Lingua Workbench

Personal English Listening Study Platform

Contents

Preface

About This Guide

This documentation provides comprehensive guidance for using Lingua Workbench, a personal language learning platform for studying spoken English from authentic audio sources.

Who Should Use This Guide

This guide is intended for:

- **Language learners** who want to improve their English listening skills through TV shows and movies
- **Developers** who want to set up and customize the platform
- **Contributors** who want to understand the system architecture

How This Guide is Organized

This guide is divided into the following parts:

- **Getting Started** – Installation and first-run instructions
- **User Guide** – Step-by-step procedures for using all features
- **Reference** – Lookup information for controls, tags, and technical details
- **Troubleshooting** – Solutions to common problems
- **Appendix** – Additional reference materials

Conventions Used

This documentation uses the following conventions:

- **Bold text in boxes** indicates UI buttons or controls
- `Monospace text` indicates code, commands, or file paths
- **Bold text** indicates important terms or emphasis

Notices

Third-Party Services

This application uses the following third-party services:

- **OpenAI API** – For phonetic analysis and dictionary features
- **Whisper API** – For speech-to-text transcription

Usage of these services is subject to their respective terms of service and pricing.

Revision History

This section documents the revision history of Lingua Workbench.

Version	Date	Author	Changes
1.0	2026-01-08	Edith Tang	• Initial release • Audio management (upload, chunk, slice) • Whisper transcription integration • AI phonetic analysis (Sound Script) • Dictionary with bilingual examples • Playback speed control • Favorite marking • Time adjustment arrows • Independent slice deletion

Part I. Getting Started

Chapter 1. Product Overview

Lingua Workbench is a personal language learning platform for studying spoken English.

What is Lingua Workbench?

Lingua Workbench helps you analyze and study spoken English from audio sources like TV shows and movies. It provides tools for:

- Uploading and managing audio files organized by drama/season/episode
- Creating precise audio slices from longer recordings
- AI-powered transcription and phonetic analysis
- Context-aware dictionary lookups

Key Features

The platform includes:

- **Audio Slicer** - Interactive waveform selection and region management
- **Sound Script Analysis** - AI-generated phonetic breakdown of phrases
- **Dictionary Integration** - Bilingual definitions with contextual examples
- **Playback Controls** - Variable speed, loop, and region playback

Target Audience

This tool is designed for English learners who want to master natural spoken English through authentic audio sources.

Chapter 2. Installing Lingua Workbench

Set up the development environment for Lingua Workbench.

Before you begin, ensure you have:

- Node.js 20+ or 22+
- Python 3.11+
- ffmpeg (for audio processing)
- Git

Lingua Workbench consists of a Vue 3 frontend and Django backend. Both need to be set up separately.

1. Clone the repository:

   ```
   git clone https://github.com/uufishtxl/lingua-workbench.git
   cd lingua-workbench
   ```

2. Set up the frontend:
   a. Navigate to the frontend directory: `cd frontend`
   b. Install dependencies: `npm install`

3. Set up the backend:

   a. Navigate to the backend directory: `cd ../backend`

   b. Create a virtual environment: `python -m venv venv`

   c. Activate the virtual environment: `venv\Scripts\activate` (Windows)

   d. Install dependencies: `pip install -r requirements.txt`

   e. Run migrations: `python manage.py migrate`
      This creates the SQLite database automatically.

   f. **Optional:** Create a superuser for admin access: `python manage.py createsuperuser`

4. Create environment files with your API keys.

Both frontend and backend are now ready to run.

Chapter 3. Running for the First Time

Start the development servers and access the application.

Complete the installation *(on page 9)* first.

1. Start the backend server:

   ```
   cd backend
   venv\Scripts\activate
   python manage.py runserver
   ```

   The Django server starts on `http://localhost:8000`

2. In a new terminal, start the frontend:

   ```
   cd frontend
   npm run dev
   ```

   The Vite dev server starts on `http://localhost:5173`

3. In a new terminal, start the Whisper transcription service:

   ```
   cd whisper
   uv run uvicorn whisper_api:app --reload --port 8001
   ```

   The Whisper API starts on `http://localhost:8001`

4. In a new terminal, start the Huey task queue consumer:

   ```
   cd whisper
   uv run huey_consumer tasks.huey
   ```

   The task queue is now processing transcription jobs.

5. Open your browser and navigate to `http://localhost:5173`

You should see the Lingua Workbench home page. Register an account to get started, then you can start uploading audio and creating slices.

Part II. User Guide

Chapter 1. Audio Management

Audio Workflow Overview

Understanding how audio is organized and processed in Lingua Workbench.

Audio Hierarchy

Audio in Lingua Workbench is organized in three levels:

1. **Source Audio** - The original uploaded file, organized by Drama → Season → Episode
2. **Audio Chunks** - Automatically created 5-minute segments of the source audio
3. **Audio Slices** - User-created regions within chunks for focused study

Processing Flow

When you upload an audio file:

1. The file is stored and associated with a drama/episode
2. It is automatically split into 1-minute chunks
3. You can then browse chunks and create slices for study

Uploading Source Audio

Upload an audio file from a TV show or movie.

You need an audio file in MP3 or WAV format.

1. Navigate to the Audio Upload page.
   From the left sidebar, click **Audio Slicer > Load Source**.

2. Select a Drama from the dropdown (e.g., "Friends").

   Restriction:
   Creating new Dramas is currently not available to users.

