

Video2Document (V2D) – Software Documentation Package

Document Version: 1.0.0

Conformance: ISO/IEC/IEEE 26514:2022 – Systems and Software Engineering – Requirements for Designers and Developers of User Documentation

Status: Release Candidate

Last Updated: 2026-02-22

Target Audience: End Users · Developers · Administrators

Technical Depth: Intermediate → Expert

Table of Contents

1.	Executive Summary
2.	Product Overview
3.	System Architecture
4.	Technology Stack
5.	Installation & Setup Guide
6.	Configuration Guide
7.	Usage Guide – End User Perspective
8.	API Documentation
9.	Data Model Description
10.	Security Considerations
11.	Performance & Scalability Notes
12.	Deployment Guide
13.	Testing Strategy
14.	Maintenance & Support Plan
15.	Limitations & Known Issues
16.	Future Improvements / Roadmap
17.	Glossary

1. Executive Summary

Video2Document (V2D) is a cross-platform desktop application that automates the conversion of video and audio recordings into structured, professional documents. It combines AI-based speech transcription, speaker diarisation, and large language model (LLM) document generation into a single, guided end-to-end pipeline.

The system addresses a concrete operational need: organisations and individuals frequently produce recordings of meetings, lectures, and collaborative sessions that remain inaccessible as raw media files. V2D transforms this latent content into actionable, shareable documents – meeting reports, agendas, sprint planning notes, and custom-format outputs – without requiring users to possess programming or AI expertise.

The application is delivered as an Electron desktop executable, runs on Windows, macOS, and Linux, and integrates with three independent LLM providers (Google Gemini, OpenAI GPT via SAIA, and Alibaba Qwen3) and two transcription backends (AssemblyAI cloud and Whisper local). Its modular architecture permits the addition of new AI providers or output formats with minimal modification to the core codebase.

Key capabilities at a glance:

Capability	Detail
Input formats	MP4, MOV, WAV, MP3, FLAC (any FFmpeg-compatible source)
Output formats	PDF, DOCX, HTML, TXT
Transcription	AssemblyAI (cloud) · Whisper.cpp (local/offline)
LLM providers	Google Gemini · OpenAI GPT · Alibaba Qwen3
Document types	Follow-up report · Agenda · Result protocol · Sprint planning note · Custom
Speaker handling	Automatic diarisation with post-hoc name substitution
Deployment	Cross-platform Electron desktop app
UI languages	English · German (extensible)

2. Product Overview

2.1 Problem Statement

Recording meetings and lectures is widespread practice. However, converting those recordings into usable documents is time-intensive, error-prone, and requires

significant cognitive effort. Manually transcribing a one-hour meeting and reformatting it as a professional report can require two to four hours of skilled effort. V2D reduces this effort to a guided six-step workflow completed in minutes, with the document quality governed by the selected LLM and document-type prompt.

2.2 Target Users

User Type	Description	Primary Need
End User (non-technical)	Meeting participants, students, knowledge workers	Generate a document from a recording with no setup
Developer	Engineers extending or integrating V2D	Understand architecture, add modules, contribute
Administrator	IT staff deploying V2D internally	Manage API keys, configure keyserver, maintain deployments

2.3 Functional Requirements

ID	Requirement	Priority
FR-01	The system shall accept video and audio files as input	Must
FR-02	The system shall extract audio from video files	Must
FR-03	The system shall transcribe audio to text with speaker labels	Must
FR-04	The system shall generate structured documents from transcripts using an LLM	Must
FR-05	The system shall allow the user to assign real names to speakers	Must
FR-06	The system shall export documents in PDF, DOCX, HTML, and TXT formats	Must
FR-07	The system shall provide at least three standard document type templates	Must
FR-08	The system shall allow users to create custom document type templates	Should
FR-09	The system shall support at least two independent transcription backends	Should
FR-10	The system shall support at least two independent LLM backends	Should
FR-11	The system shall display processing progress to the user	Should
FR-12	The system shall provide playable speaker audio previews	Could
FR-13	The system shall support UI localisation to at least English and German	Could

2.4 Non-Functional Requirements

ID	Requirement	Category
NFR-01	The application shall run on Windows 10+, macOS 12+, and Ubuntu 20.04+	Portability
NFR-02	API keys shall never be stored in source-controlled files	Security
NFR-03	The UI shall be operable by a non-technical user without documentation	Usability
NFR-04	Individual modules shall be replaceable without modifying the core pipeline	Maintainability
NFR-05	The addition of a new LLM or transcription module shall require changes to exactly one directory	Extensibility
NFR-06	Intermediate artefacts (audio, transcripts, HTML) shall be persisted on disk	Reliability
NFR-07	The application shall report errors to the user via the UI	Usability
NFR-08	All secrets shall be injected at runtime via environment variables or keyserver	Security

NFR-09	The system shall complete processing of a 30-minute recording within 10 minutes (cloud transcription)	Performance
NFR-10	The test suite shall cover all major module types	Testability

2.5 Scope and Constraints

In scope:

- Desktop application for video/audio-to-document conversion
- Modular plugin system for transcription and LLM providers
- Multi-format document export
- Speaker diarisation and name mapping

