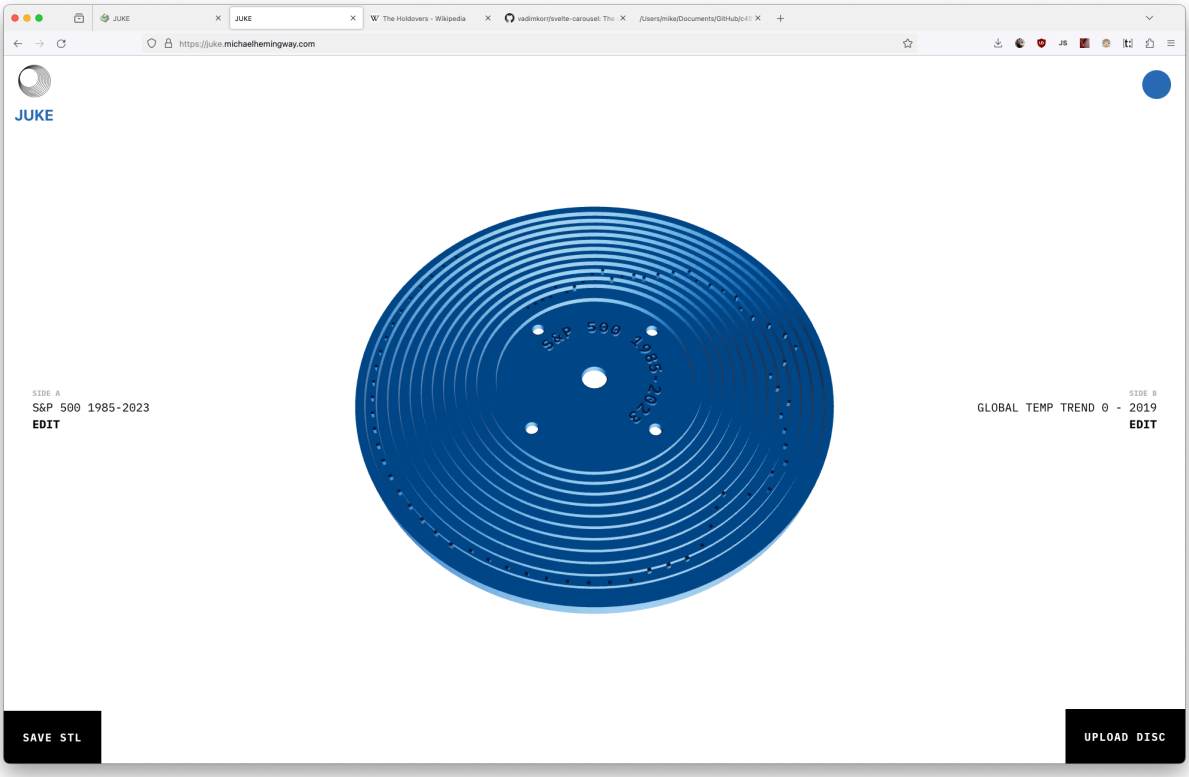
JUKE

Post-Mortem

Juke is an online STL builder and sharing platform that allows users to create custom music discs for the 1971 Fisher-Price *Music Box - Record Player*.

