📚 AI Notes Chrome Extension - Comprehensive Documentation

🔍 Project Overview

This Chrome extension generates concise, AI-powered notes from YouTube videos by extracting and summarizing their transcripts. It provides a seamless experience for users to save and access these summaries for later reference or download.

✨ Key Features

- Extracts video transcripts directly from YouTube
- Summarizes content using Google's Gemini AI
- Saves notes per video for easy reference
- Provides downloadable summaries in markdown (.md) or text format
- Clean, responsive UI built with TailwindCSS

🏗️ Architecture Overview

The extension follows a client-server architecture:

Frontend (Chrome Extension)

- **Popup Interface**: User-facing component with controls and note display
- **Content Script**: Extracts information from YouTube pages
- **Local Storage**: Persists notes per video ID

Backend (Node.js + Python)

- **Express Server**: Handles API requests and communicates with AI
- **Python Script**: Extracts video transcripts
- **Gemini API**: Processes and summarizes transcript text

💻 Technical Implementation

1. `manifest.json`

The manifest defines the extension's permissions, resources, and entry points:

json

```json
{
  "manifest_version": 3,
  "name": "YouTube Note Taker",
  "description": "Generate AI notes from YouTube videos",
  "version": "1.0",
  "permissions": ["scripting", "activeTab", "storage"],
  "host_permissions": [
    "*://www.youtube.com/*",
    "http://localhost:3000/*"
  ],
  "action": {
    "default_popup": "popup.html"
  },
  "content_scripts": [
    {
      "matches": ["*://www.youtube.com/watch*"],
      "js": ["content.js"]
    }
  ]
}
```

Line-by-line explanation:

- `"manifest_version": 3`: Specifies we're using Manifest V3, the latest Chrome extension format
- `"name": "YouTube Note Taker"`: The name displayed in the Chrome Extensions page
- `"description": "Generate AI notes from YouTube videos"`: Brief description of the extension's functionality
- `"version": "1.0"`: The extension's version number
- `"permissions": ["scripting", "activeTab", "storage"]`:
  - `scripting`: Allows running scripts on web pages
  - `activeTab`: Grants temporary access to the current tab
  - `storage`: Allows using Chrome's storage API to save notes
- `"host_permissions": [...]`: Specifies which domains the extension can interact with:
  - `"*://www.youtube.com/*"`: YouTube domain for transcript extraction
  - `"http://localhost:3000/*"`: Local backend server for API communication

- `"action": {"default_popup": "popup.html"}`: Defines popup.html as the UI when clicking the extension icon
- `"content_scripts": [...]`: Scripts that run directly in web pages:
  - `"matches": ["*://www.youtube.com/watch*"]`: Only runs on YouTube watch pages
  - `"js": ["content.js"]`: The script file to inject into matching pages

2. Backend Components

2.1 `get_transcript.py`

This Python script fetches the transcript using the `youtube_transcript_api` library:

python

```python
import sys
import json
from youtube_transcript_api import YouTubeTranscriptApi

video_id = sys.argv[1]

try:
    transcript = YouTubeTranscriptApi.get_transcript(video_id)
    texts = " ".join([entry['text'] for entry in transcript])
    print(json.dumps(texts))
except Exception as e:
    print(f"Error: {str(e)}", file=sys.stderr)
    sys.exit(1)
```

Line-by-line explanation:

- `import sys`: Imports the system module to access command-line arguments
- `import json`: Imports the JSON module for data formatting
- `from youtube_transcript_api import YouTubeTranscriptApi`: Imports the API for transcript extraction
- `video_id = sys.argv[1]`: Gets the YouTube video ID from the first command-line argument
- `try:`: Start of exception handling block
- `transcript = YouTubeTranscriptApi.get_transcript(video_id)`: Fetches the transcript data for the given video ID
- `texts = " ".join([entry['text'] for entry in transcript])`:

    - Uses list comprehension to extract the 'text' part from each transcript entry
    - Joins all text segments with spaces to create a single string
- `print(json.dumps(texts))`: Converts the text to JSON and outputs it to stdout for Node.js to capture
- `except Exception as e:`: Catches any errors during transcript retrieval
- `print(f"Error: {str(e)}", file=sys.stderr)`: Prints error message to stderr
- `sys.exit(1)`: Exits with error code 1 to signal failure to the Node.js process

