sugarlabs

Sugar Labs

Google Summer of Code 2025

Project Information:

Name: Music blocks 4 Masonry Module (here)
Length: 350 Hours (Large - 22 weeks)
Mentor: Anindya Kundu
Assisting Mentor: Walter Bender Devin Ulibarri

Student Details:

Full Name: Saumya Shahi
Email:saumya23bcy18@iiitkottayam.ac.in
GitHub: saumyashahi
LinkedIn: Saumya Shahi
Medium: Saumya Shahi
Preferred Language: I'm proficient in English for communication, both spoken and written
Location: New Delhi, India

Time Zone: Indian Standard Time (IST) (UTC +05:30)
Phone Number: +91 9569057897
Institution: Indian Institute of Information Technology Kottayam
Program: B.Tech in Computer Science and Engineering with Specialization in Cybersecurity
Stage of completion: 2nd Year (expected June 2027)

About Me:

My introduction to open source wasn't planned—it just happened. Initially, I participated in HacktoberFest, where I made my first small contributions and got familiar with **Git and GitHub**. But at that point, I didn't fully understand the essence of open-source collaboration. That changed when I got introduced to **Sugar Labs**.

I first connected with **Devin Ulibarri** through a Matrix meeting, where we had a conversation about the community and its projects. That discussion gave me my first real insight into how open-source works—not just in terms of code but in terms of collaboration, learning, and growth. I immediately felt drawn to **Sugar Labs' working style and the welcoming nature of the community**.

I didn't come into college with a background in Computer Science—I learnt everything using resources available online. At first, I explored **competitive programming**, but over time, I realized that my real interest lay elsewhere. I started experimenting with **web development** and **Quantum Computing** (under cybersecurity), building projects that helped me understand these fields better.

During my winter break last December, I dedicated time to **learning Git more deeply** and making my first contributions to Sugar Labs and its website, which is going to have a new version now (equally excited for it). While my initial contributions were small, they were incredibly valuable in helping me understand the workflow and the impact of even minor changes. Every PR, every issue, and every discussion helped me grow as a developer.

I was actively contributing to **Sugar Labs**, preparing for **Google Summer of Code (GSoC) 2025**. One of my notable contributions has been working on **Music blocks**, where I focused on improving the repository. I also collaborated with another contributor to make **Sugar Labs' website donation banner more responsive and dynamic**.

However, with the start of my next semester, my contributions slowed down due to academic commitments. But now, I'm back, **excited and ready to contribute officially through GSoC 2025**. I see GSoC as the perfect opportunity to **apply what I've learned, deepen my understanding, and actively contribute** to Sugar Labs in a meaningful way.

As I prepare for GSoC, I am also continuously learning and refining the skills required for my project. I'm eager to **take on challenges, contribute effectively, and grow alongside the Sugar Labs community**.

My general interest lies in creating products that bring impact in society and benefit the community. I am well aware of:

Technical Skills

- **Programming Languages**: C, C++, Python, Java, JavaScript, CSS3, HTML5
- **Frameworks & Libraries**: React, Redux, Tailwind CSS, Typescript, Bootstrap
- **Databases**: MySQL, MongoDB
- **Tools & Platforms**: Git, GitHub, Docker, Vercel, Vite, Qiskit, Figma
- **Special Interests**: Web Dev, Quantum, Open-Source Contributions

I also have been learning in public and here are some posts about that: Medium post**.** I have also been teaching competitive programming to my juniors, here Is the post on the introductory session : Coder's Club. A lot more stuff can be found on my socials! LinkedIn post

Some projects:

1. **Quantum Quest** (Python, Qiskit, React.js, Jupyter Notebook) Github

   - Developed a Quantum Random Number Generator with 94.7% randomness validation accuracy.
   - Built a React dashboard for interactive visualization of quantum algorithms.
   - Applied Grover's and Shor's Algorithms, showcasing my ability to work with structured programmatic logic, which is essential for developing Music blocks' visual programming interface.

2. **IIIT Kottayam Placement Cell Website** (Next.js, React.js, Tailwind CSS, Figma) Github website
   - Designed a fully functional placement portal, improving user engagement by 25%.
   - Created interactive UI elements in React.js to automate recruiter tasks, reducing manual effort by 40%.
   - Demonstrates my ability to build modular, scalable components—skills that directly translate to developing Music blocks' brick-based programming modules.

3. **Sugar Labs Contribution** (Github)
   - Developed a dynamic, responsive donation banner for Sugar Labs, enhancing UI/UX.
   - Led UX/UI improvements on PR #603 and collaborated with contributors on strategic improvements.
   - This demonstrates my familiarity with Sugar Labs' development environment and open-source collaboration, making me a strong fit for contributing to Music blocks.

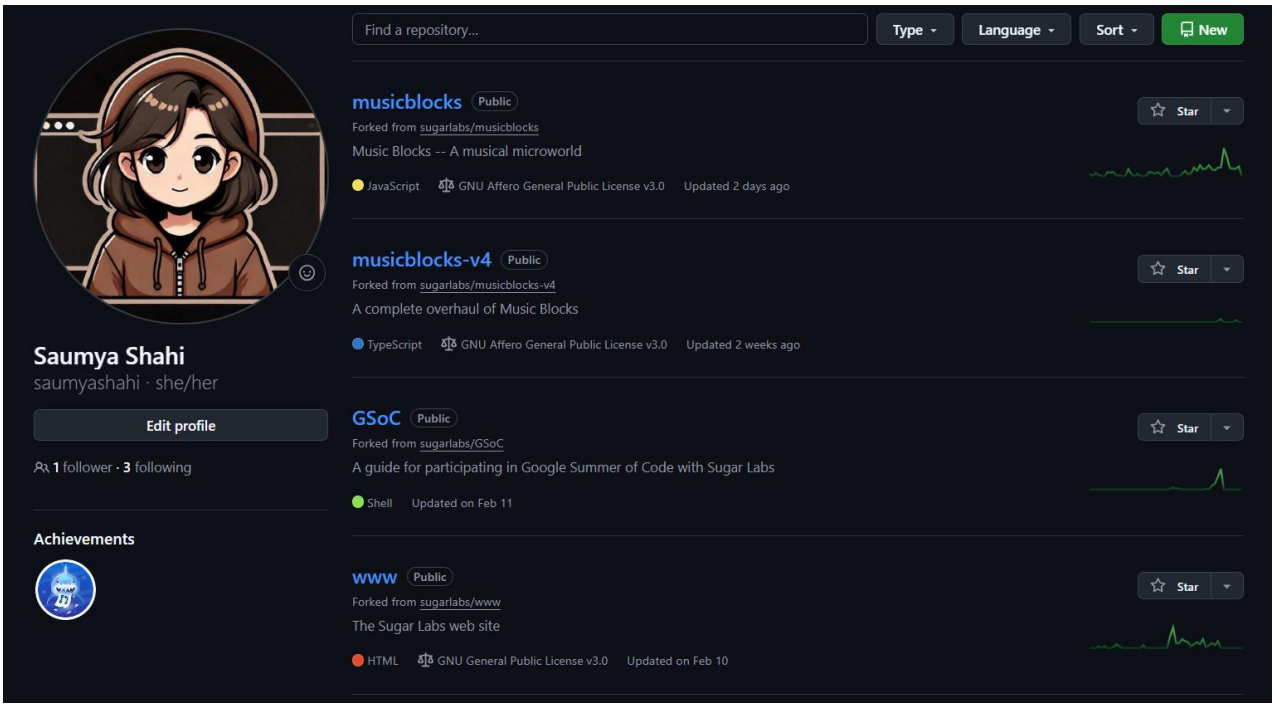
Why Sugar Labs?

My journey with Sugar Labs and the Music blocks project is a result of my love for music, technology and education. Education has been a changing point in my life. I feel my knowledge is the most valuable asset that I have and finding the best teachers and resources that help students like us grow is definitely amazing. Discovering Sugar Labs, providing this to children, all over the world, for free has not just been inspiring but has made me contribute to this organization.

As I delved into the project, I found its simplicity remarkable, and exciting to be building an enhanced version of music blocks as if it's built from scratch, enabling me to quickly immerse myself. Through experimentation and exploration within the repository, I thoroughly understood its architecture and functionality, making me contribute meaningfully to Music blocks v3. This is a program I created with Music blocks :learning to use music blocks: https://musicblocks.sugarlabs.org/index.html?id=1744057068516992&run=True

My involvement extended beyond contributions; I actively engaged with the community, sharing insights and collaborating on solutions. This collaborative environment fueled my enthusiasm, fostering a sense of camaraderie and shared purpose.

My dedication to the Music blocks project remains unwavering throughout my journey with Sugar Labs. Combining my passion for music with coding proficiency has been immensely fulfilling, driving me to contribute to democratizing musical expression and fostering creative exploration for all.



Here is a detailed summary of the work and contributions that I did before the GSoC proposal period:

Issues Raised:

- **[Bug] Sugar Lab's website navigation bar is not responsive(#579)**
  - The navigation section wasn't responsive on desktop as well as Mobile devices which led to a bad user experience as the user would not be able to view content.