Tech Stack

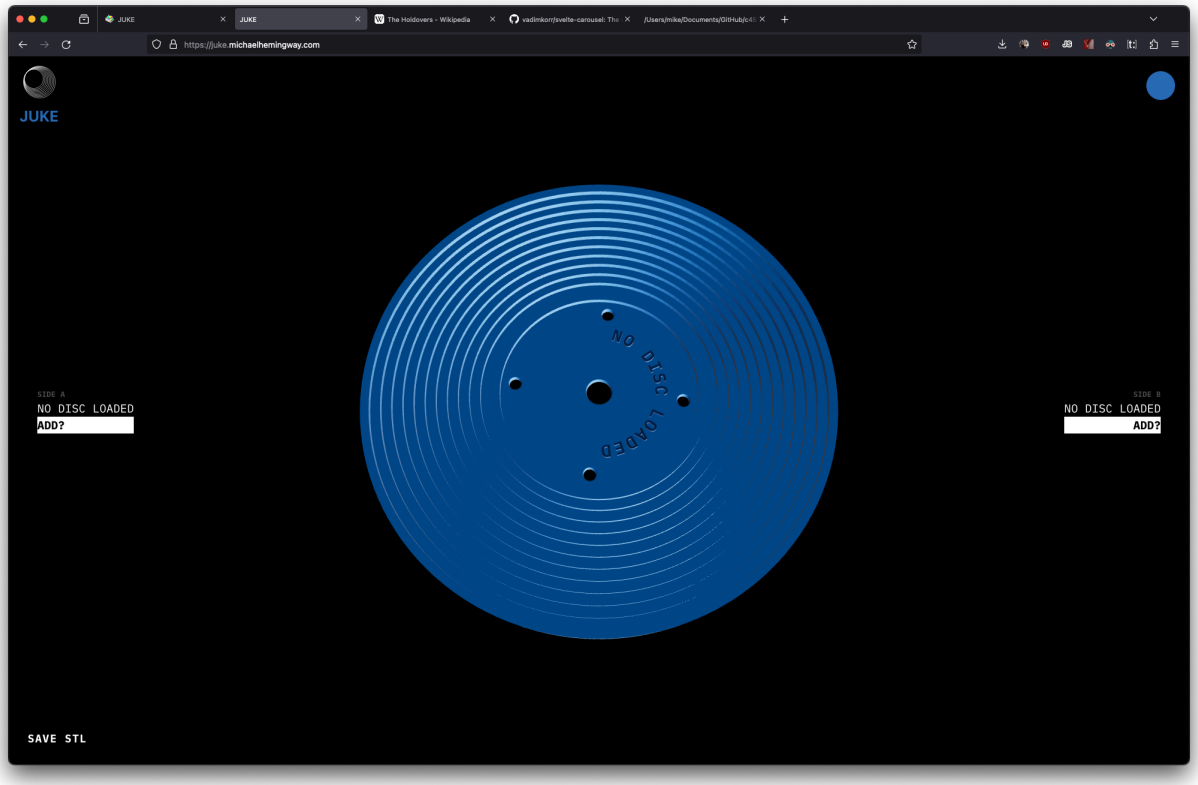


Main app interface, light theme

The app itself is built with Svelte, Vite, and Threlte for Threejs interoperability. Fonts used in the generation of 3D text were built using FaceType.js. An incomplete character set was provided to save on loading times, as extruded fonts can become enormous with little benefit if latin uppercase is all I aim to support.

Initial modelling (the disc is not entirely generated, a base model is first loaded) was done in Blender, with manual measurements and adjustments over iterations to compensate for thermal expansion and tolerances. This model is a two-sided disc with 11 groves and no text.

