Technical Documentation for Tonnetz Player

Table of Contents:

Introduction

Tonnetz Player is a browser-based app designed to help users analyze and engage with musical harmony. This tool allows users to input and modify notes, then play them using the Tonnetz Grid. The project aims to enhance music harmony lessons and support research in the field.

Key Technologies and Frameworks Used:

- React.js
- Vite.js
- Tone.js
- GSAP
- TailwindCSS

Codebase Structure

The codebase is organized into several directories, each serving a specific purpose. Below is the structure of the project:

```
index.html
public/
src/
    App.jsx
    assets/
    components/
        Chords/
            Dichords.jsx
            Notes.jsx
            Trichords.jsx
        Circles/
            CircleView.jsx
            DragZoomCircle.jsx
        Header/
            Header.jsx
        NoteMap/
            NoteMap.jsx
            noteUtils.jsx
        PianoRoll/
            ClearNotes.jsx
            Connector.jsx
            midiWorker.js
            PianoRoll.jsx
            PianoRollGrid.jsx
            PlayHead.jsx
        sidebar/
            SideBar.jsx
        ToneGenerator/
            ToneGenerator.jsx
        TonnetzGrid/
            BottomDrawer.jsx
            DragZoomSvg.jsx
            TonnetzSelector.jsx
            TonnetzView.jsx
    context/
        PlayingContext.jsx
    index.css
    main.jsx
tailwind.config.js
vite.config.js
```

```
package.json
postcss.config.js
README.md
```

Component Overview

1. App

Description: The main entry point of the Tonnetz Player application. It sets up the primary layout, initializes the tour using Intro.js, and renders the main components of the application.

Key files:

- App.jsx

Dependencies:

- **React Hooks:**
  - useState: Manages the state of notes.
  - useRef: Creates a reference for the Intro.js tour instance.
  - useEffect: Initializes the Intro.js tour when the component mounts.
- **Libraries:**
  - IntroJs: Provides an interactive tour for the application.
  - react-icons: Provides the FaInfoCircle icon for starting the tour.
- **Related Components:**
  - CircleView: Displays notes and chords in a circular layout.
  - TonnetzView: Displays the Tonnetz grid for exploring musical relationships.
  - Sidebar: Provides functionalities for uploading, editing, and exporting notes.
  - PlayingProvider: Context provider for managing the playing state.
  - Header: Displays the header of the application.

Code Flow:

- **State and References:**
  - notes: State variable to manage the list of notes.
  - tour: Reference to store the Intro.js tour instance.
- **Effect Hook:**
  - The useEffect hook initializes the Intro.js tour when the component mounts.
  - Defines the steps for the tour, including elements to highlight and descriptions.
- **Start Tour Function:**

    - startTour: Function to start the Intro.js tour when the FaInfoCircle icon is clicked.

- **Rendering:**
    - The PlayingProvider wraps the entire application to provide context for managing the playing state.
    - The Header component is rendered at the top.
    - The FaInfoCircle icon is positioned fixed at the top-left corner to start the tour.
    - The main content is displayed in a flex container, including TonnetzView, CircleView, and Sidebar.

2. Tonnetz Grid

Description: The Tonnetz grid is a visual representation of musical relationships, particularly focusing on the harmonic and melodic connections between notes. It is used to explore and manipulate musical structures interactively.

Key Files:

- **TonnetzView.jsx**: Main component rendering the Tonnetz grid.
- **DragZoomSvg.jsx**: Handles the interactive SVG rendering, including zoom and drag functionalities.
- **TonnetzSelector.jsx**: Provides a UI for selecting different Tonnetz configurations.
- **Notes.jsx**: Renders individual notes within the Tonnetz grid.
- **noteUtils.jsx**: Utility functions for note-related operations.
- **ToneGenerator.jsx**: Handles audio playback for notes and chords.

Dependencies

- **React Hooks**:
    - useState: Manages state for the Tonnetz grid, selected notes, and transformations.
    - useEffect: Handles side effects such as locking transformations.
    - useContext: Accesses the playing state from the context.
    - useRef: Creates references for SVG elements and Tone.js synthesizer.
- **Libraries**:
    - **Tone.js**: For audio synthesis and playback.
- **Related Components:**
    - **TonnetzView:** Main component that initializes and renders the Tonnetz grid.
    - **DragZoomSvg:** Provides interactive functionalities like zooming and dragging within the SVG.
    - **TonnetzSelector:** UI component for selecting different Tonnetz configurations.

