# Technical Architecture and Development Timeline: IDTAP

## Data Model and Core Entities

The IDTAP platform is built around a sophisticated data model specifically designed to represent the complex relationships and hierarchies inherent in musical transcription and analysis. At its core, the system employs a multi-dimensional approach to handling musical data, accommodating up to four simultaneous instrumental tracks while maintaining precise temporal and harmonic relationships.

### Core Musical Entities

**Trajectory** represents the fundamental unit of musical expression in IDTAP. Each trajectory encapsulates an archetypal melodic motion, containing arrays of Pitch objects that define the harmonic content, duration arrays that specify temporal subdivision, and articulation dictionaries that capture performance-specific techniques. The trajectory system includes 14 distinct types (IDs 0-13, with ID 12 representing silence): pitch-stable trajectories for sustained notes, various slide types with adjustable slopes for glissandi, vibrato patterns with configurable parameters, and specialized trajectories for ornaments and attacks. Each trajectory stores its pitch data as both frequency values and logarithmic representations, enabling efficient visualization and analysis.

**Pitch** objects encode microtonal information using a sophisticated representation system. Each pitch contains swara (scale degree), octave, and raised/lowered boolean flags, along with computed properties for frequency, logarithmic frequency, and various representational formats (sargam letters, solfege, chromatic values, Western pitch names). The pitch system supports arbitrary fundamental frequencies and custom tuning ratios, enabling accurate representation of diverse microtonal traditions.

**Phrase** serves as the organizational unit containing an array of trajectories across multiple instrumental tracks, a chikari object containing all sympathetic string information (present only in sitar transcriptions), and an array of groups for analytical segmentation. Each phrase includes comprehensive categorization systems supporting both structured and ad-hoc labeling approaches.

The structured labeling system encompasses **eight phrase types**: Mohra (opening motif), Mukra (refrain), Asthai (first section), Antara (second section), Manjha (middle section), Abhog (final section), Sanchari (development section), and Jhala (fast rhythmic section). **Thirteen elaboration types** characterize developmental techniques: Vistar (expansion), Barhat (gradual elaboration), Prastar (extended development), Bol Banao (text elaboration), Bol Alap (syllabic improvisation), Bol Bandt (rhythmic text setting), Behlava (melodic variation), Gat-kari (instrumental composition work), Tan Sapat (straight runs), Tan Gamak (ornamented runs), Laykari (rhythmic play), Tihai (three-time repetition), and Chakradar (circular patterns).

**Vocal articulation types** include Bol (syllabic singing with text), Non-Tom (non-syllabic singing), Tarana (fast syllabic improvisation), Aakar (open vowel singing), and Sargam (scale-degree singing). **Instrumental articulations** distinguish between Bol (stroke patterns mimicking vocal syllables) and Non-Bol (non-syllabic techniques). **Incidental categories** capture non-musical elements: Talk/Conversation, Praise (‘Vah’), Tuning, and Pause.

Beyond these structured categories, the system supports **ad-hoc labeling** where transcribers can create custom text labels for phrases and sections, add or remove label fields dynamically, and use free-form text for categories not covered by the predefined taxonomies. This flexibility enables the system to accommodate diverse analytical approaches while maintaining consistency for comparative studies.

**Piece** represents the highest-level musical entity, coordinating multiple phrase arrays across instrumental tracks while maintaining temporal synchronization. The piece object implements a sophisticated organizational system where each instrumental track contains its own phrase array, duration arrays specifying temporal proportions, section start arrays marking formal divisions, and section category arrays for structural analysis. Pieces maintain relationships to Raga objects for harmonic context, Meter objects for rhythmic analysis, and assemblage descriptors for advanced analytical groupings.

**Raga** encapsulates the harmonic framework through rule sets that specify which scale degrees are active (using lowered/raised boolean pairs), tuning ratios for microtonal precision, and fundamental frequency settings. The raga system supports both equal temperament and just intonation tuning systems, with methods for converting between different representational formats.

**Chikari** objects model sympathetic string information for plucked instruments, storing pitch arrays, stroke timing, and amplitude data for up to four sympathetic strings, enabling synthesis of the complex resonant textures characteristic of instruments like the sitar.

### Data Architecture and Persistence

The IDTAP database architecture employs MongoDB with 13 primary collections designed to handle the complex relational needs of musical data while maintaining query performance. The database implements sophisticated permission systems, hierarchical relationships, and rich metadata structures suitable for academic music research.