3. Enter the Season and Episode numbers.

4. Optionally add an episode title.

Restriction:
Adding an episode title is currently not available to users.

5. Click **Choose File** and select your audio file.

6. Click **Upload**.

The audio is uploaded and automatically chunked. You can now browse the chunks.

Browsing Audio Chunks

View and select audio chunks for slice creation.

1. Navigate to the Audio Workbench page.
   From the left sidebar, click **Audio Slicer > Load Source**.

2. Use the filter to select a drama or episode.

3. Click on a chunk card to open it in the Audio Slicer.

The waveform is displayed and you can begin creating slices.

Chapter 2. Creating Slices

What is an Audio Slice?

Understanding audio slices and their role in language learning.

Definition

An *audio slice* is a user-defined region within an audio chunk. Each slice represents a meaningful utterance, sentence, or phrase that you want to study.

Slice Components

Each slice contains:

- **Time boundaries** - Start and end times within the chunk
- **Transcription** - The text content (via Whisper or manual input)
- **Highlights** - Selected text portions for phonetic analysis
- **Favorite flag** - Mark important sentences for review

Best Practices

- Keep slices focused on single sentences or short exchanges
- Use the time adjustment arrows to fine-tune boundaries
- Star slices that contain challenging pronunciations

Creating a Slice

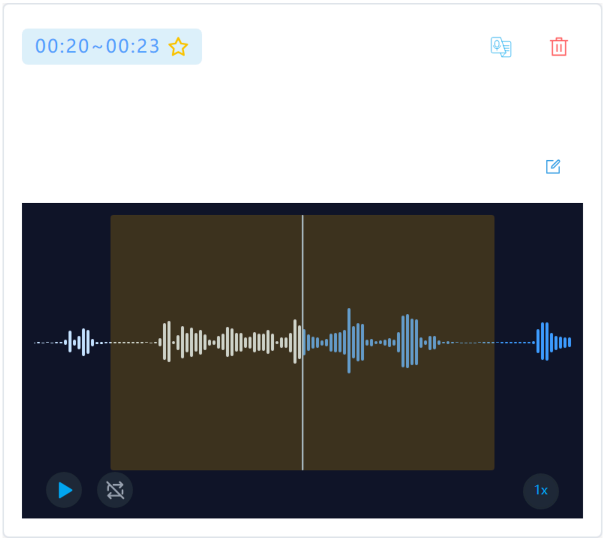
Select a region on the waveform to create an audio slice.

You must have an audio chunk open in the Audio Slicer.

1. Click and drag on the waveform to select a region.
   The selected region is highlighted in blue.

2. Release the mouse button to create the slice.
   A new SliceCard appears in the panel below.

Figure 1. Audio Slice



3. **Optional:** Fine-tune the boundaries using the ◀ and ▶ arrows on the timestamp.
   Each click adjusts the time by 0.5 seconds.

The slice is created locally. Use **Save All Changes** to persist to the server.

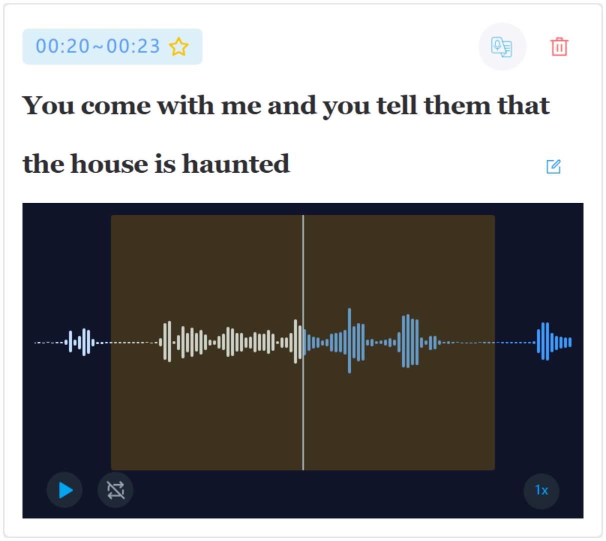
Transcribing a Slice

Use AI to automatically transcribe the audio in a slice.

1. Locate the slice you want to transcribe.
2. Click the button.
   The button shows a loading indicator while processing.
3. Wait for the transcription to complete.
   The transcribed text appears in the slice card.
4. **Optional:** Click the button to correct any transcription errors.

The slice now contains the transcribed text, ready for highlighting and analysis.

Figure 2. Text Transcription



You come with me and you tell them that the house is haunted

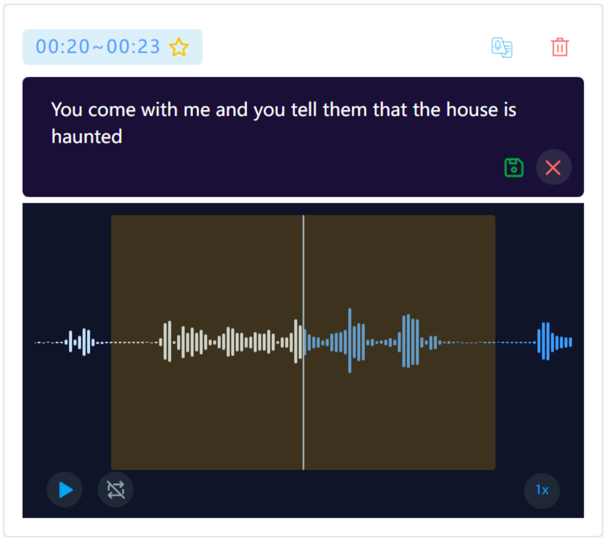
Editing Slice Text

Manually edit the transcription text of a slice.