- **[Documentation] Inconsistent Heading and Description for "Context Menu" in Music blocks Documentation (#4235)**
  - Move the context menu text from the toolbar section to its appropriate location in the context menu section.
  - Adjust structure for clarity and accuracy to ensure sections align with their intended topics.

Pull Requests:

- **Make Donation Banner Responsive and Sync Logo Colors Dynamically SugarLabs Website (#603) (merged)**
  - Fixed issue **#603, #579 , #574 , #575**
  - Add media queries to adjust the banner layout for different screen sizes.
  - Implement a JavaScript function to dynamically fetch and apply the logo color to the banner.

- **Sugar Stories Page and Authors added (#618) (merged)**
  - Create sugar-stories.html for the new website.
  - Implement markdown syntax posts into the "stories" category.
  - Automate story uploads by adding markdown files.
  - Dynamic Author Display
  - Fixes #571. Continuation of #615.

- **Refactor structure and fix broken links in documentation (#4246) (merged)**
  - Adjust structure for clarity and accuracy to ensure sections align with their intended topics.
  - Remove redundant content for Context Menu's
  - Fixed issue #4246, #2403

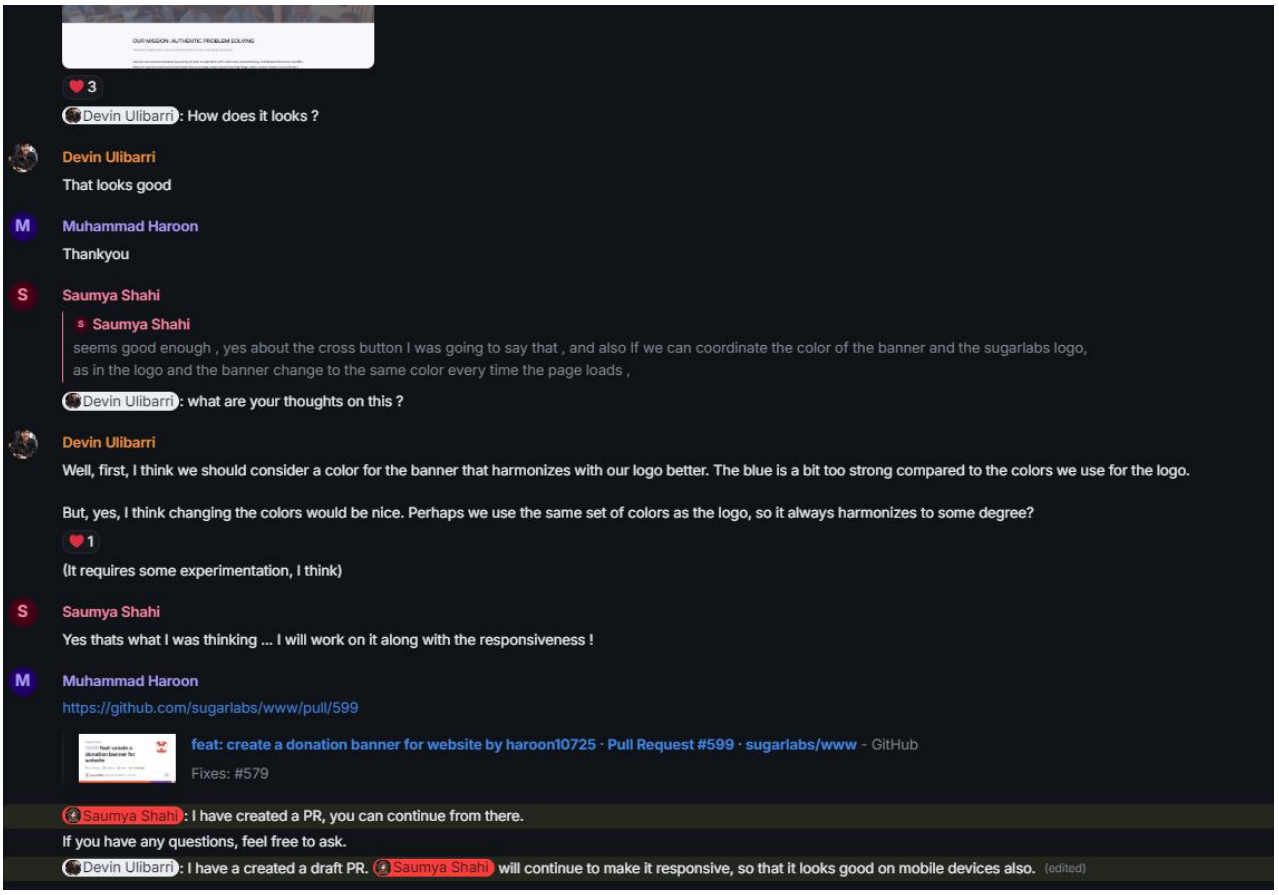
- **Fix image scaling and restore list rendering in blog posts (#627) (merged)**
  - Made images responsive by adding max-width: 100%.
  - Updated list styles to restore bullet points

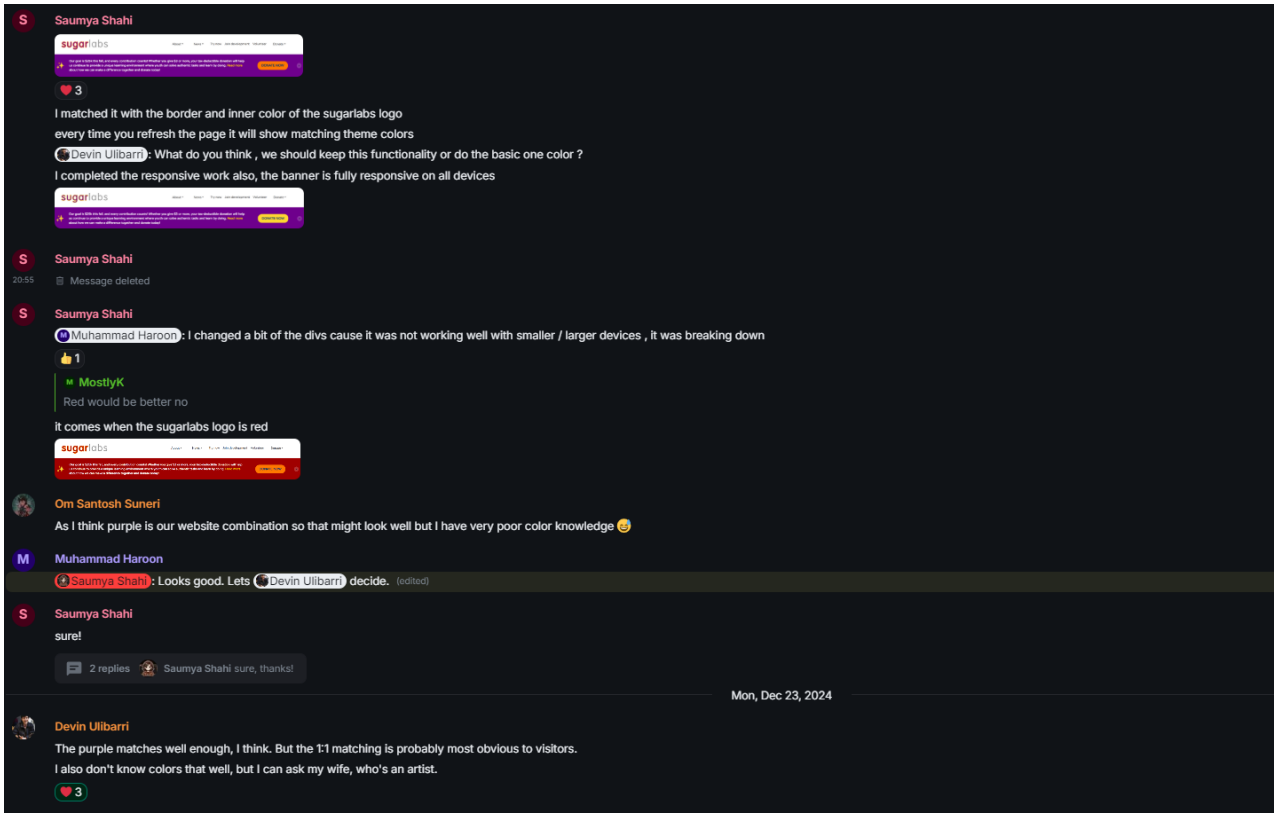
- **Other merged pull requests (Link)**
  - Fix: typos and formatting issues in GSoC 2025 Ideas document.

