



sugarlabs



Sugar Labs

Google Summer of Code 2024

Project Information:

Name: Music Blocks v4 Project Builder Integration ([here](#))

Length: 350 Hours (Large - 22 weeks)

Mentor: [Anindya Kundu](#)

Assisting Mentor: [Walter Bender](#)

Student Details:

Full Name: Karan Shailesh Palan

Email: karanpalan007@gmail.com

GitHub: [karan-palan](#)

LinkedIn: [Karan Palan](#)

Twitter: [Karan Palan](#)

Preferred Language: I'm proficient in English for communication, both spoken and written

Location: Mumbai (Maharashtra), India

Time Zone: Indian Standard Time (IST) (UTC +05:30)

Phone Number: +91 8928670471

Institution: Manipal University

Program: B.Tech in Computer Science and Engineering with Specialization in Artificial Intelligence and Machine Learning

Stage of completion: 1st Year (expected June 2027)

About Me:

I have been using open-source software for a long time, even before I knew what it was. Whether it's using [VLC media player](#) to watch videos or browsing the web with [Mozilla](#), I've always tried to contribute to open-source software in any way I can.

I didn't have any prior knowledge about Computer Science and programming before entering college. Initially, I tried my hands with Competitive programming but later realized that my interest lay somewhere else. I tried my hands-on with Web Development and Machine learning in my [First Semester](#) and did some cool projects.

My Open-source journey started when I learned about HacktoberFest 23' where I learned about Git, GitHub, and the whole workflow of Open source i.e. how to write PRs, create issues, etc. I got **7 PRs merged** at the end in an [open-source online C compiler](#). Here is a [LinkedIn post](#) about it.

I was also a contributor at [SWOC 23'](#). I am also an [MLSA](#) (Microsoft Student Learn Ambassador) and a [Postman API student expert](#).

I have participated in a few online Hackathons by MLH and an offline hackathon which I won. Here is the [LinkedIn post](#) and [Tweet](#) about the same. I have also mentored at a few events in my college. I also have been learning in public and here are some posts about that:

1. [Promises explained](#) – A made a thread about promises in JavaScript that Devs loved!
2. [Hacktoberfest Experience](#) – Made a thread about my journey of navigating through open source.
3. [30 days of Code](#) – I learned a lot in this period as I didn't have any exams in college.
4. An update on 4 months of learning to code ([Link](#)).

A lot more stuff can be found on my socials!

My general interest lies in building creative and user-centric applications that can solve real-world problems. I am well aware of:

Programming Languages: JavaScript, TypeScript, Solidity, Python, C

Libraries/Frameworks: React, Redux, Recoil, Sass, Bootstrap, TailwindCSS, Jest, Node.js, ExpressJs, Electronjs, Slickit Learn, Tensorflow

Databases: MongoDB, PostgreSQL

Tools/Platforms: Git, GitHub, Vercel, AWS, VS Code, Postman, Docker, MATLAB

Previous Open source Projects around collaboration:

I have always followed a project-based approach whilst learning any new technology. Thus, I have built a lot of projects. Here are some of them:

1. SightAi

[Website](#) - *Reactjs, TailwindCSS, [ML model](#) - Tensorflow, Coco-api etc.*

[Deployed Website](#) | [ML model](#) | [Demo](#)

- This Project helped us to qualify for *round 2 of the Microsoft Imagine Cup*. This project empowers the Blind to be more independent with the help of AI.
- Made an ML model with a teammate using Tensorflow, Yolo, Coco-API, and Pytorch which gives an accuracy of 90-95 % for static objects and 60-75% for moving objects.
- Developed the complete landing page for our product in under 5 hours using React and TailwindCSS.

2. CHAEN COLLECTIVE

HTML, CSS(Bootstrap), Javascript, Aeternity Javascript SDK

[Deployed Website](#) | [Code](#) | [Demo](#)

- We won *First Place* in the *Aeternity track* of a Hackathon with 800+ participants and had a meeting with the founder about scaling the project. [LinkedIn post](#)
- Made a DAO using the Aeternity blockchain, learned their Sophia language, and deployed the chain to Testnet.
- I was responsible for the Smart contract and the landing page, whilst I led the team to create the internal frontend using Bootstrap and make the PPT.