#### **Core Content Collections**

**transcriptions** collection stores complete musical transcription documents containing nested phrase arrays with trajectory data, duration arrays, section categorization arrays, and timing data. Each transcription includes metadata (title, creator, creation/modification dates), instrumentation arrays, raga objects with harmonic rules and tuning ratios, and comprehensive permission structures supporting both legacy string-based permissions and modern explicitPermissions objects with granular edit/view access control arrays. The collection maintains bidirectional references with collections and cross-references user ownership.

**audioRecordings** collection contains metadata for individual audio files including duration, SA (tonic) frequency estimates with verification status, octave offsets, complex musician objects mapping names to roles and instruments, hierarchical date objects, location objects with continent/country/city structure, and raga objects mapping raga names to timing information. Records include parent references to audioEvents, user ownership, and explicit permission structures. This collection is generated via aggregation pipelines from audioEvents while also supporting standalone “loose” recordings.

**audioEvents** collection serves as organizational containers for grouped recordings (concerts, sessions, performances) with nested recordings objects using dynamic keys mapped to track numbers. Each event includes user ownership, event type classification, permission structures, and serves as the source for audioRecordings aggregation pipelines with cascading update and delete operations.

#### **User and Organizational Collections**

**users** collection manages user accounts with Google OAuth integration (sub, email, profile data), waiver agreements, owned transcription arrays, saved multiQueries arrays containing complex search objects, savedSettings arrays for display preferences, defaultSettingsID references, and transcriptionsViewed objects tracking access history with timestamps.

**collections** collection enables user-curated content organization with title, creator information, purpose enumeration (Pedagogical, Research, Creative, Personal), description text, sophisticated permission objects with view/edit arrays and publicView flags, content arrays for audioRecordings/audioEvents/transcriptions, date tracking, color coding, and invite code systems for sharing. The collection maintains bidirectional references where collections reference content and content references collections.

#### **Reference and Lookup Collections**

**musicians** collection provides biographical data with hierarchical name structures (Initial Name, Full Name, First/Middle/Last Name), birth/death years, gharana (musical lineage) affiliations, gender classification, primary instrument, and comprehensive instrument arrays. The collection supports complex queries for artist information and integrates with gharana hierarchies.

**ragas** collection stores musical scale/mode definitions with name identifiers, rules objects specifying scale degree configurations, and update tracking. This collection provides the harmonic framework data for transcription analysis and tuning systems.

**instruments** collection contains reference data with name and kind classifications (melody, percussion, etc.), supporting instrument type filtering and transcription instrumentation validation.

**gharanas** collection maintains musical school/lineage information with name identifiers and members arrays, providing genealogical context for musician classification and stylistic analysis.

**phonemes** collection stores comprehensive phonetic data for vocal transcription with type classification (vowel/consonant), IPA symbols, English transliterations, and Hindi/Urdu object representations supporting multi-script vocal notation.

**location** collection implements hierarchical geographic structures with dynamic continent/country/city organization, enabling geographic analysis of musical content and performance contexts.

#### **Specialized Collections**

**audioEventTypes** and **performanceSections** collections provide standardized terminology for event classification and performance structure annotation, supporting consistent metadata entry and analytical queries.

#### **Database Architecture Patterns**

The database implements several sophisticated patterns: **aggregation-based derived collections** where audioRecordings are generated from audioEvents data; **bidirectional reference management** maintaining consistency between collections and their referenced content; **complex permission systems** supporting multiple access levels with inheritance patterns; **nested document structures** for rich musical metadata while maintaining query efficiency; **cross-collection update cascading** ensuring data consistency across related documents; and **hybrid embedding/referencing strategies** optimizing for both analytical queries and real-time editor performance.

This architecture enables IDTAP to handle complex musicological research requirements while supporting real-time collaborative editing, sophisticated analytical queries, and scalable content organization suitable for institutional deployment.

### Type System and Interoperability

IDTAP’s type system is architected for cross-language compatibility, with comprehensive TypeScript interfaces defined in shared/types.ts ensuring consistency between frontend and backend systems. The type definitions encompass 200+ interfaces covering everything from basic musical elements (PitchNameType, OutputType) to complex analytical structures (QueryType with 15 category variations, MultipleReturnType for pattern analysis).

Key type families include **Musical Types** (Pitch, Trajectory, Phrase hierarchies), **Display Types** (SargamDisplayType, VowelDisplayType, ChikariDisplayType for visualization), **Analysis Types** (QueryType, QueryAnswerType, MultipleOptionType for analytical workflows), **Audio Types** (synthesis node interfaces, parameter mappings), and **Database Types** (collection schemas, permission structures).