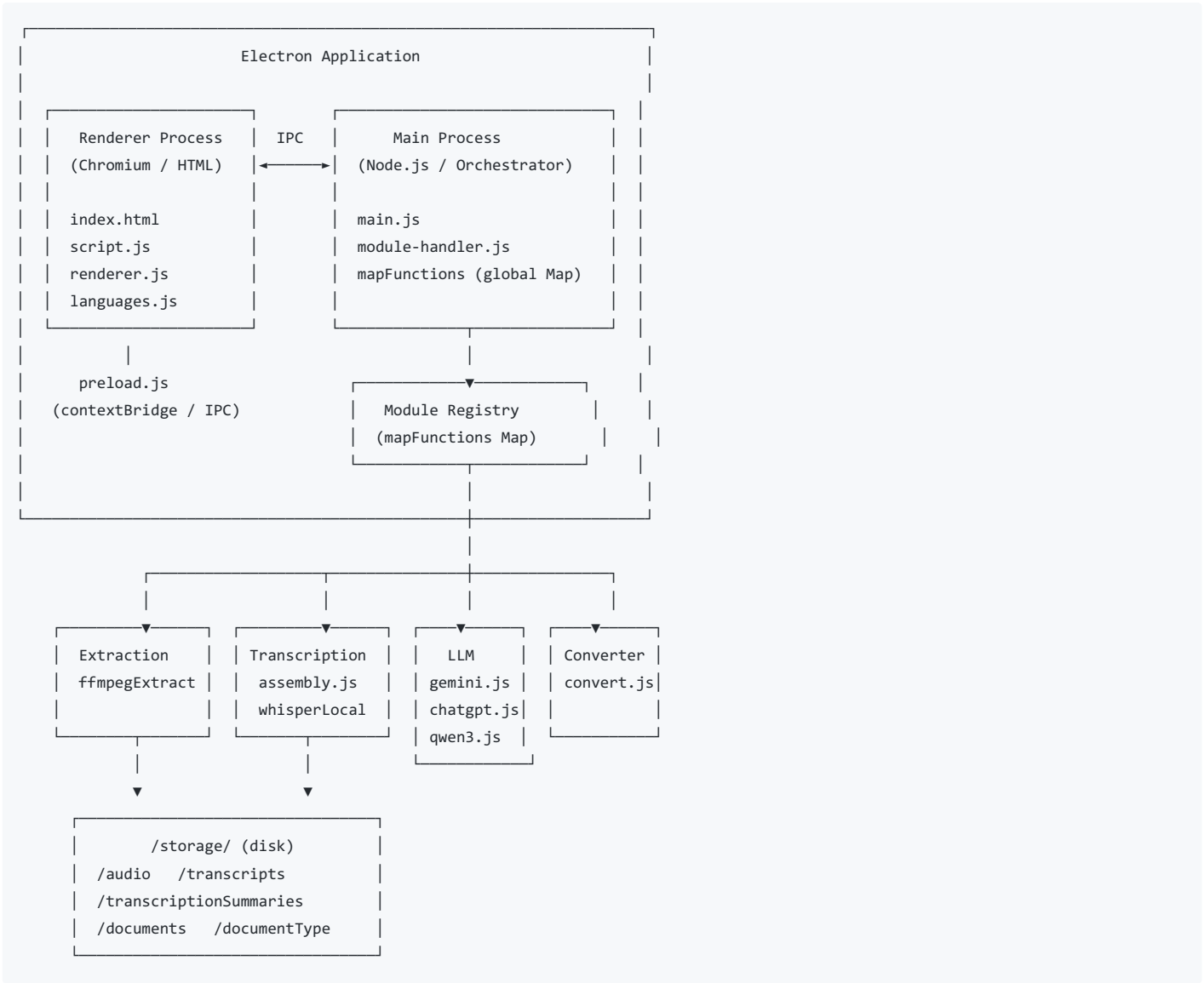
Out of scope (current version):

- Real-time (streaming) transcription
- Web-based or SaaS deployment
- Voice biometric speaker identification
- Video editing or preview
- Multi-user collaboration or cloud storage

3. System Architecture

3.1 Architectural Overview

V2D follows a **layered, modular pipeline architecture** running within a single Electron process pair (main process + renderer process). There is no external server dependency for the core workflow; all orchestration is local.



3.2 Process Architecture

Electron provides two isolated JavaScript contexts:

Process	Runtime	Responsibilities
Main Process	Node.js	File system, pipeline execution, module loading, IPC handlers, native dialogs
Renderer Process	Chromium (sandboxed)	UI rendering, user interaction, progress display
Preload Script	Node.js (bridge)	Exposes a controlled API surface to the renderer via contextBridge

Node integration in the renderer is explicitly disabled. All privileged operations are routed through the preload bridge.

3.3 Module System

The module system is the central extensibility mechanism. On startup, `main.js` traverses all subdirectories of `services/modules/`, dynamically requires each `.js` file, and registers the exported module object in a global `Map<string, Module>` keyed by `module.name`.

```
services/modules/
├─ extraction/           ffmpegExtractor.js
├─ transcription-remote/ assembly.js
├─ transcription-local/  whisperLocal.ts
├─ jsonTools/           transcriptionSummarizer2.js
├─ llm-gemini/           gemini.js
├─ llm-chat_gpt/         chatgpt.js
├─ quen3/               qwen3.js
├─ convert/              convert.js
├─ audioSnippets/        extract-speaker-snippets.js
├─ replace_speaker/      replaceSpeaker.js
└─ utility/              @startup.js · module-handler.js
```

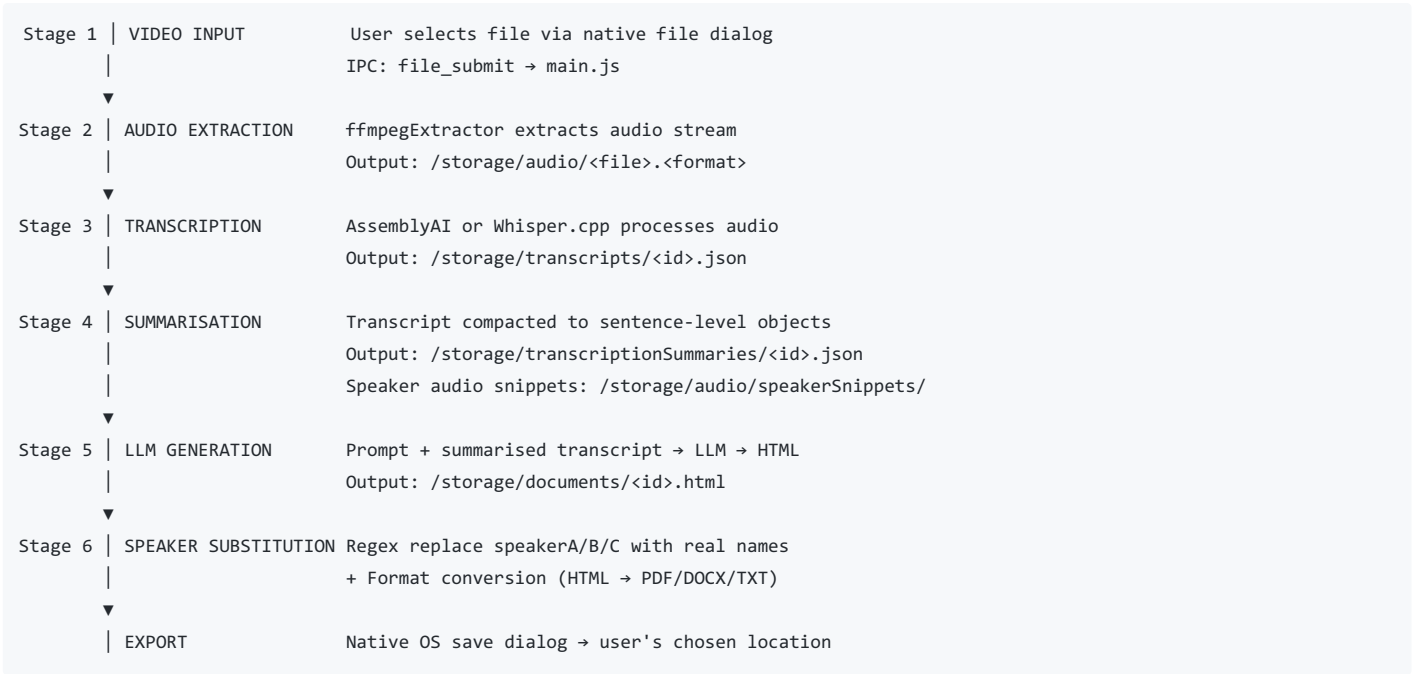
Each module conforms to the following interface:

```
module.exports = {
  name:      "unique-module-identifier", // key in mapFunctions
  type:      "llm|transcription|...",   // category label
  displayname: "Human-Readable Name",   // shown in UI dropdowns
  description: "What this module does",
  audioformat: "mp3",                   // (transcription modules only)
  async function(parameter) {           // primary entry point
    // Implementation
  }
}
```