2.2 `index.js` (Express Server)

This Node.js Express application serves as the backend for processing transcripts and generating summaries:

```
javascript
```

```
const express = require('express');
const cors = require('cors');
const { exec } = require('child_process');
require('dotenv').config();

const app = express();
const PORT = process.env.PORT || 3000;

app.use(cors());

const genAI = require('@google/generative-ai');
const genAIclient = new genAI.GoogleGenerativeAI(process.env.GEMINI_API_KEY);

app.get('/transcript', (req, res) => {
  const videoId = req.query.videoId;
  if (!videoId) return res.status(400).json({ error: 'videoId is required' });

  exec(`python get_transcript.py ${videoId}`, async (error, stdout, stderr) => {
    if (error) {
      console.error('Python Error:', stderr);
      return res.status(500).json({ error: 'Failed to fetch transcript' });
    }

    const transcript = JSON.parse(stdout.trim());
    try {
      const model = genAIclient.getGenerativeModel({ model: "gemini-2.0-flash" });
      const prompt = `Summarize the following YouTube transcript into short, clear bullet-point

      const result = await model.generateContent(prompt);
      const response = await result.response;
      const summary = response.text();

      res.json({ notes: summary, videoId });
    } catch (apiError) {
      console.error('Gemini API Error:', apiError);
      res.status(500).json({ error: 'Failed to generate notes using Gemini' });
    }
  });
});

app.listen(PORT, () => {
  console.log(`Server running on http://localhost:${PORT}`);
});
```

Line-by-line explanation:

- `const express = require('express')`: Imports the Express.js framework
- `const cors = require('cors')`: Imports CORS middleware for cross-origin requests
- `const { exec } = require('child_process')`: Imports the exec function to run external commands
- `require('dotenv').config()`: Loads environment variables from .env file
- `const app = express()`: Creates an Express application
- `const PORT = process.env.PORT || 3000`: Sets the server port (from env or default 3000)
- `app.use(cors())`: Enables CORS for all routes
- `const genAI = require('@google/generative-ai')`: Imports Google's Generative AI library
- `const genAIclient = new genAI.GoogleGenerativeAI(process.env.GEMINI_API_KEY)`: Creates a client with the API key
- `app.get('/transcript', (req, res) => {...})`: Creates a GET endpoint at /transcript
- `const videoId = req.query.videoId`: Extracts videoId from the request query parameters
- `if (!videoId) return res.status(400).json({ error: 'videoId is required' })`: Validates that videoId exists
- `exec(`python get_transcript.py ${videoId}`, async (error, stdout, stderr) => {...})`:
  - Executes the Python script with videoId as argument
  - Passes callback for handling the result
- `if (error) {...}`: Handles execution errors
- `const transcript = JSON.parse(stdout.trim())`: Parses the transcript from script output
- `try {...}`: Start of AI processing block
- `const model = genAIclient.getGenerativeModel({ model: "gemini-2.0-flash" })`: Initializes Gemini model
- `const prompt = ...`: Creates a prompt for the AI model
- `const result = await model.generateContent(prompt)`: Sends prompt to Gemini
- `const response = await result.response`: Gets the response object
- `const summary = response.text()`: Extracts the text summary

- `res.json({ notes: summary, videoId })`: Returns the summary and videoId as JSON
- `catch (apiError) {...}`: Handles AI processing errors
- `app.listen(PORT, () => {...})`: Starts the server on the specified port

3. Frontend Components

3.1 `popup.js`

The main script that handles user interaction in the popup:

javascript

```
document.addEventListener('DOMContentLoaded', async () => {
  const [tab] = await chrome.tabs.query({ active: true, currentWindow: true });
  const url = new URL(tab.url);
  const currentVideoId = url.searchParams.get("v");

  // Check if we have saved notes for this video
  chrome.storage.local.get([currentVideoId], (data) => {
    if (data[currentVideoId]) {
      const htmlNotes = marked.parse(data[currentVideoId]);
      document.getElementById('notesOutput').innerHTML = htmlNotes;
      document.getElementById('downloadBtn').hidden = false;
    }
  });
});

document.getElementById('generateNotes').addEventListener('click', async () => {
  const generateBtn = document.getElementById('generateNotes');
  const loadingDiv = document.getElementById('loading');
  const notesOutput = document.getElementById('notesOutput');
  const downloadBtn = document.getElementById('downloadBtn');

  notesOutput.innerHTML = '';
  loadingDiv.style.display = 'block';
  generateBtn.disabled = true;
  downloadBtn.hidden = true;

  let [tab] = await chrome.tabs.query({ active: true, currentWindow: true });

  chrome.scripting.executeScript({
    target: { tabId: tab.id },
    files: ['content.js']
  }, () => {
    chrome.tabs.sendMessage(tab.id, { action: 'getTranscript' }, (response) => {
      loadingDiv.style.display = 'none';
      generateBtn.disabled = false;

      if (chrome.runtime.lastError) {
        console.error("Popup: Error", chrome.runtime.lastError.message);
        notesOutput.innerText = "Error: " + chrome.runtime.lastError.message;
        return;
      }

      if (response.notes) {
        const rawNotes = response.notes;
        const htmlNotes = marked.parse(rawNotes);

        // Save per video ID
        chrome.storage.local.set({ [response.videoId]: rawNotes }, () => {
          console.log(`Saved notes for video ID: ${response.videoId}`);
        });

        notesOutput.innerHTML = htmlNotes;
        downloadBtn.hidden = false;
      } else if (response.error) {
        notesOutput.innerText = response.error;
      }
    });
  });
});

document.getElementById('downloadBtn').addEventListener('click', async () => {
  let [tab] = await chrome.tabs.query({ active: true, currentWindow: true });
  const url = new URL(tab.url);
  const currentVideoId = url.searchParams.get("v");

  chrome.storage.local.get([currentVideoId], (data) => {
    if (data[currentVideoId]) {
      const blob = new Blob([data[currentVideoId]], { type: 'text/markdown' });
      const url = URL.createObjectURL(blob);
      const link = document.createElement('a');
      link.href = url;
      link.download = 'youtube-notes.md';
      link.click();
      URL.revokeObjectURL(url);
    }
  });
});
```

Line-by-line explanation:

- `document.addEventListener('DOMContentLoaded', async () => {...})`: Runs when popup HTML is fully loaded
- `const [tab] = await chrome.tabs.query({ active: true, currentWindow: true })`: Gets the current active tab
- `const url = new URL(tab.url)`: Creates a URL object from tab's URL

- `const currentVideoId = url.searchParams.get("v")`: Extracts YouTube video ID from URL parameters
- `chrome.storage.local.get([currentVideoId], (data) => {...})`: Checks local storage for saved notes
- `if (data[currentVideoId]) {...}`: If notes exist for this video:
  - `const htmlNotes = marked.parse(data[currentVideoId])`: Converts markdown notes to HTML
  - `document.getElementById('notesOutput').innerHTML = htmlNotes`: Displays notes in output div
  - `document.getElementById('downloadBtn').hidden = false`: Shows download button
- `document.getElementById('generateNotes').addEventListener('click', async () => {...})`: Adds click handler to generate button
- `const generateBtn/loadingDiv/notesOutput/downloadBtn = ...`: Gets UI elements
- `notesOutput.innerHTML = ''`: Clears previous notes
- `loadingDiv.style.display = 'block'`: Shows loading indicator
- `generateBtn.disabled = true`: Disables generate button during processing
- `downloadBtn.hidden = true`: Hides download button
- `let [tab] = await chrome.tabs.query({ active: true, currentWindow: true })`: Gets active tab again
- `chrome.scripting.executeScript({...})`: Injects content script into active tab
- `chrome.tabs.sendMessage(tab.id, { action: 'getTranscript' }, (response) => {...})`: Sends message to content script
- `loadingDiv.style.display = 'none'`: Hides loading indicator
- `generateBtn.disabled = false`: Re-enables generate button
- `if (chrome.runtime.lastError) {...}`: Handles communication errors
- `if (response.notes) {...}`: Processes successful response:
  - `const rawNotes = response.notes`: Gets raw markdown notes
  - `const htmlNotes = marked.parse(rawNotes)`: Converts to HTML
  - `chrome.storage.local.set({ [response.videoId]: rawNotes }, () => {...})`: Saves notes with video ID as key
  - `notesOutput.innerHTML = htmlNotes`: Displays notes
  - `downloadBtn.hidden = false`: Shows download button