1. Click the ✎ button at the bottom-right of the text area.

   The text becomes editable in a textarea.

   Figure 3. Editing Slice Text

   You come with me and you tell them that the house is haunted

2. Make your changes to the text.

3. Click the button to save changes.

4. **Optional:** Click the button to discard changes.

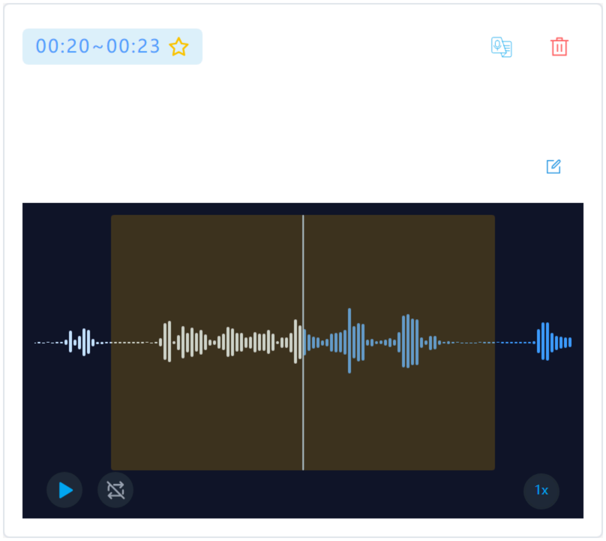
Deleting a Slice

Remove a slice from your collection.

Deleting a saved slice immediately removes it from the server. This action cannot be undone.

Click the button on the slice card.

Figure 4. Deleting a Slice



For saved slices, the deletion is immediate. For unsaved slices, only the local region is removed.

Chapter 3. Phonetic Analysis

Sound Script Analysis

Understanding AI-generated phonetic analysis.

What is Sound Script?

Sound Script is an AI-generated phonetic breakdown of spoken English phrases. It analyzes how words are actually pronounced in connected speech, identifying:

- **Reductions** - Vowels weakening to schwa sounds
- **Linking** - Words connecting across boundaries
- **Assimilations** - Sounds changing due to neighboring sounds
- **Deletions** - Sounds that are dropped (h-deletion, t-deletion)

Sound Display Notation

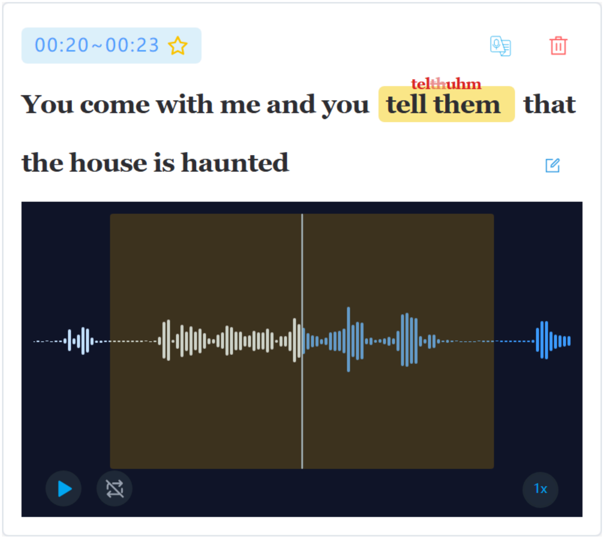
The **Sound Display** shows pronunciation using:

- Phonetic spelling (e.g., "wuh-duh-we" for "what do we")
- Strikethrough text(s) for ghost/silent sounds: `~~t~~uh`
- Hyphens to show syllable/word boundaries

Ruby Text Display

The UI displays pronunciation above the original text using Ruby annotation, making it easy to compare written and spoken forms.

Figure 5. Sound Script: Ruby Text Annotation



00:20~00:23

You come with me and you tell them that the house is haunted

tel~~th~~uhm

Highlighting Text for Analysis

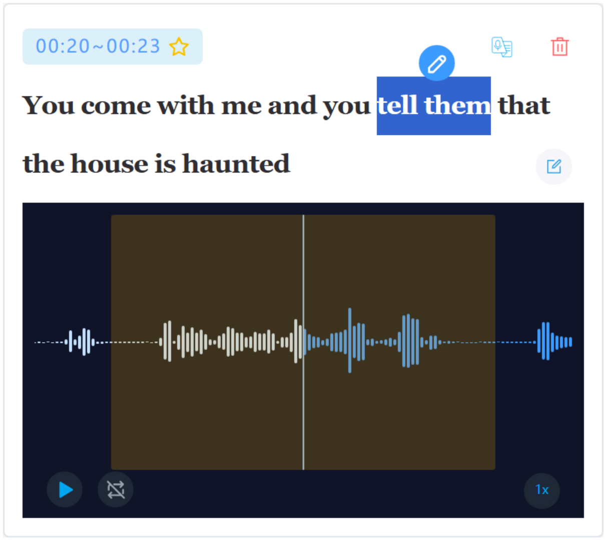
Select a phrase to analyze its pronunciation.