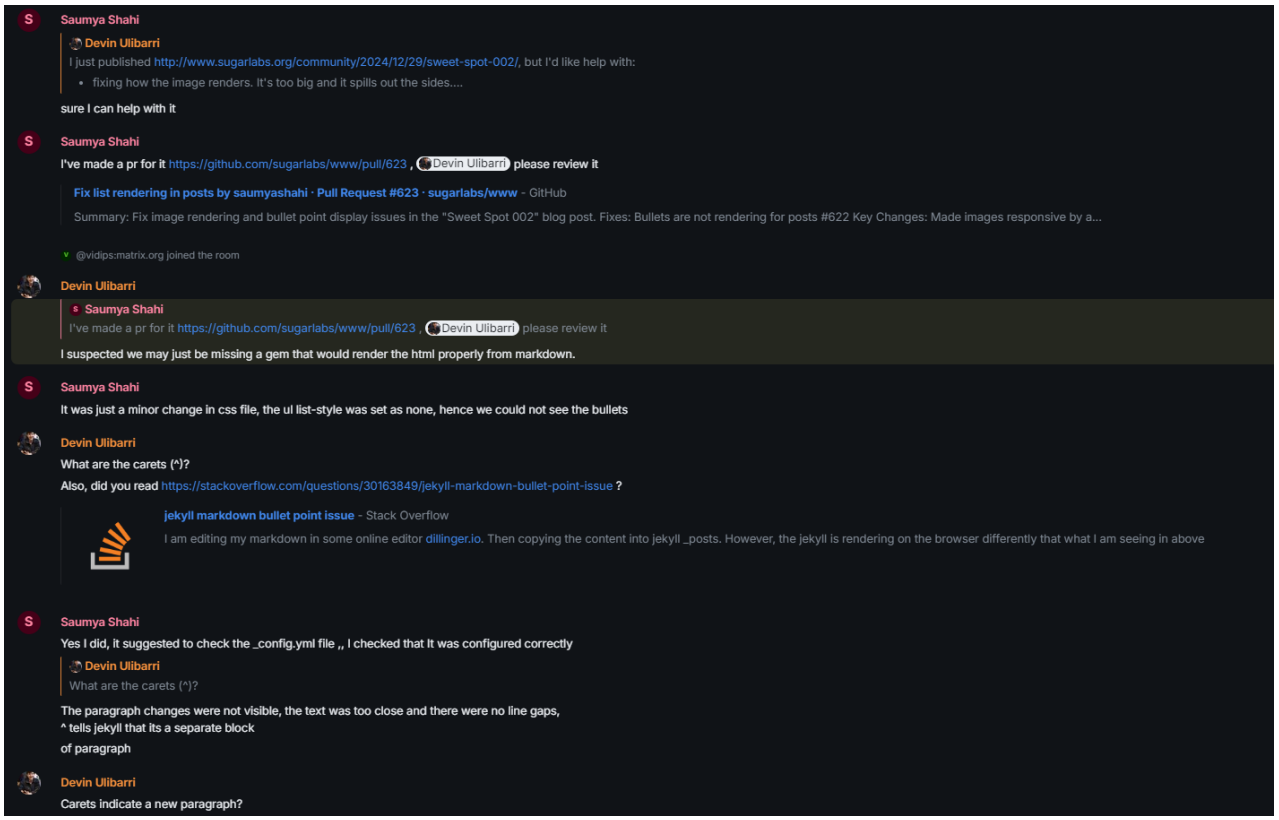
Presence in the Community:

As an open-source contributor, I prioritize assisting new contributors to engage with the project's issues. Active involvement within the community and regular participation in biweekly meetings underscore my commitment to fostering collaboration and supporting others in their contributions.

A few screenshots from the Element public conversation are here:







I have helped other developers set up the project and make contributions to GitHub. Apart from these, I **have had in-depth conversations about the project** with my mentors Anindya Kundu and Devin Ulibarri.

Project Details:

Introduction

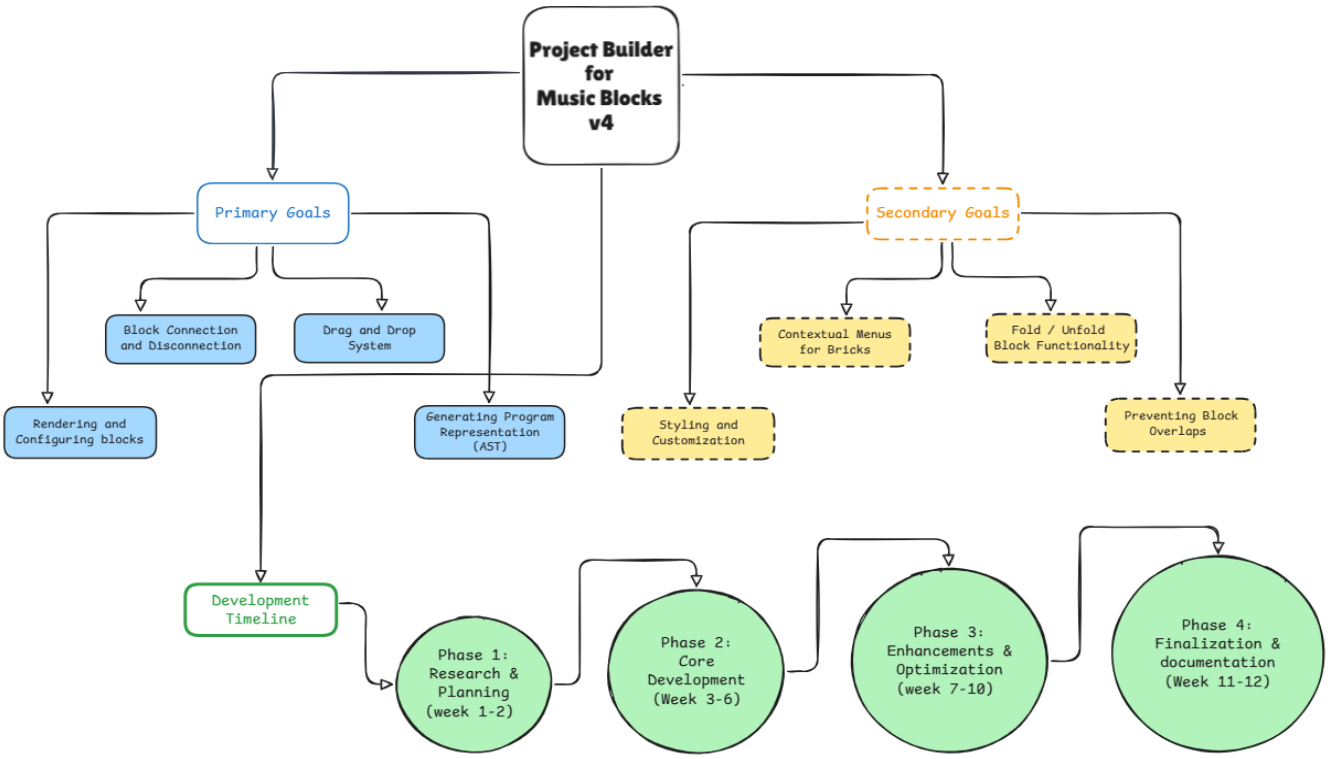
The Music blocks v4 project represents a significant evolution from its predecessor, Music blocks v3, which was originally derived from Turtle blocks, transitioning from HTML, CSS, and Vanilla JavaScript to **React v18 with Functional Components and Hooks, TypeScript, and modern state management solutions**. This shift enhances maintainability, scalability, and performance.

While the transition has laid a solid foundation, **one critical aspect still requires significant development—the Project Builder.** The Project Builder is the core graphical interface where users **visually arrange musical blocks to create compositions**. It serves as an intuitive programming environment where each block represents musical elements like rhythm, pitch, and instruments.

The Project Builder is a **fundamental module** of Music blocks v4, designed to provide a graphical, interactive environment where users can create music by visually arranging blocks. Each block represents a different musical or functional component, such as **rhythm, pitch, notes, and instruments**. The bricks are highly interactive, supporting drag-and-drop functionality, click actions for contextual options, and the ability to delete or modify blocks dynamically.

I have carefully studied the project's previous contributions, coding patterns, and best practices to identify **areas for improvement** and development of the project builder module of Music blocks . My focus will be on building essential components while ensuring the codebase remains **clean, optimized, and extensible**. The attached diagram outlines my **planned contributions over the summer**, though the execution sequence may be adjusted based on project needs and feedback:

Have a detailed look at the diagram here : Link



Problem Statement

The current Project Builder lacks essential functionalities to efficiently support **building, modifying, and managing structured music programs**. The challenges include:

- **Rendering and Configuring Bricks Dynamically** – The system must allow **any musical block to be rendered based on external configurations** while ensuring performance and clarity.
- **Drag-and-Drop Block Interaction** – Users need an intuitive way to **drag bricks from a palette, place them in a workspace, and modify them dynamically**.
- **Block Connectivity and Validation** – bricks should be **connectable and disconnectable**, with a logic system to **validate valid connections** dynamically.
- **Program Representation & Serialization** – The Project Builder must **extract a structured representation (AST) of the arranged bricks** to preserve the composition's logic.
- **Performance Optimization** – The block-building system should be lightweight and **optimized for smooth user interactions**.

Why This Matters

These improvements are crucial for making Music Blocks **more accessible and intuitive for users of all levels, from beginners to advanced composers.** By ensuring an efficient, well-structured block-building experience, users can focus on musical creativity rather than struggling with interface limitations.

