TranscribeFlow: AI-Powered Audio Transcription & Translation Platform

Technical Documentation

Executive Summary

TranscribeFlowis a Flask-basedweb application that leverages state-of-the-art AI models (OpenAI Whisper, Facebook BART, Pyannote) to provide automatic speech recognition (ASR), speaker diarization, text summarization, and multi-language translation capabilities. The platform features a modern web interface with Clerk authentication, trial mode support, and comprehensive audio analysis tools including "Sonic DNA" acoustic feature extraction.

Core Technologies: Python Flask, OpenAI Whisper (ASR), Facebook BART-CNN (Summarization), Pyannote Audio (Speaker Diarization), Deep-Translator (Translation), Clerk (Authentication), Librosa (Audio Analysis)

Key Features: Drag-and-drop file upload, real-time transcription processing, speaker diarization, multi-language translation (100+ languages), AI-powered summarization, audio analytics, user authentication with trial mode, persistent history management

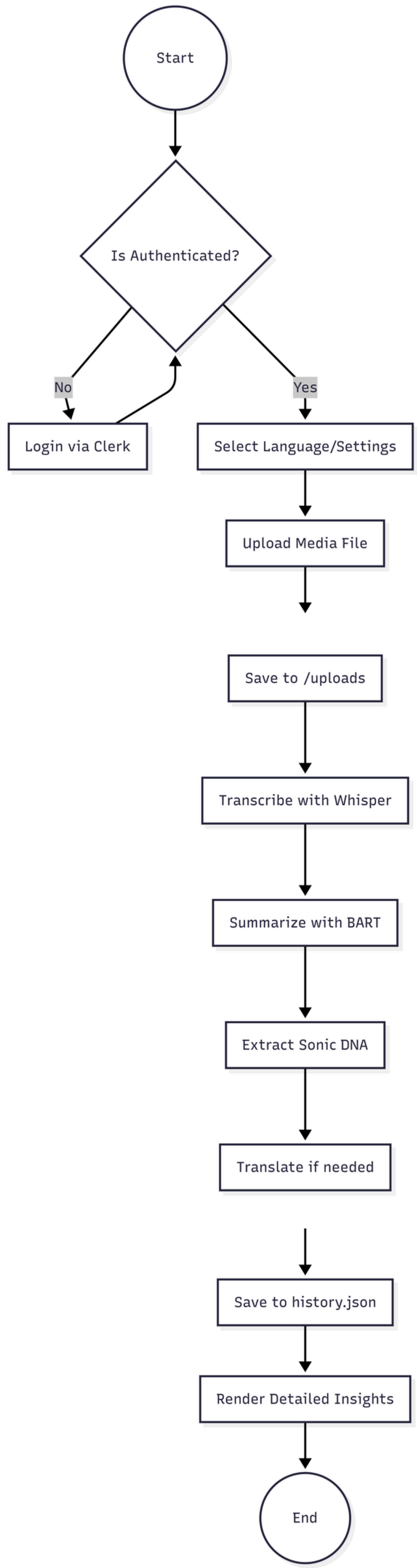
Submitted By: Group III | Infosys Virtual Internship Program

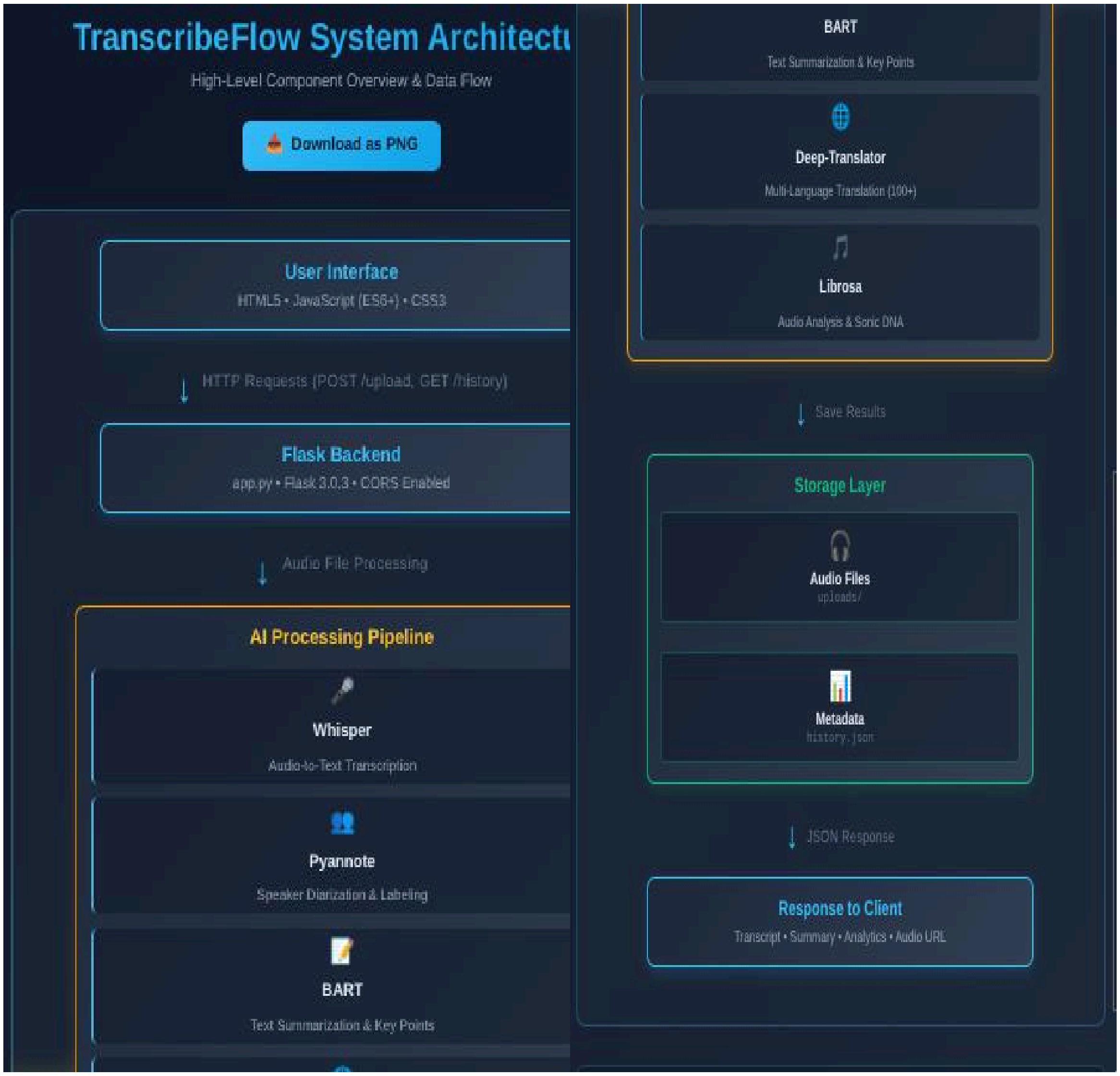
System Architecture

High-Level Overview

[User Interface (HTML/JS/CSS)]

```
       ↓ HTTP Requests
[Flask Backend (app.py)]
       ↓
[AI Processing Pipeline]
   ├── Whisper (Transcription)
   ├── Pyannote (Speaker Diarization)
   ├── BART (Summarization)
   ├── Deep-Translator (Translation)
   └── Librosa (Audio Analysis)
       ↓
[Storage Layer]
   ├── uploads/ (Audio Files)
   └── history.json (Metadata)
       ↓
[Response to Client]
```


Start
Is Authenticated?
No
Yes
Login via Clerk
Select Language/Settings
Upload Media File
Save to /uploads
Transcribe with Whisper
Summarize with BART
Extract Sonic DNA
Translate if needed
Save to history.json
Render Detailed Insights
End



