Main

This **Main** class handles the interaction between the lower level classes and the Firebase server. Inside it, instances of other classes are constructed, and login is managed.

- **Initialization and Controller Setup:**
The script sets up various controllers (`AudioModel`, `Model`, `View`, `Controller`, `MidiController`, `TouchController`) for managing different aspects of the application.
- **Firebase Integration:**
The script initializes Firebase with specific configuration parameters (API keys, domain, etc.).
The Firestore database (`db`) and Firebase Authentication service (`auth`) are set up for data storage and user authentication.
- **User Authentication:**
Functions like `logOut`, `onAuthStateChanged`, and `loginWithMail` manage user authentication states.
Users can log in with email and password, and their authentication state changes are monitored to load appropriate data.
- **Data Retrieval and Management:**
Functions `getDefaultPresets` and `getUserPresets` retrieve preset data from Firestore. This data is related to user-specific settings or configurations (presets).
`pushNewUserPreset` and `deletePreset` allow adding and deleting user-specific presets in the Firestore database.

MidiController

The **MidiController** class acts as a bridge between MIDI hardware inputs and the software logic of your application. It interprets MIDI messages from a keyboard and a control pedal, translating them into actionable commands or data

- **MIDI Setup:**
  - `initializeMIDI`: Requests access to MIDI devices and, upon success, calls `setupMIDIInput` to configure MIDI input handling.
  - `setupMIDIInput`: Iterates over available MIDI input devices and sets up event handlers for the specified keyboard and control pedal.
- **Handling MIDI Messages:**
  - `handleKeyboardMIDIMessage`: Processes MIDI messages received from the keyboard. It interprets different types of MIDI messages such as Pitch Bend, Control Change, Note On, and Note Off.
  - `handlecntrlPedalMIDIMessage`: Processes MIDI messages from the control pedal. It handles specific control changes based on the pedal's input.

TouochController

TouchController provides a touch-based interface for interacting with various elements of the synthesizer. It translates user actions like clicking and dragging on screen elements into meaningful commands that sends to the controller.

- **Touch Interaction:**
The methods `attachKeysEventListeners`, `attachPadsEventListeners`, `attachDisplayButtonsEventListeners` and `attachDisplaySplitEventListener` listen for mouse down, up, and leave events and send the corresponding commands to the controller class.

Controller

the **Controller** class serves as an intermediary between the other controllers (midi and touch input), the model (data and logic) and the view (UI).

- **Preset Management:**
`setPreset` changes the current preset based on user selection.
`synchronizeKnobs` updates the knob elements on the UI to reflect the current state of the model.
- **Instruments Handling:**
`handleNoteOn` and `handleNoteOff` manage the behavior when musical notes are played or stopped. This includes handling for different instruments like keys, bass, and arpeggio (arp).
`handleSustain` manages sustain pedal effects for notes. Functions like `handlePadOn`, `handlePadOff`, `handleControlChangeEvent`, and `handleWheel` handle various user interactions with pads, control changes, and modulation wheels. `handleArp` toggles the arpeggiator and updates the interface accordingly.
- **Instrument Configuration:**
`shiftOctave`, `waveformChanger`, `splitManager`, `turnOn`, `flipWheel`, `flipMono`, `flipSustain` are used to modify settings of the instruments like octave shifting, waveform changing, mono/polyphonic modes, and sustain settings.
- **MIDI and Preset Mapping:**
`handleMidiPresetChange` and `handleMidiMappingPresetChange` manage changes in MIDI presets and mappings.
- **Rendering and Display Updates:**
`renderAll` is a comprehensive function that updates the entire view based on the current state of the model. It updates various components like volume indicators, oscillator types, active indicators, and more.

Model

The **Model** class acts as the central repository for the application's state, handling the logic for audio parameters, user interactions, and interface changes.