Proposed Approach

To address these issues, my GSoC project will focus on:

1. **Developing a modular, extensible system** to dynamically render and manage musical bricks:
2. **Implementing an optimized drag-and-drop interface** with smooth, intuitive interactions.
3. **Integrating a robust AST-based serialization system** for program representation.
4. **Ensuring performance optimization** for seamless user experience.

This structured approach will **transform the Project Builder into a powerful, interactive music programming environment**, aligning with the goals of Music blocks v4.

Product Requirements Document (PRD)

1. Overview

Project Name: Music blocks v4 - Project Builder Module
Objective: Develop an interactive block-based programming interface for users to create musical patterns visually.

The **Project Builder Module** allows users to drag, drop, connect, and manage music programming blocks, ensuring a smooth and user-friendly composition experience.

2. Goals & Objectives

Primary Goals (Core Functionality)

- Render bricks dynamically based on configurations.
- Provide a palette of bricks with external configuration support.
- Implement drag-and-drop functionality for placing blocks on the workspace.
- Enable users to connect and disconnect bricks logically.

- Implement block deletion by dragging bricks to a designated area.
- Generate a serializable AST representation of the block program.
- Optimize performance for seamless interaction.

Secondary Goals (Enhancements & UI Improvements)

- Implement styling options for bricks.
- Introduce contextual menus for brick interactions (duplicate, extract, delete).
- Enable folding/unfolding of block structures.
- Support resizing and scaling of bricks in the workspace.
- Ensure dynamic height adjustments for parent bricks based on child elements.
- Provide visual feedback when connecting or disconnecting bricks.

Tertiary Goals (Advanced Features)

- Implement switchable brick types (e.g., If to If-Else, Pitch to Octave/Hertz).
- Add search functionality in the palette.
- Prevent block overlaps for better visual clarity.
- Add a visualizer for programming bricks.
- Integrate debugging features like step execution and breakpoints.
- Support commenting for documentation within block structures.

3. Feature Specifications

Feature	Description
Brick Rendering	Dynamically render bricks using SVGs with external configurations.
Drag-and-Drop	Allow moving bricks from the palette to the workspace.

Block Connections	Enable snapping bricks together based on logic rules.
Deletion System	Implement a trash area for removing bricks.
AST Generation	Convert the block structure into a serializable tree format.
UI Enhancements	Improve interaction with animations, styling, and layout fixes.

Major Objectives for GSoC 2025:

Here are the major objectives on which I want to work on developing Music blocks v4 Masonry Module as a part of Google Summer of Code 2025.

1. **Creation of an SRS / SDD (Design) Document: To outline the project's architecture, technical specifications, and development roadmap, ensuring clarity and alignment with project goals.**

Implementation:

1. Clearly outline the objectives, features, and deliverables of the project to establish a clear understanding of its scope after discussion with the mentors.

2. Provide detailed technical specifications for each component, including APIs, algorithms, and any external libraries or frameworks to be used.

3. Outline a development roadmap with milestones, timelines, and task breakdowns to guide the implementation process and ensure progress tracking.

2. **Be Able to Render Any Brick Given Some Configurations**

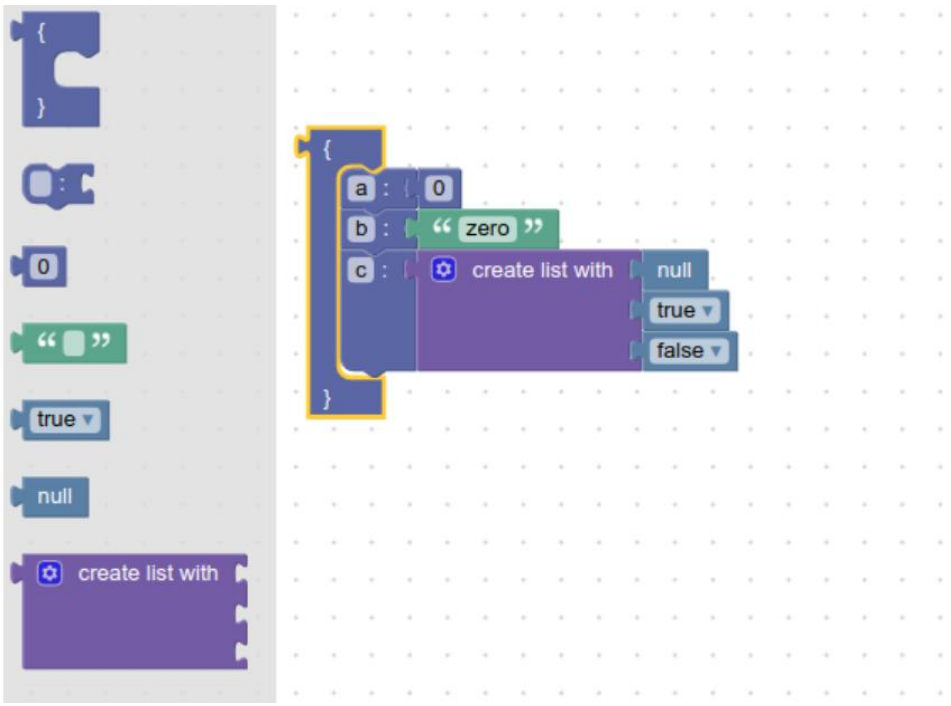
The current brick rendering process in Music blocks v3 is tightly coupled to predefined templates, making it rigid and difficult to scale. Blocks are not fully dynamic—their structure, layout, and appearance must be manually defined, which adds friction when introducing new types or adapting to new use cases. Additionally, the lack of a standardized, reusable rendering pipeline limits the ability to evolve the platform efficiently.

To address these challenges, I propose a fully configuration-driven rendering system. This system will interpret a JSON-like configuration object and dynamically generate the corresponding visual brick. Each brick's type, layout, connectors, and internal components will be determined by this object, separating logic from design and enabling easy scalability.

2.1 Configuration Object Structure

At the heart of this system will be a schema that defines the structure of each brick. This schema will include properties like:

- **Brick type and category** – Determines default styling and placement logic.
- **Input slots** – Can include text fields, dropdowns, or numerical inputs, depending on the brick's purpose.
- **Outputs and connectors** – Indicate which directions (top, bottom, lateral) the brick connects from or to.
- **Shape and layout** – Bricks can have different silhouettes, such as rounded or hexagonal, and internal arrangements like "Set Instrument to [Dropdown]".
- **Styling options** – Including font, icon, padding, background color, and border radius.
- **Event hooks** – Optional interactive behavior (e.g., `onClick`, `onDrop`) to provide dynamic user feedback.



(reference image is from blocky by google)

2.2 Modular Brick Rendering Engine

The implementation will involve building a modular rendering engine, likely using React components. This engine will be responsible for interpreting the configuration object and producing the correct DOM or SVG elements. Internally, it will:

- Parse label fragments and mix text with dynamic input components.
- Determine layout spacing and apply appropriate padding and margin.
- Render connector pins or joins as defined in the config.
- Attach event handlers for interactive behavior like click or drag actions.

This modularity will allow the rendering logic to remain clean and extensible, encouraging contributions and experimentation without risking regressions

2.3 Dynamic Input Handling

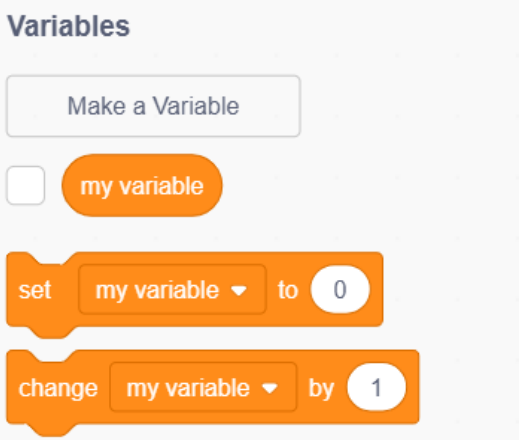
For bricks that require user interaction (e.g., selecting a value from a dropdown or entering a number), I will implement controlled React components inside each brick. These inputs will be tightly bound to the underlying brick data model, ensuring that any interaction reflects instantly in the program's state.

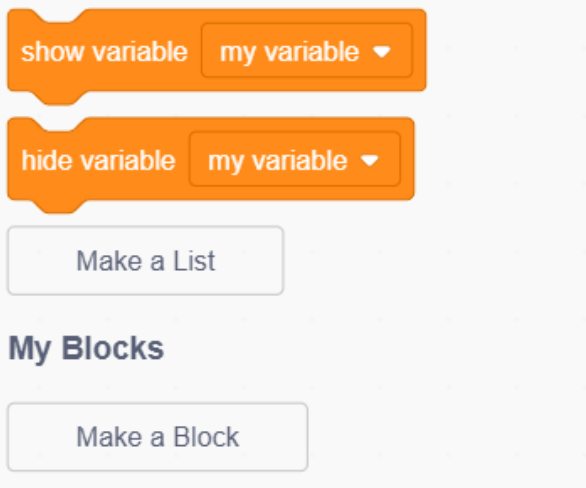
2.4 Flexible and Theme-Aware Styling

To support theming and ensure visual consistency, styles will be dynamically injected through context providers or props. I'll rely on CSS variables or inline styles to manage things like colors, font sizes, and border styling—ensuring adaptability across light/dark themes or user-customized palettes.

