**Synquencer Design Document**

Mitchell Bridwell, Zach Heskett, Aidan McDonnell, and Mohini Roplekar

Team 37



Index

* Purpose - Page 3
* Design Outline - Page 7
* Design Issues - Page 10
* Design Details - Page 14

# Purpose

Collaboration is very common in digital music production, but there are surprisingly few tools to help musicians collaborate remotely. One essential part of music production is creating note sequences to play back with virtual instruments. Musicians most commonly use piano roll sequencers within the digital audio workstation (DAW) of their choosing to write these sequences. However, since there is no convenient way to collaborate on these sequences in real time, musicians often resort to sending project files back and forth, which can be tedious and could possibly lead to unintended changes in the projects.

There are a few applications that come close to solving this problem, but each falls short in one way or another. OnlineSequencer.net, for example, allows users to edit and share MIDI sequences online, but it does not allow for real-time collaboration. Users must save their sequence and send it to another user, which is just as tedious as sending a project file. Other websites, like BandLab.com, attempt to replace the user's DAW entirely, but these web-based applications all fall short of professional desktop solutions. Additionally, both of these applications are lacking in accessibility to those with visual impairments.

The purpose of our solution is to provide a smooth web-based musical sequence editing platform. This platform will be accessible to those with visual impairments, particularly people who use screen readers. In addition, this platform will enable real-time collaboration with other users on the same project at the same time, as well as exporting sequences to the commonly-used MIDI format.

**Functional Requirements**

**Sequence Editing**

As a user,

* I would like to be able to create a new, empty sequence
* I would like to add a note to the sequence by clicking an empty point in the sequence.
* I would like to edit or remove a note from the sequence.
* I would like to move or copy a note somewhere else.
* I would like to undo an action.
* I would like to redo an action after undoing twice.
* I would like to lengthen or shorten the duration of a note.
* I would like to select multiple notes and move them all at once
* I would like to change the tempo of the sequence
* I would like to loop a specific section of the sequence (cycle) to work on
* I would like to change the time signature of the pattern
* I would like to change the length of the pattern
* I would like to change the grid resolution of the piano roll
* I would like to change the velocity of a note
* I would like to zoom in and out of the sequence
* I would like to scroll up, down, left, and right in the sequence.
* I would like to import MIDI files stored on my computer and edit them (if time allows)

**Sequence Playback**

As a user,

* I would like to use multiple different instruments in my sequence
* I can play back and listen to my project in the editor
* I would like to export my sequences to MIDI files and store them locally
* I would like to configure the sounds that the software uses to play back sequences (if time allows)

**Accessibility**

As a user,

* I would like to enter notes using only my keyboard.
* I would like to be able to change instruments and other settings using only my keyboard.
* I would like to use my computer keyboard to play the currently selected instrument.
* I would like to choose between multiple color schemes for the application.

**Collaboration**

As a user,

* I would like to be able to create a new, empty sequence
* I would like to see who is working on the same project as me
* I would like to be able to see a note placed by another user in the same project
* I would like to be able to see a note edited by another user in the same project
* I can play the sequence in time with another user
* I would like to edit a sequence alongside other users asynchronously
* I would like to edit a sequence alongside other users synchronously
* I would like to store my sequences on the server and access them later
* I would like to share my sequences with other users
* I would like to only allow specific people to edit and/or view my sequence (if time allows)

**Non-Functional Requirements**

**Client**

As a developer,

* I would like the frontend to be written using React and Next.js.
* I would like to use the MidiWriterJS library to export Synquencer sequences to MIDI files.

**Server**

As a developer,

* I would like the backend to be written using React and Next.js.
* I would like all sequences to be stored internally on the server as JSON files.

**Design**

As a developer,

* I would like the entire project to be written in TypeScript, a variant of Javascript.
* I would like to use a queue system for all user operations, to ensure that the server handles incoming operations from clients in the proper order.
* I would like to ensure that this website conforms to WCAG (Web Content Accessibility Guidelines) 2.0, which is the standard for accessible design.

**Performance**

As a developer,

* I would like to minimize latency between client and server as much as possible.
* I would like to make sure that, when working together on the same sequence, all clients agree with each other on the state of the sequence at any given time.
* I would like to minimize the data that needs to be transferred between server and client, to make sure that the amount of time spent transferring data remains small.