Modules are invoked as: `mapFunctions.get("module-name").function(params)`.

3.4 Pipeline Execution Flow

The six-step user workflow maps to the following internal pipeline stages:



3.5 IPC Communication Map

The following IPC channels are defined between the renderer and main processes:

Channel	Direction	Trigger	Payload
file_submit	Renderer → Main	User submits step 1 form	Pipeline configuration object
file_download	Renderer → Main	User clicks Download	{ type, speakerMap }
speaker_submit	Renderer → Main	User confirms speaker names	Speaker name map
progress	Main → Renderer	After each pipeline stage	Stage number (1–4)
speakerAudios	Main → Renderer	After snippet extraction	Speaker audio path map
error	Main → Renderer	Any pipeline exception	Error message string
get-module-names	Renderer ↔ Main	UI init	Module list
get-txt-files	Renderer ↔ Main	UI init	Document type filenames
save-txt-file	Renderer ↔ Main	Custom type editor save	{ filename, content }
read-txt-file	Renderer ↔ Main	Custom type editor load	Filename
delete-txt-file	Renderer ↔ Main	Custom type editor delete	Filename

4. Technology Stack

4.1 Core Runtime and Framework

Component	Technology	Version	Purpose
Runtime	Node.js	19 – 24	JavaScript execution environment
Desktop Framework	Electron	39.1.1	Cross-platform desktop application shell

HTTP Framework Component	Express Technology	5.1.0 Version	Local HTTP server (IPC supplement; API Purpose Skeleton)
Primary Language	JavaScript (ES2022)	—	All production modules
Secondary Language	TypeScript	5.9.3	Local transcription module, pipeline job examples
TypeScript Runtime	ts-node	10.9.2	Execute TypeScript without pre-compilation

4.2 AI and LLM Providers

Provider	Module	Environment Variable	Model	Mode
Google Gemini	llm-gemini	GOOGLE_API_KEY	gemini-2.0-flash (default)	Cloud
OpenAI GPT (via SAIA)	llm-saia_openai_gpt	SAIA_API_KEY	OSS 120B	Cloud
Alibaba Qwen3	qwen3-235b-a22b	SAIA_API_KEY	Qwen3-235B-A22B	Cloud

4.3 Transcription Backends

Provider	Module	Environment Variable	Mode	Speaker Labels
AssemblyAI	assembly	ASSEMBLYAI_API_KEY	Cloud (remote)	Yes
Whisper.cpp	whisperLocal	None required	Local (offline)	Limited

4.4 Audio and Video Processing

Library	Version	Purpose
fluent-ffmpeg	Latest	Node.js wrapper for FFmpeg CLI
ffmpeg-static	Latest	Bundled static FFmpeg binary

Supported input container formats: any FFmpeg-compatible format (MP4, MOV, AVI, MKV, WebM for video; WAV, MP3, FLAC, M4A for audio).
Extraction output formats: WAV (16kHz, mono), MP3, FLAC — determined per transcription module's `audioformat` property.

4.5 Document Generation and Export

Library	Purpose	Output
html-to-docx	Convert HTML to Word document	.docx
puppeteer	Headless Chromium for PDF rendering	.pdf
Native file write	Plain text export	.txt
Native file write	HTML export	.html

4.6 Networking and Utilities

Library	Purpose
axios	HTTP client for external API calls

Library	Purpose
Load environment variables from .env file	

4.7 Development and Test Dependencies

Library	Purpose
mocha (v11.7.5)	Unit and integration test runner
@types/node (v24.9.2)	TypeScript Node.js type definitions
@types/fluent-ffmpeg	TypeScript FFmpeg type definitions
typescript (v5.9.3)	TypeScript compiler
ts-node (v10.9.2)	Direct TypeScript execution

5. Installation & Setup Guide

5.1 Prerequisites

Before installing V2D, ensure the following are present on the target machine:

Requirement	Minimum Version	Notes
Node.js	19.x	Versions 19 through 24 are supported. Use <code>node -v</code> to verify.
npm	9.x	Bundled with Node.js.
Git	Any recent	Required to clone the repository.
Internet connection	—	Required for cloud transcription and LLM services.
FFmpeg	Any recent	Bundled via <code>ffmpeg-static</code> ; a separate system install is not required.

Note: A separate system-level FFmpeg installation is not required. The `ffmpeg-static` package provides a platform-appropriate binary that is automatically used by `fluent-ffmpeg`.

5.2 Obtaining the Source Code

```
git clone <repository-url>
cd video2document
```

Replace `<repository-url>` with the actual repository URL provided by your administrator.

5.3 Installing Dependencies

From the project root directory:

```
npm install
```

This installs all dependencies declared in `package.json`, including Electron, Puppeteer, and all service libraries. The installation may take several minutes on the first run due to Puppeteer downloading a Chromium binary.

5.4 Environment Configuration

V2D requires API keys for external services. These are provided via environment variables.

Option A — Local `.env` file (development / personal use):

Create a file named `.env` in the project root:

```
GOOGLE_API_KEY=your_google_gemini_key
ASSEMBLYAI_API_KEY=your_assemblyai_key
SAIA_API_KEY=your_saia_platform_key
```

The `.env` file is Git-ignored and will never be committed to the repository.

Option B — Keyserver (organisational deployment):

If your organisation operates the V2D keyserver, set the following variables instead:

```
auth_username=your_keyserver_username
auth_password=your_keyserver_password
```

On startup, the application contacts `keyserver.dommymommy.xyz:443` and retrieves all required API keys automatically. See [Section 6.2](#) for details.