The slice must have transcribed text.

1. Select text in the slice by clicking and dragging.

   A highlighter icon appears near your selection.

   Figure 6. Highlighting Text for Analysis

   00:20~00:23

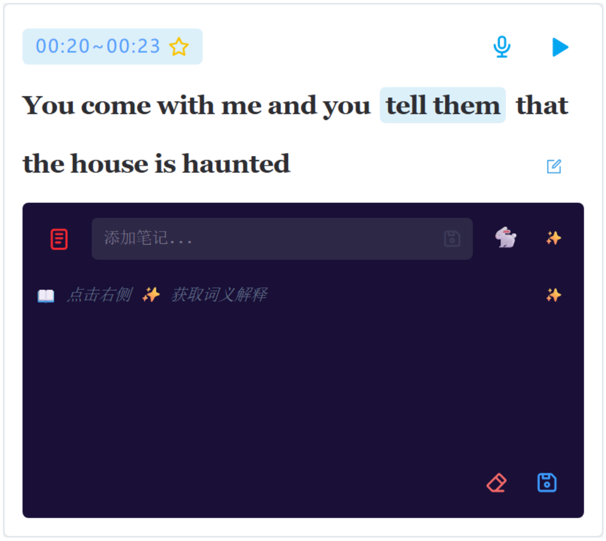
   You come with me and you tell them that the house is haunted

2. Click the icon to create a highlight.

The text is highlighted and the Highlight Editor opens below.

As you can see from the image below, the text is highlighted in blue.

Figure 7. Highlighting Text: Highlight Editor



You come with me and you tell them that the house is haunted

The highlight is created and you can now request AI analysis.

Getting AI Analysis

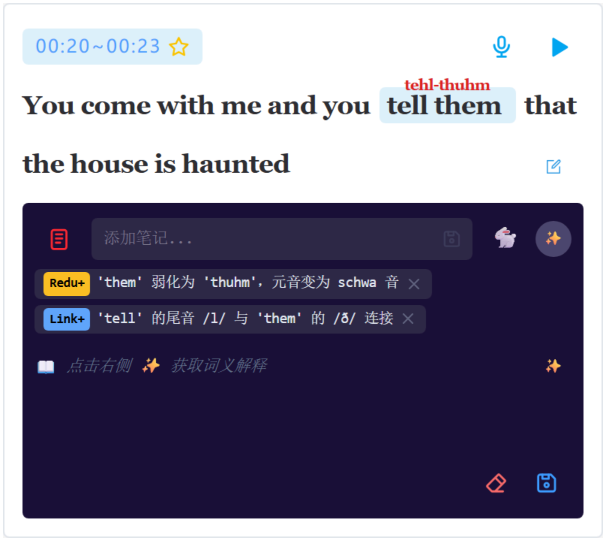
Request phonetic analysis for a highlighted phrase.

You must have a highlight selected.

1. With a highlight active, click the ✨ button in the Highlight Editor.
2. Review the phonetic tags and sound display.

The analysis shows phonetic breakdown with Ruby text annotations above the original text.

Figure 8. Reviewing Phonetic Analysis



tehl-thuhm

You come with me and you tell them that the house is haunted

Redu+ 'them' 弱化为 'thuhm'，元音变为 schwa 音

Link+ 'tell' 的尾音 /l/ 与 'them' 的 /ð/ 连接

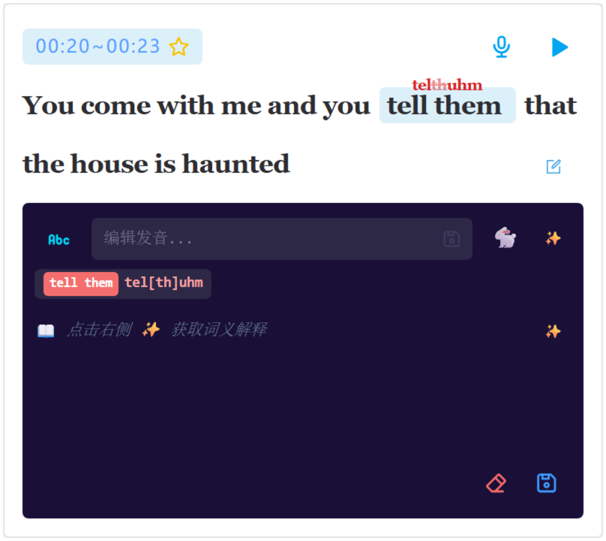
Editing Sound Display

Correct or refine the AI's pronunciation analysis.

Sometimes the AI makes mistakes in phonetic analysis. You can manually correct the sound_display values.

1. Switch to Sound Display mode by clicking the 📄 button.
2. Click on a segment to select it for editing.
   The segment's sound_display appears in the input field.
3. Edit the pronunciation text.
4. Click 💾 inside of the text area to apply changes.