The system’s Python API porting project systematically translates these TypeScript classes into Python equivalents, enabling computational researchers to work with identical data structures across both web-based and programmatic analysis environments.

## Technical Architecture Overview

The Interactive Digital Transcription and Analysis Platform (IDTAP) represents a comprehensive reimagining of how musical transcription and analysis can be approached in the digital humanities. Rather than adapting existing Western notation systems to non-Western musical traditions, we built IDTAP from the ground up to authentically represent the melodic contours, microtonal inflections, and expressive nuances that characterize oral musical traditions like Hindustani classical music.

### Platform Philosophy and Core Innovation

IDTAP’s central innovation lies in its trajectory-based transcription system. Instead of treating fixed-pitch notes as the atomic units of musical structure, our platform organizes musical representation around “trajectories”—formally specified archetypal paths between pitches, across series of pitches, or sustained on fixed pitches. This approach provides what we call a “middle-level representation” that captures the finely calibrated glissandi essential to many non-Western musical traditions while avoiding the over-granular nature of raw fundamental frequency data and the over-simplified representation of MIDI.

Our trajectory system includes thirteen distinct types for plucked string instruments like the sitar and eight types for vocal and bowed string instruments like the sarangi. Each trajectory can be adjusted for slope and microtonal positioning, allowing transcribers to match the precise melodic contours they hear in recordings. This flexibility enables authentic representation of musical features like *shrutis* (microtones between scale degrees) and *andolans* (expressive movements between pitches) that are central to Hindustani musical practice but impossible to accurately capture in traditional notation.

### Frontend Architecture and Component System

IDTAP’s user interface is built as a Vue.js 3 application with TypeScript, organized around a sophisticated component-based architecture that supports multiple concurrent user workflows. The application leverages Vue’s Composition API throughout, with reactive state management via Vuex and Vue Router for navigation between the platform’s five primary functional areas.

**Editor Components** form the heart of the transcription experience. The EditorComponent serves as the primary coordinator, integrating multiple specialized sub-components through a complex event-driven architecture. The Renderer component manages the visual display of transcriptions on a logarithmic frequency scale, implementing custom D3.js visualizations with zoom and pan capabilities. TranscriptionLayer handles trajectory placement and editing through sophisticated mouse and keyboard interaction systems, supporting drag-and-drop trajectory manipulation, real-time visual feedback during editing operations, and undo/redo functionality through command pattern implementation.

SpectrogramLayer and MelographLayer provide algorithmic pitch analysis overlays through integration with Web Workers and WebGL-accelerated rendering pipelines. The SpectrogramLayer component implements lazy loading via the Intersection Observer API, rendering spectrogram data only for visible portions of the transcription. This optimization enables handling of hour-long audio recordings without performance degradation.

EditorAudioPlayer controls audio playback with synthesis capabilities, implementing precise audio-visual synchronization through the Web Audio API. The component manages multiple audio contexts for original recordings and synthesized playback, with sub-sample precision timing and support for looping, variable playback speed, and pitch shifting.

**Analysis Components** provide quantitative tools for examining completed transcriptions. The AnalyzerComponent includes sophisticated query systems for filtering transcriptions by musical criteria, supporting complex multi-parameter queries with Boolean logic operators. The query system can search across 15 different musical categories (pitch sequences, trajectory types, section labels, performance techniques) with four designator types (includes, excludes, starts with, ends with).

Pitch prevalence visualization tools implement both emic (culturally-informed) and etic (measurement-based) analytical approaches. The system can segment transcriptions by duration, phrase boundaries, or section boundaries, generating heatmap visualizations that reveal melodic emphasis patterns. Pattern analysis capabilities identify recurring melodic sequences of user-specified lengths, implementing efficient substring matching algorithms to reveal stylistic consistencies within and across performances.

**Collection Components** enable users to organize recordings and transcriptions into research-focused groupings, supporting collaborative workflows for pedagogical, research, and creative applications. The collection system implements sophisticated permission management with role-based access control, invitation systems, and real-time collaborative features.

**Audio Recording Components** handle file uploads through drag-and-drop interfaces, metadata entry with validation, and basic audio playback outside the editor environment. The upload system supports multiple audio formats with automatic conversion, peak normalization, and metadata extraction.

**File Management Components** provide interfaces for creating new transcriptions, managing permissions, and linking transcriptions to their source recordings. The system includes version control capabilities, enabling users to track changes over time and maintain multiple versions of transcriptions.