3. DECENDRIVE

Reactjs, Solidity, IPFS

[Github](#)

- Learned Solidity, how to write smart contracts in 6 hours, and built this project collaboratively.
- This project facilitates decentralized image upload and sharing on the blockchain using Solidity for the smart contract and React for the front-end interface.
- It enables users to securely upload images to IPFS (InterPlanetary File System) and share access with specific users through smart contract functionality.

4. NEXUS SPHERE

Reactjs, Solidity, Infura, Web3js, Truffle, Ganache-cli, TailwindCSS

[Github](#) | [Demo](#) | [Pitch](#)

- Made this for the Soonami Ventureathon, got shortlisted in 2 rounds and *ranked 11th across the world and 2nd at IIT Delhi*.
- Expertise in DAO Governance: Highlight your proficiency in designing, implementing, or participating in DAO governance structures. This involves experience with voting mechanisms, proposal creation and evaluation, tokenomics, and overall strategic decision-making within decentralized organizations.

These projects helped me understand how to lead a team, collaborate, and build great stuff with other people.

Some Personal projects:

1. The Insight Isle

Reactjs, MongoDB, Nodejs, Expressjs

[Github](#)

- This is a blogging website supporting CRUD operations.
- It has a well-styled React frontend which uses React-Quill as the editor.
- Learned the use of JWT, hashing, and cookies whilst building this project.

2. AI Paragraph Summarizer

Reactjs, TailwindCSS, Redux

[Github](#) | [Tweet](#)

- It summarizes any website or paragraph for you. Just enter the link and get the summarized pdf.
- It uses the Article-API-key from Rapid API.
- Learned the Redux toolkit while building this project.

3. Some Ad Hoc projects

- a. [Sundarone Hostel Website](#) - Website for the hostel I reside in. Still under development. *Uses Reactjs, Node+Express, Razorpay API.* [Live](#)
- b. [Google Docs clone](#) - Made this quick project to understand how Web sockets work. *Uses React, Node+Express, Web-sockets.* [Tweet](#).
- c. [PayTM clone](#) - This project is under development. Understood database transactions whilst making this project. *Uses React, Node+Express, MongoDB, and Zod for input validation.*

I have a good grip on the MERN stack, APIs, Redux, and TypeScript whilst building these projects.

Why Sugar Labs?:



My journey with Sugar Labs and the Music Blocks project is deeply rooted in my passion for music and technology. Music has always been a significant part of my life, inspiring me to explore where technology intersects with musical creativity. Discovering Music Blocks, a platform seamlessly integrating coding and music composition, immediately captivated me with its innovative approach.

As I delved into the project, I found its simplicity remarkable, enabling me to quickly immerse myself. Through experimentation and exploration within the repository, I thoroughly understood its architecture and functionality, empowering me to contribute meaningfully to Music Blocks v3. This is a Paint program I created with MusicBlocks - <https://musicblocks.sugarlabs.org/index.html?id=1711022100132843&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.

The transition to [Music Blocks v4](#), with its adoption of React and TypeScript, marked an exciting new chapter. Embracing this evolution, I eagerly immersed myself in learning and adapting to the updated stack, broadening my technical skills and reinforcing my commitment to advancing Music Blocks' capabilities.

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] Take a tour section is not responsive ([#3776](#))**
 - The take-a-tour section wasn't responsive on Mobile devices which led to a bad user experience as the user would not be able to understand the Project.
- **[Bug] The Mouse pointer gets stuck on a block which leads it to scroll and doesn't provide a good experience to the user in Widgets ([#3412](#))**
- **[Enhancement] Host the README file content as a set of web pages hosted as a section in the main website ([#3751](#))**
 - I suggested hosting the whole Music Blocks guide which was in a README file at that point, as a set of webpages in Music Blocks.
- **[Enhancement] Making a PDF guide for music blocks and improving the guide webpage ([#3765](#))**
 - As per the suggestion of a mentor, [Walter Bender](#), created an issue for Making a downloadable PDF guide for music blocks, improving the Guide webpage, and adding developer documentation for it.