2.5 Reusability and Extensibility

A major design goal is to make adding new bricks as easy as defining a new configuration object. No additional custom rendering logic should be necessary unless the brick has very specialized behavior. This level of abstraction will allow designers, educators, and developers to iterate faster without diving into low-level code.





(Image from scratch
My blocks)

3. Drag-and-Drop Functionality with Intuitive Placement

A seamless drag-and-drop experience is crucial to making Music blocks intuitive and accessible, especially for young learners and first-time users. At present, the dragging mechanism feels disjointed and often results in imprecise placement, with bricks landing in arbitrary positions. To address this, I plan to implement a responsive and structured drag-and-drop system that blends freedom with control, allowing users to explore while maintaining organized logic flows.

This system will not only support freeform interactions but also gently guide users through real-time visual feedback, alignment assistance, and smart collision handling.

3.1 Smooth Drag-and-Drop Mechanism

To improve the user experience, the dragging system will rely on low-level pointer event handling (`mousedown`, `mousemove`, `mouseup`) for real-time tracking. This allows for fluid motion across devices and screen sizes. While users should feel free to drag a block anywhere, grid snapping will gently align the blocks to a structured layout, improving visual consistency.

- Capture drag motion via precise pointer events for pixel-perfect tracking
- Snap block movement to a configurable grid for cleaner arrangement
- Use event batching to optimize re-renders during continuous motion
- Provide fine-tuned responsiveness across browsers and input devices

3.2 Visual Feedback for Placement Assistance

One major gap in the current UI is the lack of feedback when placing a block. Without clear visual cues, users are left guessing where a block will land or whether it will connect. To make placement intuitive, I will introduce a ghost or preview block, as well as dynamic highlights to indicate valid drop zones and connection points.

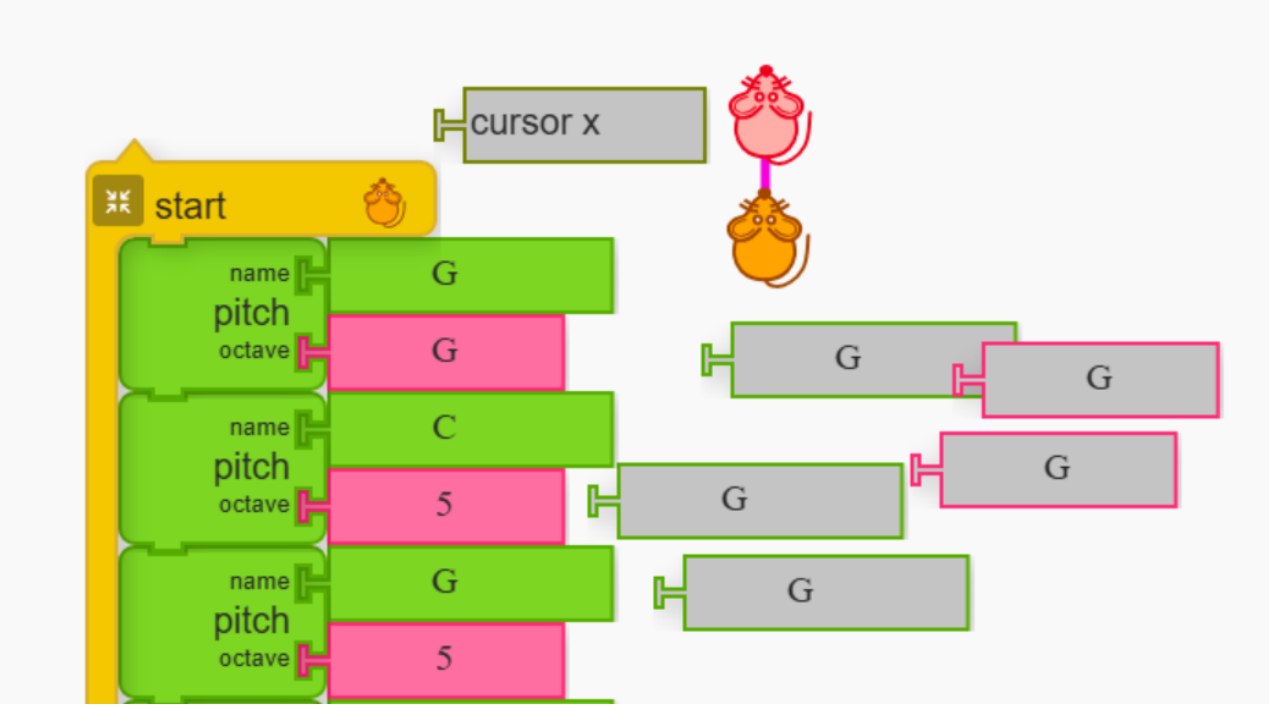
- Display a semi-transparent preview of the dragged block at its intended drop location

- Dynamically highlight nearby valid connection points when a block is in proximity
- Add magnetic guidelines or helper lines to encourage clean alignment
- Animate snapping transitions when a block connects, improving clarity and satisfaction

3.3 Collision Detection for Overlapping blocks

Currently, users can unintentionally stack or overlap bricks, leading to visual clutter and logical confusion. To prevent this, bounding box collision detection will be implemented. It will check for overlaps in real-time and resolve them by gently nudging bricks to nearby valid positions. This ensures that every block has its own space and remains accessible.

- Continuously detect bounding box intersections between the dragged block and others
- If overlap occurs, auto-adjust the dragged block to the nearest available position
- When no valid spot is found, apply a soft error cue (e.g., red border or shake)
- Allow users to override this behavior manually in case of intentional overlap scenarios

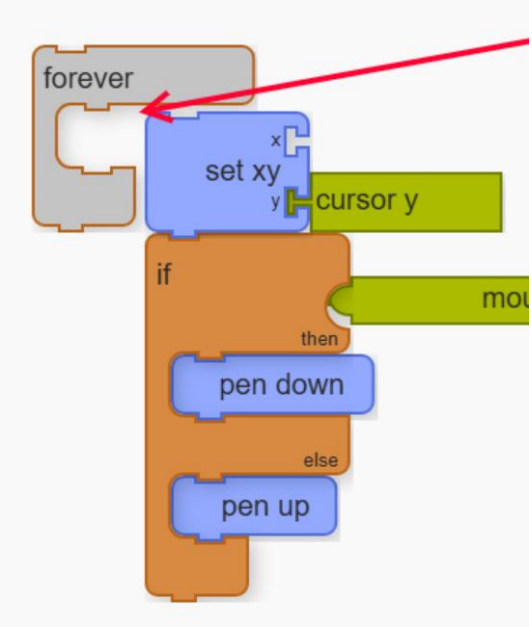


3.4 Undo and Redo System for Drag Actions (Future Enhancement)

While not essential for the initial version, having an undo/redo system will be a valuable future addition. Users should be able to quickly revert accidental placements without having to manually reposition the block.

- Maintain a history stack of movement actions for each block
- On undo, revert the last move and reapply snap-to-grid logic
- Implement redo to reinstate an undone move if needed
- Support keyboard shortcuts (Ctrl+Z and Ctrl+Y) for efficient editing

4. **Implementing, Improving the collision detection and Brick functionalities based on distance between bricks:** The goal is to create an intuitive, responsive, and visually clear system that facilitates brick management through drag-and-drop, collision detection, and seamless block connections. By implementing features like visual feedback during block connections, intuitive selection and highlighting, minimalistic context menus, and an improved deletion mechanism, we will significantly improve the usability and user interface of the platform. These changes will ensure that users can interact with the workspace more naturally, improving overall productivity and satisfaction.