  - **Notes:** Renders individual notes and handles interactions like clicks.
  - **ToneGenerator:** Plays notes and chords based on user interactions.
  - **noteUtils:** Utility functions for note calculations and transformations.

Code Flow

TonnetzView.jsx

- **State and References:**
  - **graph:** State variable to manage the Tonnetz intervals and type.
  - **lock:** State variable to manage whether the grid is locked for transformations.
- **Effect Hook:**
  - Initializes the Tonnetz grid and handles locking transformations.
- **Rendering:**
  - Renders the DragZoomSvg component with the Tonnetz grid.
  - Displays a message if the grid is not connected.

DragZoomSvg.jsx

- **State and References:**
  - Manages transformations (tx, ty, scale) and interactions (captureMouse, clickedPos).
  - References for SVG element and Tone.js synthesizer.
- **Effect Hook:**
  - Handles pan events and updates transformations.
- **Functions:**
  - **zoomInOut:** Handles zooming based on mouse wheel events.
  - **drag:** Handles dragging interactions.
  - **captureOn and captureOff:** Manage mouse capture for dragging.
  - **panTo:** Pans the view to a specific position.
- **Rendering:**
  - Renders the Tonnetz grid with notes and chords.
  - Handles note interactions and plays corresponding sounds using ToneGenerator.

TonnetzSelector.jsx

- **Rendering:**
  - Provides buttons for selecting different Tonnetz configurations.

  - Updates the graph state in TonnetzView based on user selection.

Notes.jsx

- **State and References:**
  - Manages the clicked state of notes.
- **Functions:**
  - **handleMouseDown and handleMouseUp:** Handle note interactions and trigger sound playback.
- **Rendering:**
  - Renders individual notes with appropriate colors and labels.

noteUtils.jsx

- **Functions:**
  - **getStartingNote:** Calculates the starting note for each line in the Tonnetz grid.
  - **notes:** Array of note names used in the grid.

ToneGenerator.jsx

- **Functions:**
  - **playNoteWithY:** Plays a note based on its y-coordinate.
  - **playMidi:** Plays a note based on its MIDI number.
  - **playChord:** Plays a chord based on an array of notes.

3. Circle Layout

Description: The Circle View is an interactive component that displays musical notes arranged in circular patterns. It allows users to explore and interact with notes, providing functionalities like zooming, dragging, and playing notes.

Key Files

- **DragZoomCircle.jsx:** Handles the interactive SVG rendering, including zoom and drag functionalities for the circular view.
- **CircleView.jsx:** Main component rendering the Circle View.

Dependencies

- **React Hooks:**
  - **useState:** Manages state for transformations, interactions, and selected notes.
  - **useEffect:** Handles side effects such as locking transformations.
  - **useRef:** Creates references for SVG elements and Tone.js synthesizer.

- **Libraries:**
    - **Tone.js:** For audio synthesis and playback.

Related Components

- **DragZoomCircle**: Provides interactive functionalities like zooming and dragging within the SVG.
- **Notes**: Renders individual notes and handles interactions like clicks.
- **ToneGenerator**: Plays notes and chords based on user interactions.
- **noteUtils**: Utility functions for note calculations and transformations.

Code Flow:

DragZoomCircle.jsx

- **State and References:**
    - Manages transformations (tx, ty, scale) and interactions (captureMouse, clickedPos).
    - References for SVG element and Tone.js synthesizer.
- **Effect Hook:**
    - Handles lock changes and updates transformations.
- **Functions:**
    - zoomInOut: Handles zooming based on mouse wheel events.
    - drag: Handles dragging interactions.
    - captureOn and captureOff): Manage mouse capture for dragging.
    - handleNoteClick and handleNoteRelease): Handle note interactions and trigger sound playback.
    - calculateCirclePoints): Calculates note positions around a circle.
- **Rendering:**
    - Renders two circles of notes with interactive functionalities.
    - Handles note interactions and plays corresponding sounds using ToneGenerator.

CircleView.jsx

- **State and References:**
    - Manages the state for the Tonnetz grid, selected notes, and transformations.
- **Rendering:**
    - Renders the DragZoomCircle component with the circular view.

   - Displays a message if the grid is not connected.

4. Sidebar and Piano Roll

Description: The Sidebar component provides a UI for interacting with the Piano Roll, which is a grid-based interface for creating and editing musical notes. The Piano Roll allows users to import, export, play, pause, and stop MIDI sequences, as well as record and download audio.