Technology Stack

Layer	Technology	Purpose
Backend Framework	Flask 3.0.3	HTTP server, routing, request handling
Speech Recognition	OpenAI Whisper (tiny model)	Audio-to-text transcription
Speaker Diarization	Pyannote.audio 3.1	Multi-speaker detection and labeling
Summarization	Facebook BART-Large-CNN	Text summarization and key point extraction
Translation	Deep-Translator (Google)	Multi-language translation

Audio Processing	Librosa 0.10.2	Audio feature extraction and analysis
Authentication	Clerk Backend API	User management and session verification
Frontend	HTML5, JavaScript (ES6+), CSS3	User interface and client-side logic
Deep Learning	PyTorch 2.4.1, Transformers 4.45.1	Model inference backend

---

Module Architecture

TranscribeFlow is organized into **4 core modules** with clear separation of concerns:

Module 1: File Upload & Management

Responsibilities: File handling, drag-and-drop interface, upload validation, success/failure messaging

Components:

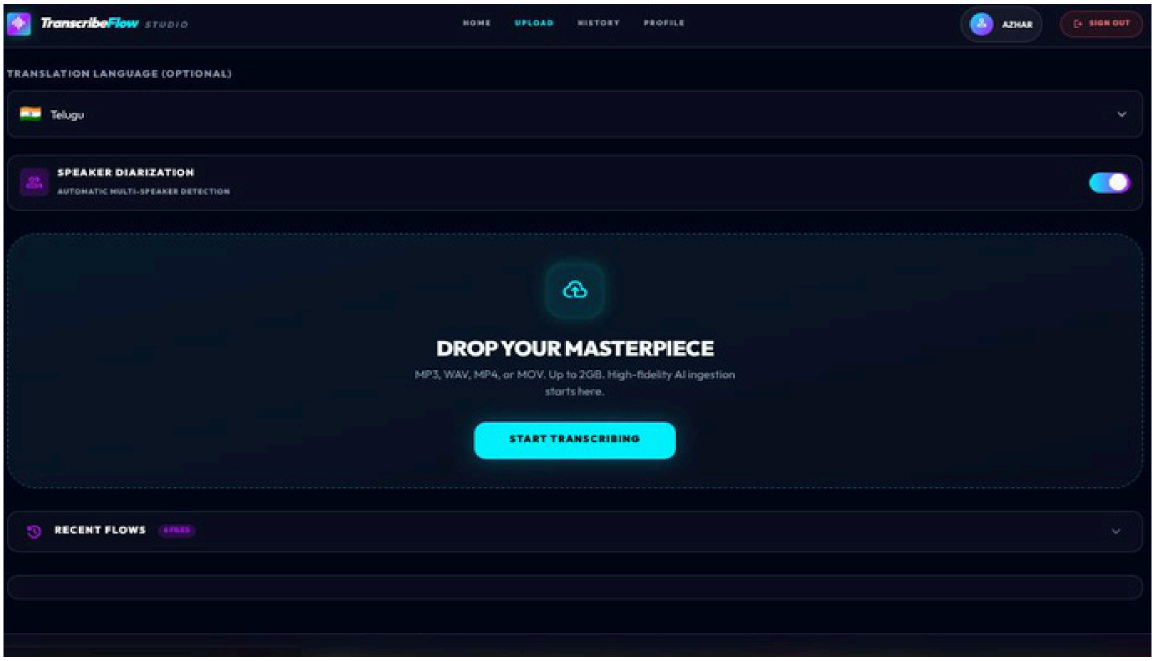
- **Frontend:** `upload.html`, `upload.js`
- **Backend:** `/upload` endpoint in `app.py`
- **Storage:** `uploads/` directory for audio files

Key Features:

- Drag-and-drop file upload with visual feedback
- Supported formats: MP3, WAV, MP4, MOV (up to 2GB)
- Real-time upload progress and status notifications
- File validation and error handling
- Trial mode upload limit enforcement (2 uploads for non-authenticated users)

Data Flow:

```
User Drops File → handleDrop() → uploadFile() → FormData Construction
  → POST /upload → File Saved to uploads/ → Processing Pipeline
  → Success/Error Response → displayResults() / notify.error()
```



Code Snippet (upload.js):

```
function uploadFile(file) {

  //Trial limit check
  if(window.trialManager && !window.trialManager.canUpload()) {
    notify.warning('Free trial limit reached!');
    return;
  }

  const formData = new FormData();

  formData.append('audio', file);
  formData.append('target_lang', targetLang);
  formData.append('enable_diarization', enableDiarization);

  fetch('/upload', {

    method: 'POST',
    headers: { 'Authorization': `Bearer ${token}` },
    body: formData
  })
  .then(response => response.json())
  .then(data => displayResults(data, file.name));
}
```

Module 2: Transcription & Translation

Responsibilities: Audio-to-text conversion, speaker diarization, multi-language translation

Components:

- **Backend:** perform_diarization(), format_transcript_with_speakers(), translate_texts() in app.py

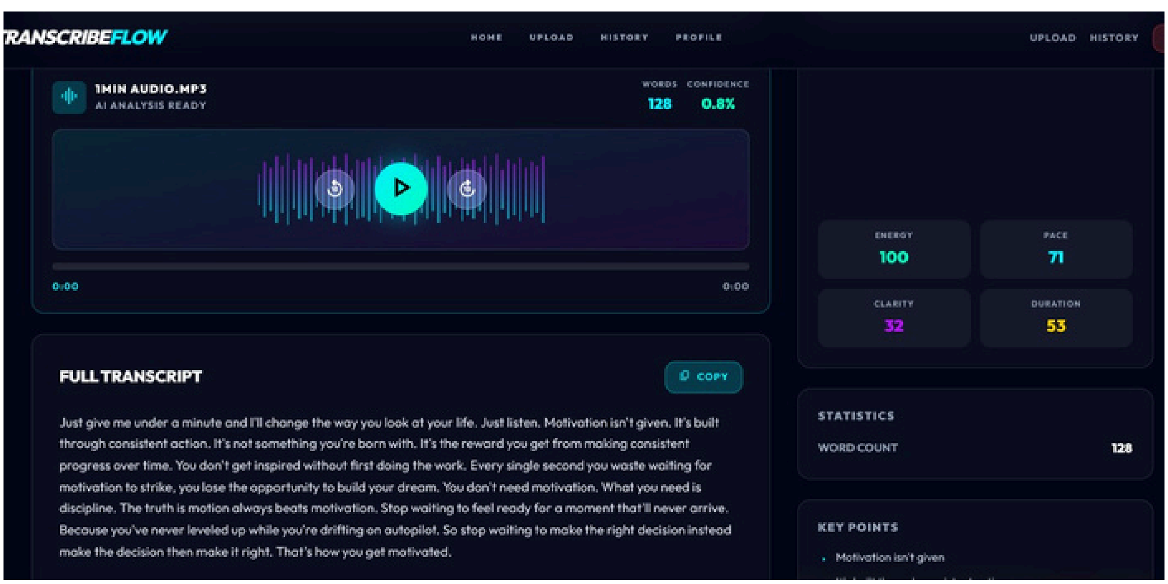
- **AI Models:** Whisper (ASR), Pyannote (diarization), Deep-Translator (translation)
- **Template:** diarization_template.py (implementation reference)

Key Features:

- **Transcription:** OpenAI Whisper "tiny" model for fast, accurate speech-to-text
- **Speaker Diarization:** Automatic detection of multiple speakers with timestamp-aligned labels
- **Translation:** Support for 100+ languages via Google Translate API
- **Confidence Scoring:** Per-segment confidence metrics from Whisper logprobs

Processing Pipeline:

1. **Audio Ingestion:** File saved to uploads/ directory
2. **Whisper Transcription:** asr_model.transcribe(file_path) generates text + segments with timestamps
3. **Speaker Diarization (Optional):**
   - o Load audio with Librosa (bypasses torchcodec issues)
   - o Convert to PyTorch tensor format
   - o Run Pyannote pipeline to detect speaker segments
   - o Match Whisper segments to speaker timestamps
   - o Format transcript with "Speaker 1:", "Speaker 2:" labels
4. **Translation (Optional):** GoogleTranslator translates transcript and summary to target language
5. **Confidence Scoring:** Average logprobs from Whisper segments converted to percentage



Code Snippet (Speaker Diarization):

```
def perform_diarization(audio_file_path):

    # Load audio with librosa (bypasses torchcodec)
    waveform, sample_rate = librosa.load(audio_file_path, sr=16000, mono=True)
    waveform_tensor = torch.from_numpy(waveform).unsqueeze(0)

    # Create audio dict for pyannote
    audio_dict = {'waveform': waveform_tensor, 'sample_rate': sample_rate}
    # Initialize pipeline
    pipeline = Pipeline.from_pretrained("pyannote/speaker-diarization-3.1")
    diarization = pipeline(audio_dict)

    # Extract speaker segments
    speaker_segments = []
    for turn, _, speaker in diarization.itertracks(yield_label=True):

        speaker_segments.append({
            "start": turn.start,
            "end": turn.end,
            "speaker": speaker
        })

    return {"num_speakers": len(set(...)), "segments": speaker_segments}
```

Translation Support:

- **Default:** Original language (no translation)
- **Available Languages:** Arabic, Spanish, French, German, Chinese, Hindi, Japanese, and 100+ more
- **Character Limit:** 4999 characters per translation request (Google API limit)
- **Error Handling:** Falls back to original text if translation fails

---

Module 3: Summarization & Authentication

Responsibilities: AI-powered text summarization, user authentication, trial mode management

Components:

- **Summarization:** Facebook BART-Large-CNN model via Transformers pipeline
- **Authentication:** Clerk API integration for user management
- **Frontend:** `login.html`, `auth.js`, `trial-mode.js`
- **Backend:** `require_auth()` decorator, Clerk session verification

3.1 Summarization Engine

Features:

- Extracts key points from transcripts using Facebook BART-Large-CNN
- Generates bullet points (top 3 sentences)
- Identifies top 5 keywords by frequency
- Minimum 50 words required for summarization

Code Snippet:

```
# Summarization
input_text = transcript[:3000] # First 3000 chars
summary_res = summarizer(input_text, max_length=150, min_length=40, do_sample=False)
summary = summary_res['summary_text']

# Bullet Points
bullet_points = summary.split('. ')[:3]
# Keyword Extraction
words = [w.lower() for w in transcript.split() if len(w) > 5]
word_freq = Counter(words)
keywords = [pair.title() for pair in word_freq.most_common(5)]
```

3.2 Authentication System

Clerk Integration:

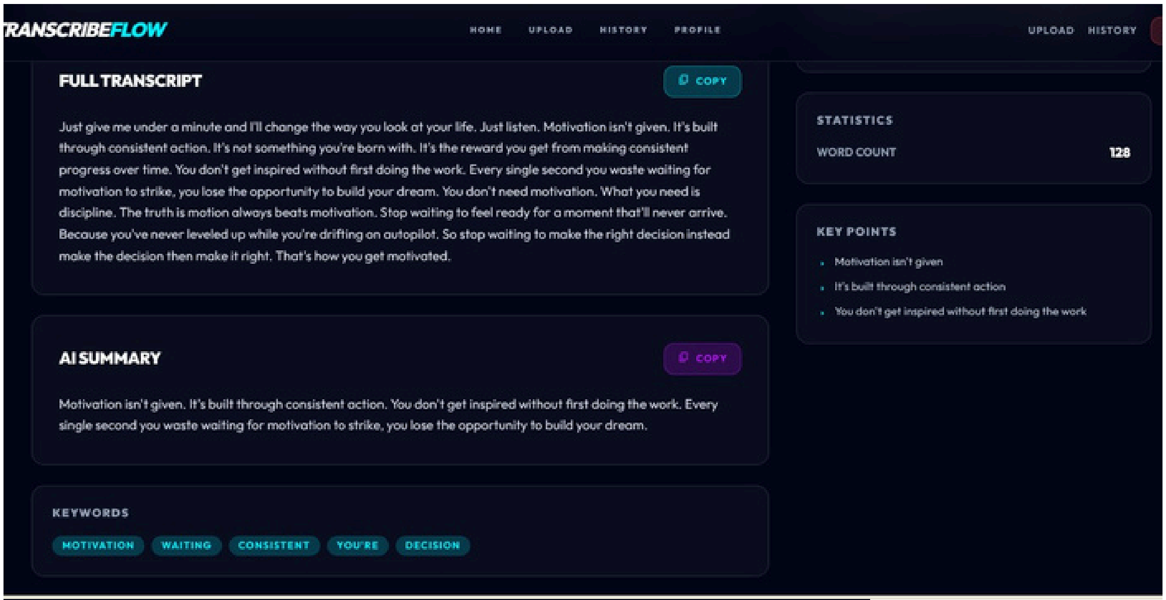
- **Provider:** Clerk Backend API (production-grade authentication)
- **Session Handling:** JWT token verification via `clerk_client.sessions.verify_token()`
- **Token Sources:** Authorization header (`Bearer {token}`) or `__session` cookie
- **Protection:** `@require_auth` decorator for protected routes

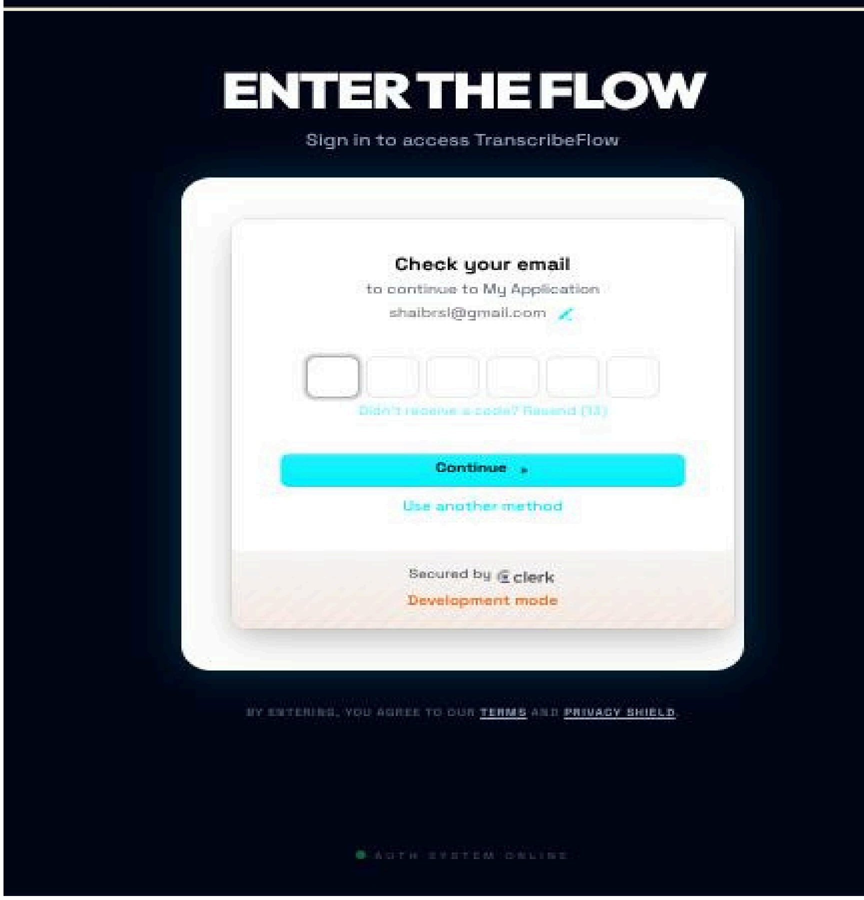
Trial Mode:

- **Limit:** 2 uploads for non-authenticated users
- **Storage:** `localStorage` tracks trial count client-side
- **Enforcement:** `trial-mode.js` checks `canUpload()` before processing
- **UI Feedback:** Displays remaining uploads, shows limit banner when exhausted

Authentication Flow:

```
User Visits Site → Clerk.load() → Check Session
    ├── Authenticated → Full Access (unlimited uploads)
    └── Guest → Trial Mode (2 uploads limit)
        └── Upload Attempt → Check localStorage
            ├── Count < 2 → Allow + Increment
            └── Count >= 2 → Block + Show Login Prompt
```





Code Snippet (Auth Decorator):

```
def require_auth(f):

    @wraps(f)
    def decorated_function(*args, **kwargs):
        auth_header = request.headers.get('Authorization', '')
```

```
        session_token = auth_header.replace('Bearer ', '') or request.cookies.get('__session')

        ifnot session_token:
            return jsonify({"error": "Unauthorized"}), 401

        session = clerk_client.sessions.verify_token(session_token)
        ifnot session:
            return jsonify({"error": "Invalid session"}), 401

        request.user_id = session.get('sub')

        return f(*args, **kwargs)
    return decorated_function
```

Module 4: Frontend API & User Profile

Responsibilities: Complete frontend interface, API integration, user profile management, history tracking

Components:

- **Pages:** index.html, home.html, upload.html, history.html, details.html, profile.html
- **JavaScript:** upload.js, history.js, details.js, notifications.js, dropdown.js
- **Styling:** style.css, Tailwind CSS classes
- **Backend:** Flask route handlers for page rendering and API endpoints

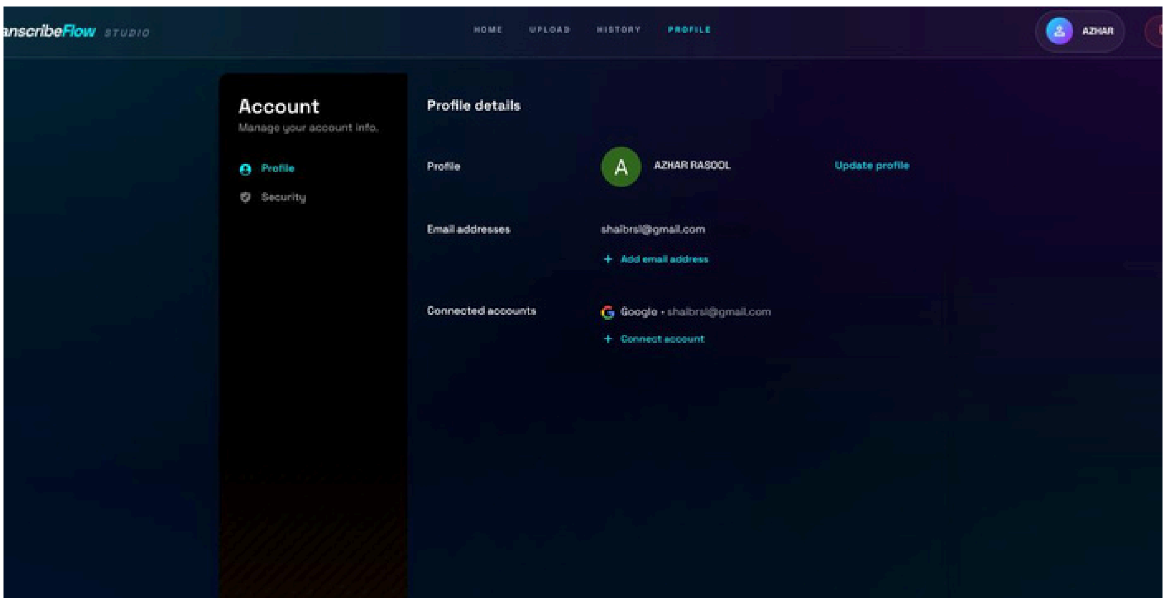
4.1 Frontend Pages

Page	Route	Purpose
Landing	/ (index.html)	Marketing page with product features and call-to-action
Login	/login (login.html)	Clerk authentication interface
Home Dashboard	/home (home.html)	User dashboard with stats and quick actions
Upload	/upload-page (upload.html)	Main transcription interface with drag-and-drop
History	/history-page (history.html)	List of all processed transcripts
Details	/details (details.html)	Full transcript view with analytics
Profile	/profile (profile.html)	User account settings and preferences

4.2 API Endpoints

Method	Endpoint	Authentication	Purpose
POST	/upload	Optional	Upload audio file for processing
GET	/history	Optional	Retrieve all transcription history
DELETE	/history/<id>	Optional	Delete specific history item by ID
DELETE	/history/delete-all	Optional	Clear entire history
DELETE	/delete/<filename>	Optional	Delete audio file and history entry
GET	/uploads/<filename>	None	Serve uploaded audio files

Note: Authentication is currently enforced on the frontend via Clerk. The `@require_auth` decorator exists but is commented out in production to allow trial mode access.



4.3 History Management

Storage Format (history.json):

```
[
 {
  "id": 1709097834,
  "filename": "meeting_recording.mp3",
  "timestamp": "2024-02-28 14:30:34",
  "transcript": "Full transcript text...",
  "summary": "Meeting discussed Q1 goals...",
  "sonic_dna": {
   "energy": 68,
   "pace": 72,
   "clarity": 81,
   "duration": 240,
   "rms": 45,
```

```
    "raw_pace": 145
  },
  "bullet_points": ["Key point 1", "Key point 2", "Key point 3"],
  "keywords": ["Q1", "Goals", "Strategy", "Budget", "Timeline"],
  "confidence_score": 94.8,
  "word_count": 582
 }
]
```

History Operations:

- **Create:** save_to_history() inserts new entry at position 0 (newest first)
- **Read:** GET /history returns full JSON array
- **Delete Single:** DELETE /history/<id> filters out matching ID
- **Delete All:** DELETE /history/delete-all overwrites with empty array
- **Persistence:** JSON file updated synchronously after each operation

4.4 Sonic DNA Analytics

Purpose: Provide acoustic analysis metrics for uploaded audio

Metrics Calculated:

- **Energy (0-100):** RMS (Root Mean Square) amplitude scaled to 0-100 range
- **Pace (0-100):** Words per minute (WPM) normalized to 0-100 scale (200 WPM = 100)
- **Clarity (0-100):** Spectral centroid (frequency brightness) as proxy for clarity
- **Duration:** Total audio length in seconds
- **Raw Pace:** Actual WPM value (not normalized)

Code Snippet:

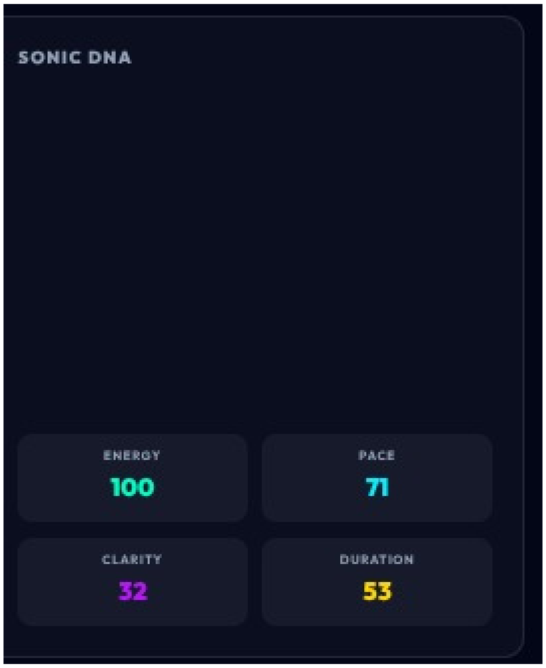
```
def calculate_sonic_dna(audio_path, transcript):
  y, sr = librosa.load(audio_path)
  duration_seconds = librosa.get_duration(y=y, sr=sr)

  # Energy from RMS
  rms = librosa.feature.rms(y=y)
  energy = min(int(np.mean(rms) * 1000), 100)

  # Pace from word count
  word_count = len(transcript.split())
  pace = int(word_count / (duration_seconds / 60)) # WPM
  pace_score = min(int((pace / 200) * 100), 100)
  # Clarity from spectral centroid
  centroid = librosa.feature.spectral_centroid(y=y, sr=sr)
  clarity = min(int(np.mean(centroid) / 50), 100)
```

```
return {"energy": energy, "pace": pace_score, "clarity": clarity, ...}
```

Visualization: Radar chart displays energy, pace, and clarity on `details.html` page



4.5 User Interface Components

Notifications System (`notifications.js`):

```
const notify = {
  success: (msg, duration = 3000) => showToast(msg, 'success', duration),
  error: (msg, duration = 5000) => showToast(msg, 'error', duration),
  warning: (msg, duration = 4000) => showToast(msg, 'warning', duration),
  info: (msg, duration = 3000) => showToast(msg, 'info', duration)
};
```

Custom Dropdown (`dropdown.js`):

- Language selector for translation
- Custom-styled select dropdowns
- Keyboard navigation support

Audio Player:

- Custom HTML5 audio controls
- Waveform visualization (visual feedback)
- Playback speed controls
- Download button

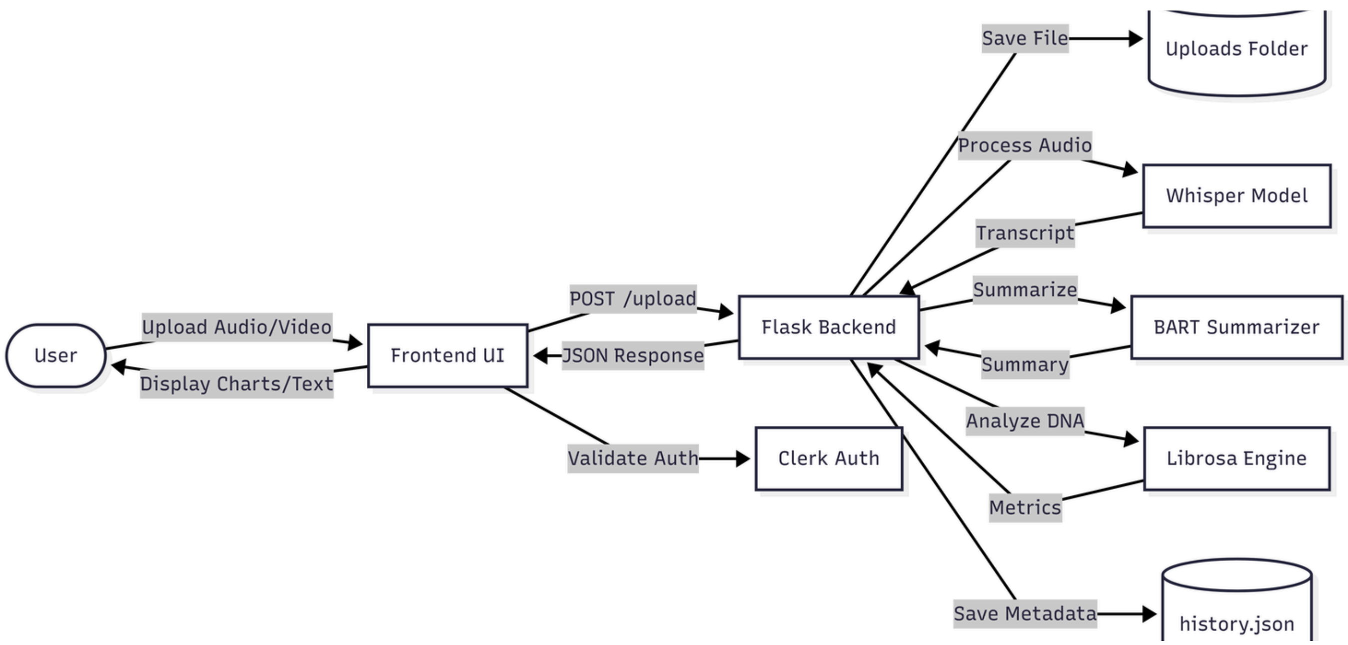
Data Flow Architecture

Complete Request-Response Cycle

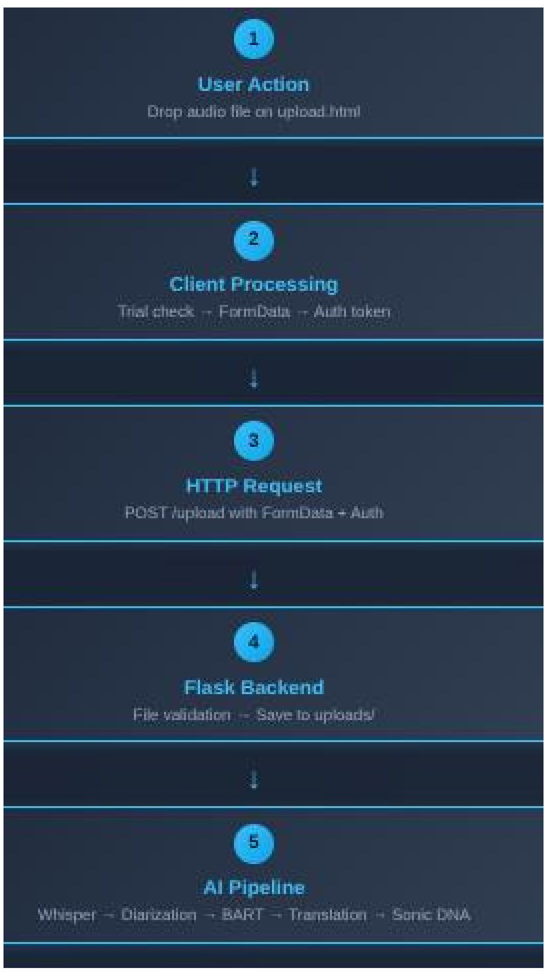
[1. User Action]