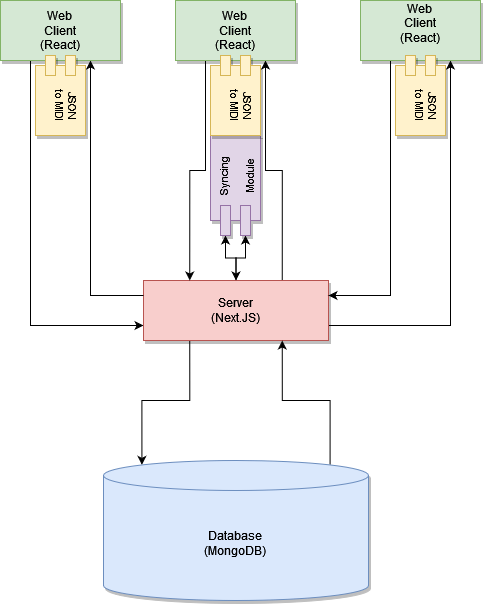
# 

# Design Outline

**High Level Overview**

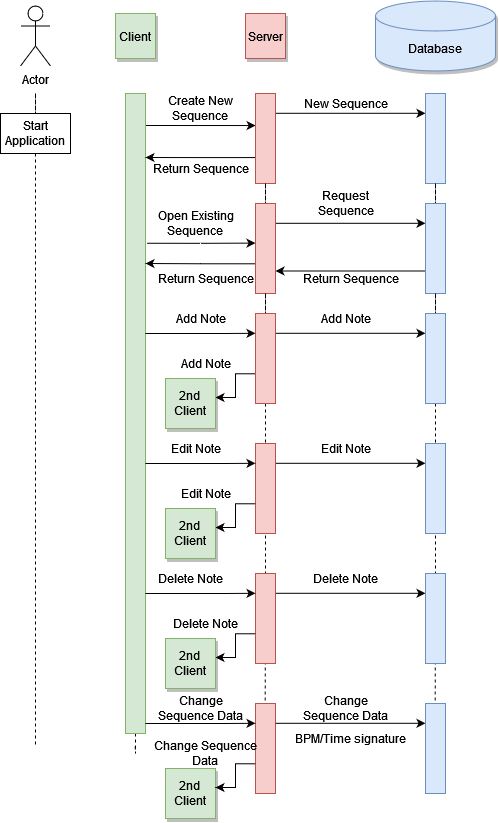
Our project is a web application that allows users to write musical sequences, both individually and collaboratively with other users. Our implementation will use a Client-Server model, in which the sequence is stored on the server and all clients make changes to the sequence stored on the server by sending all commands to the server, which carries out those changes on its own copy of the sequence. The server will store all sequences in a database, which can be accessed by using the specific link associated with the sequence.

1. Client
   1. The client will be a webpage that allows each user to view and edit a given sequence
   2. The client will display the current sequence, provide an interface for editing the sequence, and allow users to play back the sequence
   3. The client will send commands to the server and receive commands from the server in JSON format
2. Server
   1. The server will be written in Javascript and will handle receiving commands from clients, modifying the sequence, and sending the modified sequence out to all clients
   2. The server will send and receive all communications with clients using the JSON format
   3. The server will access the database whenever someone visits the URL associated with a specific sequence, will retrieve the sequence from the database, and will send the sequence to the client(s) that requested it
3. Database
   1. The database will use MongoDB to store sequences when sessions end and restore them when users request them again
   2. The database will store the JSON data structure for each sequence so that it can be accessed later via a link



**Sequence of Events Overview**

The following sequence diagram demonstrates how the client, server, and database interact, starting from when the user starts the application by visiting the webpage and creates a new sequence. The server creates a new sequence and sends this blank sequence data to the client. From then on, whenever the user performs an operation on the sequence, this operation is sent as a command to the server, which receives the command, performs the corresponding operation on its own copy of the sequence, and sends the change back to all other clients, which applies the change to its own version of the sequence. The server then saves the modified sequence to the database. Whenever a client attempts to access a sequence that already exists, the server will query the database for this sequence, retrieve this sequence, and send the sequence to the client.



# Design Issues

# **Functional Issues**

Issue 1: How do we maintain that all changes that one user makes to a sequence are accurately synchronized to all users who are editing that sequence?

* Option 1: Each client carries out all commands on its own version of the sequence first, and the client sends its changed version of the sequence to the server, which adopts the most recent version of the sequence as its own
* Option 2: Each user’s commands are first sent to the server, which applies the respective changes to its own version of the sequence and sends this changed version of the sequence back to all users
* Choice: Option 1
* Discussion: Although option 2 is simpler to implement, the latency is too high for our desired experience. Users would have to wait for the server to process every new note before it appears on their screen, and since sequences are made up of many notes, this would slow down the composing process significantly. Option 1 also aligns more closely with many of the syncing frameworks available today.