Server-Side



Dark theme, showcasing initial state with a blank disc

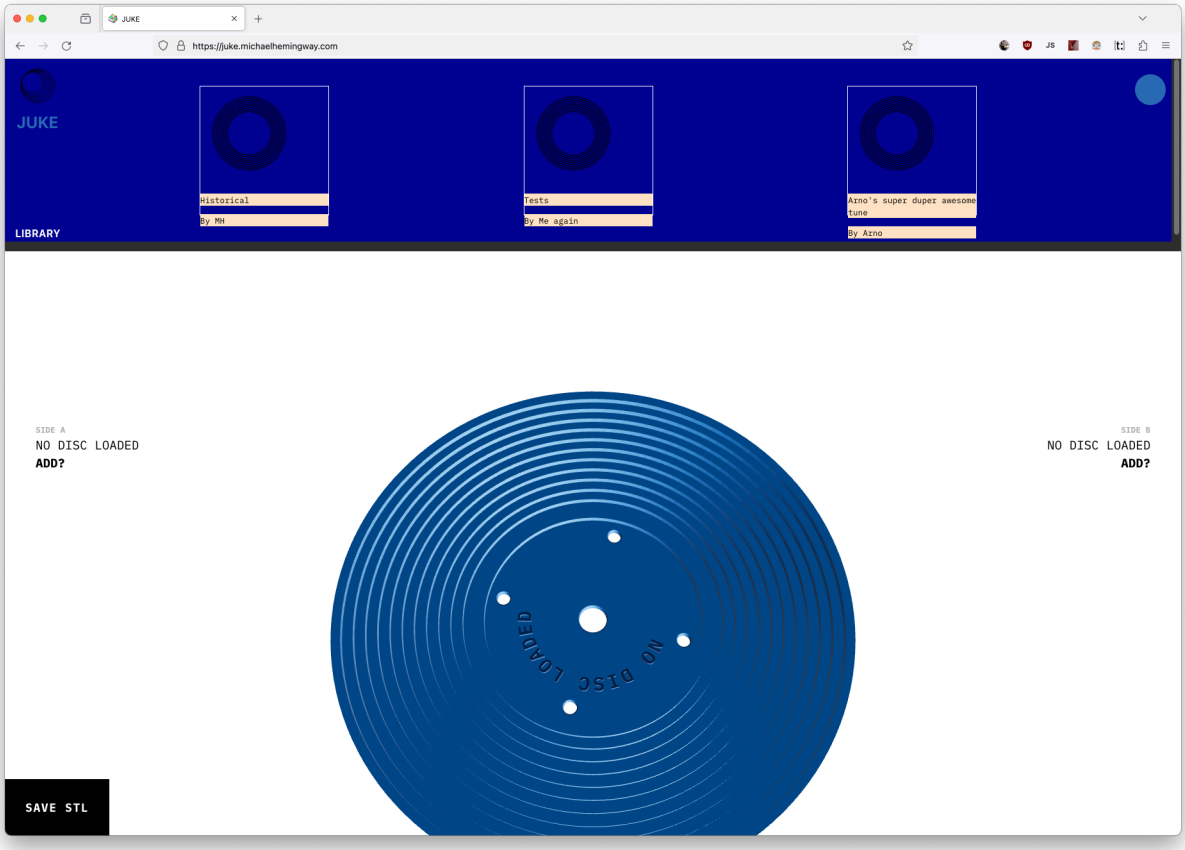
By virtue of convenience, this is running on a tiny Olimex A20 in my home office, though Nginx. A small Node server passes requests to MongoDB. This node app is *daemonized* via PM2. My home internet connection is no longer static, so random outages may occur, but I endeavour to fix that.

The app's Node Server is technically open to CORS requests (set up as such before the site itself was to be hosted on the same server that servers the /add and /discs endpoints.

Further work is necessary to support ranking features, pagination of database results, input sanitization and more for a public release. The server component has been deliberately omitted from the release package.

Storage has been optimized by encoding the 75 possible notes spaces and 16 possible notes into zero-padded hexadecimal string arrays. Further optimizations are possible at the cost of code legibility. Base64 unfortunately carries too much overhead, but a blob format would be possible.