Implementation:

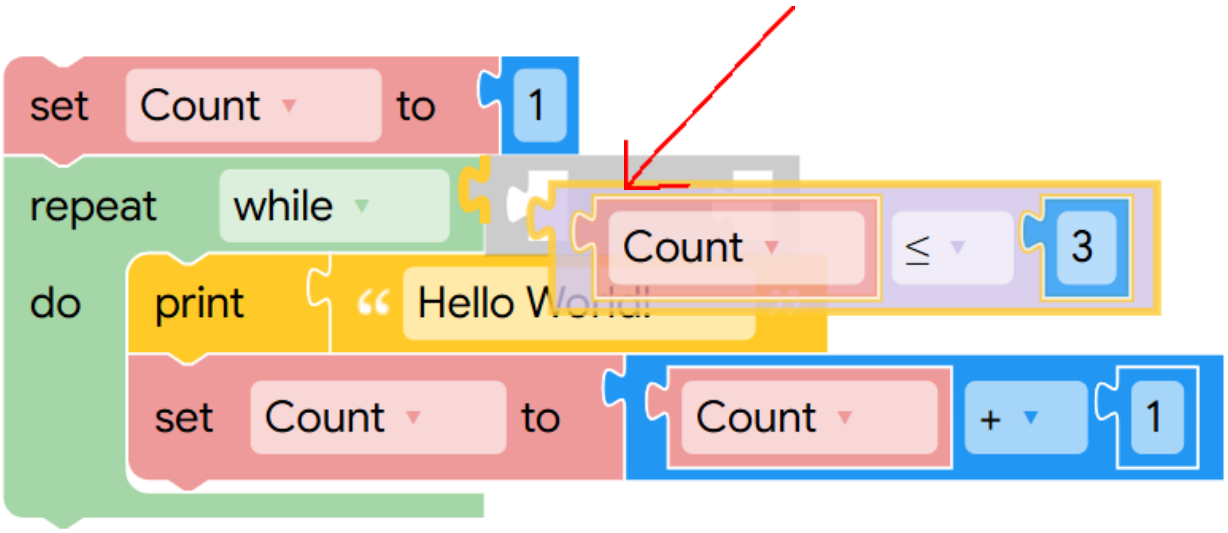
4.1 Block Selection and Highlighting:

- **Current Issue:** In Music blocks v3, block selection does not provide any clear visual indication when a block is selected.
- **Proposed Solution:** When a user selects a block, it should be visually highlighted. This can be achieved by outlining the selected block with a glowing border or changing its background color to indicate that it is active. This will make it clear to users which block they are interacting with and will improve overall usability.

4.2 Collision Detection with Preview:

- **Current Issue:** There is no visual feedback when dragging blocks near one another, making it difficult for users to see where bricks might collide or fit together.

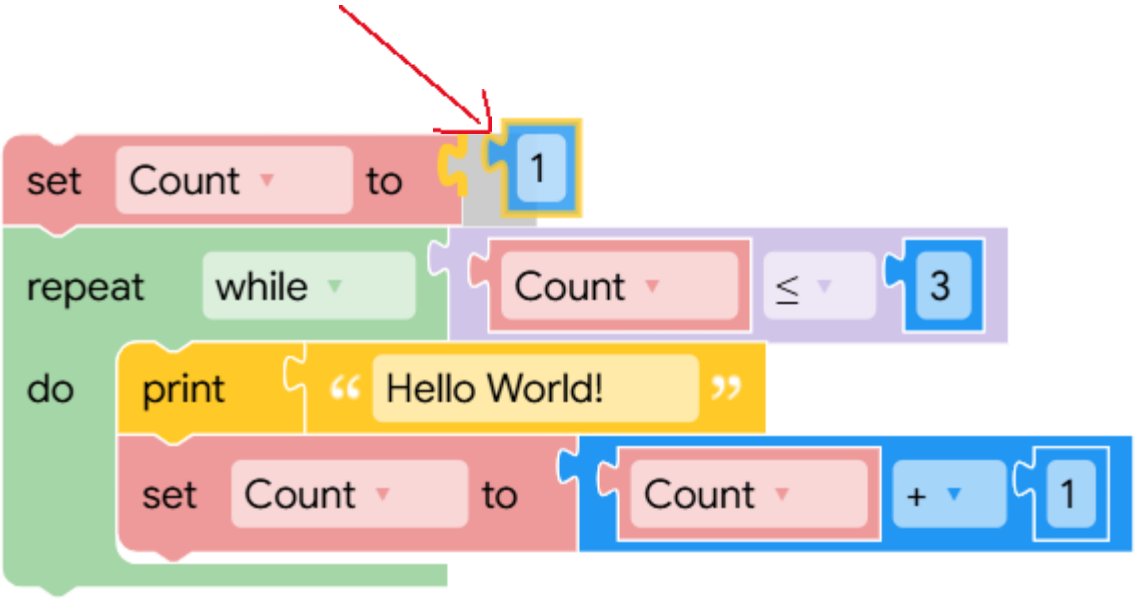
- **Proposed Solution:** When a block is being dragged, a transparent preview of the block should be shown. As it gets close to another block, it should show visual feedback (e.g., a shadow or an outline) indicating potential connections. If a collision is detected, the bricks should snap into place, creating a smooth and intuitive joining experience. This feedback should follow the behaviour of other block-based programming tools like Blocky.



1. Highlighting Connection Points:

- **Current Issue:** Users may not know where bricks will connect or snap together.

- **Proposed Solution:** When dragging a block over another, the target block's connection points (where it can join with others) should be highlighted. These connection points can be visually emphasized by changing the block's color or adding a subtle effect (like a pulse or glow). This will provide clear guidance to users about where blocks can be attached, ensuring smoother and more accurate block connections.



4.3 Improved Context Menu:

- **Current Issue:** The current circular context menus are difficult to read and don't offer a smooth user experience.

- **Proposed Solution:** Replace the circular context menus with a more minimalist and traditional dropdown menu. This will make the menu items easier to read and navigate. The dropdown should be aligned with the block and offer clear, concise options such as duplicate, resize, delete, etc. The new menu should also support keyboard navigation for power users.

4.4 Collision Detection Optimization (Quadtree):

- **Current Issue:** As the number of bricks increases, collision detection can lead to lag and slow performance.

- **Proposed Solution:** Implement a Quadtree algorithm for collision detection. The Quadtree will partition the workspace into regions, allowing for faster detection of collisions by limiting the number of checks needed. This will reduce performance issues, especially when handling large numbers of blocks. The algorithm will update the Quadtree structure dynamically as bricks are moved and resized, ensuring that collision detection remains efficient even with a large number of objects.

4.5 Dynamic Resizing of bricks Based on Distance:

- **Current Issue:** bricks are static in size, and as they get close to each other, it may be difficult to fit them together properly.

- **Proposed Solution:** Introduce dynamic resizing of bricks based on the distance between them. When bricks are near each other, they should automatically resize to create enough space for proper fitting. This resizing will be based on the distance detected between bricks and will create a smoother and more visually appealing interaction, allowing users to connect bricks even if they are closely packed.

5. Be able to delete bricks/blocks by dragging them to a corner of the workspace

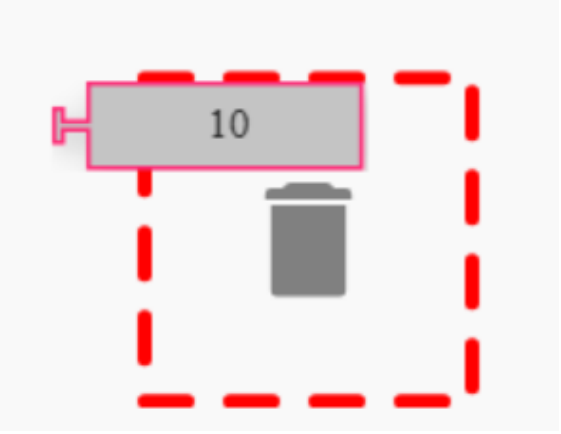
Deleting bricks should be just as intuitive as placing them. Right now, there's no clear visual or interactive way to remove bricks from the workspace. Beginners, especially kids, may not be familiar with right-click menus or keyboard shortcuts, making deletion a friction point. To solve this, I plan to introduce a “Trash Zone” — a dedicated area of the workspace that lets users drag and drop blocks for deletion, making the interaction more visual and discoverable.

This drag-to-delete interaction mimics familiar metaphors (like dragging files to a trash bin) and provides clarity through animations and feedback, ensuring the action feels intentional and reversible.

5.1 Trash Zone Integration

The first step is to introduce a clear and accessible trash zone in one corner of the workspace — most likely the bottom-right. This will either always remain visible or appear subtly when dragging starts. The zone should visually stand out using a semi-transparent trash icon or label, but not obstruct the working area.

- Place a fixed trash zone in a corner (e.g., bottom-right) of the canvas
- Display a minimal icon or box labeled "Trash" with hover effects
- Fade or animate the zone into focus only when a drag begins



5.2 Drag-and-Drop Detection for Deletion

The drag-and-drop system will be extended to recognize when a block is being dragged into the trash area. Once a brick enters this region, the trash zone will respond with visual cues like background color shifts or a gentle pulse animation to signal that the block is about to be deleted.

- Detect bounding box intersection between the dragged brick and the trash zone
- Highlight the trash area when a brick hovers near it
- Use visual feedback like glow or pulse to indicate readiness to delete

5.3 Handling Deletion of Single Bricks and Block Trees

When a brick is released over the trash zone, the system should confirm and execute deletion. If the brick is part of a larger connected structure, all downstream (child) bricks should be deleted automatically, unless a user setting disables that behavior. This prevents leftover orphan bricks and keeps the workspace clean.

- On drop, delete the brick or the full connected block beneath it
- If part of a chain, traverse and remove all linked children
- Animate the fade-out of bricks to indicate deletion
- Optionally, show a toast saying “Block Deleted” for feedback

5.4 Safeguards and Edge Case Handling

Accidental deletions must be avoided. Simply hovering over the trash zone shouldn’t trigger deletion — only dropping should. To support error recovery, the system should also track recent deletions and allow an undo operation. For large brick groups, a confirmation prompt or soft undo can be added.