Pull Requests:

- **Take a tour section made responsive for all mobile devices ([#3777](#)) (merged)**
 - Fixed issue [#3776](#)
- **Host the README file content as a set of web pages hosted as a section in the main website ([#3757](#)) (merged)**
 - Fixed issue [#3751](#)
- **Other merged pull requests ([Link](#))**
 - Added JsDoc Style comments whilst understanding the codebase. Reference issue [#2630](#).
- **Documented the components of the architecture diagram [[MB4 project](#)] ([#145](#)) (open)**
 - Documented all the components we have in the Block Diagram of the Project architecture. Fixes [#144](#).

Discussions in which I participated:

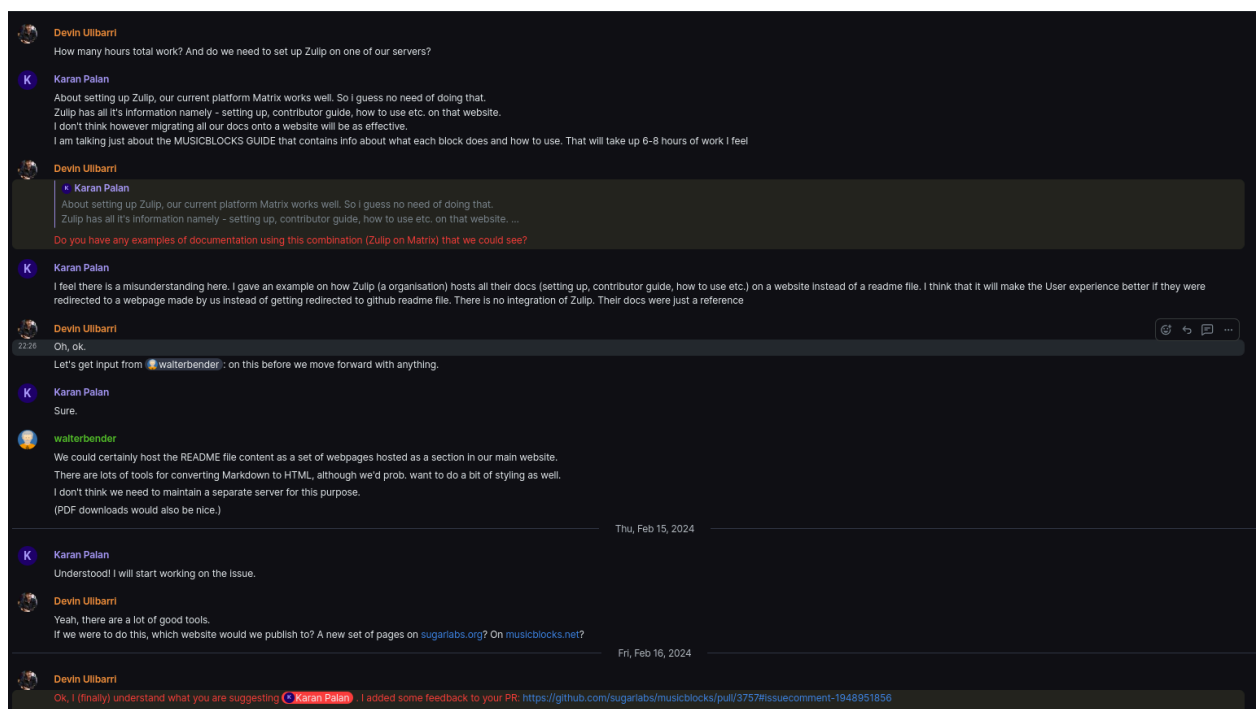
<https://github.com/sugarlabs/musicblocks-v4/discussions/390>