5.5 Starting the Application

Linux / macOS:

```
./start.sh
```

Or manually:

```
npm install # if not already done
npm start
```

Windows:

```
start.bat
```

Or manually from Command Prompt or PowerShell:

```
npm install
npm start
```

The Electron window will open automatically after a brief initialisation period.

5.6 Verifying the Installation

After launch, the V2D main window should appear showing Step 1 (file selection). If the window does not appear:

- 1. Check the terminal / command prompt for Node.js error output.
- 2. Confirm Node.js version with `node -v`.
- 3. Confirm `npm install` completed without errors.
- 4. Confirm at least one set of API credentials is present in `.env` or via keyserver.

6. Configuration Guide

6.1 Environment Variables Reference

Variable	Required By	Description
<code>GOOGLE_API_KEY</code>	<code>llm-gemini</code> module	Google Cloud API key with Generative Language API enabled
<code>ASSEMBLYAI_API_KEY</code>	<code>assembly</code> transcription module	AssemblyAI account API key
<code>SAIA_API_KEY</code>	<code>llm-saia-openai-gpt</code> , <code>qwen3</code> modules	SAIA platform API key for hosted LLM access
<code>auth_username</code>	Keyserver auth	Username credential for automatic key retrieval
<code>auth_password</code>	Keyserver auth	Password credential for automatic key retrieval

Only the variables corresponding to the modules you intend to use are strictly required. The application will report a clear error if a module is invoked without its required key.

6.2 Keyserver Authentication

The keyserver is a proprietary key distribution service hosted at `keyserver.dommymommy.xyz:443`. When credentials are provided, the startup module (`@startup.js`) performs the following sequence:

1. Issues a `GET` request to the keyserver endpoint with `auth_username` and `auth_password` as HTTP headers.
2. The server responds with a JSON object containing API key-value pairs.
3. Keys are merged into `process.env` within the main process.
4. Keys are never forwarded to the renderer process.

This mechanism allows teams to rotate keys centrally without redistributing `.env` files to individual users.

6.3 Document Type Templates

Document type templates control the instructions given to the LLM when generating output. They are stored as plain text files in `/storage/documentType/`.

Built-in templates:

Filename	Purpose
<code>followup_report.txt</code>	Professional post-meeting follow-up report
<code>agenda.txt</code>	Reconstructed meeting agenda from transcript
<code>result_protocol.txt</code>	Formal meeting minutes / result protocol
<code>sprint_planning_note.txt</code>	Agile sprint planning session documentation
<code>custom_document.txt</code>	Placeholder for user-defined format

Creating a custom template:

Templates are editable directly through the V2D UI (Custom Type Editor, accessible from the document type step) or by editing the `.txt` files directly in `/storage/documentType/`.

Template authoring guidelines:

- Write instructions for the LLM in clear, imperative sentences.
- Reference speaker tokens using the exact placeholder strings `speakerA`, `speakerB`, `speakerC`, etc.
- Specify the desired output structure (headings, bullet lists, tables).
- Include language instructions if multilingual output is required.
- Do **not** include the transcript itself — it is automatically appended by the pipeline.

6.4 Module Configuration

Modules are self-describing. To enable or disable a module, add or remove its `.js` file from its directory under `services/modules/`. The module will be included or excluded from all UI dropdowns automatically on next application start.

To change the default Gemini model, edit the `model` constant in `services/modules/llm-gemini/gemini.js`.

6.5 Storage Paths

All artefact directories are located under `/storage/` at the project root. These paths are currently hard-coded and are not configurable without modifying the relevant module source files. Future versions may expose these as configuration parameters.

7. Usage Guide — End User Perspective

7.1 Workflow Overview

V2D presents users with a linear six-step wizard. Steps must be completed in order. Each step requires a selection or action before the next step becomes available.

Step 1 → Step 2 → Step 3 → Step 4 → [Processing] → Step 5 → Step 6					
File	Transcrip-	Document	Output	Preview	Download
Select	tion Svc	Type	Format	& Names	

7.2 Step-by-Step Instructions

Step 1 — Select Video or Audio File

1. Click **Browse** or drag and drop a file onto the designated area.
2. Accepted formats: any FFmpeg-compatible video or audio file (MP4, MOV, WAV, MP3, FLAC, etc.).
3. The selected filename is displayed for confirmation.
4. Click **Next** to proceed.

Step 2 — Select Transcription Service

1. A dropdown lists all registered transcription modules.
2. Select the desired backend:
 - **AssemblyAI** — cloud-based, highest accuracy, requires internet and API key.
 - **Whisper Local** — offline, no API key, lower accuracy for multi-speaker content.
3. Click **Next** to proceed.

Step 3 — Select Document Type

1. A dropdown lists all document type templates found in `/storage/documentType/`.
2. Select the template matching your desired output format (e.g., "Follow-up Report").
3. Optionally, click **Edit / Create Custom Type** to open the template editor.
4. Click **Next** to proceed.

Step 4 — Select Output Format and LLM

1. Select the export format: **PDF, DOCX, HTML, or TXT**.
2. Select the LLM provider from the dropdown.
3. Click **Generate Document** to begin processing.

Processing Phase

A progress bar advances through four stages:

Stage	Description
1 / 4	Extracting audio from video
2 / 4	Transcribing audio (cloud or local)
3 / 4	Generating document with LLM
4 / 4	Converting to selected output format

Processing time depends on recording length and network speed. A 30-minute recording typically completes in 3–8 minutes with cloud services.

Step 5 — Review Document and Assign Speaker Names

1. A preview of the generated document is displayed.
2. For each detected speaker (labelled **Speaker A, Speaker B**, etc.):
 - A short audio clip of that speaker is playable for identification.
 - A text field allows entry of the speaker's real name.
3. Click **Apply Names** to replace all speaker placeholders in the document.
4. Review the updated preview.

Step 6 — Download / Export

1. Click **Download**.
2. A native OS save dialog opens.
3. Choose the destination path and filename.
4. The file is saved in the format selected in Step 4.

7.3 UI Language Selection

The UI language can be changed via the language selector (flag icons). Currently supported languages: English, German. The selection applies immediately to all UI text.

7.4 Help Page

A built-in help page is accessible from the main toolbar. It provides concise per-step guidance and explains the purpose of each configuration option.

7.5 Error Messages

If the pipeline encounters an error, a notification is displayed in the UI with a description of the failure. Common causes include:

- Missing or invalid API key for the selected service

- Network connectivity loss during cloud transcription
- Unsupported or corrupted input file
- Insufficient disk space in the `/storage/` directory

8. API Documentation

8.1 Status

The REST API component (`/API-Skeleton/`) is a **non-integrated template**. It defines the intended endpoint structure for a future server-mode version of V2D. The current production application communicates exclusively via Electron IPC.