- Only trigger deletion on drop, not on accidental overlap
- Add undo capability (Ctrl+Z or UI button) for deleted bricks
- Consider a confirmation setting for deleting complex structures

5.5 Performance & Clean-Up

Deleting bricks isn't just visual — the logic should also clear up memory, remove event listeners, and update internal data models (such as the AST). The process should be fast and avoid triggering unnecessary full re-renders.

- Ensure memory references to deleted bricks are cleaned
- Trigger minimal and scoped re-renders
- Update the AST and other internal models to reflect deletion

6. Abstract Syntax Tree (AST) Integration for Brick-Based Programming

Current Issue: The existing system lacks a structured representation of the workspace, making it difficult to analyze, modify, or export block-based programs efficiently. Without an AST (Abstract Syntax Tree), there is no clear way to represent relationships between blocks, which affects execution, debugging, and external integrations.

Proposed Solution: Implement an AST to represent the hierarchical structure of blocks programmatically. This will ensure:

- A structured and scalable way to interpret block-based logic.
- Easy serialization and deserialization of programs for saving/loading.
- Compatibility with code generation and execution engines.
- Efficient modifications such as rearranging or replacing parts of the program.

Implementation Plan:

1. AST Representation and Structure

- Define a standardized AST format, where each brick corresponds to a node in the tree.
- Nodes should store relevant data, such as the brick type, connections (parent-child relationships), and configurable parameters.
- Use a hierarchical model to distinguish between control structures (loops, conditionals) and standard action bricks.

2. Brick-to-AST Conversion

- Each time a brick is placed, moved, or deleted, update the AST in real time.
- Implement a parser that converts workspace blocks into an AST representation.
- Ensure blocks with nested structures (loops, conditionals) properly reflect their child relationships in the tree.

3. AST-to-Brick Rendering

- Implement a mechanism to generate visual blocks from an AST, allowing for seamless reloading of saved programs.
- When loading a project, reconstruct the workspace layout using AST data.
- Maintain synchronization between the AST and UI elements to prevent inconsistencies.

4. Performance Considerations

- Optimize updates to avoid redundant re-processing of the entire AST when minor changes occur.
- Implement efficient diffing algorithms to track changes without full re-computation.
- Ensure real-time updates without introducing performance bottlenecks.

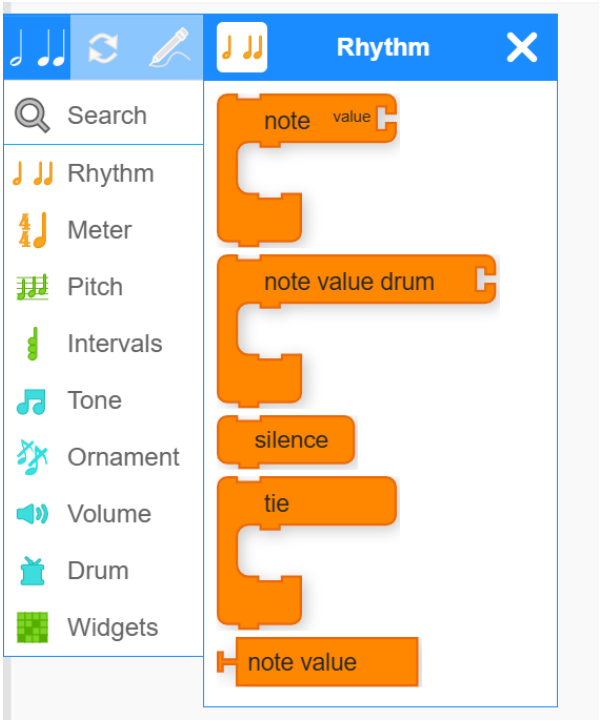
This AST integration will serve as the backbone of the block-based programming system, enabling a structured, scalable, and efficient way to represent and manipulate programs.

***Palette Feature and Infinite Scroll Enhancement: the palette feature will be introduced, along with infinite scroll functionality, allowing users to effortlessly browse and add bricks to their workspace. By organizing blocks into categories and implementing drag-and-drop functionality, users will be able to navigate and utilize music blocks in a more intuitive and efficient way**

Implementation:

1. We can load some hard coded bricks for each type of option and use cases. We can detach the block from the palette which has been dragged into the scene by updating the values and detecting collisions with the already existing blocks on the scene. Also, we need to refill the block into the palette which has been detached, by again simply adding the same block with the same values and state into the specific palette array.

2. The palette will display a categorized set of bricks, grouped into sections such as Music, Flow, and Graphics. This will help users quickly locate the bricks they need, ensuring the interface remains organized and uncluttered. The block categories will be clearly labeled with tabs or dropdowns for easy navigation, making it easier for users to find and select the right block for their needs. I will be taking inspiration from the original palette:



3. As users scroll through the palette, more blocks will load dynamically, allowing them to access an extensive range of options without overwhelming the interface. Infinite scroll will ensure that the loading of additional bricks feels seamless, with no visible pauses or loading indicators that disrupt the user experience.

4. The drag-and-drop functionality will allow users to easily select and add bricks to their workspace. This interaction will be intuitive, enabling users to drag a block from the palette and drop it into the workspace with minimal effort. The dragged blocks will automatically snap into place, aligning with the grid or predefined layout, ensuring that the workspace remains organized.

5. Both the palette and the workspace will be treated as child components within a parent container. A draggable overlay will allow users to move bricks freely between the two components. The layout will be designed to make the palette easily accessible, while also enabling smooth transitions when adding bricks to the workspace.

6. To ensure the effectiveness of these features, user testing will be conducted to identify any pain points or areas for improvement. Feedback will be gathered on the usability of the drag-and-drop functionality, infinite scroll behavior, and the overall organization of the palette.

7. Integration of the Project Builder into Music blocks v4: The goal is to seamlessly integrate the Project Builder framework into Music blocks v4, ensuring all programmatic features and visual components work cohesively within the upgraded system. This will consolidate development efforts and enhance the application's overall functionality.

Implementation:

1. **Framework Incorporation:**
    - Integrate the **Project Builder** framework into **Music blocks v4** by ensuring all necessary dependencies, configurations, and module imports are correctly structured.
    - Align the **Builder's architecture** with Music blocks' existing state management and rendering flow.
2. **Feature Validation & Compatibility Testing:**
    - Ensure that all interactive blocks (e.g., **Start Block, Rhythm Block, Note Block, Pitch Block**) function as expected within **Music blocks v4**.
    - Test interactivity features like **drag-and-drop**, **block snapping**, **context menus**, and **customizable block parameters** to confirm smooth user experience.
    - Identify and resolve **any compatibility issues** between the existing **Music blocks architecture** and the newly integrated **Project Builder framework**.
3. **Extensive Testing & Debugging:**
    - Conduct **unit tests** to verify the correctness of individual modules.
    - Perform **system and integration testing** to ensure the Builder operates within Music blocks v4 without breaking existing functionality.
    - Address any **bugs, performance bottlenecks, or unexpected behaviors** that arise during integration.

This integration will solidify **Music blocks v4** as a complete, interactive music composition tool **(MVP)** , providing users with a structured and intuitive block-based programming experience.

Secondary Goals

1. **Customizable Brick Styling**
   Bricks should have configurable styles, including color, border-radius, font size, and icons. This will help differentiate categories like loops, conditionals, and sound bricks. A style object in the brick configuration will define these properties, applied via inline styles or class names, with sensible defaults based on category.

2. **Context-Specific Menus**
   Each brick type should have a right-click (or long-press) context menu with relevant options such as "Duplicate," "Extract," or "Convert." A context menu manager will handle this dynamically, with brick configurations specifying available actions. The menu system will trigger these actions through an API integrated with the main application.

3. **Brick Duplication and Extraction**
   Users should be able to duplicate bricks, extract child bricks from their parent, or collapse blocks. This enhances workflow efficiency and provides greater flexibility in structuring programs. Duplication will use deep cloning, while extracted bricks will be repositioned independently in the workspace.

4. **Collapsible bricks**
   To manage visual clutter, compound structures like loops and conditionals should be collapsible. A toggle icon will allow users to fold/unfold bricks. When collapsed, child blocks will be hidden and the block shrunk; expanding will restore them to their original size.

5. **Workspace Scaling**
   Users should be able to zoom in and out of the workspace to adjust the view based on the complexity of their program. This will be implemented using CSS transformations or canvas scaling while ensuring that drag-and-drop interactions remain smooth and precise.