Figure 9. Editing Sound Display



00:20~00:23

You come with me and you tel~~th~~uhm tell them that the house is haunted

Abc 编辑发音...

tell them tel[th]uhm

📖 点击右侧 ✨ 获取词义解释

Chapter 4. Dictionary

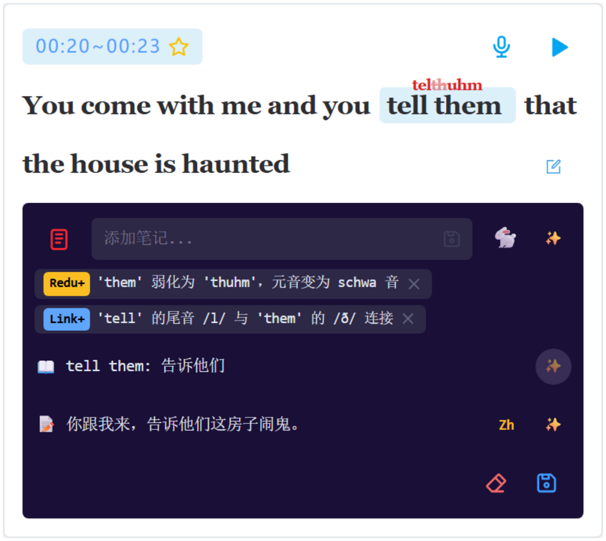
Looking Up a Phrase

Get dictionary definitions for a highlighted phrase.

1. With a highlight active, click the ✨ button.
2. Review the definition and example sentences.

The dictionary panel shows bilingual definitions with contextual examples.

Figure 10. Dictionary Analysis



00:20~00:23

telthuhm

You come with me and you tell them that the house is haunted

添加笔记...

Redu+ 'them' 弱化为 'thuhm'，元音变为 schwa 音

Link+ 'tell' 的尾音 /l/ 与 'them' 的 /ð/ 连接

tell them: 告诉他们

你跟我来，告诉他们这房子闹鬼。

Refreshing Example Sentences

Generate new example sentences for a dictionary entry.

In the Dictionary panel, click the ✨ button.

The AI generates new contextual examples in both English and Chinese.

Part III. Reference

Chapter 1. UI Reference

SliceCard Controls Reference

Reference for all controls available on a SliceCard.

Timestamp Bar

Icon	Action	Description
<	Adjust Start	Move start time 0.2s earlier
>	Adjust End	Move end time 0.2s later
★	Toggle Favorite	Mark/unmark as favorite

Action Buttons

Button	Function
🎙	Send audio to Whisper for transcription
💾	Save your changes
✕	Cancel and quit
🗑	Remove the slice (immediate for saved slices)
✎	Enter text editing mode
≡	Toggle AI Note Mode off to edit sound display
Abc	Toggle Sound Display Mode off and display the notes

Playback Controls Reference

Reference for audio playback controls.

Speed Control

Available playback speeds: 0.5x, 1x

Loop Mode

When enabled, the active region plays repeatedly until stopped.

Region Playback

Click on a Sliced Region to play that region only.

Editor Modes Reference

Reference for Highlight Editor modes.

AI Note Mode

Displays phonetic tags with their notes. Click a tag to edit its annotation.

Sound Display Mode

Displays script segments in format: [original] sound_display. Click a segment to edit its pronunciation.

Mode Toggle

Click / Abc to switch between the two modes.

Chapter 2. Phonetic Notation

Phonetic Tags Reference

Reference for phonetic tag types used in analysis.

Tag	中文	Description
reduction	弱化	Vowel reduced to schwa
linking	连读	Consonant-vowel linking across words
assimilation	同化	Sound changes due to adjacent sounds
h_deletion	H删除	Initial /h/ dropped in function words
flap_t	闪音T	T/D flapped between vowels
glottal_stop	喉塞音	T replaced with glottal stop
elision	省略	Sounds deleted entirely

Ghost Sounds Notation

Understanding the notation for silent or omitted sounds.

Bracket Notation

Sounds enclosed in square brackets `[x]` represent "ghost sounds" - phonemes that are weakened or omitted in natural speech.

Display

Ghost sounds are displayed with strikethrough styling and reduced opacity in the UI.

Examples

Notation	Meaning
`[h]e`	The /h/ in "he" is silent
`wha[t]`	The /t/ in "what" is dropped

Notation	Meaning
`uh[p]`	The /p/ is barely pronounced

Part IV. Troubleshooting

Chapter 1. Transcription Fails or Returns Empty

Resolve issues with Whisper transcription.

Symptom: Clicking the button results in empty text or an error.

Possible Causes and Solutions:

1. Check that the Whisper service is running.
   Ensure the Whisper API container is started.

2. Verify the audio region is not empty.
   The selected region must contain audible speech.

3. Check browser console for errors.
   Network or API errors will appear in the console.

4. Ensure the audio format is supported (MP3, WAV).

Chapter 2. AI Analysis Errors

Resolve issues with phonetic analysis.

Symptom: Analysis fails or returns unexpected results.

Troubleshooting Steps:

1. Check your OpenAI API key is valid and has credits.
2. Ensure the highlighted text is meaningful (not just punctuation).
3. Try a shorter phrase if the text is very long.
4. Check the Django server logs for API errors.

Keyboard Shortcuts

Quick reference for keyboard shortcuts.

Available Shortcuts

Shortcut	Action
Space	Play/Pause audio
Esc	Close active highlight editor

Note:
More shortcuts may be added in future versions.

Technology Stack

Technical details about the technologies used.