This section documents both the existing IPC interface (authoritative) and the planned REST API (informational).

8.2 IPC API (Production)

The IPC interface is the authoritative communication mechanism between the renderer and main processes. It is accessed via the `window.electron` object injected by the preload script.

`window.electron.submitFile(payload)`

Initiates the full processing pipeline.

Direction: Renderer → Main

Parameters:

```
interface SubmitFilePayload {
  video: {
    module: string;          // e.g. "extraction-video-to-audio"
    inputVideoPath: string; // Absolute path to input file
  };
  transcription: {
    module: string;          // e.g. "assembly"
  };
  document: {
    module: string;          // e.g. "llm-gemini"
    type: string;            // Document type template name (without .txt)
    outputType: "pdf" | "docx" | "html" | "txt";
  };
}
```

Example:

```
window.electron.submitFile({
  video: {
    module: "extraction-video-to-audio",
    inputVideoPath: "/Users/name/recordings/meeting.mp4"
  },
  transcription: { module: "assembly" },
  document: {
    module: "llm-gemini",
    type: "followup_report",
    outputType: "pdf"
  }
});
```

`window.electron.downloadFile(payload)`

Triggers the final document export.

Direction: Renderer → Main

Parameters:

```
interface DownloadPayload {
  type: "pdf" | "docx" | "html" | "txt";
  speakerMap: Record<string, string>; // e.g. { "speakerA": "Alice", "speakerB": "Bob" }
}
```

`window.electron.onProgress(callback)`

Registers a listener for pipeline progress events.

Direction: Main → Renderer

Callback parameter: (stage: number) => void — stage is an integer 1 through 4.

`window.electron.onSpeakerAudios(callback)`

Receives speaker audio clip paths after diarisation.

Direction: Main → Renderer

Callback parameter:

```
(speakerMap: Record<string, { src: string; name: string }>) => void
```

`window.electron.onError(callback)`

Registers a listener for pipeline error events.

Direction: Main → Renderer

Callback parameter: (message: string) => void

`window.electron.getModuleNames()`

Returns the list of registered module names, grouped by type.

Direction: Renderer ↔ Main (invoke/handle)

`window.electron.getTxtFiles()`

Returns filenames of all document type templates.

Direction: Renderer ↔ Main (invoke/handle)

`window.electron.saveTxtFile(filename, content)`

Persists a new or updated document type template.

Direction: Renderer ↔ Main (invoke/handle)

`window.electron.readTxtFile(filename)`

Returns the content of a named document type template.

Direction: Renderer ↔ Main (invoke/handle)

`window.electron.deleteTxtFile(filename)`

Deletes a named document type template.

Direction: Renderer ↔ Main (invoke/handle)

8.3 REST API Skeleton (Planned — Not Integrated)

Base URL: `http://localhost:3000`

Authentication: Not yet defined.

Health Check

```
GET /api/v1/healthcheck
```

Response: 200 OK — "running"

Video Endpoints

Method	Path	Description	Status
POST	/videos	Upload a video file for processing	Planned
GET	/videos	List all processed videos	Planned
GET	/videos/:id	Retrieve a specific video record	Planned
DELETE	/videos/:id	Delete a video record	Planned

Document Endpoints

Method	Path	Description	Status
GET	/documents	List all generated documents	Planned
GET	/documents/:id	Retrieve a specific document	Planned
DELETE	/documents/:id	Delete a document record	Planned

9. Data Model Description

9.1 Primary Data Artefacts

V2D does not use a relational database in the current version. All persistent data is stored as files in structured directories under `/storage/`.

9.2 Raw Transcript (AssemblyAI Output)

Stored in: `/storage/transcripts/<transcript_id>.json`

```
interface TranscriptWord {
  text: string;      // Individual word token
  speaker: string;   // Speaker label, e.g. "A", "B"
  start: number;     // Word start time, milliseconds
  end: number;       // Word end time, milliseconds
  confidence?: number;
}

interface RawTranscript {
  id: string;
  status: "completed" | "error";
  text: string;      // Full transcript as single string
  words: TranscriptWord[];
  language_code?: string;
  utterances?: TranscriptUtterance[];
}
```

9.3 Summarised Transcript

Stored in: `/storage/transcriptionSummaries/<id>.json`

Produced by `transcriptionSummarizer2.js`. Word-level tokens are grouped into complete sentences per speaker.

```
interface SummarisedSegment {
  speaker: string; // Speaker label, e.g. "A"
  sentence: string; // Reconstructed sentence text
  start: number; // Segment start time, milliseconds
  end: number; // Segment end time, milliseconds
}

type SummarisedTranscript = SummarisedSegment[];
```

9.4 Speaker Audio Snippet Map

Stored in memory; audio files written to `/storage/audio/speakerSnippets/`.

```
interface SpeakerSnippetMap {
  [speakerKey: string]: { // e.g. "speakerA"
    src: string; // Absolute path to WAV snippet file
    name: string; // Same as speakerKey
  };
}
```

9.5 Pipeline Input Payload

Passed via IPC from renderer to main process when the user initiates processing.

```
interface PipelinePayload {
  video: {
    module: string;
    inputVideoPath: string;
  };
  transcription: {
    module: string;
  };
  document: {
    module: string;
    type: string;
    outputType: "pdf" | "docx" | "html" | "txt";
  };
}
```

9.6 Module Registry Entry

Each entry in the global `mapFunctions` Map conforms to:

```
interface ModuleDefinition {
  name: string;
  type: string;
  displayname: string;
  description: string;
  audioformat?: string;
  function: (parameter: unknown) => Promise<unknown>;
}
```

9.7 Document Type Template

Stored in: `/storage/documentType/<name>.txt`

Plain text files containing LLM prompt instructions. No structured schema; content is author-defined. Templates are prepended to the summarised transcript before submission to the LLM.

10. Security Considerations

10.1 Secret Management

Threat: API key exposure in source code or version control.

Mitigations in place:

- `.env` is listed in `.gitignore` and is never committed.
- API keys are loaded at runtime via `dotenv` or the keyserver, not hard-coded.
- Keys are stored only in `process.env` within the main process.
- The preload script's `contextBridge` exposes no key values to the renderer process.
- The GitLab CI/CD pipeline injects keys via protected CI/CD variables, not plain-text files.

Residual risk: If a user manually adds API keys to a tracked source file, they will be committed. Developer education and pre-commit hook enforcement (e.g., `git-secrets`) are recommended mitigations.