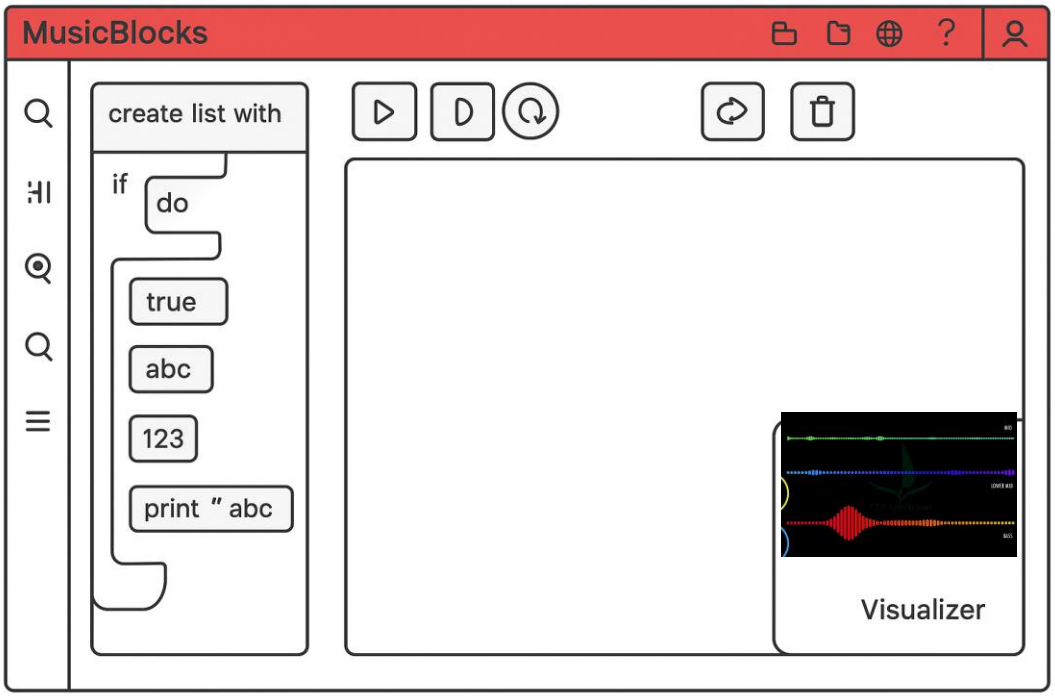
6. **Dynamic Parent Brick Resizing**
   Parent bricks, such as loops or conditionals, should automatically resize based on the height of nested child bricks. This prevents overlapping or clipping in deeply nested structures. Using `ResizeObserver` or manual recalculations, the system will adjust parent dimensions dynamically.

7. **Visual Feedback for Brick Connections**
   When connecting or disconnecting blocks, users should receive clear visual cues. Valid connection points will be highlighted on hover, successful placements will trigger snap-in animations, and invalid placements will be indicated through effects like shaking or flashing warnings.

A tertiary goal (Good for future) can be Visualizer for Real-Time Feedback: The **small box in the bottom/top right corner** will serve as a **visualizer** that:

- **Displays musical patterns and rhythms in real time.**
- **Animates based on user interactions** (e.g., when a block is executed, it triggers a corresponding animation).
- **Provides a waveform view or note progression** to help users see the output of their musical sequences.
- **Includes playback controls** for pausing, replaying, or stepping through compositions.

This feature will make the experience **more interactive and engaging**, allowing users to **see and hear their compositions** as they build them.



Wireframe Representation

Other objectives as part of GSoC 2025 are:

1. Bug Fixes and stability Improvements: Ensuring the stability and reliability of the **Code Builder module** in Music blocks v4 will be a critical aspect of this project. As part of my contributions, I will focus on identifying and resolving any inconsistencies in **brick rendering, interaction behavior, connection logic, and deletion workflows**. Addressing these issues early will improve the overall user experience and make the builder more robust and dependable for visual programming and music composition. Stability will be continuously prioritized throughout the development cycle.

2. Comprehensive Test Coverage: To maintain a high-quality and future-proof codebase, I will extend test coverage for the module by introducing targeted **unit and integration tests**. I plan to use **Vitest** and leverage **GitHub Actions** (already set up for continuous integration) to ensure stability across builds. I will contribute comprehensive test cases for rendering logic, drag-and-drop behavior, connection constraints, and AST serialization. These tests will act as safeguards against regressions, promote confidence in changes, and support future contributors in extending the module.

Impact on Sugar Labs:

The development of the **Code Builder (Masonry) module** will have a significant and lasting impact on Sugar Labs and the broader educational ecosystem. As a core component of **Music blocks v4**, this visual programming interface enables learners to build complex musical compositions using intuitive, modular “blocks.” By visually organizing and connecting musical logic through drag-and-drop components, students can develop both musical and computational thinking without the barriers of traditional text-based programming.

This module will enhance **accessibility and engagement**, especially for younger students and those in under-resourced environments. Its intuitive design lowers the entry barrier for creative coding, and its integration with a performant workspace ensures that Music blocks remain usable even on low-powered devices.

Beyond the user-facing features, this project will also introduce **developer-focused enhancements**—including reusable UI components, documented Storybook stories, and a minimal, composable API. This will help simplify the learning curve for future contributors and promote consistency across the application. The performance optimizations planned will further ensure smooth user interactions, even as complexity increases.

By aligning with Sugar Labs' mission to create open-source educational tools for learners worldwide, this project serves both students and educators by offering a creative, playful, and structured way to explore programming through music. It bridges the gap between **technical fluency** and **artistic expression**, embodying the values of open education and hands-on learning.

As someone committed to open-source development and collaborative learning, I am eager to take on the responsibility of integrating these features for the v4 application and completing it. I am confident that I can help the entire Sugar Labs community get started with this feature and contribute to their mission of promoting learning through technology.

Timeline:

I will be working on an existing/new branch and implementing all the changes there. Also, I will be documenting everything I do and adding tests for them alongside, I have not mentioned them in the timeline.

Time Frame	Tasks
Pre GSoC period April 08 - April 29	• Deep dive into the existing Music blocks v4 work and builder module. • Explore libraries like **React Aria**, **React Spectrum**, and **Storybook**. • Build and experiment with UI primitives. • Start outlining reusable components and design constraints.

Community Bonding Period May 8 - June 01	• Much time won't be wasted as I'm aware of most parts of the codebase and know the mentors well. • Discuss and finalize workflows with mentors (event handling, rendering strategy, connection logic). • Finalize design document structure. • Investigate best practices for performant rendering and drag interactions.
Coding Officially Begins ! June 02 – June 21 (2.5 weeks)	• Finalize and submit the **Design Document**. • Implement the **brick rendering engine** (configurable brick shapes, labels, slot visuals). • Establish base structure for workspace and palette integration.
June 22 - July 10 (3 weeks)	• Implement **drag and drop** from palette to workspace. • - Develop logic for **brick-to-brick connections and disconnections**. • - Add visual **feedback during interactions** (highlight, snap indicators, etc.). • - Begin serializing block trees (AST representation).
July 11 - July 17	• Bug fixing and code polishing. • Test all major interactions (drag, snap, delete, AST extraction). • Prepare for Phase 1 evaluation.
Phase 1 Evaluation July 14 - July 18	• Submit Phase 1 deliverables with documentation and recorded demos (if needed).
July 18 - August 05 (2.5 weeks)	• Begin work on the **Palette component UI**. • Implement brick deletion via workspace corner drag. • Improve connection constraints (via external validation methods). • Make the **workspace infinite/scrollable**.
August 06 - August 11 (~1 weeks)	• Perform full integration test: render → drag → connect → delete → serialize. • Improve performance using batching or layout optimizations. • Start documentation and developer notes.

August 12 - August 15 (~1 weeks)	• Finalize **user-facing and dev documentation** for rendering, connections, and AST. • Conduct final round of bug fixes and refinements.
Phase 2 Evaluation By August 25	• Submit final report and evaluation. • Add interaction and rendering tests. • Clean up codebase and remove unused logic/components.
August 26 - November 17	• Tackle legacy or low-priority issues (e.g., recording bugs, history tracking). • Help onboard contributors and document architectural decisions. • Explore additional advanced features or extensions, if requested by mentors.

Availability:

I plan to dedicate 35-45 hours per week to the project and will be most active between Friday and Sunday from 8 AM to 7 PM IST. I will have my End-semester exams between 15 April - 5 May i.e. the pre-GSoC and community bonding period and will be able to dedicate 2-3 hours a day during that period.

Progress Report:

I will be updating the mentors daily on Matrix chat and demonstrating my work through biweekly meetings. Additionally, I will write a blog after every evaluation done about my progress and share it on my medium blog and LinkedIn profiles.

Post-GSoC Plans:

After my completion of GSoC, my plan is to thoroughly review the issues and pull requests raised by other developers. In addition, I will explore new issues to address. My primary goal is to add more functionality to the Music blocks v4 project to make it more robust and feature-rich.

Along with my work on Music blocks v4, I also plan to explore and contribute to other Sugar Labs projects, including the new Sugar Labs Website launched, after the GSoC period.

Conclusion:

Thank you for reading. I have provided a detailed overview of my project and how I plan to execute it. For GSoC 2025, my main goal is to further enhance my understanding of the project by building on my practical experience and research.

As for the technology stack, I am well-versed in all the necessary technologies required for this project. I have extensive experience working with React, TypeScript, and other state management solutions. I am confident that I can complete this project within the given timeline and take full responsibility for implementing all the crucial and valuable features that will take Music blocks v4 to the next level.

I am 100% dedicated to SugarLabs and have no plans whatsoever to submit a proposal to any other organization.