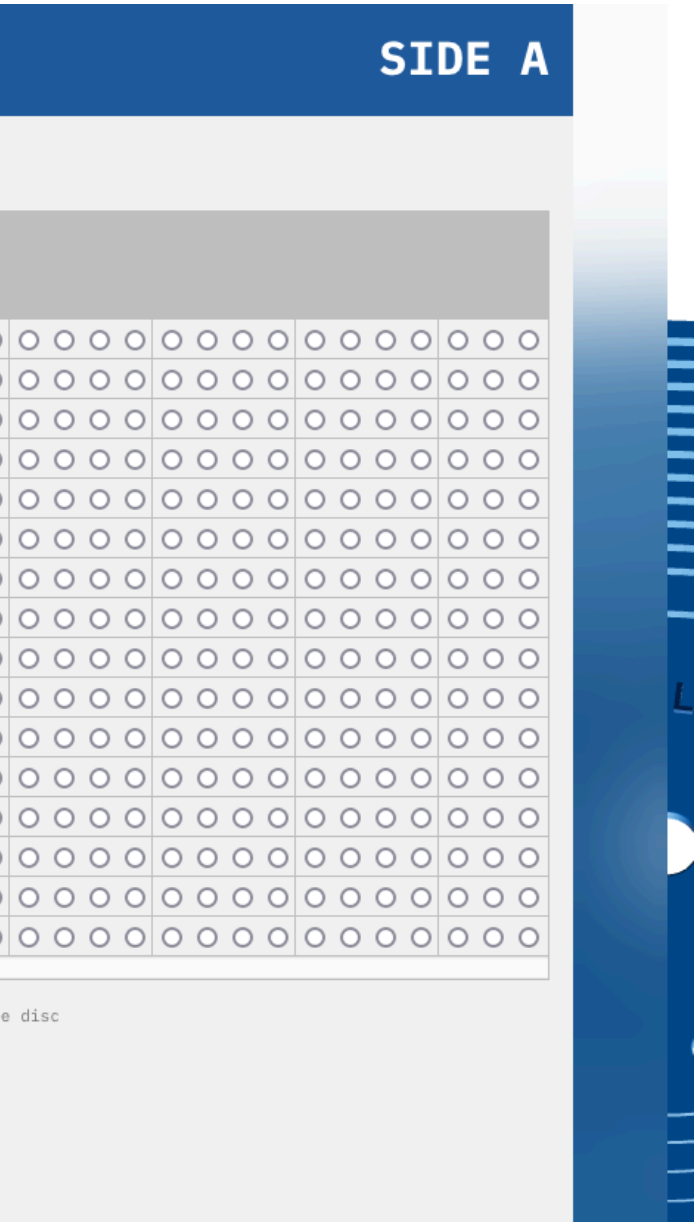
Features



User Catalogue visible

Whilst planned from the start, the user catalogue was relatively rushed, as I had spent a great amount of time with two other backend implementations in Strapi and Directus (both self-hosted). These were too resource-intensive for the shared VPS I planned to host on and on my own low powered machine (my own home server is dedicated to other tasks and isn't exposed to the internet). This is reflected in the design and lack of real input sanitization, but performance is acceptable and loading of other users' data is instantaneous (as demonstrated in the feature video).

User input was planned as a notation app, using conventional Western sheet music. This did not pan out as the differences between individual units (notices by comparing videos to mine) do not reliably produce tones in the way that sheet music would convey. The notches on the disc are instead rough approximations of particular notes.

