

**# Part III: Generate** [**NOTES.md**](http://NOTES.md) **for AI IDE Agent**

**Goal:** To generate a structured NOTES.md file that summarizes key information and provides a high-level implementation plan based on the provided Product Requirements Document (PRD) and Technical Design Document (Tech Design Doc). This NOTES.md file will serve as the primary guide for an AI IDE agent (like GitHub Copilot Chat or Cursor) during the initial code generation phase.

**AI Role:** Act as a meticulous Technical Project Manager. Your task is to thoroughly analyze the referenced PRD and Tech Design Doc files and synthesize them into a clear, actionable NOTES.md file formatted for an AI agent.

**Reference Documents:**

* **Product Requirements Document (PRD):** Please read the attached/provided PRD file ([Specify PRD Filename Here, e.g., PRD-MVP.md]). This contains the *what* – requirements, user stories, features, scope.
* **Technical Design Document (Tech Design Doc):** Please read the attached/provided Tech Design Doc file ([Specify Tech Design Doc Filename Here, e.g., Tech-Design-MVP.md]). This contains the *how* – tech stack, architecture, data flow, implementation approach.

**Instructions for AI:**

1. **Analyze Thoroughly:** Carefully read and understand *both* the PRD and Tech Design Doc in their entirety.
2. **Synthesize Information:** Extract and combine relevant information from both documents.
3. **Structure Output:** Generate a markdown file named NOTES.md containing the following sections:
   * **## Project Overview**:
     + Product Name: (From PRD)
     + Core Purpose: (Concise summary based on PRD)
     + MVP Goal: (Primary goal for the MVP, from PRD)
     + Target Audience: (From PRD)
   * **## Technical Specifications (from Tech Design Doc)**:
     + Platform:
     + Tech Stack (Frontend):
     + Tech Stack (Backend/Core):
     + Key Libraries/APIs:
     + Architecture Overview: (Brief description or key components)
     + Data Handling Notes: (Key points on privacy/storage)
     + Error Handling Approach: (Brief summary)
   * **## Core MVP Features & Implementation Plan (from PRD & Tech Design Doc)**:
     + For *each* core MVP feature listed in the PRD:
       - Create a sub-section: ### Feature: [Feature Name]
       - Description: (From PRD)
       - Key Acceptance Criteria/User Story: (Link to main user story if applicable, or list key criteria)
       - Technical Implementation Notes: (Summarize approach from Tech Design Doc, mention key components/files involved if specified)
       - Agent Implementation Steps (Suggested): (Provide a *high-level*, logical checklist for the agent, e.g., "1. Create function X in file [Y.py](http://Y.py)", "2. Add route Z in main app file", "3. Implement UI element based on UI concept")
   * **## UI/UX Concept (from PRD)**:
     + Brief description of the look, feel, or key elements.
   * **## Out of Scope for MVP (from PRD)**:
     + List features explicitly excluded.
   * **## Key Agent Instructions**:
     + "Agent: Please generate the MVP codebase based on the details above."
     + "Prioritize implementing the features exactly as specified in the 'Core MVP Features' section."
     + "Strictly adhere to the 'Technical Specifications' regarding platform, stack, and architecture."
     + "Refer to the full PRD ([PRD Filename]) and Tech Design Doc ([Tech Design Doc Filename]) files in the project root for complete details if needed."
     + "Create files and directory structures as logically required by the Tech Design Doc and implementation plan."
     + "Add comments to explain complex logic."
4. **Clarity and Conciseness:** Ensure the generated NOTES.md is clear, well-organized, and provides actionable guidance for the AI agent. Use markdown formatting effectively.
5. **Filename Placeholders:** Remember to include the actual filenames of the PRD and Tech Design Doc in the "Key Agent Instructions" section where indicated.

**Output:**

* The complete content for the NOTES.md file, formatted in markdown.

**Execution Request:**

Please generate the NOTES.md content by analyzing the referenced PRD ([Specify PRD Filename Here]) and Tech Design Doc ([Specify Tech Design Doc Filename Here]) files according to the instructions above.