### Advanced Frontend Technologies

The frontend employs several cutting-edge technical innovations to meet the platform’s specialized requirements:

**Web Audio API Integration** powers the real-time synthesis system through custom AudioWorklet implementations. The karplusStrong.worklet.js implements physical modeling synthesis for plucked strings, using ring buffer delay lines and filtered feedback loops to simulate string resonance. The klattSynth2.worklet.js provides formant synthesis for vocal modeling, implementing the complete Klatt synthesizer with cascade and parallel formant branches, aspiration modeling, and dynamic formant frequency control.

**D3.js Visualization Framework** enables sophisticated interactive visualizations throughout the platform. The transcription renderer uses D3’s path generation, zoom behavior, and event handling systems to create responsive musical notation displays. Analysis visualizations leverage D3’s statistical and color interpolation capabilities to generate publication-quality charts and graphs.

**Web Workers for Background Processing** handle computationally intensive tasks without blocking the user interface. The spectrogramWorker.ts implements a sophisticated pipeline for processing compressed spectral data, including real-time scaling, intensity adjustment, and colorization using D3 color maps. The worker system implements priority queuing to ensure visible spectrogram regions are rendered first while background processing continues for off-screen areas.

**Intersection Observer API** enables lazy loading of visual components based on viewport intersection. This optimization is crucial for handling long-form musical content, ensuring that only visible portions of spectrograms, transcriptions, and analysis visualizations are actively rendered.

**Custom Audio Worklets** provide high-performance audio processing within the browser. The worklet system implements parameter automation for dynamic synthesis control, enabling real-time modification of synthesis parameters during playback. The chikari synthesis worklet models up to four sympathetic strings with independent frequency and amplitude control.

### Backend Infrastructure and API Design

Our backend is implemented as a Node.js Express server that coordinates between the web application, MongoDB database, and Python audio processing components. The server architecture implements a microservices-inspired pattern while maintaining deployment simplicity through monolithic hosting.

**Database Management** uses MongoDB with sophisticated aggregation pipelines for derived data generation. The system implements compound indexing strategies for complex musical queries, enabling sub-second search times across large transcription databases. Collection design balances document embedding for frequently accessed related data with references for stable entities.

The database implements automated maintenance through scheduled aggregation tasks. The generateAudioRecordingsDB pipeline transforms hierarchical audioEvent documents into flattened audioRecording documents, enabling efficient queries across recording metadata. This approach provides the benefits of both hierarchical organization and flat query performance.

**Audio Processing Pipeline** spawns Python scripts for computationally intensive tasks like pitch extraction, spectrogram generation, and melograph creation. The system implements job queuing for long-running audio analysis tasks, with progress reporting and error handling. Python processes communicate through file-based data exchange using compressed numpy arrays and JSON metadata.

The audio pipeline includes automated spectral analysis using constant-Q transforms for music-appropriate frequency resolution, fundamental frequency estimation using multiple algorithms (SWIPE, CREPE) with confidence scoring, and automated tonic identification for Hindustani music analysis.

**File Handling and Storage** manages audio uploads, transcription exports, and generated visualization files. The system supports multiple audio formats through server-side conversion using FFmpeg, implementing automatic quality adjustment and normalization. Generated spectral data is compressed using pako (gzip) compression and served through CDN distribution for optimal performance.

**Analysis Services** power the query system by executing complex searches across transcription databases and generating real-time visualizations of musical patterns and prevalence data. The search engine implements full-text indexing for lyrical content, structural indexing for musical patterns, and geospatial indexing for performance locations.

### Audio Synthesis Engines and Processing Infrastructure

IDTAP implements a sophisticated audio synthesis system using custom AudioWorklet processors that provide real-time, low-latency audio generation specifically designed for Indian classical music transcription verification. The system employs three specialized synthesis engines, each optimized for different instrumental timbres and performance techniques.

#### **Physical Modeling Synthesis Engines**

**Karplus-Strong Algorithm for Sitar Synthesis** (src/audioWorklets/karplusStrong.worklet.js) implements the classic Karplus-Strong algorithm with Indian classical music-specific enhancements. The system uses a circular delay buffer of 2048 samples with efficient bit-masking for array indexing (writePtr = (readPtr + period \* sampleRate) & 2047), enabling precise frequency control from 50-2000 Hz. The implementation includes a first-order lowpass filter for realistic string damping, with cutoff parameter controlling the rate of energy decay. The synthesis supports real-time frequency modulation for glissando effects and burst excitation for plucking articulations.

