

**1.** [**yt-indir.com**](http://yt-indir.com)

1. it lets users download yt videos
2. everyone
3. the users opens the website, inputs a youtube url, gets presented with video options, and downloads one of them.
4. website/webapp
5. erro handling, download button, different quality and format options
6. batch, video editing, accounts
7. more than 50 users a month
8. Single input field centered on page, format dropdown, progress bar during processing
9. no, but react sounds fine, check the report before this
10. this youtube url is invalid
11. No persistent storage - URLs processed transiently.
12. a couple of weeks

**Product Requirements Document (PRD) - MVP**

**1. Introduction & Goals**

**Product Name:** [yt-indir.com](http://yt-indir.com)  
**Purpose:** Enables users to download YouTube videos in various formats/qualities without complex interfaces.  
**MVP Goals:**

* Achieve 50+ monthly active users
* Maintain <5% error rate during video processing
* Deliver 90% successful download completion rate

**2. Target Audience**

General consumers needing offline video access, including:

* Students archiving educational content
* Travelers preparing entertainment for limited connectivity
* Content creators backing up their uploads

**3. User Stories**

**Primary Journey:**  
"As a mobile user, I want to paste YouTube URLs and download MP4s for offline viewing during my commute."

**4. Features & Requirements**

**Core Features:**

1. URL validation with regex pattern matching (YouTube URL formats) [[1]](#fn1)[[2]](#fn2)
2. Dynamic format/quality selector using YouTube's adaptive streams [[1]](#fn1)[[3]](#fn3)
3. Progress visualization during video processing [[4]](#fn4)[[5]](#fn5)
4. Error handling for invalid URLs/age-restricted content [[6]](#fn6)[[7]](#fn7)

**UI/UX Concept:**

* Centered input field with paste detection
* Dropdown showing available resolutions (360p/720p/1080p)
* Animated progress bar during file processing
* One-click download button upon readiness

**5. Out of Scope**

* User accounts/profiles
* Video editing tools
* Browser extensions
* Batch processing

**6. Success Metrics**

1. 50+ monthly active users
2. 85% first-attempt success rate
3. <2.5s average processing time

**Technical Design Document (Tech Design Doc) - MVP**

**1. System Overview**

**Architecture:**

* Static frontend + Serverless backend
* Decoupled processing pipeline

**Platform:** Web app (PWA-capable)

**2. Tech Stack**

**Frontend:**

* React + Vite (lightweight build)
* HeadlessUI for accessible components

**Backend:**

* Cloudflare Workers (Node.js runtime)
* yt-dlp WASM build [[8]](#fn8)[[7]](#fn7)

**Processing:**

* Stream-based video handling [[9]](#fn9)[[7]](#fn7)
* FFmpeg.wasm for format conversion

**3. Architecture & Data Flow**

graph LR  
 A[User] --> B(React Frontend)  
 B --> C{Cloudflare Worker}  
 C --> D[YouTube API]  
 C --> E[yt-dlp Processor]  
 E --> F[Streaming Response]  
 F --> A

**4. Feature Implementation**

**URL Validation:**

// Worker logic  
const YT\_REGEX = /(youtu.be\/|v\/|u\/\w\/|embed\/|watch\?v=|\&v=)([^#\&\?]\*).\*/;  
if (!url.match(YT\_REGEX)) return new Response('Invalid URL', {status: 400});

**Stream Handling:**

// Cloudflare Worker streaming  
async function handleRequest(url) {  
 const videoStream = await fetchYouTubeVideo(url);  
 return new Response(videoStream.body, {  
 headers: {  
 'Content-Disposition': `attachment; filename="video.mp4"`,  
 'Content-Type': 'video/mp4'  
 }  
 });  
}

**5. Error Handling**

**Common Scenarios:**

* Invalid URL: Toast notification with example format [[6]](#fn6)
* Age-restricted content: Modal explaining limitations [[1]](#fn1)
* Processing timeout: Retry button with simplified quality option

**6. Data Handling**

* Zero persistence architecture
* URL parameters transiently stored in memory <60s [[10]](#fn10)[[7]](#fn7)
* Cloudflare DLP for accidental PII detection [[10]](#fn10)[[11]](#fn11)

**7. Development Timeline**

**Phase 1 (Week 1):**

* Frontend scaffolding with Vite+React
* Basic Worker URL validation

**Phase 2 (Week 2):**

* yt-dlp integration testing
* Stream handling implementation

**Phase 3 (Week 3):**

* UI polish with loading states
* Error handling implementation

**Buffer Week:**

* Load testing & optimization

**Implementation Notes**

1. Use Cloudflare's Streams API for chunked processing [[9]](#fn9)[[7]](#fn7)
2. Implement Wrangler for local dev/prod parity [[4]](#fn4)[[5]](#fn5)
3. Set compatibility date to 2024-06-01 for Node.js 20 features
4. Use Durable Objects if state management becomes necessary

**Legal Safeguards**

* Age gate for videos marked 18+
* Robots.txt disallow for search engines
* Clear disclaimer about YouTube ToS compliance

This architecture keeps costs at $0/month (Cloudflare free tier) while handling ~10k requests/month. For scale beyond MVP, consider:

* Adding Queue for request batching
* Implementing per-IP rate limiting
* Exploring Cloudflare R2 for temporary storage

⁂

1. <https://gist.github.com/hizkifw/ae229eb0c5ff809fc2a4a88735bfd604>

1. <https://stackoverflow.com/questions/74633545/cloudflare-worker-download-a-file-using-fetch>

1. <https://developers.cloudflare.com/stream/viewing-videos/download-videos/>

1. <https://www.youtube.com/watch?v=snMPPXyf73M>

1. <https://www.youtube.com/watch?v=b63kbdGWFmg>

1. <https://community.cloudflare.com/t/how-to-get-a-binary-file-and-post-it-somewhere-else/313115>

1. <https://community.cloudflare.com/t/streaming-large-remote-files/14501>

1. <https://forum.cloudron.io/topic/6415/switching-to-yt-dlp>

1. <https://developers.cloudflare.com/workers/runtime-apis/streams/>

1. <https://blog.cloudflare.com/improving-data-loss-prevention-accuracy-with-ai-context-analysis/>

1. <https://dev.to/tylerlwsmith/12-things-i-learned-about-hosting-serverless-sites-on-cloudflare-2dml>