Issue 2: How do we make sure that commands get executed in the same order they are sent by the users?

* Option 1: Add all incoming commands to the server to a queue, which the server then executes in order and sends back out to the clients.
* Option 2: Execute all commands locally, then send them to the server. The server resolves conflicts and sends commands to fix client data structures if necessary.
* Choice: Option 2
* Discussion: Option 1 would be easier to implement, but it would add far too much latency. It would also cause issues if the client had an intermittent internet connection. Option 2 allows most of the processing to be done locally, and the server would only have to send changes in the event of a conflict.

Issue 3: How do we implement the ability to undo and redo commands?

* Option 1: Store sequence states after every command and restore those states when a command is undone/redone
* Option 2: Have each command implement undo/redo functionality specific to that command and store the changes made by the command with the command itself so that undoing/redoing the command has a different effect for each command
* Choice: Option 2
* Discussion: The first option would require the least computation, but it would take the longest time when the time needed to transfer data between server and client is factored in. Implementing the second option would require writing the code for the undo/redo operation for each command which would need to implement this functionality, but it would be the fastest at runtime, since it would allow undo and redo operations to be treated like any other command.

Issue 4: How do we make our application accessible for the visually impaired?

* Option 1: Create a separate view specific to screen readers that works as closely as possible to the main view, while only showing the buttons on screen necessary for a screen reader to use the software.
* Option 2: Implement screen reader and keyboard accessibility directly in with the application, figuring out as time moves along how to best ensure that each action that can be performed with a mouse can also be performed using keyboard shortcuts and a screen reader.
* Choice: Option 2
* Discussion: While Option 1 may allow for more design freedom (in the fact that buttons and actions can be designed however we want), the accessibility, if left untested, could feel "tacked-on," as software bugs could arise based on how each page (accessible and inaccessible) accesses the backend. Option 2, on the other hand, would require less time in the creation and upkeep, as all of the ways that an action can be triggered would all be pointing directly to that action. This ensures that if any bugs do arise, they will be easier to fix, and less time will be wasted only debugging one page instead of two.

Issue 5: How will we retrieve MIDI sequences for export or playback?

* Option 1: Create the MIDI data on the client-side from its cached copy of the sequence in JSON format
* Option 2: Download the MIDI data directly from the server
* Choice: Option 1
* Discussion: We initially thought that Option 2 would be the better approach to saving sequences, as it allows them to be easily grabbed from the server to be viewed through a person's web browser. This is the traditional way to download files from the internet. However, we ended up deciding on option 1 because MIDI sequences are miniscule (rarely coming close to half a megabyte), meaning that if we are able to create and store them on the client side, we wouldn’t need to send unnecessary data over the server.

**Non-Functional Issues**

Issue 1: What programming language will we use to develop this application?

* Option 1: C++
* Option 2: Javascript
* Option 3: Typescript
* Choice: Option 3
* Discussion: Although all of us have the most experience with C++, and have little experience with either Javascript or Typescript, we decided not to use C++ because we would already have to use HTML for the web page, and both Javascript and Typescript interface much better with HTML than C++ does. Initially we intended on using Javascript, but after consulting with our project coordinator we determined that Typescript would be a better option, as it has full compatibility with all Javascript code but implements static typing, which makes writing and debugging code much easier.

Issue 2: How do we update each client’s copy of the sequence to match the server’s copy?