Frontend

Technology	Purpose
Vue 3	UI framework with Composition API
TypeScript	Type-safe JavaScript
Vite	Build tool and dev server
Element Plus	UI component library
Tailwind CSS	Utility-first CSS
WaveSurfer.js	Audio waveform visualization
Pinia	State management
Axios	HTTP client

Backend

Technology	Purpose
Django	Web framework
Django REST Framework	API development
SimpleJWT	JWT authentication
LangChain	LLM orchestration
OpenAI API	AI analysis and dictionary

Services

Service	Purpose
Whisper API	Speech-to-text transcription
OpenAI GPT-4	Phonetic analysis and examples

Developer Guide

Technical documentation for developers contributing to or extending Lingua Workbench.

About This Guide

This section contains architecture documentation, component designs, and implementation details intended for developers working on the Lingua Workbench codebase.

Important:
This documentation is intended for developers only. End users should refer to the main user guide.

Frontend Components

Key Vue components and their architecture:

- BaseWaveSurfer *(on page 36)* - Audio waveform visualization and region management
- AudioSlicer *(on page 38)* - Region selection workspace and slice persistence
- SliceCard *(on page 40)* - Audio slice editing with highlighting and AI analysis

Composables

Reusable logic extracted from components:

- useRecording *(on page 41)* - Microphone recording and playback
- useTranscription *(on page 43)* - Whisper API transcription
- useHighlightSelection *(on page 45)* - Text selection and highlight creation

Additional Resources

- Internal jottings: `jottings/` directory
- Tech stack reference: Technology Stack *(on page 34)*

BaseWaveSurfer Component Architecture

Technical documentation for the BaseWaveSurfer Vue component, which provides waveform visualization and region management.

Overview

The `BaseWaveSurfer` component is a Vue 3 wrapper around WaveSurfer.js, providing audio waveform visualization with region selection and loop playback capabilities.

This component serves as the foundation for all audio interactions in the application.

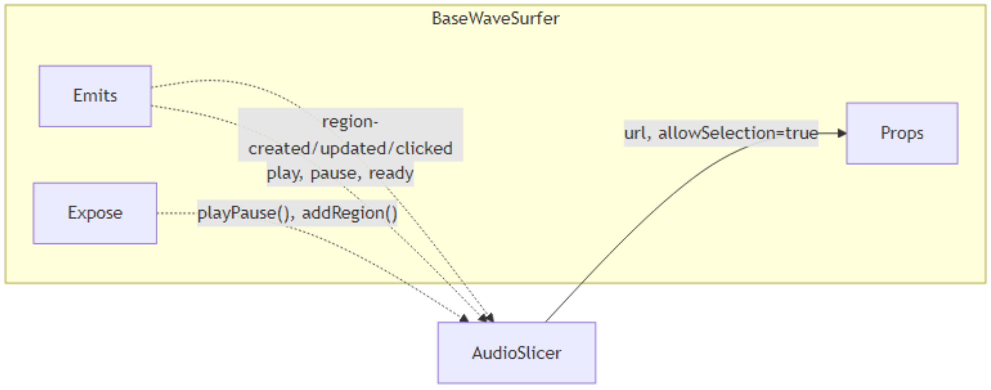
Usage Scenarios

Scenario	Parent Component	Props	Purpose
Main Waveform	`AudioSlicer`	`url`	Users drag to create multiple regions
Slice Card Waveform	`SliceCard`	`url`, `start`, `end`	Display fixed segment with loop playback

Data Flow: AudioSlicer

In the AudioSlicer context, users interact with the waveform to create regions:

Figure 11. AudioSlicer ↔ BaseWaveSurfer Data Flow

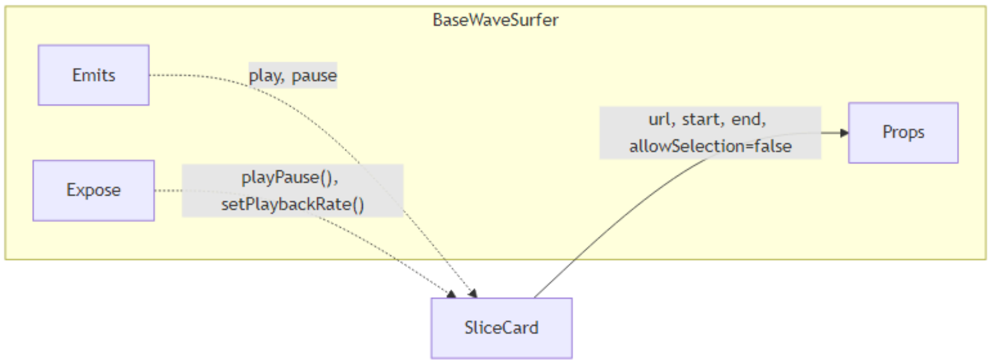


- **Props**: `url`, `allowSelection=true`
- **Events**: `region-created`, `region-updated`, `region-clicked`, `play`, `pause`, `ready`
- **Exposed Methods**: `playPause()`, `addRegion()`

Data Flow: SliceCard

In the SliceCard context, a fixed region is displayed with loop playback:

Figure 12. SliceCard ↔ BaseWaveSurfer Data Flow


BaseWaveSurfer
Emits
play, pause
Expose
playPause(), setPlaybackRate()
url, start, end, allowSelection=false
Props
SliceCard

- **Props**: `url`, `start`, `end`, `allowSelection=false`
- **Events**: `play`, `pause`
- **Exposed Methods**: `playPause()`, `setPlaybackRate()`

Key Concepts

Managed Region (Sentinel Region)

When `start` and `end` props are provided, the component creates a special region with ID `'start-end-segment'`. This region is used for visual highlighting, loop playback, and smart cursor positioning. Its `region-created` event is filtered out to avoid interfering with user-created regions.