10.2 Electron Security Model

Control	Status	Notes
<code>nodeIntegration: false</code>	Enabled	Renderer has no direct Node.js access
<code>contextIsolation: true</code>	Enabled	Renderer and preload run in separate contexts
<code>sandbox</code> (preload)	Enabled	Limits preload capabilities
<code>webSecurity</code>	Default (enabled)	Standard Chromium web security applies
Content Security Policy	Not explicitly set	Recommended for production hardening

Recommendation: Add an explicit Content Security Policy header to the `index.html` file to prevent XSS attacks and restrict resource loading origins.

10.3 Data Handling

Local data: Transcripts, audio snippets, and generated documents are stored in the local `/storage/` directory. These files contain the full text of the processed meeting or lecture.

Implications:

- Files persist after the session ends and are not automatically purged.
- On shared machines, other users with file system access can read these artefacts.
- Organisations handling sensitive content (e.g., confidential meetings) should restrict operating system file permissions on the V2D installation directory.
- A storage clean-up mechanism (scheduled or on-demand) is recommended for future versions.

Cloud data: When using cloud transcription (AssemblyAI) or cloud LLMs (Gemini, SAIA), audio and transcript content is transmitted to external services. Users should review the data retention and privacy policies of each provider before processing sensitive content.

10.4 Network Security

- All external API calls use HTTPS.
- The keyserver endpoint (`keyserver.dommommy.xyz:443`) uses TLS.
- No inbound network ports are opened by the application in normal operation.
- The API Skeleton (Express server) would open port 3000 if started; this is not part of the default startup.

10.5 Input Validation

- File path inputs are passed to FFmpeg via the `fluent-ffmpeg` library, which provides a layer of sanitisation.
- Document type template names are used to construct file paths; input from the UI should be validated to prevent path traversal (e.g., filenames containing `../`). This is an identified hardening item.

10.6 Risk Assessment Summary

Risk	Likelihood	Impact	Mitigation Status
API key committed to version control	Low	High	Mitigated (<code>.gitignore</code> , CI variables)
Sensitive meeting content transmitted to cloud	Medium	High	Documented; user decision
Local artefact exposure on shared machines	Medium	Medium	Partially mitigated (user responsibility)
Path traversal in template file naming	Low	Medium	Not yet mitigated – requires input sanitisation

Risk in generated HTML document preview	Likelihood	Impact	Mitigation Status
	Medium	Medium	Partial: mitigated by Electron sandbox
Keyserver unavailability blocking startup	Medium	Medium	Fallback: local <code>.env</code>

11. Performance & Scalability Notes

11.1 Performance Characteristics

Stage	Bottleneck	Typical Duration (30-min recording)
Audio extraction	Disk I/O + FFmpeg CPU	10–30 seconds
Cloud transcription (AssemblyAI)	Network + provider queue	2–5 minutes
Local transcription (Whisper)	CPU / memory	5–20 minutes
Transcript summarisation	CPU (JavaScript)	< 5 seconds
LLM document generation	Network + provider latency	20–90 seconds
Document conversion (PDF)	Puppeteer / Chromium	5–15 seconds
Speaker snippet extraction	Disk I/O + FFmpeg	10–30 seconds

Total (cloud, typical): 3–8 minutes for a 30-minute recording.

Total (local Whisper): 8–25 minutes depending on hardware.

11.2 Memory Consumption

- Electron base overhead: ~150–300 MB RAM.
- Puppeteer (PDF generation): additional ~200–400 MB during conversion.
- Long transcripts loaded into memory as JSON: typically < 10 MB.
- Speaker audio snippets: 1–5 MB each; total depends on speaker count.

Peak memory usage for a typical session: 500–800 MB. Ensure the host machine has at least 4 GB RAM available for comfortable operation.

11.3 Disk Space Requirements

Artefact Type	Typical Size per Session
Extracted audio (WAV 16kHz mono, 30 min)	~50 MB
Raw transcript JSON	0.5–2 MB
Summarised transcript JSON	0.1–0.5 MB
Speaker audio snippets (WAV)	0.5–5 MB total
Generated HTML document	10–100 KB
PDF export	50–500 KB
DOCX export	50–200 KB

Accumulated storage can grow significantly over time if artefacts are not periodically cleared.

11.4 Scalability Constraints

The current architecture is **single-user, single-session**. It is not designed for concurrent pipeline executions and has the following inherent constraints:

- Only one pipeline job can run at a time within a single Electron instance.
- The `mapFunctions` global Map and `storage/` directory are shared across all sessions of the same instance.
- The application does not implement job queuing.

For multi-user or server-side deployment, the REST API Skeleton provides a foundation, but significant development is required to add persistence, concurrency control, and job management.

11.5 Operational Impact Analysis

Scenario	Impact
AssemblyAI API outage	Transcription unavailable; local Whisper fallback is available
SAIA platform outage	Qwen3 and GPT modules unavailable; Gemini fallback is available
Google Gemini API outage	Gemini module unavailable; SAIA-hosted models are available
Keyserver outage	Application cannot retrieve keys; local <code>.env</code> fallback required
Puppeteer Chromium failure	PDF export unavailable; HTML, DOCX, and TXT exports remain functional
FFmpeg binary failure	Audio extraction fails; entire pipeline blocked

12. Deployment Guide

12.1 Local Development Deployment

Follow [Section 5](#) exactly. This is the standard and only fully supported deployment mode.

12.2 Electron Package Build (Distribution)

V2D can be packaged into a distributable executable using `electron-builder` or `electron-forge`. Neither is currently configured in `package.json`; the following steps describe the integration procedure.

Install a build tool:

```
npm install --save-dev electron-builder
```

Add build configuration to `package.json`:

```
{
  "build": {
    "appId": "com.your-org.video2document",
    "productName": "Video2Document",
    "directories": { "output": "dist" },
    "files": ["main.js", "requires.js", "electron/", "services/", "storage/documentType/"],
    "mac": { "target": "dmg" },
    "win": { "target": "nsis" },
    "linux": { "target": "AppImage" }
  }
}
```

Build:

```
npx electron-builder build --mac --win --linux
```

Important: Do not bundle `.env` files or API keys into the distribution package. The keyserver mechanism or user-provided `.env` files should be used in distributed builds.

12.3 GitLab CI/CD Pipeline

The `.gitlab-ci.yml` file defines a single `test` stage:

```
image: node:latest

stages:
  - test

job-test:
  stage: test
  script:
    - npm install
    - echo "ASSEMBLYAI_API_KEY=$apikey_assembly" > .env
    - echo "GOOGLE_API_KEY=$apikey_gemini" >> .env
    - echo "SAIA_API_KEY=$apikey_saia" >> .env
    - npm test
```

CI/CD variables required (set in GitLab project settings → CI/CD → Variables):

Variable	Description
apikey_assembly	AssemblyAI API key
apikey_gemini	Google Gemini API key
apikey_saia	SAIA platform API key

Mark all variables as **Protected** and **Masked** in GitLab settings to prevent exposure in job logs.