**Sarangi Bowed String Synthesis** (src/audioWorklets/sarangi.worklet.js) implements a complex dual delay line architecture with cascaded filtering to model the acoustic characteristics of bowed chordophones. The system uses two synchronized delay lines with feedback gain of 0.98, creating the characteristic sustain and resonance of bowed instruments. A custom BiquadFilter class provides resonant filtering at five specific frequencies ([185, 275, 405, 460, 530] Hz) that simulate the harmonic resonances of sarangi sympathetic strings. The bow gain parameter controls the excitation strength, while bandpass filtering on white noise input simulates the complex friction dynamics of bow-string interaction.

**Klatt Formant Synthesizer for Vocal Modeling** (src/audioWorklets/klattSynth2.worklet.js) implements a sophisticated formant synthesizer based on the Klatt model, extensively modified for singing voice production. The system supports 47 controllable parameters including six formant frequencies with adjustable bandwidths and amplitudes. The implementation includes both cascade and parallel synthesis branches, enabling realistic modeling of voiced and unvoiced components. The system supports vowel morphing through 15 different vowel configurations with real-time interpolation, essential for phonetic transcription using IPA symbols embedded in trajectory articulations.

#### **Dynamic Synthesis Deployment Architecture**

The synthesis system automatically deploys appropriate engines based on transcription instrumentation (src/comps/editor/audioPlayer/Synths.vue). For sitar transcriptions, the system spawns both main string synthesis (Karplus-Strong) and sympathetic string synthesis (chikari) nodes. The chikari system models up to four sympathetic strings with independent frequency control and strum timing. Sarangi transcriptions utilize bowed string synthesis with dynamic bow pressure control, while vocal transcriptions employ formant synthesis with automatic vowel target selection based on phonetic annotations.

#### **Real-time Audio Capture and Looping System**

IDTAP implements a sophisticated audio capture system (src/audioWorklets/captureAudio.worklet.js) that enables real-time recording of synthesis output for looping playback. The capture processor uses configurable buffer sizes (up to 20 seconds at 48kHz) with separate channels for main synthesis and sympathetic string outputs. The system implements edge fading (1000-sample crossfade) to prevent clicks during loop transitions, enabling seamless repetition of synthesized transcription segments for detailed analysis and verification.

#### **Pitch Manipulation and Time-Domain Processing**

**Pitch Shifting Implementation** uses phase vocoder-based processing (python/pitch\_shifting/pitch\_shift.py) with librosa’s STFT processing using 2048-point FFT and 512-sample hop length. The algorithm supports ±200 cents pitch adjustment while maintaining temporal characteristics, crucial for comparative analysis of transcriptions at different tuning standards. The implementation includes time-domain resampling to maintain original duration after pitch modification.

**Time-Stretching Capabilities** utilize the RubberBand library (src/audioWorklets/rubberband-processor.worklet.js) for high-quality tempo modification without pitch alteration. The system supports speed ranges from 0.5x to 2x with minimal latency, enabling detailed analysis of rapid melodic passages. The implementation includes quality settings optimized for musical content rather than speech.

#### **Python-based Audio Analysis Pipeline**

**Spectrogram Generation Using Constant-Q Transform** (python/visualization\_tools/make\_spec\_data.py) employs Essentia’s NSGConstantQ algorithm for logarithmic frequency analysis specifically suited to musical content. The system generates 72 bins per octave for microtonal precision, with configurable frequency ranges typically spanning 75-2400 Hz. The implementation includes hierarchical compression using gzip encoding for efficient web transmission, with automatic colorization using perceptually uniform color spaces.

**Multi-Algorithm Pitch Extraction** (python/essentia/make\_melograph.py) implements multiple concurrent pitch detection algorithms: PitchYin for fundamental frequency estimation, PitchYinProbabilistic for confidence scoring, and PredominantPitchMelodia for melody extraction in polyphonic content. The system uses 1024-point analysis windows with 512-sample hop size, providing sub-10ms temporal resolution for detailed melodic contour analysis.

**Indian Classical Music Tonic Estimation** (python/process\_audio.py) uses Essentia’s TonicIndianArtMusic algorithm specifically designed for Indian classical music analysis. The algorithm automatically detects the tonic frequency with a maximum frequency limit of 200 Hz, appropriate for typical SA positioning in Hindustani music. The system integrates tonic estimates with the raga tuning system for automatic pitch calibration and microtonal analysis.