- **Preset Management:**
`setPresets` and `setPreset` are used to manage presets, which are predefined settings for the application.
`getPresetNames` provides a list of available preset names.
- **Instrument Settings:**
Methods like `flipSplit`, `flipMono`, `flipSust`, and `flipWheel` toggle different modes for instruments (e.g., split mode, mono/polyphonic mode, sustain, wheel control).
`setWaveform` updates the waveform type for different instruments based on user selection.
- **Knob and Control Pedal Handling:**
`updateKnobLevel` and `handleControlChangeEvent` manage changes in knob levels and control changes from the MIDI or touch input.
`connectPedalKnobs` links control pedals to specific knob functions.
- **Audio Parameter Updates:**
`refreshAudioParameters` updates the `audioModel` with the current state of various controls like gain, filter frequencies, delay settings, etc.
- **Note Handling for Instruments:**
Methods like `handleNoteOn`, `handleNoteOff`, `handleBassOn`, `handleBassOff`, and related methods manage the playing and stopping of notes for keys, bass, and arpeggiator.
`deleteAllNotes` and `deleteAllSustainedNotes` are used to stop all currently playing or sustained notes.
- **Arpeggiator Control:**
`handleArpeggioOn`, `handleArpeggioOff`, `playArpSequence`, and `handleArpSustain` control the behavior of the arpeggiator, including note playing and sustaining.
- **Drum Pad Handling:**
`handlePadOn` and `handlePadOff` manage interactions with drum pads, triggering different drum sounds.
- **MIDI Preset Changes:**
`handleMidiPresetChange` changes presets based on MIDI input.

View

View class is focused on the visual aspects of the application. It manages the dynamic updating of UI components based on user interactions and audio processing events. This includes visual feedback on controls like knobs and buttons, visualizations of audio signals, and general UI theming and layout adjustments.

- **LED, Key, and Pad Control:**
Methods like `flipLed`, `flipKey`, and `flipPad` handle the visual toggling of LEDs, keys, and pads on the interface, indicating their active or inactive states.
- **Knob Interaction:**
`rotateKnob` updates the visual representation of a knob element based on the rotation value, reflecting changes in settings like volume or filter cutoff.
- **Display Updates:**
  - `updateDisplayOctave` updates the display of the current octave for an instrument.
  - `renderActiveIndicator` visually indicates whether an instrument (like a synthesizer voice) is active or not.
  - `showOscillatorType` updates the display to show the current type of oscillator (sine, square, etc.) being used for a particular instrument.
  - `flipButton` toggles the state of buttons on the interface.
`updateSplitDot` updates the visual state of a split indicator.
  - `updateDisplayVolumeIndicator` updates the display of volume indicators for different instruments.
- **Amplitude Visualization:**
`drawAmplitudePlot` and `animateAmplitudePlot` are responsible for creating and animating an amplitude plot on the canvas. These methods visualize the audio signal's amplitude over time.
- **User Interface Color Theming:**
`interfaceColorGradient` adjusts the color scheme of various interface elements like knobs, pads, and buttons based on the audio signal's properties.

AudioModel

AudioModel acts as the audio engine for the application, generating and manipulating sound based on user interactions and control settings. It provides a wide range of functionalities for synthesizing musical notes, applying audio effects like filters and delays, and generating drum sounds.

- **Gain Control:**
Methods like `setMainGain`, `setInstGain`, `setDrumGain`, `setKeyGain`, `setBassGain`, and `setArpGain` control the volume levels of different audio sources.
- **Filter and Delay Configuration:**
`setLowPassFilterFrequency` and `setHiPassFilterFrequency` configure the frequencies of low-pass and high-pass filters for keys and bass.
`setDelayTime` and `setDelayFeedback` adjust the delay time and feedback amount for keys and bass delay effects.
- **Note Handling and Sound Generation:**
`playNote` creates an `OscillatorNode` and `GainNode` to play a musical note. It sets the oscillator frequency based on the note and instrument type and connects it to appropriate audio nodes.
`stopNote` gracefully fades out and stops a playing note.
- **Drum Sounds:**
`playKick`, `playSnare`, `playClosedHiHat`, and `playCrashCymbal` generate various drum sounds using techniques like white noise generation and filter application.
- **Amplitude Analysis:**
`getAmplitude` analyzes the audio signal to calculate the amplitude for visualization purposes.
- **Oscillator Configuration:**
`setNoteOscillator` sets the type of waveform to be used for note oscillators.