TABSENSE AI

Automated Guitar Tablature Transcription via Multi-Modal LLMs

Final Technical Report

Anas Draoui
Lead Full-Stack Engineer

Submitted: December 11, 2025

Abstract

This report documents the architectural design, implementation methodology, and deployment strategy of **TabSense AI**, a full-stack web application developed to automate the transcription of guitar music into ASCII tablature. By integrating the Google Gemini 2.5 Flash model with a custom verification protocol termed the "Hierarchy of Truth," the system addresses the stochastic nature of Large Language Models (LLMs) to produce musically accurate notation. The application was engineered over a four-week sprint, evolving from a client-side prototype into a scalable MERN-stack architecture (MongoDB, Express.js, React, Node.js). Key technical contributions include a robust error-handling mechanism for non-JSON responses and a spectral analysis pipeline for key verification.

Contents

1 Introduction

1.1 Project Scope

TabSense AI serves as a specialized integrated development environment (IDE) for musicians and transcribers. Traditional music transcription is a labor-intensive process requiring significant auditory training. While automated solutions exist, they frequently lack editability or fail to capture the nuance of guitar-specific techniques. TabSense addresses these limitations by leveraging multi-modal Generative AI to synthesize audio analysis with structured text generation.

1.2 Core Objectives

The development lifecycle was driven by four primary technical objectives:

1. **Algorithmic Accuracy:** Implementation of a transcription engine using the Google Gemini API with minimized hallucination rates.
2. **User Experience:** Development of a browser-based, latency-optimized interface for synchronized audio playback and tablature editing.
3. **Persistence:** Establishment of a secure full-stack architecture for user data management.
4. **Robustness:** Implementation of strict verification protocols to ensure musical validity.

2 System Architecture

The application employs a decoupled client-server architecture based on the MERN stack. This ensures separation of concerns between the presentation layer and the data processing layer.

2.1 Technology Stack

- **Frontend:** React 19 (Vite Build Tool), TypeScript, Tailwind CSS.
- **Backend:** Node.js runtime environment, Express.js framework.
- **Database:** MongoDB (NoSQL) accessed via Mongoose ODM.
- **AI Layer:** Google Gemini 2.5 Flash via `@google/genai` SDK.

2.2 Directory Structure

The codebase utilizes a monorepo structure, separating server logic from client assets.

```
/root
          backend/              # Server-side Environment
                models/           # Data Schemas (User, Tab)
                db.js             # Database Connection Logic
                server.js         # API Routes & Auth Middleware
          frontend/             # Client-side Environment
                components/       # UI Elements & Logic
                services/         # Service Layer
                      backend.ts    # SafeFetch Implementation
                      gemini.ts     # AI Prompt Engineering
                App.tsx           # Application Entry Point
          .env                  # Environment Configuration
          vite.config.ts        # Proxy Configuration
```

Listing 1: Project Directory Hierarchy

2.3 Data Flow Pipeline

1. **Client Request:** The user initiates an upload or query.
2. **Proxy Layer:** Vite redirects requests from port 3000 to port 5000, mitigating Cross-Origin Resource Sharing (CORS) issues during development.
3. **Server Validation:** Express.js middleware validates the JSON Web Token (JWT).
4. **Data Persistence:** MongoDB performs CRUD operations on the tablature documents.
5. **Inference:** The system constructs a context-aware prompt and interfaces with the Gemini API.

3 Methodology: The "Hierarchy of Truth"

A significant challenge in applying LLMs to music transcription is "hallucination"—the generation of syntactically correct but musically nonsensical data. To mitigate this, a tiered verification protocol was implemented:

1. **Tier 1: Grounding (Web Research).** The model utilizes Google Search tools to locate verified transcriptions from authoritative databases (e.g., Ultimate-Guitar).
2. **Tier 2: Spectral Verification.** The model analyzes the audio input's frequency spectrum to determine the root key and tempo, cross-referencing this against the Tier 1 data.

3. **Tier 3: Synthesis.** The final output is generated only when Tier 1 and Tier 2 data converge, ensuring the ASCII output is musically coherent.

4 Development Lifecycle

The project adhered to an Agile methodology, divided into four one-week sprints.

4.1 Phase 1: Prototyping and Feasibility

Initial efforts focused on the integration of the `@google/genai` SDK.

- Implementation of binary file handling for audio uploads.
- **Optimization:** Transitioned from Gemini 3 Pro to Gemini 2.5 Flash to resolve API Quota (Error 429) bottlenecks and reduce inference latency.

4.2 Phase 2: Interface Design and Simulation

The user interface was developed to facilitate intuitive interaction. A mock service layer using ‘localStorage‘ was created to simulate backend interactions, allowing frontend development to proceed in parallel with backend setup.

4.3 Phase 3: Full-Stack Integration

The mock services were deprecated in favor of a live Node.js environment.

- **Authentication:** Implemented stateless authentication using JWT and bcrypt for password hashing.
- **Seeding:** Developed automated scripts to initialize the database with administrative credentials upon startup.
- **Error Handling:** Created a ‘safeFetch‘ utility to parse non-standard HTTP responses (e.g., HTML error pages) preventing runtime crashes.

5 Technical Challenges Solutions

5.1 Handling Non-JSON Server Responses

A critical failure point occurred when the backend returned HTML (500 Internal Server Error) instead of JSON, causing the frontend JSON parser to throw an exception.

Solution: A wrapper function was implemented to inspect the ‘Content-Type‘ header. If the response is not ‘application/json‘, the system extracts the text body and throws a controlled error, maintaining application stability.

5.2 Module Resolution in Modern Bundlers

Integration of legacy Node.js modules with Vite caused import conflicts within the document object model. This was resolved by configuring explicit alias definitions in ‘vite.config.ts‘, removing the need for manual import maps in the HTML entry point.

6 Conclusion and Future Work

TabSense AI demonstrates the efficacy of Large Language Models in complex, creative transcription tasks when bounded by strict verification protocols. The system successfully bridges the gap between raw audio signals and editable musical notation.

Future iterations will prioritize:

1. **Deployment:** Migration of the database to MongoDB Atlas and the API to a serverless container environment.
2. **Interoperability:** Implementation of MusicXML export to ensure compatibility with industry-standard notation software such as Sibelius and Guitar Pro.
3. **Scalability:** Integration of AWS S3 for offloading audio binary storage from the application server.