#### **Audio File Format Management and Storage**

**Multi-Format Audio Pipeline** supports comprehensive format handling: WAV files (44.1kHz, stereo) for analysis, MP3 (192kbps) for web streaming, and Opus for high-efficiency transmission. The system uses FFmpeg for format conversions with quality-optimized encoding parameters. Storage architecture organizes files by format (/audio/wav/, /audio/mp3/, /audio/opus/) with automated peak data extraction for waveform visualization.

**Hierarchical Peak Data Generation** (python/process\_audio.py) generates multi-level peak data using numpy’s reduceat operations with configurable block sizes (2^11 base, 5 levels). This hierarchical approach enables efficient waveform visualization at multiple zoom levels without performance degradation, crucial for handling hour-long recordings.

This comprehensive audio infrastructure enables IDTAP to provide real-time synthesis verification of transcriptions while supporting the computational analysis requirements of digital musicology research. The combination of browser-based synthesis and server-side analysis creates a unique platform capable of bridging qualitative transcription practices with quantitative analytical methodologies, specifically designed for the microtonal and expressive characteristics of Indian classical music traditions.

### Spectrogram and Visualization System

The platform’s visualization system implements a sophisticated pipeline for processing and displaying spectral analysis data:

**Spectrogram Worker Pipeline** processes compressed spectral data through multiple stages: initial decompression of pako-compressed numpy arrays, logarithmic frequency scaling for musical visualization, dynamic intensity adjustment through power function mapping, and colorization using D3 color scales with perceptually uniform color spaces.

The worker implements a task dispatching system that prioritizes visible spectrogram regions while background processing continues for off-screen areas. This approach enables smooth panning and zooming across hour-long recordings while maintaining responsive user interaction.

**Canvas-Based Rendering** divides large spectrograms into manageable canvas segments (maximum 1000 pixel width) to avoid browser canvas size limitations. The rendering system implements smooth scrolling through canvas recycling and predictive pre-rendering of adjacent regions.

**Intersection Observer Integration** triggers spectrogram rendering only for visible canvas segments, dramatically reducing memory usage and rendering overhead for long-form musical content. The system implements intelligent pre-loading margins to ensure smooth scrolling while minimizing resource usage.

### Database Architecture and Query Optimization

The MongoDB database architecture implements sophisticated indexing and aggregation strategies optimized for musical data queries:

**Collection Design Strategy** balances document embedding for frequently accessed related data (musicians within recordings) with references for stable entities (raga names, user IDs). This hybrid approach optimizes for both complex analytical queries and real-time editor performance.

**Aggregation Pipeline Architecture** generates derived collections for query optimization. The audioRecordings collection is generated from audioEvents through complex aggregation pipelines that flatten hierarchical data while preserving relationships. This approach enables efficient queries across recording metadata while maintaining organizational hierarchy.

**Indexing Strategy** implements compound indexes for musical pattern queries, geospatial indexes for location-based searches, and text indexes for lyrical content. The indexing system balances query performance with storage efficiency, implementing partial indexes for frequently queried subsets.

**Permission System Architecture** implements two-tier permissions with legacy string-based permissions and modern explicitPermissions objects with granular access control arrays. The system includes invite-based sharing, role-based access control, and audit logging for collaborative research workflows.

### Analysis and Query Systems

The analysis suite provides both emic (culturally-informed) and etic (measurement-based) analytical approaches through sophisticated query and visualization systems:

**Query Engine Architecture** allows filtering by musical criteria including pitch sequences, trajectory types, section labels, and performance techniques. Users can construct complex queries combining multiple criteria with Boolean logic operators, choosing between strict matching (exact sequences) or loose matching (allowing gaps in patterns).

The query system implements efficient substring matching algorithms for melodic pattern detection, using suffix arrays and rolling hash techniques for real-time pattern identification across large transcription databases.

**Pitch Prevalence Analysis** splits transcriptions into durational windows using either time-based segmentation (etic) or phrase/section-based segmentation (emic), generating visualizations that show the relative time spent on different pitches across a performance. The system implements statistical analysis of pitch distributions with support for both absolute and proportional measures.

**Pattern Analysis Implementation** identifies recurring melodic sequences of user-specified lengths, implementing efficient algorithms for pattern discovery across multiple dimensions (pitch, rhythm, articulation). The system reveals stylistic consistencies within and across performances through statistical clustering and visualization.

## Development Timeline and Major Milestones

The development of IDTAP has unfolded across distinct phases, each marked by significant technical achievements and architectural decisions that shaped the platform’s current capabilities.

