# Content Aware Note Taker (CANT):Architecture Write-up

**Sanath Swaroop Mulky**
Graduate Student, MS Data Science
Rochester Institute of Technology
sm6184@rit.edu

**Sai Akash Avunoori**
Graduate Student, MS Data Science
Rochester Institute of Technology
sa6440@rit.edu

**Ashique R. KhudaBukhsh**
Assistant Professor, Department of Software Engineering
Rochester Institute of Technology
axkvse@rit.edu

## 1 Introduction

CANT (Content Aware Note Taker) is an innovative system designed to enhance accessibility and productivity in educational and professional settings. By integrating real-time speech-to-text processing, context-aware corrections, and iterative summarization, CANT ensures accurate and structured note generation.

The system adopts a privacy-first approach with its sandbox mode. Sandbox mode creates a separate Docker container for each user session, ensuring complete data isolation. These containers are configured with a 24-hour timeout, after which the session data is automatically destroyed. This guarantees that user data exists only within the Docker container and is never retained beyond the session.

## 2 Architecture Overview

CANT's architecture seamlessly integrates file processing, speech transcription, and summarization, leveraging Dockerized environments for secure and isolated workflows.

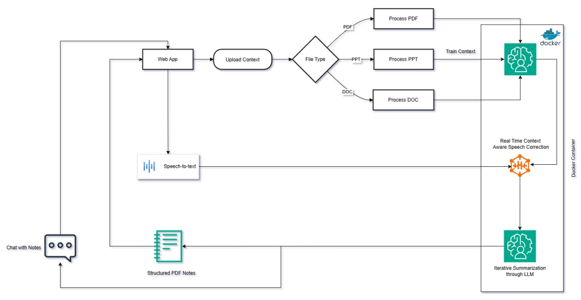

Figure 1: CANT Architecture Diagram Placeholder

### 2.1 Architecture Layers

**Web Application Layer:**

- Provides a dynamic and user-friendly interface for file uploads, transcript interactions, and sandbox configuration.
- Built with React.js for modular and scalable front-end development.

**File Processing Layer:**

- Extracts structured content from PDFs, PPTs, and DOCs using specialized libraries such as PDF.js, Python-pptx, and Docx.

**Speech-to-Text and Correction Layer:**

- Captures real-time speech input or processes uploaded audio files via web-based APIs.
- Refines transcription accuracy through context-aware corrections derived from uploaded files.

**Summarization Layer:**

- Deployed within a Docker container to ensure secure processing without internet access.
- Processes transcripts iteratively to respect token limits while generating cohesive summaries.

**Output Generation Layer:**

- Exports final notes as structured PDFs or provides access through an interactive chatbot.
- Sandboxed sessions ensure complete data isolation and automatic destruction of session data after 24 hours.

## 3 Sandbox Mode

The sandbox mode is a key feature of CANT, prioritizing user privacy by isolating each session in a dedicated Docker container. This mode ensures:

- **Data Isolation:** All session data exists only within the Docker container, preventing external access.

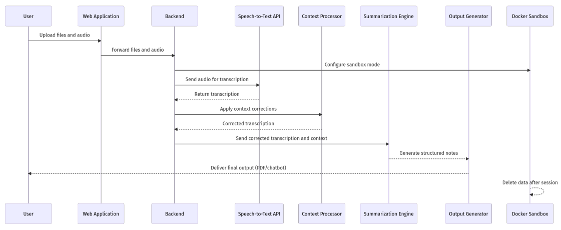

Figure 2: Sequence Diagram for Output Generation Layer

- **Automatic Data Deletion:** Containers have a 24-hour timeout, after which all session data is permanently destroyed.
- **Ephemeral Processing:** Data processed in sandbox mode never persists beyond the container's lifecycle.

## 4 Software Components and Frameworks

**Front-End Framework: React.js**

- Features:
  - File uploads with support for PDFs, PPTs, and DOCs.
  - Real-time progress indicators for processing.
  - Sandbox mode configuration for secure data handling.

**Back-End Framework: Node.js**

- Handles API interactions, communication with Dockerized components, and ephemeral storage for sandboxed sessions.

**File Processing Tools:**

- PDF.js, Python-pptx, and Docx for extracting structured content from uploaded files.

**Speech-to-Text API:**

- Web Speech API or Google Speech-to-Text for real-time transcription.

**Summarization Integration:**

- Operates securely within a Docker container, processing text iteratively while respecting token limits.
- Communicates with the back end through an Nginx-proxied single port.

**Database: MongoDB**

- Stores metadata, parsed files, and transcription data.
- Sandboxed sessions use in-memory storage to ensure no long-term data retention.

## 5 Workflow Description

**Step 1: File Upload and Processing**

- Users upload PDFs, PPTs, or DOCs via the web application.
- Files are parsed and stored temporarily for processing.

**Step 2: Real-Time Speech-to-Text**

- Speech input is captured or uploaded as audio files.
- Transcriptions are processed and stored temporarily.

**Step 3: Context-Aware Speech Correction**

- Uploaded files train the system to recognize domain-specific terminology, refining transcription accuracy.

**Step 4: Iterative Summarization**

- Transcripts are chunked into manageable sections based on token limits.
- Summaries are generated iteratively and combined into cohesive outputs.

**Step 5: Output Generation**

- Notes are exported as PDFs or accessed via chatbot.
- All sandboxed data is deleted after 24 hours or at the end of the session.

## 6 Technical Challenges and Solutions

**Challenge 1: Token Limits in Summarization**

- **Solution:** Transcripts are split into manageable chunks with overlapping sections to preserve context.

**Challenge 2: Secure Sandbox Operations**

- **Solution:** Docker containers ensure session isolation with automatic deletion after 24 hours.

**Challenge 3: Secure Communication**

- **Solution:** Nginx proxies all interactions with Docker containers through a single port.

## 7 Reproducibility

**To replicate CANT:**

1. Install Node.js, React.js, MongoDB, and Docker.
2. Set up Python libraries such as PDF.js, Python-pptx, and Docx.
3. Pull the Docker image for the summarization model.
4. Configure Nginx to proxy communication between the back end and Docker container.
5. Obtain credentials for speech-to-text APIs.
6. Start MongoDB, launch the Docker container, and initialize the backend and frontend services.
7. Test workflows, ensuring both standard and sandboxed sessions operate as expected.