Voice-to-Notes. Technical Design Report

1. Overview

Voice-to-Notes is a web application that converts voice recordings into structured text notes.

The system automates the full pipeline:

1. Record or upload an audio file in the browser
2. Transcribe speech to text
3. Generate a short summary
4. Transform the raw transcript into a structured Markdown document
5. Store the result as a note attached to the authenticated user

The main target is **English transcription**. Other languages were used only for internal tests.

---

2. Problem and Goal

2.1 Problem

Voice notes are convenient for capturing ideas quickly, but hard to use afterwards:

- no full text representation;
- no structure (headings, lists, key points);
- difficult to scan or search;
- time-consuming manual transcription and formatting.

2.2 Goal

The goal of the project is to provide a simple way to get:

1. A full **raw transcript** of the voice recording
2. A short **summary** (2–3 sentences)
3. A **structured Markdown note** with headings, bullet points and highlighted key ideas

All results are saved and can be viewed and managed later.

---

3. User Flow

1. The user opens the application and signs in with email and password.
2. After authentication, the user goes to the notes area.
3. On the main notes screen, the user can:
   - start recording audio via the browser, or
   - upload an existing audio file.
4. During recording:
   - the browser asks for microphone access;
   - an ASCII-style waveform and a timer are shown.
5. When the user stops recording, the audio is sent for processing.
6. The backend:
   - transcribes the audio with Whisper,
   - generates a summary,
   - generates a structured Markdown version of the text.

7. A new note is created in the database.
8. The user is redirected to the note details page, where they can see:
    - title,
    - summary,
    - raw transcript,
    - structured Markdown content.
9. The user can:
    - view a list of their notes,
    - open a single note,
    - delete a note.

Editing is currently possible only via the API (PUT endpoint), not via a dedicated UI form.

---

4. Architecture Overview

The system is split into two main parts: **frontend** (Next.js) and **backend** (FastAPI in Python).

4.1 Frontend (Next.js)

Responsibilities:

- Email/password authentication (using NextAuth);
- Protected routes for notes;
- Audio recording via `MediaRecorder` API;
- ASCII waveform visualization via Web Audio API;
- UI for:
    - recording and uploading audio,
    - listing notes,
    - viewing a single note,
    - deleting notes;
- Communication with the backend via Next.js API routes;
- Persistence of notes via Prisma and PostgreSQL.

Main screens:

- `/login` – authentication page;
- `/notes` – main recording screen (RecordPanel, ASCII waveform, status text);
- `/notes/list` – list of notes for the current user;
- `/notes/[id]` – single note view (summary, transcript, structured content, delete button).

Routes under a protected segment are guarded so that non-authenticated users are redirected to `/login`.

4.2 Backend (FastAPI, Python)

Responsibilities:

- Receiving audio files from the Next.js API;
- Running the speech-to-text model (Whisper);
- Calling OpenRouter LLM for summarization;
- Calling OpenRouter LLM for structuring;
- Returning JSON back to the frontend with:
    - transcript,
    - summary,
    - structured Markdown.

Main endpoints (FastAPI):

- `GET /health` – health check;
- `POST /transcribe` – transcription only;
- `POST /summary` – summarization only;
- `POST /structure` – structuring only;
- `POST /process-audio` – full pipeline: transcription → summary → structuring.

Processing is implemented as a synchronous pipeline inside `POST /process-audio`.

4.3 Data Storage (PostgreSQL + Prisma)

Main tables:

`User`

- `id`
- `email` (unique)
- `password` (hashed)
- `emailVerified` (optional)
- `createdAt`
- `updatedAt`

Relations to:

- `notes`
- NextAuth `accounts`
- NextAuth `sessions`

`Note`

- `id`
- `userId` (foreign key to `User`)
- `title` (optional; usually derived from summary)
- `summary` (optional)
- `transcript` (optional; raw ASR output)
- `contentMd` (optional; structured Markdown)
- `audioUrl` (optional; currently not used)
- `createdAt`
- `updatedAt`

NextAuth tables (`Account`, `Session`, `VerificationToken`) are also present for authentication, but not directly used in the note logic.

Stored data:

- Raw transcript (`transcript`),
- Summary (`summary`),
- Structured Markdown (`contentMd`),
- Metadata (timestamps, user linkage).

Audio files are **not** stored; only a placeholder `audioUrl` field exists in the schema but is not used in the current version.

---

5. GenAI Pipeline

The GenAI pipeline consists of three steps applied to the audio recording.

5.1 ASR: Speech-to-Text (Whisper)

- Model: **OpenAI Whisper**, used via the `whisper` Python library.
- Model size: configurable via `WHISPER_MODEL_SIZE` (default: `base`).