12.4 Adding the Application to a New System

For end-user distribution without requiring source code access:

1. Build the Electron package per Section 12.2.
2. Distribute the installer (*.dmg, *.exe, or *.AppImage).
3. Instruct users to create a .env file in the application's user data directory, or configure keyserver credentials.

Note: The storage/ directory must be writable at runtime. Electron packages typically store user data in the OS-standard application data directory. Path configuration may need adjustment for packaged builds.

12.5 Server-Side Deployment (Future)

The API Skeleton (/API-Skeleton/) provides an Express server template for eventual server-side deployment. To productionise:

1. Implement the stub endpoints in video_Router.js and document_Router.js.
2. Add a persistent database (e.g., PostgreSQL with an ORM).
3. Add authentication middleware.
4. Run behind a reverse proxy (nginx, Caddy) with TLS termination.
5. Containerise with Docker for consistent deployments.

13. Testing Strategy

13.1 Overview

V2D employs Mocha as the test runner. Tests are located in /test/unit/test.js (comprehensive module tests) and /test/integration/ (service-specific integration tests).

13.2 Test Suite Structure

Suite	File	Test Count	Coverage Area
Audio Extraction	test/unit/test.js	4	FFmpeg extraction, format variants, error cases
Audio Transcription	test/unit/test.js	2	AssemblyAI integration, invalid input handling
Transcript			

Summarisation Suite	test/unit/test.js	4	Summariser v1 and v2, error cases
LLM Document Generation	test/unit/test.js	Test Count 6	Coverage Area ChatGPT, Gemini, Qwen3, missing template error
Audio Snippets	test/unit/test.js	3	Speaker snippet extraction, error handling
AssemblyAI Integration	test/integration/assembly.test.js	Variable	End-to-end transcription
Summariser Integration	test/integration/transcriptionSummarizerTest.js	Variable	Summariser with real transcript
Gemini Integration	test/integration/gemini/test-gemini.js	Variable	End-to-end Gemini generation

13.3 Running Tests

Full test suite:

```
npm test
```

This executes: `mocha ./test/unit/test.js`

Individual integration tests:

```
node test/integration/assembly.test.js
node test/integration/gemini/test-gemini.js
```

13.4 Test Timeouts

Category	Timeout
Audio extraction	3,000 ms
Transcription (cloud)	120,000 ms
LLM generation	120,000 ms
Document conversion	30,000 ms

13.5 Test Environment Requirements

All tests that invoke external APIs require valid credentials in `.env` or via the keyserver. Tests will fail with authentication errors if credentials are absent.

CI/CD injects credentials as GitLab CI variables (see [Section 12.3](#)).

13.6 Coverage Gaps

Area	Status	Notes
UI / Renderer process	Not tested	No UI automation tests exist
IPC channel handling	Not tested	No mock IPC tests
Local Whisper transcription	Incomplete	Module is TypeScript, partially implemented
Document conversion	Not tested independently	Covered indirectly by LLM tests
Speaker name replacement	Not tested	<code>replaceSpeaker.js</code> lacks unit tests
API Skeleton endpoints	Not tested	Skeleton only

13.7 Recommended Additional Tests

- 1. Unit tests for `replaceSpeaker.js` with edge cases (names with special characters, multiple occurrences).
- 2. Unit tests for `transcriptionSummarizer2.js` with synthetic transcript JSON.
- 3. Mocked IPC tests for the main process handlers in `main.js`.
- 4. End-to-end UI tests using a framework such as Playwright for Electron.
- 5. Input validation tests for file path injection and template name sanitisation.

14. Maintenance & Support Plan

14.1 Dependency Management

Activity	Frequency	Responsibility
Review <code>npm outdated</code> for security advisories	Monthly	Developer/Admin
Update Electron to latest stable version	Per major release	Developer
Update Puppeteer Chromium dependency	Per major release	Developer
Review external API provider changelog (AssemblyAI, Gemini, SAIA)	Monthly	Developer
Rotate API keys	Per organisation policy	Administrator

Run `npm audit` regularly and address high-severity advisories promptly.

14.2 Storage Maintenance

The `/storage/` directory accumulates artefacts indefinitely. A maintenance schedule is recommended:

Artefact	Retention Recommendation
Raw audio extracts	Delete after session or after 7 days
Raw transcripts (JSON)	Retain for 30 days for debugging
Summarised transcripts	Retain for 30 days
Speaker audio snippets	Delete after session
Generated HTML documents	Retain until user exports final document

A clean-up script or UI option to purge artefacts is recommended for future versions.

14.3 Adding a New Module

To add a new LLM or transcription provider:

- 1. Create a directory under `services/modules/<module-category>/`.
- 2. Create a `.js` file following the module interface template.
- 3. Export the module object with a unique `name` field.
- 4. Add the required API key to `.env` and to the keyserver.
- 5. Restart the application — the module is auto-discovered and appears in UI dropdowns.
- 6. Write unit tests in `test/unit/test.js` following existing patterns.

No changes to `main.js`, the module handler, or the UI are required.

14.4 Monitoring and Logging

V2D currently logs to `console.log` / `console.error` in the main process. In a production distribution, these logs are accessible via:

- **macOS:** `~/Library/Logs/<AppName>/main.log`
- **Windows:** `%APPDATA%\<AppName>\logs\main.log`
- **Linux:** `~/.config/<AppName>/logs/main.log`

(Exact paths depend on the `app.getPath('userData')` value set in the packaged application.)

Structured logging (e.g., using the `winston` library) and log rotation are recommended for operational deployments.