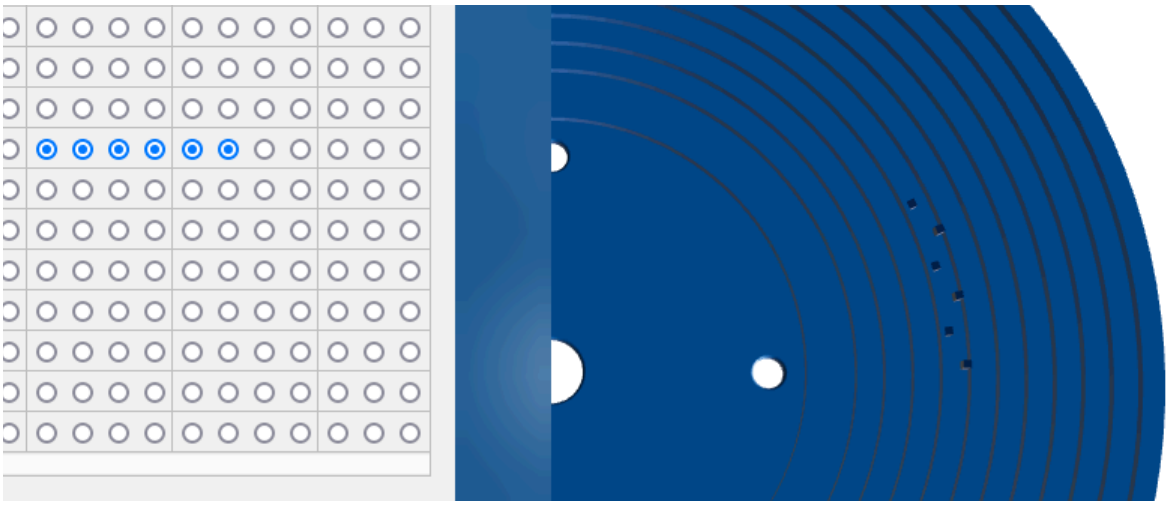


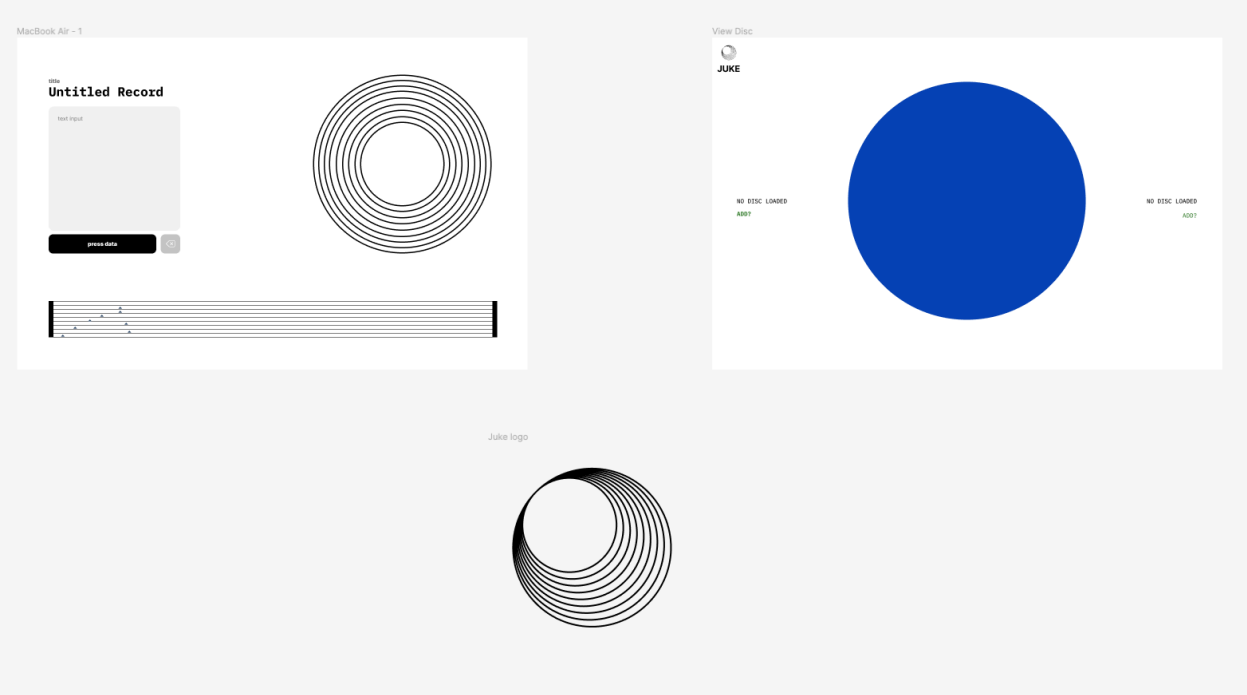
Another planned-but-not-implemented user input method is that of a direct file-to-notch data upload function, with special attention to JSON and CSV. While arbitrary quantization is possible (and was performed in one of the demo discs), the complexity required in a sanitization interface made it unfeasible.

Instead, the only current implementation of user notch input is a full presentation of all the possible notes (but not the possible spaces) that may be placed one one side of the disc. This is the case as I noticed during testing that 6 of the middle lanes actually played the same note, and when arranged in factory discs, closely repeated notes would alternate across groves to improve response time. I implemented this algorithm in the data editor. It is visible when placing repeated notes immediately following one another.