Processing:

1. The backend saves the uploaded audio to a temporary file.
2. Whisper is called with this file:
   `result = whisper_model.transcribe(tmp_file_path, fp16=False, language=language)`
3. The transcript is extracted from `result["text"]`.
4. The temporary file is removed.
5. The transcript is returned as a plain string and later stored in `Note.transcript`.

Whisper is chosen because:

- it is open-source;
- it works locally (no external ASR API);
- it supports multiple languages;
- it provides good quality for English speech.

There is no additional post-processing of the transcript in the current version (fillers, hesitations, etc. are not removed automatically).

5.2 Summarization (LLM via OpenRouter)

- Provider: **OpenRouter API**
- Default model: `openai/gpt-4o-mini`.

Prompt (simplified):

- System prompt: "You are a helpful assistant that creates concise summaries of voice transcripts."
- User prompt:

> "Please provide a brief 2-3 sentence summary of the following transcript.
> Focus on the main ideas and key points:
> [transcript]"

Objective:

- 2–3 sentences;
- main ideas only;
- no extra details.

The result is stored in `Note.summary`.

5.3 Structuring (LLM via OpenRouter)

The same model (`openai/gpt-4o-mini`) is used for structuring.

Prompt (simplified):

- System prompt:

> "You are a helpful assistant that transforms unstructured voice transcripts into well-organized markdown documents."

- User prompt:

> "Transform the following voice transcript into a well-structured markdown document.
> Requirements:
>
> - Organize content by topics with clear headings (##)
> - Use bullet points for lists
> - Use **bold** for key ideas and important points
> - Maintain the original meaning and information
> - Make it easy to read and scan
>
> Transcript:
> [transcript]"

Output:

- Markdown-like text with:
  - `##` headings,
  - `-` bullet lists,
  - `**bold**` highlights.

The result is stored in `Note.contentMd`. No additional validation or post-processing is performed.

---

6. Implementation Highlights

6.1 Audio Capture and Visualization

- Recording: `navigator.mediaDevices.getUserMedia({ audio: true })` + `MediaRecorder`.
- Recorded chunks are collected into a single `Blob` when recording stops.
- The same UI also supports uploading audio files via `<input type="file" accept="audio/*">`.

ASCII waveform:

- Implemented via Web Audio API (`AudioContext`, `AnalyserNode`).
- Frequency data is sampled and mapped to a fixed grid.
- The grid is rendered using ASCII-like characters to mimic a waveform.

6.2 API Integration

Frontend → Next.js API:

- A `FormData` object is created and the audio `Blob` is appended.
- A POST request is sent to `/api/transcribe`.

Next.js → FastAPI:

- The Next.js API route forwards the file to the Python backend endpoint `/process-audio`.
- The backend returns JSON: `{ transcript, summary, structured_content }`.

Next.js → Database:

- The API route uses Prisma to create a new `Note` with:
  - `userId`,
  - `title` (if defined),

- `summary`,
- `transcript`,
- `contentMd`.

- The new note ID is used to redirect the user to `/notes/[id]`.

---

7. Evaluation (Qualitative)

7.1 ASR

Quality observations (based on manual checks):

- Good results for clear English speech;
- Typical issues:
  - proper names,
  - technical terms,
  - noisy background,
  - very fast speech.

No automatic metrics (such as WER) are implemented in the project.

7.2 Summarization

- Usually produces a correct and concise summary of the main idea;
- Sometimes too generic for long or complex recordings.

7.3 Structuring

- Produces readable Markdown with headings, lists and bold highlights;
- The structure is generally useful for note-taking;
- For weak or very fragmented transcripts, the structure can become too formal or slightly misgrouped.

Evaluation is purely manual; no automated scoring (e.g. ROUGE) is implemented.

---

8. Limitations

Current main limitations:

- Audio files are not stored; only text is saved.
- Processing is synchronous; no task queue or background workers.
- No UI for editing notes (only API-level update).
- No search, tags or categories for notes.
- No versioning of notes.
- No streaming transcription; processing starts only after recording stops.
- No automatic quality metrics for ASR or LLM outputs.

---

9. Future Improvements

Potential next steps:

- Store audio files (`audioUrl` field backed by actual storage).
- Add a task queue for asynchronous processing and better scalability.
- Provide an in-browser editor for notes (Markdown editor).

- Implement search across notes.
- Add export options (e.g. Markdown, PDF, TXT).
- Add basic quality metrics and a feedback mechanism for users.

---

10. Conclusion

Voice-to-Notes implements a complete pipeline from voice recording to structured text note:

- Frontend: Next.js application with audio recording, waveform visualization, authentication and note management.
- Backend: FastAPI service using Whisper for transcription and an OpenRouter LLM (GPT-4o-mini) for summarization and Markdown structuring.
- Storage: PostgreSQL with Prisma for users and notes.

The project achieves its main goal: turning raw voice input into a readable, structured text note with minimal user effort, and provides a solid base for further extensions such as editing, search, export and more advanced evaluation.