Smart Zoom

When creating a sentinel region, the component automatically calculates zoom level so the region occupies approximately 70% of the viewport width.

Manual Loop

Loop playback is implemented manually via the `audioprocess` event, seeking back to `start` when playback reaches `end`.

Source Code

Component location: `frontend/src/components/BaseWaveSurfer.vue`

For detailed API documentation, see the component's internal jottings at `jottings/BaseWaveSurfer.md`.

AudioSlicer Component Architecture

Technical documentation for the AudioSlicer Vue component, which manages audio region selection and slice persistence.

Overview

The `AudioSlicer` component is the main workspace for creating and managing audio slices. It integrates `BaseWaveSurfer` for waveform visualization and renders multiple `SliceCard` instances for each selected region.

Component Hierarchy

```
AudioWorkbench (parent)
    └── AudioSlicer
        ├── BaseWaveSurfer (waveform)
        └── SliceCard[] (region cards)
```

Data Flow

The component manages a bidirectional data flow between the waveform and the region list:

1. **Initialization**: Parent passes `initialSlices` → component populates `regionsList` → syncs to WaveSurfer
2. **User creates region**: WaveSurfer emits `region-created` → `handleRegionCreated` adds to `regionsList`
3. **User deletes region**: SliceCard emits `delete` → `removeRegion` calls `region.remove()` → WaveSurfer emits `region-removed` → `handleRegionRemoved` filters from `regionsList`
4. **Save**: `saveRegions` collects data from all SliceCard refs and sends batch API request

Key State

State	Type	Purpose
`regionsList`	`ref<RegionInfo[]>`	All regions with metadata (id, dbId, start, end, etc.)
`sortedRegionsList`	`computed`	Regions sorted by start time for display
`sliceCardRefs`	`Map<number, SliceCard>`	References to SliceCard instances by index
`isDirty`	`ref<boolean>`	Tracks unsaved changes
`activeRegion`	`let`	Currently playing region (non-reactive)

Delete Flow (Event Chain)

The delete operation demonstrates a two-step event chain:

```
User clicks delete button on SliceCard

    ↓

SliceCard emits 'delete' event with region id

    ↓

removeRegion(id) called

    ├── Sends DELETE request to backend (if dbId exists)

    └── Calls region.remove() on WaveSurfer

            ↓

WaveSurfer emits 'region-removed' event

    ↓

handleRegionRemoved filters regionsList
```

Auto-Save Features

- **Token expiration**: Listens for `save-before-expiration` window event
- **Page unload**: Warns user via `beforeunload` event if there are unsaved changes

Source Code

Component location: `frontend/src/components/AudioSlicer.vue`

SliceCard Component

Technical documentation for SliceCard, the core audio slice editing component.

Overview

SliceCard is a comprehensive component for editing individual audio slices. It provides text transcription, highlight annotation, AI-powered analysis, and audio playback within a single card UI.

Props

Prop	Type	Description
`url`	string	Audio chunk URL
`start / end`	number	Slice time range (seconds)
`region`	object	Region info (id, start, end, originalText)
`initialHighlights`	HighlightData[]	Saved highlight data from backend
`initialPronunciationHard`	boolean	Pronunciation difficulty marker
`initialIdiom`	boolean	Idiom/vocabulary marker

Emits

- `delete` - Delete this slice
- `adjust-start` - Adjust start time by offset
- `adjust-end` - Adjust end time by offset
- `update-markers` - Update pronunciation/idiom markers

Composables Used

- useRecording *(on page 41)* - Microphone recording
- useTranscription *(on page 43)* - Whisper transcription
- useHighlightSelection *(on page 45)* - Text selection & highlighting

Key Features

Text Transcription

Uses Whisper API via `useTranscription` to convert audio to text

Text Highlighting

Select text to create highlights, managed by `useHighlightSelection`

AI Analysis

Sound script analysis and dictionary lookup via HighlightEditor

Audio Playback

Embedded BaseWaveSurfer with loop and speed controls

Recording

Record pronunciation practice via `useRecording`

Code Structure

The component follows Vue SFC best practices:

```
imports

props / emits

refs (state)

computed

composables

methods

watch

onMounted / onUnmounted

defineExpose
```

Exposed Methods

`getSliceData()`

Returns server-ready data structure for saving. Called by parent AudioSlicer.

Source Code

Location: `frontend/src/components/SliceCard.vue`

useRecording Composable

Technical documentation for the useRecording composable, which handles microphone access, audio recording, and playback.

Overview

The `useRecording` composable provides audio recording functionality using the browser's MediaStream Recording API. It manages microphone permissions, records audio to memory, and provides playback controls.