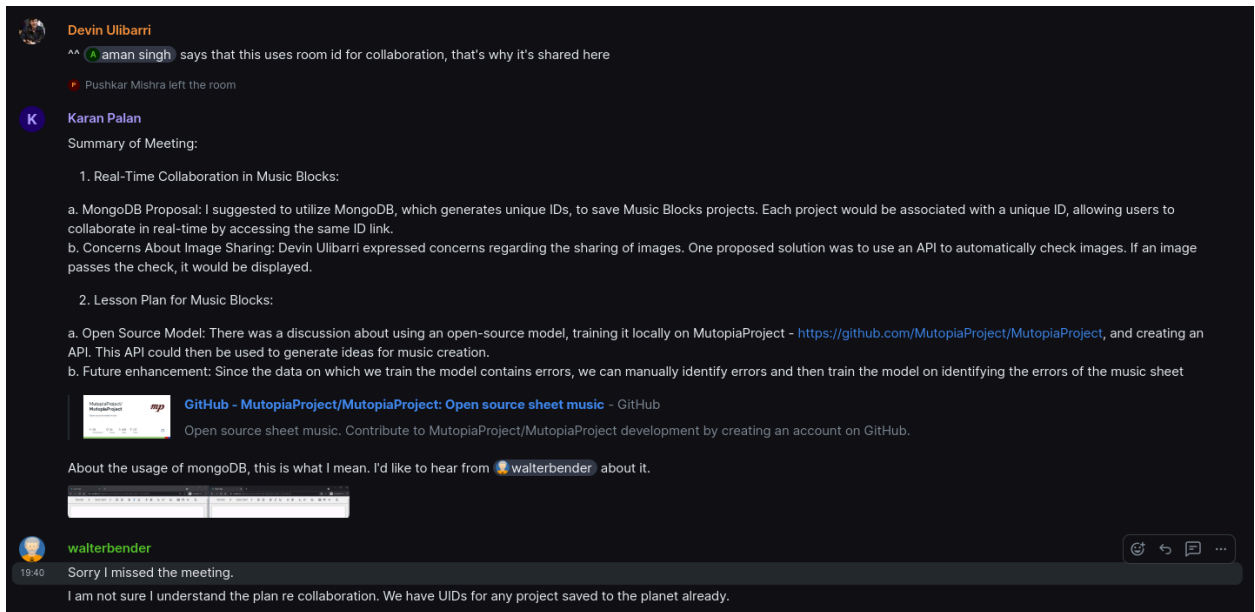
Through my consistent code contributions, I have developed a comprehensive understanding of the underlying architecture, build procedures, and intricacies within the codebase. This experience has endowed me with the confidence to swiftly identify the pertinent files requiring modification or integration of new components. Whether addressing bugs or implementing new features, I am adept at navigating the codebase and effectuating precise solutions.

Presence in the Community:

As an open-source contributor and maintainer, 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:





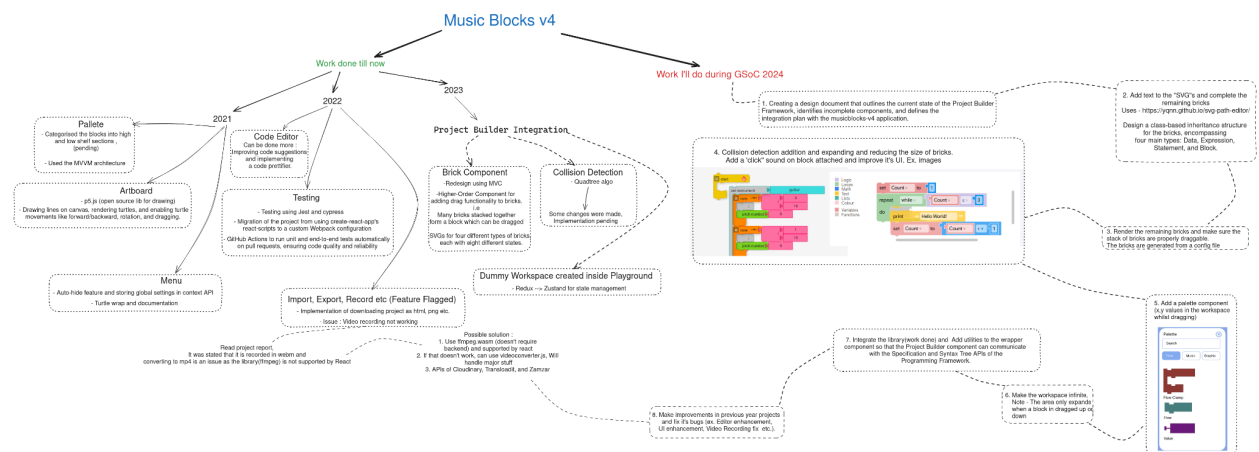
I have helped other developers set up the project and make contributions to GitHub. Apart from these, I have had in-depth conversations about the project with my mentors Anindya Kundu and Walter Bender.