Key Files

- **Sidebar.jsx:** Main component rendering the sidebar with a button to open the Piano Roll.
- **PianoRoll.jsx:** Main component rendering the Piano Roll interface.
- **PianoRollGrid.jsx:** Component rendering the grid for the Piano Roll.
- **Playhead.jsx:** Component rendering the playhead that moves across the grid.
- **ClearNotes.jsx:** Component providing a button to clear all notes.
- **Connector.jsx:** Component connecting the Piano Roll with the Tonnetz grid.
- **Notes.jsx**: Component rendering individual notes.
- **MidiWorker.js:** Web worker for processing MIDI files.

Dependencies

- **React Hooks:**
  - **useState:** Manages state for the sidebar, Piano Roll, and notes.
  - **useEffect**: Handles side effects such as event listeners and MIDI processing.
  - **useRef:** Creates references for DOM elements and Tone.js synthesizer.
  - **useContext**: Accesses the playing state from the context.
- **Libraries:**
  - **Tone.js:** For audio synthesis and playback.
  - **@tonejs/midi:** For MIDI file processing.
  - **react-icons:** For icons used in the UI.
  - **file-saver:** For saving files to the user's device.
  - **gsap:** For animations.

Related Components

- **Sidebar**: Provides a button to open the Piano Roll.
- **PianoRoll**: Main component that initializes and renders the Piano Roll interface.

- **PianoRollGrid**: Provides the grid layout for the Piano Roll.
- **Playhead**: Renders the playhead that moves across the grid.
- **ClearNotes**: Provides a button to clear all notes.
- **Connector**: Connects the Piano Roll with the Tonnetz grid.
- **Notes**: Renders individual notes within the grid.
- **MidiWorker**: Web worker for processing MIDI files.

Code Flow:

Sidebar.jsx

- **State and References:**
  - Manages the state for the sidebar's open/close status.
  - References for the sidebar DOM element.
- **Effect Hook:**
  - Adds and removes event listeners for detecting clicks outside the sidebar.
- **Rendering:**
  - Renders a button to open the sidebar.
  - Renders the Piano Roll inside the sidebar when open.

PianoRoll.jsx

- **State and References:**
  - Manages the state for the Piano Roll, including notes, BPM, and playback status.
  - References for the Tone.js synthesizer and MIDI worker.
- **Effect Hook:**
  - Initializes the Tone.js synthesizer and MIDI worker.
- **Functions:**
  - **handleNoteChange**: Handles changes to notes.
  - **handleNoteCreate**: Handles the creation of new notes.
  - **handleNoteDelete**: Handles the deletion of notes.
  - **playNotes**: Handles playing and pausing notes.
  - **stopNotes**: Handles stopping notes.
  - **importMidi**: Handles importing MIDI files.
  - **exportMidi**: Handles exporting MIDI files.
  - **handleStartRecording**: Handles starting audio recording.

    - **handleStopRecording**: Handles stopping audio recording.
    - **handleDownloadRecording**: Handles downloading the recorded audio.
- **Rendering:**
    - Renders buttons for playing, pausing, stopping, importing, exporting, and recording.
    - Renders the Piano Roll grid.

PianoRollGrid.jsx:

- **State and References:**
    - Manages the state for the grid layout and note interactions.
    - References for the grid DOM element.
- **Functions:**
    - **renderNote**: Renders individual notes within the grid.
    - **renderGrid**: Renders the grid layout for notes.
    - **renderTimeGrid**: Renders the time grid layout.
- **Rendering:**
    - Renders the Playhead component to display the playhead.
    - Renders the pitch grid and time grid.
    - Renders the notes as draggable and resizable components within the grid.
    - Handles double-click events to create new notes.

Connector.jsx

- **State and References:**
    - Manages the state for rendered notes and a debug clock.
- **Effect Hook:**
    - Handles the scheduling and rendering of notes based on the playback status.
- **Functions:**
    - **midiToNote**: Converts MIDI number to note name.
    - **replaceSharpWithSymbol**: Replaces sharp symbols in note names.
- **Rendering:**
    - Renders the notes as circles on the grid.
    - Displays a debug clock for note timings.

ClearNotes.jsx

- **Functions:**
    - **handleClearNotes:** Clears the currently playing notes and resets the total beats to 16. If the number of playing notes exceeds 100, it temporarily disables and re-enables the piano roll rendering to force a re-render.
- **Rendering:**
    - Renders a button with a trash icon. When clicked, it triggers the handleClearNotes function to clear the notes.