- `else if (response.error) {...}`: Displays any error message
- `document.getElementById('downloadBtn').addEventListener('click', async () => {...})`: Adds click handler to download button
- `let [tab] = await chrome.tabs.query({...})`: Gets active tab
- `const url = new URL(tab.url)`: Creates URL object
- `const currentVideoId = url.searchParams.get("v")`: Gets current video ID
- `chrome.storage.local.get([currentVideoId], (data) => {...})`: Retrieves notes from storage
- `if (data[currentVideoId]) {...}`: If notes exist:
  - `const blob = new Blob([data[currentVideoId]], { type: 'text/markdown' })`: Creates a file blob
  - `const url = URL.createObjectURL(blob)`: Creates a downloadable URL
  - `const link = document.createElement('a')`: Creates a download link
  - `link.href = url`: Sets link URL
  - `link.download = 'youtube-notes.md'`: Sets filename
  - `link.click()`: Triggers download
  - `URL.revokeObjectURL(url)`: Cleans up the URL object

3.2 `content.js`

Script injected into YouTube pages to extract information and communicate with the backend:

javascript

```javascript
chrome.runtime.onMessage.addListener((request, sender, sendResponse) => {
  if (request.action === 'getTranscript') {
    const videoId = new URLSearchParams(window.location.search).get('v');
    fetch(`http://localhost:3000/transcript?videoId=${videoId}`)
      .then(res => res.json())
      .then(data => {
        sendResponse({
          notes: data.notes,
          videoId: videoId
        });
      })
      .catch(error => {
        sendResponse({
          error: `Error fetching transcript: ${error.message}`
        });
      });
    return true; // Keep message channel open for async response
  }
});
```

Line-by-line explanation:

- `chrome.runtime.onMessage.addListener((request, sender, sendResponse) => {...})`: Sets up a listener for messages from the extension
- `if (request.action === 'getTranscript') {...}`: Checks if the message is requesting a transcript
- `const videoId = new URLSearchParams(window.location.search).get('v')`: Gets video ID from current page URL
- `fetch(`http://localhost:3000/transcript?videoId=${videoId}`)`: Makes request to backend server with video ID
- `.then(res => res.json())`: Parses JSON response
- `.then(data => {...})`: Handles successful response:
  - `sendResponse({notes: data.notes, videoId: videoId})`: Sends notes back to popup
- `.catch(error => {...})`: Handles errors:
  - `sendResponse({error: `Error fetching transcript: ${error.message}`})`: Sends error message
- `return true`: Keeps the message channel open for asynchronous response (crucial for fetch operations)

4. `popup.html`

The extension's user interface:

html

```html
<!DOCTYPE html>
<html>
<head>
  <title>YouTube Note Taker</title>
  <script src="https://cdn.jsdelivr.net/npm/marked/marked.min.js"></script>
  <style>
    body {
      font-family: Arial, sans-serif;
      width: 400px;
      padding: 16px;
      margin: 0;
    }
    .container {
      display: flex;
      flex-direction: column;
      gap: 16px;
    }
    h1 {
      margin-top: 0;
      color: #333;
      font-size: 20px;
    }
    #loading {
      display: none;
      text-align: center;
      padding: 10px;
    }
    .spinner {
      border: 4px solid #f3f3f3;
      border-top: 4px solid #3498db;
      border-radius: 50%;
      width: 30px;
      height: 30px;
      animation: spin 2s linear infinite;
      margin: 0 auto;
    }
    @keyframes spin {
      0% { transform: rotate(0deg); }
      100% { transform: rotate(360deg); }
    }
    #notesOutput {
      border: 1px solid #ddd;
      border-radius: 4px;
      padding: 12px;
      max-height: 300px;
      overflow-y: auto;
      background-color: #f9f9f9;
    }
    button {
      padding: 8px 16px;
      border: none;
      border-radius: 4px;
      cursor: pointer;
      font-weight: bold;
    }
    #generateNotes {
      background-color: #4285f4;
      color: white;
    }
    #downloadBtn {
      background-color: #34a853;
      color: white;
    }
    .buttons {
      display: flex;
      gap: 8px;
    }
  </style>
</head>
<body>
  <div class="container">
    <h1>YouTube Note Taker</h1>

    <div id="loading">
      <div class="spinner"></div>
      <p>Generating notes...</p>
    </div>

    <div id="notesOutput"></div>

    <div class="buttons">
      <button id="generateNotes">Generate Notes</button>
      <button id="downloadBtn" hidden>Download Notes</button>
    </div>
  </div>

  <script src="popup.js"></script>
```

```
</body>
</html>
```