Project Details:

Introduction:

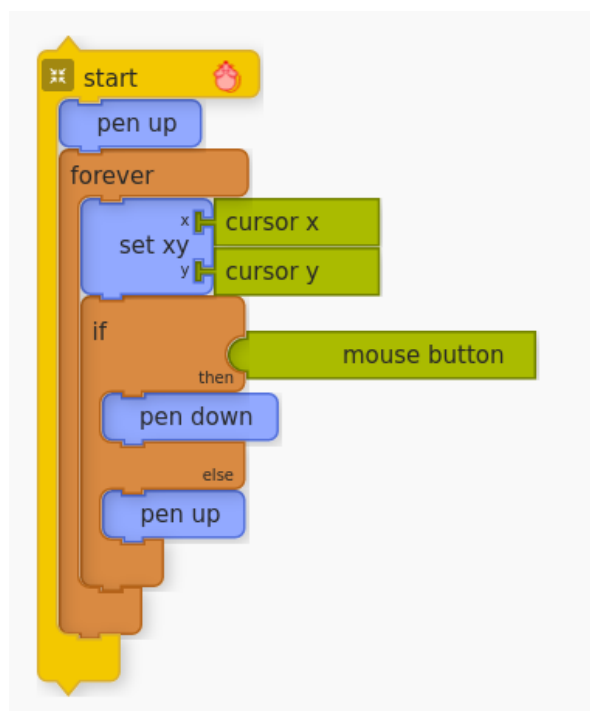
The [Music Blocks v4](#) project is a major upgrade from [Music Blocks v3](#), which was originally derived from the Turtle Blocks project. Music Blocks v3 was built entirely using HTML, CSS, and Vanilla Javascript. However, in today's world, these technologies are not very sustainable and there are better options available. That's why Music Blocks v4 was initiated, which employs modern tech stacks such as React v18 with Functional Components based approach and hooks, TypeScript, JavaScript DOM API for the DOM manipulation based on the different events, Canvas API to render the different blocks on the screen, some state management solutions (eg. Redux), different JavaScript/TypeScript libraries. As a result, the project is now more sustainable and is still a work in progress.

There have been many things that have been adapted to v4 so far, but most of the major features are still missing. I have thoroughly studied the previous additions to the project and the code practices followed and *added what can be improved*. This diagram gives a brief about the work and **work I'll do over the summer**, however the order of work might change.



Have a detailed look at the diagram here : [Link](#)

One of the main features of Music Blocks is the **“Project Builder”**, which enables children to view blocks graphically and generate music. The Project Builder is the graphical blocks manager module that can be used to create Music Blocks programs. Now there are different types of blocks and each of them represents a different type of functionality, like the Start block, Rhythm block, Note block, Pitch block, different kinds of instrument blocks (eg. Drum), and to pick them, etc. The code for them is present [here](#). These blocks are completely interactive, which means we can drag and drop different blocks, click them to take actions or open any context menu, or do some other stuff like delete any specific block. This is an example from the v3 project:

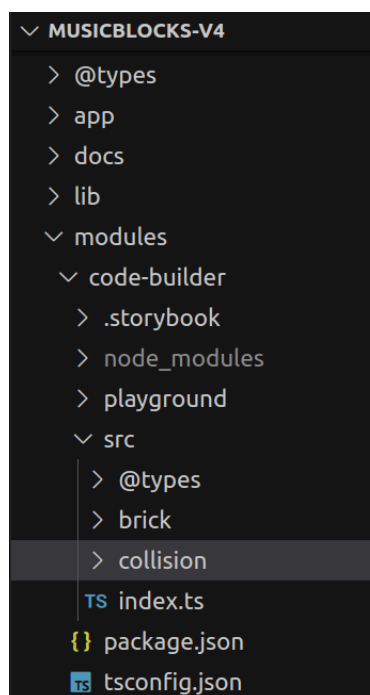


The GSoC proposal for music-blocks aims to enhance user experience and functionality. It includes adding text to bricks(SVGs), implementing dynamic editing, improving UI with collision detection, integrating the Project Builder Framework with the new Music Blocks and wrapping it in a component to facilitate easy integration with the Specification and Syntax Tree API, adding a palette feature, and fixing previous bugs. These updates promise a more comprehensive and user-friendly music composition tool.