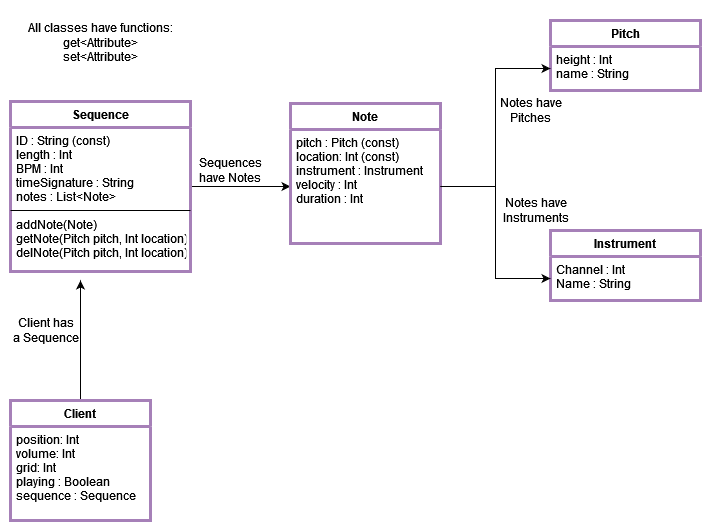
* Option 1: Send the entire sequence from the server to each client every time the server’s version of the sequence does not match a client’s version
* Option 2: Individually send over all changes made to the sequence from the server to each client
* Option 3: Individually send over changes from server to client under most circumstances, but occasionally send over the entire sequence under certain conditions
* Choice: Option 3
* Discussion: We realized early on that refreshing every client’s entire sequence every time the server’s sequence changed would incur too much latency, so we heavily leaned towards Option 2. However, there are certain circumstances where sending over a fresh copy of the sequence from the server is faster, such as when the client is first connecting to the server, or more secure, such as when connection problems cause a client’s version of the sequence to desynchronize significantly from that of the server. As such, we decided that the best option was to mix the two approaches, sending over fast and lightweight change information under most circumstances but sending over the entire sequence whenever conditions make that the better option.

Issue 3: How do we internally handle and store note data?

* Option 1: Store each note as an instance of a data structure in a list, and represent the sequence as a list of these notes
* Option 2: Store the sequence as a list of intervals when each instrument is playing for each note
* Choice: Option 1
* Discussion: We considered Option 1 initially, but for some time we thought of going with Option 2, to avoid the overhead associated with instantiating each note as an individual object. However, we realized that storing each note as only an interval and nothing else made it difficult to associate necessary metadata with a given note, as well as making it difficult for commands to refer to specific notes in the sequence. As a result, we decided that each note should be stored in a list, sorted from earliest starting position within the sequence to latest.

**Design Details**

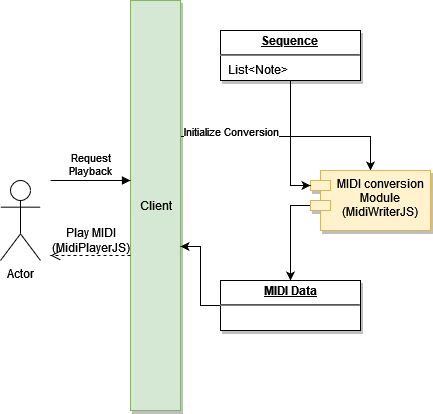
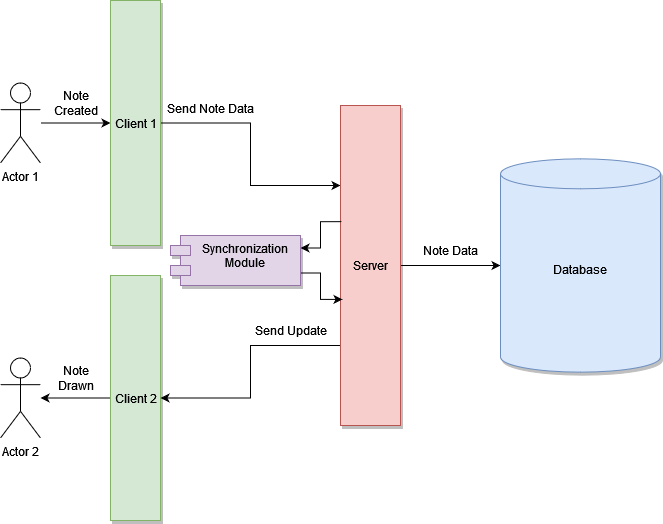
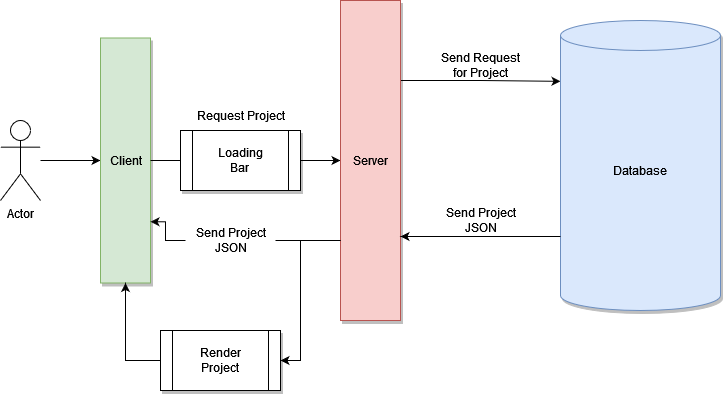
**Class Design**