14.5 Support Channels

Channel	Use
GitLab Issues	Bug reports, feature requests
GitLab CI/CD	Automated regression testing
README.md	End-user setup guidance
documentation.md	Developer architectural reference
This document	Comprehensive operational reference

15. Limitations & Known Issues

15.1 Functional Limitations

ID	Limitation	Affected Feature	Workaround
LIM-01	Local Whisper transcription is not fully implemented	Transcription	Use AssemblyAI
LIM-02	REST API is a non-integrated skeleton	API access	Use Electron IPC (desktop app)
LIM-03	No database persistence; artefacts are flat files	Session history	Manual file management
LIM-04	Single concurrent pipeline only	Throughput	Process files sequentially
LIM-05	Speaker identification is label-based (A, B, C), not biometric	Speaker accuracy	Manual name assignment in Step 5
LIM-06	Custom template names are not validated for path traversal	Security	Avoid special characters in template names
LIM-07	Storage paths are hard-coded in module source files	Portability	Edit source files to relocate storage
LIM-08	No automatic artefact clean-up	Disk usage	Manually clear <code>/storage/</code> directories
LIM-09	Packaged Electron distribution not yet configured	Distribution	Run from source
LIM-10	No Content Security Policy defined	Security	Manual CSP header addition

15.2 Known Issues

ID	Issue	Severity	Status
KI-01	Whisper local module (<code>whisperLocal.ts</code>) is incomplete; invoking it will fail	High	Open
KI-02	Long transcripts may exceed LLM context window limits, truncating output	Medium	Open
KI-03	Puppeteer may fail on systems with strict sandbox policies (e.g., Docker without <code>--no-sandbox</code>)	Medium	Workaround: add <code>--no-sandbox</code> flag
KI-04	Speaker diarisation accuracy decreases with more than 4 concurrent speakers	Medium	Open
KI-05	Mixed JS/TS codebase causes inconsistent linting and type coverage	Low	Open

KI-06	New	Validation that input file is readable/non-corrupt before pipeline start	Severity	Status

16. Future Improvements / Roadmap

The following improvements are identified based on current limitations, security analysis, and operational requirements. Items are grouped by priority.

16.1 Priority 1 — Critical Gaps

Item	Description
Complete Whisper local transcription	Fully implement <code>whisperLocal.ts</code> for offline, privacy-preserving operation
Input path validation	Sanitise all user-provided file paths and template names to prevent traversal attacks
Content Security Policy	Add explicit CSP headers to <code>index.html</code>
Artefact lifecycle management	Implement automatic or on-demand clean-up of <code>/storage/</code> artefacts

16.2 Priority 2 — Important Improvements

Item	Description
Electron distribution packaging	Configure <code>electron-builder</code> for one-click installer distribution
Structured logging	Replace <code>console.log</code> with a structured logger (<code>winston</code>) with log rotation
Context-length handling	Split long transcripts into chunks if they approach LLM token limits
TypeScript migration	Progressively convert all modules to TypeScript for type safety and tooling
Coverage expansion	Add UI automation tests (Playwright for Electron), IPC mock tests, and replacement module tests

16.3 Priority 3 — Feature Enhancements

Item	Description
REST API integration	Integrate the API Skeleton into the production application for headless/server operation
Session history	Implement a lightweight SQLite database to persist session metadata and enable document retrieval
Multi-language transcription	Explicitly expose language selection to the user for improved transcription accuracy
Concurrent pipeline support	Add a job queue to allow multiple files to be processed sequentially without UI blocking
Streaming progress	Display detailed sub-step progress (e.g., transcription percentage) rather than four discrete stages

16.4 Priority 4 — Strategic / Long-Term

Item	Description
Web / SaaS mode	Deploy V2D as a web application for browser access without installation
Speaker biometrics	Integrate voice-print identification to automatically name known speakers
Custom LLM hosting	Support self-hosted LLM endpoints (Ollama, LM Studio) for full on-premise operation

Document templates marketplace	Allow sharing and importing community-created document type templates
Enterprise SSO	Integrate with SAML/OIDC for organisational user management

17. Glossary

Term	Definition
API (Application Programming Interface)	A defined interface that allows software components to communicate with each other.
AssemblyAI	A cloud-based AI transcription service that converts audio to text with speaker diarisation support.
Artefact	An intermediate file produced and consumed by a pipeline stage (e.g., extracted audio, raw transcript, HTML document).
Context Bridge	Electron's mechanism (<code>contextBridge</code>) for safely exposing main-process APIs to the renderer without granting full Node.js access.
Diarisation (Speaker Diarisation)	The process of partitioning a transcript by speaker identity, labelling segments as "Speaker A", "Speaker B", etc.
Dotenv	A utility that loads environment variables from a <code>.env</code> file into <code>process.env</code> at application startup.
Electron	An open-source framework by GitHub/Microsoft for building cross-platform desktop applications using web technologies (HTML, CSS, JavaScript).
FFmpeg	A widely used open-source multimedia processing library supporting encoding, decoding, and conversion of audio and video.
fluent-ffmpeg	A Node.js library providing a fluent, chainable API for invoking FFmpeg operations.
IPC (Inter-Process Communication)	The mechanism Electron uses to pass messages between the main process (Node.js) and the renderer process (Chromium).
Keyserver	A custom internal service (<code>keyserver.dommymommy.xyz</code>) that distributes API keys to authenticated V2D instances at runtime.
LLM (Large Language Model)	A class of AI model trained on large text corpora, capable of generating coherent text given a prompt. Examples: GPT, Gemini, Qwen3.
mapFunctions	The global JavaScript <code>Map</code> object in V2D's main process that stores all registered module objects, keyed by module name.
Module	A self-contained unit of functionality in V2D that conforms to the standard module interface and is auto-discovered at startup.
Module Handler	The utility (<code>module-handler.js</code>) responsible for dynamically loading module files and populating <code>mapFunctions</code> .
Mocha	A JavaScript test framework for Node.js that provides test suite structure, assertions, and reporting.
npm	Node Package Manager — the default package registry and dependency management tool for Node.js.
Preload Script	A Node.js script (<code>preload.js</code>) that runs in a privileged context in Electron and bridges the renderer and main processes via <code>contextBridge</code> .
Prompt Template	A plain-text file containing LLM instructions that define the structure and style of the generated document.
Puppeteer	A Node.js library that controls a headless Chromium browser, used in V2D for HTML-to-PDF conversion.
Renderer Process	The Chromium browser process in Electron that renders the application UI. It runs in a sandboxed environment without direct Node.js access.
	An academic cloud computing platform that hosts selected LLM models (including Qwen3 and GPT

SAIA Term	Definition
Speaker Placeholder	varjants) accessible via API. A temporary token (e.g., speakerA , speakerB) inserted by the LLM generation step and later replaced with the speaker's real name.
Transcript	The textual output of the transcription step, containing the spoken words with associated speaker labels and timestamps.
V2D	Abbreviation for Video2Document – the name of this software system.
Whisper / Whisper.cpp	An open-source speech recognition model by OpenAI. Whisper.cpp is a C++ port that runs locally without network access.

End of Document

Document Control

Field	Value
Title	Video2Document – Software Documentation Package
Version	1.0.0
Standard	ISO/IEC/IEEE 26514:2022
Generated	2026-02-22
Project	video2document (WISE 2025/2026)
Status	Release Candidate