### Phase 1: Foundational Development (2021 - Early 2022)

The project began with JM’s initial exploration of trajectory-based musical representation, establishing the theoretical framework that would guide all subsequent technical development. This foundational period focused on proving the basic concept could work in a web environment, with early commits revealing the establishment of core Vue.js architecture and the first implementations of trajectory placement in a browser-based editor.

**Core Architecture Establishment** during this phase included adopting Vue.js 3 with the Composition API for reactive interface management, establishing the basic data structures for representing trajectories and pitches through TypeScript interfaces, and implementing the initial mathematical functions for trajectory interpolation and pitch calculation. The git history shows early experimentation with SVG-based rendering and the first attempts at audio-visual synchronization.

**Initial Audio Integration** emerged during this period with basic audio player implementation using the Web Audio API and preliminary trajectory morphing capabilities. Key technical decisions included choosing logarithmic frequency representation for pitch visualization and establishing the coordinate system that would persist throughout the platform’s development.

**Database Foundation** was established using MongoDB with initial collection designs for transcriptions and user data. Early commits show the development of basic CRUD operations and the first attempts at complex aggregation queries for musical data.

### Phase 2: Full-Time Development Acceleration (Summer 2022)

The transition to full-time development in summer 2022 marked a dramatic acceleration in platform capabilities. This period saw fundamental overhauls of the editor interface and the implementation of core features that define IDTAP’s current functionality.

**Editor Redesign and Phrase Management** became a major focus, with the development of moveable phrase divisions, trajectory dragging capabilities, and integrated *chikari* string transcription. The git history shows intensive work on phrase positioning, with commits like “trajectory morphing,” “phrase divs,” and “moveable phrase” indicating the emergence of the flexible editing environment users experience today.

Technical achievements during this period included implementing drag-and-drop trajectory manipulation with sub-pixel precision, developing the phrase grid system for multi-instrument transcription, and establishing the event-driven architecture that coordinates between editing components. The system gained support for real-time collaboration features and comprehensive undo/redo functionality.

**Synthesis System Implementation** represented another major milestone. The integration of dampening controls, vibrato trajectories, and multi-instrument synthesis capabilities emerged during this period. Commits referencing “dampener playback,” “synth damping,” and “vibrato adding” document the development of the custom synthesis engines that allow transcribers to aurally verify their work.

The synthesis system implementation involved extensive research into physical modeling algorithms, culminating in custom implementations of Karplus-Strong synthesis for plucked strings and early versions of formant synthesis for vocal modeling. The system gained the ability to process complex trajectory data in real-time, with synthesis parameters automatically derived from transcription content.

**Audio Integration and Visualization** saw significant advancement with improvements to spectrogram display, audio playback synchronization, and looping capabilities. The development of proper phrase-to-audio alignment and the integration of Python-generated spectrograms into the web interface occurred during this intensive development period.

Key technical developments included implementing Web Workers for background spectrogram processing, developing the canvas-based rendering system for large spectral displays, and establishing the data compression and streaming protocols that enable handling of hour-long recordings.

### Phase 3: Feature Expansion and Analysis Tools (Late 2022 - 2023)

This phase focused on expanding IDTAP beyond basic transcription toward comprehensive analysis and collaboration capabilities.

**Collections System Development** enabled users to organize transcriptions into research-focused groupings. The implementation of permission systems, collaborative features, and collection-based organization reflected the platform’s evolution from a personal transcription tool toward a community resource supporting diverse research methodologies.

Technical achievements included developing role-based access control with invite-based sharing, implementing real-time collaborative editing features, and establishing the audit logging system for tracking transcription changes. The collections system gained sophisticated filtering and search capabilities, enabling users to quickly locate content across large archives.

**Analysis Suite Implementation** introduced the query system, pitch prevalence visualization, and pattern analysis tools that transform IDTAP from a transcription platform into an analytical environment. Commits documenting “analysis visualizing segments,” “query save past queries,” and “downloadable analysis outputs” mark the emergence of quantitative capabilities that support empirical musicological research.

The analysis system development involved implementing efficient algorithms for melodic pattern detection, developing statistical analysis tools for pitch distribution studies, and creating interactive visualization components using D3.js. The query system gained support for complex Boolean logic queries across multiple musical dimensions.

**Multi-instrument Support** expanded beyond the initial sitar focus to include vocal and sarangi transcription capabilities. This required developing instrument-specific trajectory sets, articulation systems for phonetic transcription using IPA symbols, and the custom sarangi synthesis engine.

Technical challenges included extending the data model to support multiple concurrent instrument tracks, developing the grid system for coordinating temporal relationships across instruments, and implementing instrument-specific synthesis parameters and trajectories.