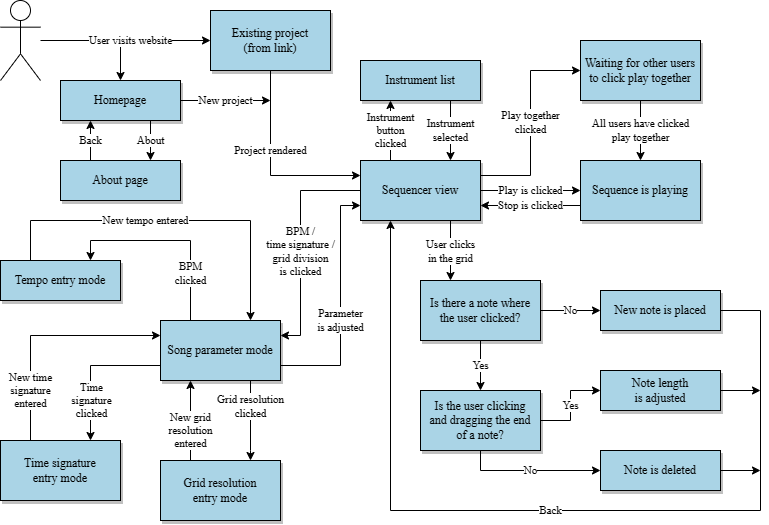
**Description of Classes and Interaction between Classes**

* **Client**
  + Contains information related to the user’s current session. This includes:
    - Position: Location of playhead in user’s session
    - Volume: Current volume of the playback, initialized to 0
    - Grid: Grid resolution of changing duration of notes
    - Playing: Whether the client is currently playing the sequence or not
    - Sequence: The current sequence being worked on
* **Sequence**
  + Contains information related to the sequence/project currently being worked on. This information contains the fields:
    - ID: Identification of the Sequence. This is important for creating unique links to new and existing sequences.
    - Length: The total duration of the sequence
    - BPM: Beats per Minute. The speed at which the sequence plays notes. Default is 120 BPM.
    - Time Signature: The time signature of the sequence. A unique string identifies different signatures such as “4/4”, “3/4", etc. Defaults to 4/4.
    - Notes: A list of all the notes in the sequence.
  + Also includes the unique functions:
    - Add Note: Adds a new note to the sequence.
    - Get Note: Returns a note from the sequence based on location and pitch.
    - Del Note: Deletes a note from the sequence based on location and pitch.
* **Note**
  + Contains unique information related an individual note in a sequence:
    - Pitch: The pitch of the note (how high or low it is).
    - Location: Where in the sequence the note starts.
    - Instrument: Which instrument this note should be played as
    - Velocity: The velocity of the note. Also known as the volume of the note. Can be set between 1 and 127.
    - Duration: How long the note lasts in the sequence.
* **Pitch**
  + Identifier for the pitch of a note:
    - Height: An integer representing where in the GUI the note should be placed. (0-87)
    - Name: The traditional name of this pitch. In the format: “C4”, “D3”, etc. Important for constructing MIDI data from the pitch.
* **Instrument**
  + Identifier for the instrument of a note
    - Channel: A unique ID for the instrument. Also specifies what channel this instrument should be put in for MIDI data.
    - Name: The name of the instrument. “Piano”, “Bass”, etc.

**Sequence Diagrams**

* Sequence of Events when a user requests playback of their project
* Sequence of Events when a user adds a note to their sequence
* Sequence of Events when a user opens a project

**Navigation Flow Map**

****

**UI Mockups**

**Homepage**



* **New Composition:** Opens a new, blank sequence with a unique URL that can be shared with collaborators
* **About:** Displays information about the app.

**Sequencer view**



* **Play together:** Allows the user to play the sequence in time with collaborators. The button shows how many users in the session have clicked it. Once every user in the session has clicked it, the sequence will start playing for every user at once.
* **Note preview:** Allows the user to preview a note without placing it by clicking the corresponding key in the preview bar
* **Playhead:** Shows the position of the currently playing sequence

**Sequencer view (cont.)**



* **Tempo, time signature, and grid resolution:** Allows the user to adjust the tempo in beats per minute (BPM), time signature, and grid resolution (how precisely the user can adjust note start and length) for the current sequence.
* **Instrument Selector:** Allows the user to select the current instrument with which to write notes from a list of preset instruments