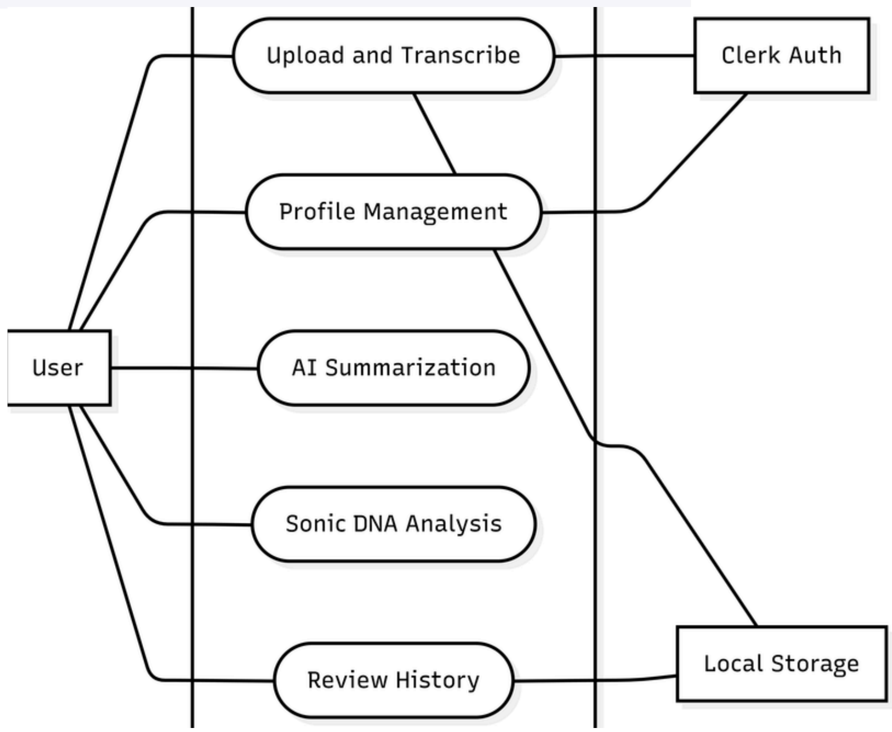
```
  User drops audio file on upload.html
     ↓
[2. Client-Side Processing]
  upload.js: handleDrop() → uploadFile()
  - Trial limit check (trialManager.canUpload())
  - FormData construction (audio, target_lang, enable_diarization)
  - Clerk token retrieval (Clerk.session.getToken())
     ↓
[3. HTTP Request]
  POST /upload with FormData + Authorization header
     ↓
[4. Flask Backend (app.py)]
  @app.route('/upload', methods=['POST'])
  - File validation and saving to uploads/
  - Extract form parameters (target_lang, enable_diarization)
     ↓
[5. AI Processing Pipeline]
   ├─ Whisper Transcription (asr_model.transcribe())
   │  → Returns: transcript text + segments with timestamps
   ├─ Speaker Diarization (if enabled)
   │  → perform_diarization() → format_transcript_with_speakers()
   ├─ Confidence Scoring (np.mean(avg_logprobs))
   ├─ BART Summarization (summarizer(input_text))
   │  → Returns: summary text
   ├─ Bullet Points & Keywords Extraction
   │  → split summary, count word frequency
   ├─ Sonic DNA Calculation (calculate_sonic_dna())
   │  → Librosa: RMS, WPM, spectral centroid
   └─ Translation (if target_lang != 'original')
     → translate_texts() using GoogleTranslator
     ↓
[6. Persistence]
  save_to_history() → Append entry to history.json
     ↓
[7. JSON Response]
  {
   "transcript": "...",
   "summary": "...",
   "sonic_dna": {...},
   "bullet_points": [...],
   "keywords": [...],
   "confidence_score": 94.8,
```

DATA FLOW DIAGRAM



```
    "word_count": 582,
    "audio_url": "/uploads/filename.mp3",
    "num_speakers": 2
  }
      ↓
[8. Client-Side Rendering]
  upload.js: displayResults(data, filename)
  - Show results-section
  - Update transcript, summary, sonic DNA radar chart
  - Display bullet points, keywords, confidence
  - Load audio player with audio_url
  - Increment trial counter (if not authenticated)
  - Show success notification
```





Core Algorithms & Implementation Details

1. Speaker Diarization Pipeline

ChallengeSolved: Pyannote'sdefaultaudioloadingusestorchcodec, which has FFmpeg DLL conflicts on Windows.

Solution: Load audio with Librosa, convert to PyTorch tensor, pass pre-loaded audio dict to Pyannote pipeline.

Implementation Steps:

1. Load audio at 16kHz mono with `librosa.load()`
2. Convert NumPy array to PyTorch tensor with channel dimension
3. Create audio dict: `{'waveform': tensor, 'sample_rate': 16000}`
4. Initialize Pyannote pipeline with HuggingFace token
5. Run diarization on in-memory audio (bypasses file I/O)
6. Extract speaker segments with timestamps
7. Match Whisper word segments to speaker timestamps
8. Format transcript with "Speaker 1:", "Speaker 2:" labels

Technical Note: Single-speaker audio returns `None` (no diarization applied). Multi-speaker audioformats transcript as dialogue.

2. FFMPEG Configuration for Whisper

Problem: Whisper calls ffmpeg executable via subprocess, but imageio-ffmpeg installs as ffmpeg-{platform}.exe

Solution:

```
ffmpeg_exe = imageio_ffmpeg.get_ffmpeg_exe() # Get platform-specific exe
ffmpeg_dir = os.path.dirname(ffmpeg_exe)
target_ffmpeg = os.path.join(ffmpeg_dir, "ffmpeg.exe") # Create copy with exact name

if not os.path.exists(target_ffmpeg):

    shutil.copy(ffmpeg_exe, target_ffmpeg)

os.environ["PATH"] += os.pathsep + ffmpeg_dir # Inject into PATH
```

This ensures Whisper's `subprocess.call(['ffmpeg', ...])` finds the executable.

3. Translation Character Limit Handling

Google Translate API Limit: 5000 characters per request

Mitigation:

```
translated_transcript = translator.translate(transcript[:4999]) if len(transcript) <= 4999 else
transcript
```

Transcripts >4999 characters return untranslated. Summary translation always attempted (typically <1000 chars).

4. Trial Mode Client-Side Enforcement

Implementation: `trial-mode.js` singleton instance

Key Methods:

- `initialize()`: Detects Clerk authentication state
- `canUpload()`: Returns `true` if logged in OR trial count < 2
- `incrementTrial()`: Increments `localStorage` counter after successful upload
- `showLimitBanner()`: Displays "Sign in to continue" banner
- `disableUpload()`: Disables file input and upload button

Security Note: Trial enforcement is client-side only (easily bypassed). Production should enforce server-side via database-backed rate limiting.

Configuration & Deployment

Environment Setup

Required Python Version: 3.8+

Installation:

```
pip install -r requirements.txt
```

Dependencies (requirements.txt):

```
Flask==3.0.3 Flask-Cors==5.0.0 imageio-
ffmpeg==0.5.1 openai-
whisper==20231117
transformers==4.45.1
librosa==0.10.2.post1 torch==2.4.1
numpy==2.1.1 deep-translator==1.11.4
clerk-backend-api==1.1.0
requests==2.32.3
```

Additional Requirements:

- **Pyannote.audio:** Install separatelywith HuggingFace token (not in requirements.txt)
- **FFmpeg:** Automatically handled viaimageio-ffmpeg

Configuration Variables

Flask Configuration (app.py):

```
basedir = os.path.dirname(os.path.abspath(__file__))
template_dir = os.path.join(basedir, "templates")
static_dir = os.path.join(basedir, "static")
UPLOAD_FOLDER = os.path.join(basedir, "uploads")
HISTORY_FILE = os.path.join(basedir, "history.json")
```

Clerk Authentication:

```
CLERK_SECRET_KEY = "sk_test_VZJlwKWegCCcq6SftOAp5fXDJcMXSEZDmIUr1Zs2TL"
clerk_client = Clerk(bearer_auth=CLERK_SECRET_KEY)
```

Frontend Clerk Key (trial-mode.js, auth.js):

```
const clerkPubKey = "pk_test_Y29vcC1naWxsLTQ0LmNsZXJrLmFjY291bnRzLmRldiQ";
await Clerk.load({ publishableKey: clerkPubKey });
```

Pyannote Token (app.py):

```
pipeline = Pipeline.from_pretrained(
  "pyannote/speaker-diarization-3.1",
```

```
 token="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"
)
```

Running the Application

Development Mode:

```
python app.py
```

Server Configuration:

```
if __name__ == "__main__":
    app.run(debug=True, port=5000, use_reloader=False)
```

Access: Navigateto `http://127.0.0.1:5000` or `http://localhost:5000`

GitHub Codespaces: Flaskautomatically forwards port 5000 to public `*.github.dev` URL