**Python Integration** for computationally intensive audio processing established the hybrid architecture that continues to characterize IDTAP. The development of Python scripts for pitch extraction using multiple algorithms (SWIPE, CREPE), spectrogram generation using constant-Q transforms, and melograph creation for melodic contour visualization enabled sophisticated audio analysis while maintaining responsive web interface performance.

The Python integration involved developing robust data exchange protocols between Node.js and Python processes, implementing job queuing for long-running analysis tasks, and establishing the compressed data formats that enable efficient storage and transmission of spectral analysis results.

### Phase 4: Refinement and Testing Infrastructure (2024)

The most recent development phase has emphasized code quality, testing, and preparation for broader community adoption.

**TypeScript Migration and Testing** established comprehensive test coverage across the codebase with over 750 commits including extensive testing infrastructure. This period reflects a maturation from prototype to production-ready software. The implementation of Vitest for frontend testing and Pytest for Python components ensures reliability as the platform scales to support more users and larger datasets.

Testing infrastructure development included implementing unit tests for all core musical data structures, integration tests for the synthesis system, end-to-end tests for the transcription workflow, and performance tests for large dataset handling. The test suite gained continuous integration support with automated testing on multiple browser platforms.

**Python API Porting Project** began the systematic translation of core TypeScript models into Python equivalents, enabling computational researchers to work with IDTAP data using familiar scientific computing tools while maintaining data consistency with the web application.

The porting project involved developing Python classes that mirror the TypeScript model hierarchy, implementing serialization and deserialization methods for cross-platform data exchange, and establishing the testing framework that ensures ongoing compatibility between TypeScript and Python implementations.

**Performance Optimization** addressed scalability concerns as transcription databases grew larger. Improvements to spectrogram rendering through WebGL acceleration, database query optimization using advanced indexing strategies, and audio worklet efficiency optimizations prepared the platform for deployment at institutional scale.

Performance improvements included implementing lazy loading for large transcription datasets, optimizing MongoDB aggregation pipelines for complex analytical queries, developing client-side caching strategies for frequently accessed musical data, and integrating Web Workers for background audio processing (src/js/stretcherWorker.js) using SoundTouch.js time-stretching with priority queuing to maintain responsive user interaction. The system also implements Intersection Observer-based lazy loading for spectrogram rendering, triggering processing only for visible canvas segments to dramatically reduce memory usage and processing overhead.

**Documentation and Community Preparation** included the development of comprehensive API documentation, user guides, and the technical documentation infrastructure necessary for open-source community engagement. The platform gained extensive inline documentation, automated API documentation generation, and comprehensive user tutorials.

### Current State and Future Directions

IDTAP now represents a mature platform capable of supporting diverse research methodologies in digital musicology. The platform successfully hosts transcriptions across multiple musical traditions, supports collaborative research workflows, and provides both qualitative and quantitative analytical capabilities.

Technical achievements include real-time synthesis verification of transcriptions with sub-sample timing precision, flexible microtonal tuning systems supporting just intonation and custom temperaments, comprehensive query and analysis tools with statistical export capabilities, and data export capabilities that integrate with existing computational research workflows including R, Python, and MATLAB.

The platform’s trajectory-based representation system has proven effective for capturing musical nuances that traditional notation systems cannot adequately represent, with successful transcriptions of complex ornamental passages, microtonal melodic movements, and multi-layered performance textures that would be impossible to notate using conventional methods.

Looking forward, the established architecture provides a foundation for expansion to additional musical traditions through the development of new trajectory types and articulation systems, integration with emerging AI/ML technologies for assisted transcription and pattern recognition, and the development of pedagogical applications that could democratize access to detailed musical analysis tools.

The combination of web-based accessibility and computational power positions IDTAP to contribute significantly to the digital transformation of musical scholarship and education. The platform’s open-source trajectory creates opportunities for community-driven development and adaptation to diverse musical traditions worldwide.

The development process itself offers lessons for digital humanities projects attempting to create alternatives to established representational systems. Our experience demonstrates that building from first principles—rather than adapting existing tools—can produce more authentic and effective solutions for representing non-Western cultural practices in digital environments, albeit requiring sustained development effort and deep domain expertise.

The successful integration of sophisticated audio processing, real-time synthesis, collaborative editing, and analytical capabilities within a single web-based platform demonstrates the potential for browser-based applications to handle traditionally desktop-bound computational tasks, opening new possibilities for accessible digital humanities tools.