Line-by-line explanation:

- Basic HTML structure with head and body
- `<script src="https://cdn.jsdelivr.net/npm/marked/marked.min.js"></script>`: Imports the Marked library for markdown processing
- CSS section defines the styling:
  - Sets popup width, font, and spacing
  - Creates a loading spinner animation
  - Styles the notes output area with scrolling
  - Defines button appearances and layout
- Body contains:
  - `<h1>YouTube Note Taker</h1>`: Title of the extension
  - `<div id="loading">`: Loading indicator with spinner
  - `<div id="notesOutput">`: Container for displaying notes
  - Button container with:
    - `<button id="generateNotes">`: Triggers note generation
    - `<button id="downloadBtn" hidden>`: Downloads notes (initially hidden)
- `<script src="popup.js"></script>`: Loads the popup interaction script

🎨 UI Implementation

The extension features a clean, functional interface with:

1. **Header**: Clearly identifies the extension's purpose
2. **Loading Indicator**: Shows animation during processing
3. **Notes Display Area**: Scrollable container for output
4. **Control Buttons**:
   - Generate Notes: Primary action button
   - Download Notes: Secondary action for saving content

The design prioritizes usability with visual feedback during operations and clear delimitation of interactive elements.

🧰 Tech Stack

Frontend

- **HTML/CSS/JavaScript**: Core web technologies
- **Chrome Extensions API**: For browser integration
- **Marked.js**: Markdown parsing and rendering

Backend

- **Node.js**: JavaScript runtime for server-side code
- **Express.js**: Web framework for Node.js
- **Python**: Used for transcript extraction
- **youtube-transcript-api**: Python library for YouTube transcripts
- **Gemini 2.0 Flash**: Google's AI model for summarization

Tools & Libraries

- **CORS**: Cross-origin resource sharing
- **dotenv**: Environment variable management
- **Child Process (exec)**: For Python script execution from Node.js

🐛 Common Issues & Solutions

Issue	Description	Solution
Notes overwriting	Different videos' notes were overwriting each other	Used video ID as storage key
CORS issues	Frontend couldn't directly access Gemini API	Implemented Node.js backend to handle API calls
Response timeouts	Long transcript processing caused timeout errors	Used `return true` in message listener to keep connection open
UI overflow	Notes content too large for display area	Added max-height and scrolling to notes container
API quota limits	Initial OpenAI implementation hit rate limits	Switched to Gemini API for more generous free tier

📁 Project Structure

```
📁 youtube-note-taker/
│
├── manifest.json          # Extension configuration
├── popup.html             # User interface
├── popup.js               # Popup logic
├── content.js             # YouTube page script
│
└── backend/               # Server components
    ├── index.js           # Express server & Gemini integration
    ├── get_transcript.py  # Transcript extraction script
    ├── package.json       # Node dependencies
    └── .env               # Environment variables (API keys)
```

🚀 Setup Instructions

1. **Backend Setup**:

   bash

   ```bash
   cd backend
   npm install
   pip install youtube-transcript-api
   # Create .env file with GEMINI_API_KEY=your_api_key
   node index.js
   ```

2. **Extension Setup**:
   - Open Chrome and navigate to `chrome://extensions`
   - Enable "Developer mode"
   - Click "Load unpacked" and select the extension folder
   - Navigate to a YouTube video and click the extension icon

🔧 Future Improvements

1. **Offline Support**: Cache previously generated notes for offline access
2. **Custom Prompts**: Allow users to customize the AI summarization prompt
3. **Export Options**: Add more export formats (PDF, HTML, etc.)
4. **UI Themes**: Implement light/dark mode and customizable themes
5. **Note Editing**: Allow users to edit/annotate generated notes

📄 Conclusion

This YouTube Note Taker Chrome Extension provides a powerful tool for students, researchers, and casual learners who want to extract valuable information from YouTube videos efficiently. By combining modern web technologies with AI capabilities, it transforms video content into concise, accessible notes with minimal effort.