Glossário de Tecnologias**

* Python (3.11+): Linguagem principal do backend.
* Flask: Microframework web utilizado para construir o servidor, gerenciar rotas e renderizar a interface.
* yt-dlp: Biblioteca de linha de comando e Python, fork do youtube-dl, usada para baixar vídeos e extrair metadados do YouTube e centenas de outros sites.
* HTML5: Linguagem de marcação para estruturar a página web.
* Tailwind CSS: Framework de CSS "utility-first" para estilização rápida e moderna da interface.
* JavaScript (Vanilla): Utilizado no frontend para interatividade, como fazer requisições AJAX para a rota de progresso e manipular o DOM para atualizar as barras de progresso.
* FFmpeg: Software de processamento de mídia essencial, usado pelo yt-dlp para unir os fluxos de áudio e vídeo e para converter áudios para o formato MP3.
* Git & GitHub: Sistema de controle de versão para gerenciar o código-fonte e plataforma de hospedagem para o repositório.

**2.2. Ícones e Recursos Visuais**

* Os ícones que representam o tipo de download e a ordem na fila são armazenados em static/images/ e referenciados no template HTML.
* A aplicação usa a fonte "Inter" do Google Fonts para uma tipografia limpa e moderna.

**3. Arquitetura e Fluxo de Trabalho**

**3.1. Estrutura de Diretórios e Arquivos**

down-dot-you-be/

│

├── downloads/ # Pasta onde os arquivos baixados são salvos

│ ├── videos/

│ └── audios/

│

├── logs/ # Registros de todas as operações

│ └── master\_log.csv

│

├── static/ # Arquivos estáticos acessíveis pelo navegador

│ └── images/

│ ├── 1\_icon.png # Ícones de ordem

│ ├── video\_icon.png # Ícones de tag

│ └── ...

│

├── templates/ # Templates HTML renderizados pelo Flask

│ └── index.html

│

├── .gitignore # Arquivos e pastas ignorados pelo Git

├── app.py # Lógica principal da aplicação Flask (rotas, downloads, etc.)

└── requirements.txt # Lista de dependências Python para o projeto

**3.2. Fluxo de Operação da Aplicação**

1. Acesso Inicial: O usuário acessa a rota / e o Flask renderiza o index.html.
2. Adicionar à Fila: O usuário insere uma URL, seleciona o tipo (vídeo/áudio) e clica em "Adicionar à Fila".
3. Extração de Metadados: O formulário envia uma requisição POST para a rota /add. O backend usa yt-dlp para extrair os metadados do vídeo/playlist (sem baixar) e os armazena na session do usuário, junto com um ID único. A página é recarregada, mostrando o item na tabela.
4. Iniciar Download: O usuário clica em "Iniciar Downloads".
5. Disparo da Thread: O JavaScript envia uma requisição POST para a rota /download\_queue. O backend:
   * Copia a fila da sessão para uma estrutura de dados global de progresso.
   * Limpa a fila da sessão.
   * Inicia uma thread de trabalho (download\_worker) para processar os downloads, retornando imediatamente uma resposta de sucesso para o JavaScript.
6. Monitoramento: O JavaScript começa a enviar requisições GET para a rota /progress a cada segundo.
7. Atualização da UI: A rota /progress retorna o status atual de cada item. O JavaScript usa esses dados para atualizar a largura, cor e texto das barras de progresso na tabela.
8. Conclusão: Quando a thread de trabalho termina, ela atualiza o status final. O JavaScript detecta essa mudança, aguarda alguns segundos e recarrega a página.

**3.3. Lógica da Barra de Progresso**

A atualização em tempo real é o recurso mais complexo da aplicação e funciona através de uma dissociação entre a tarefa de download e a interface do usuário:

* progress\_data (Dicionário Global): Funciona como um "quadro de avisos" compartilhado. A thread de download escreve o progresso nele.
* progress\_hooks (yt-dlp): É o mecanismo que permite que a nossa função my\_progress\_hook seja executada pelo yt-dlp sempre que um bloco de dados é baixado, permitindo-nos obter a porcentagem de progresso.
* Polling (JavaScript): O setInterval no frontend é a técnica de "polling" (sondagem), que pergunta repetidamente ao servidor: "Qual é o status agora?". Essa é uma abordagem eficaz para obter atualizações sem a complexidade de WebSockets.

**4. Guia de Instalação e Execução**

**4.1. Pré-requisitos**

* Python 3.10+
* Git
* FFmpeg: Essencial para o funcionamento da aplicação.
  + Windows: Instale via [Chocolatey](https://chocolatey.org/): choco install ffmpeg
  + macOS: Instale via [Homebrew](https://brew.sh/): brew install ffmpeg
  + Linux (Debian/Ubuntu): sudo apt update && sudo apt install ffmpeg

**4.2. Passos para Instalação**

1. Clone o repositório do GitHub:
2. git clone https://github.com/seu-usuario/down-dot-you-be.git
3. cd down-dot-you-be
4. Crie e ative um ambiente virtual:
5. # Windows
6. python -m venv venv
7. .\venv\Scripts\activate
8. # macOS / Linux
9. python3 -m venv venv
10. source venv/bin/activate
11. Instale as dependências do projeto:
12. pip install -r requirements.txt

**4.3. Executando a Aplicação**

1. Com o ambiente virtual ativo, inicie o servidor Flask:
2. flask run
3. Abra seu navegador e acesse a URL: http://127.0.0.1:5000

**5. Base de Conhecimento do Projeto**

**5.1. Refinamento das Solicitações Chave**

Durante o desenvolvimento, as seguintes solicitações foram cruciais para definir a arquitetura final do projeto:

* "Quero criar uma fila de download": Essa foi a diretriz inicial que moveu o projeto de um downloader simples para uma aplicação mais complexa, exigindo o uso de sessões para gerenciar o estado da fila.
* "Para cada futura funcionalidade (tag) anexar um ícone": Levou à criação da pasta static/images e à lógica no template para exibir ícones dinamicamente com base no tipo de conteúdo.
* "Agora preciso criar uma barra de progresso": Essa solicitação foi um ponto de virada, introduzindo a necessidade de tarefas em segundo plano (threading), uma rota de API para progresso (/progress) e JavaScript para polling (AJAX).
* "Insira a barra dentro da nossa tabela de filas": Refinou a experiência do usuário, exigindo uma refatoração para rastrear o progresso de cada item individualmente através de IDs únicos, resultando em uma UI mais clara e profissional.

**5.2. Código Fonte Completo**

O código-fonte final e atualizado da aplicação (app.py) e da interface (templates/index.html) pode ser encontrado diretamente nos arquivos do projeto neste repositório. Eles refletem a implementação de todas as funcionalidades descritas neste guia.