Directory Structure

```
project-root/
├── app.py                  # Main Flask application
├── diarization_template.py    # Diarization implementation reference
├── requirements.txt          # Python dependencies
├── history.json             # Transcription history (auto-generated)
├── uploads/                # Audio file storage (auto-generated)
├── templates/               # HTML pages
│   ├── index.html
│   ├── login.html
│   ├── home.html
│   ├── upload.html
│   ├── history.html
│   ├── details.html
│   └── profile.html
└── static/                 # JavaScript and CSS
├── js/
│   ├── upload.js
│   ├── history.js
│   ├── details.js
│   ├── auth.js
│   ├── trial-mode.js
│   ├── notifications.js
│   └── dropdown.js
└── css/
    └── style.css
```

API Reference

POST /upload

Purpose: Upload audiofile for transcription, summarization, and analysis

Authentication: Optional (Clerk Bearer token or __session cookie)

Request Format: multipart/form-data

Form Parameters:

Parameter	Type	Required	Description
audio	File	Yes	Audio file (MP3, WAV, MP4, MOV, up to 2GB)
target_lang	String	No	Target language code (default: 'original')
enable_diarization	String	No	'true' or 'false' (default: 'false')
upload_mode	String	No	'authenticated' or 'trial' (client metadata)

Response (200 OK):


```
{
 "transcript": "Full transcript text with speaker labels if diarization enabled...",
 "summary": "AI-generated summary of the transcript...",
 "original_transcript": "Untranslated transcript...",
 "original_summary": "Untranslated summary...",
 "sonic_dna": {
  "energy": 68,
  "pace": 72,
  "clarity": 81,
  "duration": 240,
  "rms": 45,
  "raw_pace": 145
 },
 "word_count": 582,
 "bullet_points": ["Point 1", "Point 2", "Point 3"],
 "keywords": ["Keyword1", "Keyword2", "Keyword3", "Keyword4", "Keyword5"],
 "confidence_score": 94.8,
 "audio_url": "/uploads/filename.mp3",
 "num_speakers": 2
}
```


Error Response (400/500):


```
{
 "error": "No file part"
```

```
}
```

GET /history

Purpose: Retrieve alltranscription history entries

Authentication: Optional

Response (200 OK):


```
[
  {
    "id": 1709097834,
    "filename": "meeting.mp3",
    "timestamp": "2024-02-28 14:30:34",
    "transcript": "...",
    "summary": "...",
    "sonic_dna": {...},
    "bullet_points": [...],
    "keywords": [...],
    "confidence_score": 94.8,
    "word_count": 582
  }
]
```

DELETE /history/<id>

Purpose: Delete specifichistoryentrybyID

Authentication: Optional

URL Parameters:

- `id` (integer): History item ID (Unix timestamp)

Response (200 OK):


```
{
  "success": true,
  "message": "Item deleted successfully"
}
```


Error Response (404):


```
{
  "success": false,
  "error": "Item not found"
```

```
}
```

DELETE /history/delete-all

Purpose: Clearall history entries

Authentication: Optional

Response (200 OK):

```
{
  "success": true,
  "message": "All history deleted successfully"
}
```

DELETE /delete/<filename>

Purpose: Delete audiofile andassociatedhistoryentry

Authentication: Optional

URL Parameters:

- `filename` (string): Audio filename in `uploads/` directory

Response (200 OK):

```
{
  "message": "File deleted"
}
```

GET /uploads/<filename>

Purpose: Serveaudiofilesfor playback

Authentication: None (public access)

URL Parameters:

- `filename` (string): Audio filename

Response: Binary audio file with appropriate MIME type

Security Considerations

Current Implementation

1. **Authentication:** Clerk integrationpresent but routes NOT protected (commented out `@require_auth`)
2. **Trial Mode:** Client-side enforcement only (localStorage) - easily bypassed
3. **File Upload:** No server-side validation beyond Flask's file handling
4. **CORS:** Enabled globally (`CORS(app)`) - accepts requests from any origin
5. **APIKeys:**Hardcodedin sourcecode (Clerksecret, Pyannote token)

Production Recommendations

Critical:

- [ ] Move API keys to environment variables (`os.getenv('CLERK_SECRET_KEY')`)
- [ ] Enable `@require_auth` decorator on all sensitive routes
- [ ] Implement server-side rate limiting (e.g., Flask-Limiter)
- [ ] Add file type validation beyond extension checking
- [ ] Implement file size limits at application layer
- [ ] Configure CORS to allow only trusted domains
- [ ] Add HTTPS enforcement (redirect HTTP to HTTPS)

Moderate Priority:

- [ ] Implement database-backed history (replace JSON file)
- [ ] Add user-specific history isolation (currently shared across all users)
- [ ] Sanitize filenames to prevent path traversal attacks
- [ ] Add virus scanning for uploaded files (e.g., ClamAV)
- [ ] Implement audit logging for all API operations

Low Priority:

- [ ] Add CSRF protection for form submissions
- [ ] Implement content security policy (CSP) headers
- [ ] Add request signing for API calls

Performance Optimization

Current Performance Characteristics

ModelLoading (One-Time Startup):

- Whisper "tiny" model: ~2 seconds

- BART summarizer: ~3 seconds
- Pyannote pipeline: ~5 seconds (on-demand, per diarization request)

Per-Request Processing Time:

Operation	Duration (approx.)	Scaling Factor
Whisper Transcription	0.1-0.3× audio duration	Linear with audio length
Speaker Diarization	0.5-1.0× audio duration	Linear with audio length
BART Summarization	2-5 seconds	Linear with transcript length
Translation	1-2 seconds	Linear with text length
Sonic DNA Calculation	0.5-1 second	Linear with audio length

Example: 5-minute audio file with diarization:

- Transcription: ~90 seconds
- Diarization: ~240 seconds
- Summarization: ~3 seconds
- Translation: ~2 seconds
- **Total:** ~6 minutes

Optimization Strategies

Immediate Improvements:

1. **Use Larger Whisper Model for Accuracy:** Upgrade from "tiny" to "base" or "small" (trade-off: 2-3× slower)
2. **Implement Async Processing:** Use Celery + Redis for background task queue
3. **Cache Model Inference:** Store results in Redis to avoid reprocessing identical files
4. **Optimize Librosa Loading:** Use `sr=None` to avoid resampling if audio already at 16kHz

Advanced Optimizations:

1. **GPU Acceleration:** Run Whisper and Pyannote on CUDA-enabled GPU (10-30× speedup)
2. **Model Quantization:** Use INT8 quantized models to reduce memory and increase throughput
3. **Batch Processing:** Process multiple files in parallel using multiprocessing pool
4. **CDN for Static Assets:** Serve HTML/CSS/JS from CDN to reduce server load

Code Example (Async with Celery):

```
from celery import Celery

celery = Celery('tasks', broker='redis://localhost:6379')
```

```
@celery.task
def process_audio_async(file_path, target_lang, enable_diarization):
    # Existing processing pipeline
    return result_dict

@app.route('/upload', methods=['POST'])
def upload_file():
    # Save file
    task = process_audio_async.delay(file_path, target_lang, enable_diarization)
    return jsonify({"task_id": task.id, "status": "processing"})

@app.route('/status/<task_id>')
def check_status(task_id):
    task = process_audio_async.AsyncResult(task_id)
    if task.ready():
        return jsonify({"status": "complete", "result": task.result})
    return jsonify({"status": "processing"})
```

Testing Strategy

Unit Tests (Recommended)