Left: A blank input panel, with “Side A” visible

Below: The alternating notch pattern





Figma: Initial interface concept, logo, final design concept

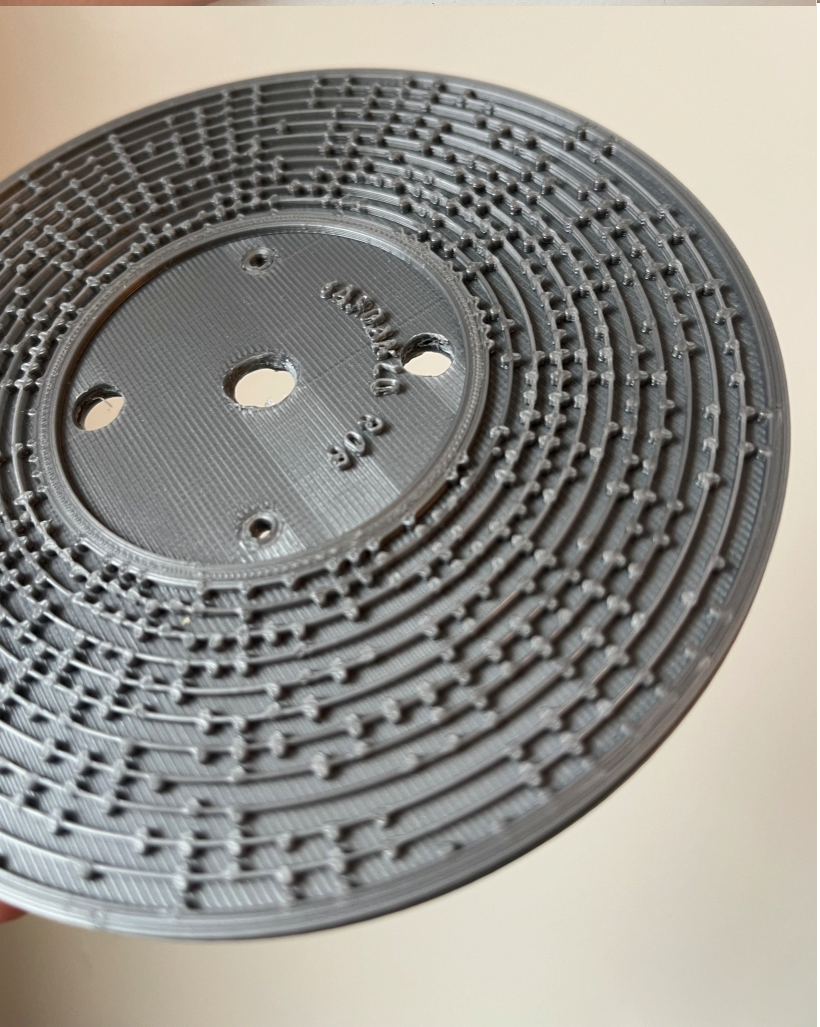
Lastly, missing from intended features is a audio feedback mechanism. An audio file exists on GitHub to slice into individual notes and play back on placement. This is an essential component for a public release as one usually doesn't want to commit to 3D printing for hours only for an acoustic dud.

Hardware

In the process of building this project, I managed to find a bug in the STL Exporter in three-stdlib. I expect to write a patch for this soon.

3D printed discs are highly variable in quality but there is not much I can do from the web interface to account for that. I have tried to optimize for both print speed and ease of printing (without supports on the underside) without sacrificing playability. Given the original toy is meant for children, bad print jobs are tolerated by the rather hardy machine. The FDM printing target (my own printer, the most common) is deliberate, and higher-quality resin printing may break given the shape I chose for the notches (cuboids), when printed at sufficiently high resolution, might create a barrier to the play arm's needles without the rounding-off effect FDM printing has on fine model geometry.

Michael Hemingway



Top Left: A printed disc

Top Right: A factory Disc “Humpty Dumpty 1”

Left: Detail of a random data print. Note holes drilled to avoid tolerance issues.

Following Page: The toy with a 3D printed record obtained via the web interface

Other Works & Conclusion

This was fun to make. Other attempts at this idea (discovered after I had already settled on the project), while good, only served as a benchmark to beat. I did not make use of their code nor execution. The disc shape itself, being relatively simple, was not copied but instead made from scratch with measurements, as I had the luxury of owning the toy myself and could thus study it at my leisure. IBM Plex should be acknowledged for the excellent typeface.

I hope, upon public launch, that this project's resultant discs are as fun to listen to as this was to make.

Thanks,

Michael