State

State	Type	Description
`isRecording`	`ref<boolean>`	Whether recording is in progress
`recordedAudioUrl`	`ref<string|null>`	Object URL of recorded audio (for playback)

Internal variables (not exposed):

- `mediaRecorder` - MediaRecorder instance from MediaStream Recording API
- `audioChunks` - Array of Blob chunks collected during recording

Methods

Method	Description
`startRecording()`	Request mic permission, set up MediaRecorder, start recording
`stopRecording()`	Stop recording, release microphone hardware
`toggleRecording()`	Start or stop based on current state
`playRecording()`	Play recorded audio from URL
`clearRecording()`	Revoke URL and free memory

Recording Flow

```
startRecording()

    ↓

navigator.mediaDevices.getUserMedia({ audio: true })

    ↓ (user grants permission)

new MediaRecorder(stream)

    ↓

mediaRecorder.ondataavailable → push chunks to array

mediaRecorder.onstop → create Blob → create Object URL

    ↓
```

```
mediaRecorder.start()

    ↓

isRecording = true
```

Key Browser APIs

navigator.mediaDevices.getUserMedia()

Prompts user for microphone/camera permission, returns MediaStream

MediaRecorder

Converts live media stream into storable data chunks (Blobs)

URL.createObjectURL(blob)

Creates temporary browser-accessible URL from Blob

URL.revokeObjectURL(url)

Releases memory when URL is no longer needed

Lifecycle

The composable automatically cleans up on component unmount via `onUnmounted` hook, releasing the recorded audio URL to prevent memory leaks.

Source Code

Location: `frontend/src/composables/useRecording.ts`

useTranscription Composable

Technical documentation for the useTranscription composable, which handles audio extraction and Whisper API transcription.

Overview

The `useTranscription` composable provides audio-to-text transcription using the Whisper API. It handles extracting audio segments, submitting to the API, and polling for results.

State

State	Type	Description
`isTranscribing`	`ref<boolean>`	Whether transcription is in progress

Methods

`transcribe(options): Promise<string | null>`

Transcribe an audio segment. Returns transcribed text or null if failed/cancelled.

Options:

- `audioUrl` - Chunk audio URL
- `startTime` - Segment start time (seconds)
- `endTime` - Segment end time (seconds)

Transcription Flow

```
transcribe(options)
    ↓
[Guard] if isTranscribing → return null (prevent duplicate calls)
    ↓
isTranscribing = true
    ↓
extractAudioSegment(url, start, end) → WAV Blob
    ↓
transcribeAudio(blob) → { task_id }
    ↓
pollTaskUntilComplete(task_id) → poll until complete
    ↓
return result (string) or catch → return null
    ↓
finally: isTranscribing = false
```

Dependencies

- `extractAudioSegment` - Utility to extract WAV Blob from time range
- `transcribeAudio` - Submits audio to Whisper API, returns task_id
- `pollTaskUntilComplete` - Polls task status until result is ready

Error Handling

- **Duplicate prevention**: Returns null immediately if already transcribing
- **API errors**: Caught and logged, returns null
- **Cleanup**: `isTranscribing` always reset in `finally` block

Source Code

Location: `frontend/src/composables/useTranscription.ts`

useHighlightSelection Composable

Handles text selection and highlight creation within SliceCard.

Overview

The `useHighlightSelection` composable manages text selection, highlighter icon positioning, and highlight creation. Extracted from SliceCard.vue during refactoring.

Options

Option	Type	Description
`containerRef`	Ref<HTMLElement>	Reference to text container element
`currentText`	() => string	Function returning current text content
`isEditingMode`	() => boolean	Function checking if in edit mode
`onHighlightCreated`	(Hili) => void	Callback when highlight is created

Returns

`selectedTextInfo`

Info about currently selected text (start, end, rect)

`highlighterIconVisible`

Whether highlighter icon is shown

`highlighterIconPosition`

Reactive object with top/left position

`handleTextSelection()`

Call on mouseup to process text selection

`handleHighlighterClick()`

Call when highlighter icon clicked

`resetSelection()`

Hide highlighter icon and clear selection

Selection Flow

```
User selects text (mouseup on container)

    ↓

handleTextSelection()

    ↓

[Guard] isEditingMode? → return

    ↓

window.getSelection() → validate selection

    ↓

indexOf(selectedText) in currentText → get position

    ↓

Calculate icon position relative to container

    ↓

highlighterIconVisible = true

    ↓

User clicks highlighter icon

    ↓

handleHighlighterClick() → create Hili object

    ↓

onHighlightCreated(highlight) → callback to parent

    ↓

resetSelection()
```

Key Implementation Details

- **Ruby text handling**: Uses string indexOf instead of DOM offset to avoid interference
- **100ms delay**: handleWindowMouseUp delays check to let highlighter click register first
- **Auto cleanup**: Registers/unregisters global mouseup listener via lifecycle hooks

Source Code

Location: `frontend/src/composables/useHighlightSelection.ts`

Glossary

Chunk

A fixed-duration segment (e.g., 30 seconds) automatically generated when source audio is uploaded.

Chunks serve as browsable units from which users can create slices.

Drama

A TV show, movie, or podcast series that serves as the source of audio materials.

Dramas

Each drama contains multiple episodes organized by season.

Slice

A short audio segment extracted from a source file, used as a study unit for listening practice.

Slices

Each slice contains a transcription and can be analyzed for phonetic features.

Sound Script

An AI-generated phonetic analysis that shows how words are actually pronounced in connected speech.

Sound scripts highlight features like liaisons, reductions, and assimilations.

Transcription

The text representation of spoken audio, generated by AI (Whisper) or entered manually.

Transcriptions can be edited and serve as the basis for phonetic analysis.