# NOTES.md - yt-indir.com MVP Implementation Guide  
  
## Project Overview  
\*\*Product Name\*\*: yt-indir.com   
\*\*Core Purpose\*\*: Enable frictionless YouTube video downloads through minimalist web interface   
\*\*MVP Goal\*\*: Serve 50+ MAUs with <5% error rate   
\*\*Target Audience\*\*: General consumers needing offline video access   
  
## Technical Specifications  
\*\*Platform\*\*: Web app (PWA-ready)   
\*\*Tech Stack (Frontend)\*\*:   
- React 18 + Vite   
- HeadlessUI components   
- Axios for API calls   
  
\*\*Tech Stack (Backend)\*\*:   
- Cloudflare Workers (Node.js 20)   
- yt-dlp WASM build   
- FFmpeg.wasm   
  
\*\*Key Libraries/APIs\*\*:   
- YouTube player\_iframe API   
- Cloudflare Streams   
- WebAssembly toolchain   
  
\*\*Architecture Overview\*\*:   
1. Static frontend hosted on Cloudflare Pages   
2. Serverless Workers handle video processing   
3. Stream-based response pipeline   
  
\*\*Data Handling Notes\*\*:   
- Zero persistent storage   
- URL parameters held in memory <60s   
- DLP scanning for accidental PII   
  
\*\*Error Handling Approach\*\*:   
- Contextual toast notifications   
- Fallback to lower-resolution streams   
- Automated retries for timeout errors   
  
## Core MVP Features & Implementation Plan  
  
### Feature: URL Validation  
\*\*Description\*\*: Regex-based YouTube URL verification   
\*\*Acceptance Criteria\*\*:   
- Rejects non-YouTube URLs with clear error   
- Supports all valid YouTube URL formats   
  
\*\*Technical Implementation\*\*:   
- Worker-side regex validation   
- Client-side paste detection   
  
\*\*Agent Steps\*\*:   
1. Create `src/utils/validation.js` with YT\_REGEX pattern   
2. Implement URL check in `worker.js` route handler   
3. Add client-side input sanitization in `InputField.jsx`   
  
### Feature: Dynamic Format Selector   
\*\*Description\*\*: Resolution/format dropdown based on available streams   
\*\*Acceptance Criteria\*\*:   
- Displays 360p/720p/1080p options   
- Auto-detects available formats   
  
\*\*Technical Implementation\*\*:   
- yt-dlp --list-formats equivalent   
- Adaptive stream selection   
  
\*\*Agent Steps\*\*:   
1. Create `streamParser.js` worker module   
2. Build `FormatDropdown.jsx` component   
3. Implement format mapping in `worker.js`   
  
### Feature: Progress Visualization   
\*\*Description\*\*: Animated progress bar during processing   
\*\*Acceptance Criteria\*\*:   
- Real-time download % estimation   
- Smooth animation transitions   
  
\*\*Technical Implementation\*\*:   
- ReadableStream with progress events   
- React state management   
  
\*\*Agent Steps\*\*:   
1. Create `ProgressBar.jsx` with animation logic   
2. Implement stream progress in `useDownloader` hook   
3. Add event listeners in `worker.js`   
  
### Feature: Error Handling System   
\*\*Description\*\*: User-friendly error messaging   
\*\*Acceptance Criteria\*\*:   
- Covers invalid URLs/age restrictions   
- Provides recovery options   
  
\*\*Technical Implementation\*\*:   
- Custom error classes in Workers   
- Error boundary components   
  
\*\*Agent Steps\*\*:   
1. Create `errors.js` with YTError classes   
2. Build `ErrorToast.jsx` component   
3. Implement error boundaries in `App.jsx`   
  
## UI/UX Concept   
- Centered card layout with glassmorphism effect   
- Animated gradient background   
- Single-step workflow: Input → Select → Download   
  
## Out of Scope for MVP   
- User authentication   
- Download history   
- Video editing tools   
- Browser extensions   
  
## Key Agent Instructions   
\*\*Agent\*\*: Generate MVP codebase per these specifications   
1. Prioritize core features in implementation order listed   
2. Strictly use React + Cloudflare Workers stack   
3. Reference full documents:   
 - [PRD-MVP.md](PRD-MVP.md)   
 - [TECH-DESIGN-MVP.md](TECH-DESIGN-MVP.md)   
4. Create following directory structure:

/src  
/components  
/hooks  
/workers  
/utils

5. Add JSDoc comments for complex logic   
6. Prefer functional components with hooks   
7. Implement Cloudflare Worker middleware chain   
  
\*\*Initial File Creation Priority\*\*:   
1. `worker.js` - Core processing logic   
2. `App.jsx` - Main UI component   
3. `useDownloader.js` - Download state management   
4. `ErrorBoundary.jsx` - Global error handling   
  
\*\*Testing Requirements\*\*:   
- Vitest unit tests for validation logic   
- Playwright e2e test for core user journey