Backend (pytest):

```
def test_calculate_sonic_dna():
    result = calculate_sonic_dna('test_audio.mp3', 'test transcript')
    assert 0 <= result['energy'] <= 100
    assert 0 <= result['pace'] <= 100
    assert 0 <= result['clarity'] <= 100

def test_translate_texts():
    transcript, summary = translate_texts('Hello', 'Summary', 'es')
    assert transcript != 'Hello' # Translated

def test_save_to_history():
    history = save_to_history('test.mp3', 'transcript', 'summary', {}, [], [], 0.95, 100)
    assert len(history) > 0
    assert history['filename'] == 'test.mp3'
```

Frontend (Jest):

```
test('uploadFile checks trial limit', () => {
    window.trialManager = { canUpload: () => false };
    uploadFile(mockFile);
    expect(notify.warning).toHaveBeenCalledWith(expect.stringContaining('trial limit'));
});
```

```
test('displayResults updates UI elements', () => {
  const data = { transcript: 'Test', confidence_score: 95.5 };
  displayResults(data, 'test.mp3');
  expect(document.getElementById('confidence-display').innerText).toBe('95.5%');
});
```

Integration Tests

API Endpoints:

```
def test_upload_endpoint(client):
  with open('test_audio.mp3', 'rb') as f:
    response = client.post('/upload', data={'audio': f})
  assert response.status_code == 200
  assert 'transcript' in response.json

def test_history_endpoint(client):
  response = client.get('/history')
  assert response.status_code == 200
  assert isinstance(response.json, list)
```

Manual Testing Checklist

Module 1 (Upload):

- [ ] Drag-and-drop file upload works
- [ ] File validation rejects invalid formats
- [ ] Trial limit enforced (2 uploads max)
- [ ] Progress indicator displays during processing
- [ ] Success/error notifications appear correctly

Module 2 (Transcription):

- [ ] Single-speaker audio transcribes correctly
- [ ] Multi-speaker audio detects speakers and labels transcript
- [ ] Translation produces correct target language
- [ ] Confidence scores are reasonable (>80%)

Module 3 (Summarization):

- [ ] Summary captures key points from transcript
- [ ] Bullet points extract top 3 sentences
- [ ] Keywords identify most frequent terms

Module 4 (Frontend):

- [ ] History page displays all past transcripts
- [ ] Details page shows full analytics (Sonic DNA radar chart)
- [ ] Audio player controls work (play, pause, seek)
- [ ] Profile page loads user information from Clerk

Troubleshooting Guide

Common Issues

Issue 1: "FFmpeg not found" error

Cause: Whisper cannot locate ffmpeg executable in PATH

Solution:

```
# Check if ffmpeg.exe exists in imageio-ffmpeg directory
ffmpeg_dir = os.path.dirname(imageio_ffmpeg.get_ffmpeg_exe())
target_ffmpeg = os.path.join(ffmpeg_dir, "ffmpeg.exe")
print(f"FFmpeg path: {target_ffmpeg}")
print(f"Exists: {os.path.exists(target_ffmpeg)}")

# Manually add to PATH if missing
os.environ["PATH"] += os.pathsep + ffmpeg_dir
```

Issue 2: "torchcodec DLL load failed" during diarization

Cause: Pyannote's default audio loading uses torchcodec with FFmpeg DLL conflicts

Solution: Use Librosa-based audio loading (already implemented in `perform_diarization()`)

```
# Don't pass file path directly to pipeline
diarization = pipeline(audio_file_path) # Fails with torchcodec error
# Use pre-loaded audio dict instead
waveform, sr = librosa.load(audio_file_path, sr=16000, mono=True)
audio_dict = {'waveform': torch.from_numpy(waveform).unsqueeze(0), 'sample_rate': sr}
diarization = pipeline(audio_dict) # Works
```

Issue 3: Translation fails silently

Cause: Google Translate API character limit (5000 chars) or network timeout

Solution:

```
try:
    translated = translator.translate(text[:4999]) # Respect 5000 char limit
except Exception as e:
    print(f"Translation error: {e}")
    translated = text # Fallback to original
```

Issue 4: History not persisting across restarts

Cause: `history.json` in `.gitignore` or not writable

Solution:

```
# Ensure history file is writable
HISTORY_FILE = os.path.join(basedir, "history.json")
if not os.path.exists(HISTORY_FILE):
    with open(HISTORY_FILE, 'w') as f:
        json.dump([], f) # Initialize empty history
```

Issue 5: CORS errors in browser console

Cause: Frontend served from different origin than Flask backend

Solution:

```
from flask_cors import CORS
CORS(app, origins=["http://localhost:3000", "https://yourdomain.com"])
```

Issue 6: Whisper model download fails

Cause: Network connectivity or HuggingFace API quota

Solution:

```
# Pre-download model to cache
import whisper
whisper.load_model("tiny", download_root="./models")

# Use local cache

asr_model = whisper.load_model("tiny", download_root="./models")
```

Future Enhancements

Short-Term Improvements (1-3 months)

1. Real-Time Transcription

- WebSocket-based streaming transcription
- Incremental result updates during processing
- Live confidence scoring display

2. Advanced Speaker Identification

- Speaker name labeling (custom labels instead of "Speaker 1")
- Speaker voice embeddings for cross-file identification
- Speaker profile management

3. Enhanced Translation

- Support for additional translation providers (DeepL, Azure Translator)
- Translation quality scoring
- Side-by-side original/translated view

4. Export Formats

- PDF export with formatting and branding
- SRT/VTT subtitle file generation
- Word document export with speaker labels

5. Audio Preprocessing

- Noise reduction and audio enhancement
- Automatic volume normalization
- Background music separation

Medium-Term Enhancements (3-6 months)

6. Multi-Modal Input

- Video file upload with audio extraction
- YouTube URL direct transcription
- Screen recording integration

7. Collaborative Features

- Shared transcripts with team members
- Commenting and annotation system
- Version control for edited transcripts

8. Advanced Analytics

- Sentiment analysis per speaker
- Topic modeling and clustering

- Speaker talk-time visualization
- Interruption detection

9. Custom Model Training

- Fine-tune Whisper on domain-specific vocabulary
- Custom keyword extraction models
- Personalized summarization styles

10. Integration APIs

- Zapier integration for workflow automation
- Slack/Teams bot for meeting transcription
- Google Drive/Dropbox file sync

Long-Term Vision (6-12 months)

11. Mobile Applications

- Native iOS/Android apps
- Voice recording and instant transcription
- Offline mode with local processing

12. Enterprise Features

- SAML/SSO authentication
- Role-based access control (RBAC)
- Usage analytics dashboard
- White-labeling and custom branding

13. AI-Powered Search

- Semantic search across all transcripts
- Natural language queries ("Find all mentions of Q1 budget")
- Cross-transcript topic discovery

14. Advanced Diarization

- Emotion detection per speaker
- Language switching detection (code-switching)
- Speaker age/gender classification

15. Legal & Compliance

- GDPR compliance tools (data export, right to erasure)
- HIPAA compliance for medical transcription
- Audit trails for all data access

Conclusion

TranscribeFlow demonstrates a production-ready architecture for AI-powered audio transcription with modular design, comprehensive feature set, and clear separation of concerns. The 4-module structure (Upload, Transcription/Translation, Summarization/Auth, Frontend API) provides maintainability and extensibility for future enhancements.

Key Strengths:

- Robust AI pipeline leveraging state-of-the-art models
- Modern web interface with excellent UX
- Flexible authentication with trial mode support
- Comprehensive audio analytics (Sonic DNA)
- Multi-language support (100+ languages)

Areas for Improvement:

- Security hardening (environment variables, rate limiting)
- Async processing for better scalability
- Database-backed persistence
- Comprehensive test coverage

This documentation provides a complete technical reference for developers to understand, maintain, and extend the TranscribeFlow platform.

[SPACE FOR TECHNOLOGY STACK DIAGRAM]

[SPACE FOR MODULE INTERACTION FLOWCHART]

[SPACE FOR DEPLOYMENT ARCHITECTURE]