I would like to implement this feature end-to-end so that users can start creating different musical patterns using this “Project Builder”.

Detailed breakdown of the code-builder:

The project builder consists of different types of blocks which can be grouped together in the workspace to perform a musical sequence. The code can be found here </modules/code-builder>.



The [@types folder](#) encapsulates TypeScript declaration files crucial for maintaining consistent data structures and interfaces throughout the Music Blocks project.

The **brick.d.ts** TypeScript declaration file encapsulates the foundational structure and characteristics of bricks within the Music Blocks project. It defines essential types and interfaces, facilitating consistent representation and management of various brick components, including their styles, arguments, and states. Details about it can be found in this [gist](#).

```

/**
 * @type
 * Data type (boolean, number, string, any) returned by an argument brick.
 */
export type TBrickArgDataType = 'boolean' | 'number' | 'string' | 'any';

/**
 * @type
 * Bounding box dimensions of a brick.
 */
export type TBrickExtent = {
  width: number;
  height: number;
};

```

The `collision.d.ts` TypeScript declaration file defines interfaces and types essential for managing collision detection within the Music Blocks project. It includes definitions for collision objects and an interface specifying methods for configuring collision space properties, adding and removing objects, checking collisions, and resetting the collision space.

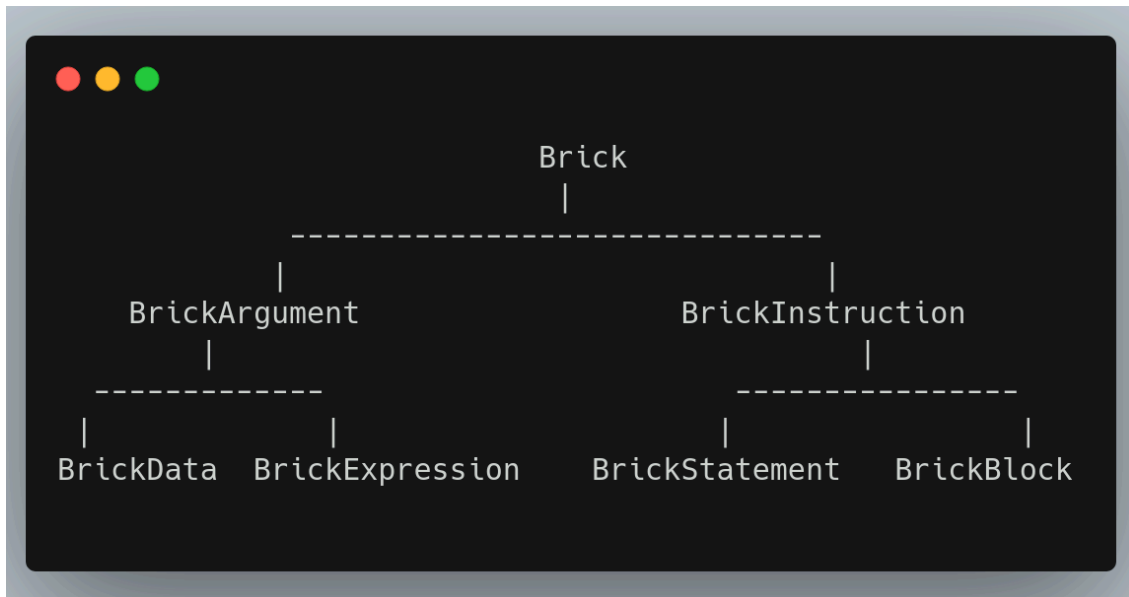
```

export type TCollisionObject = {
  /** unique ID of the object in the collision space */
  id: string;
  /** x-coordinate of the centre of the object */
  x: number;
  /** y-coordinate of the centre of the object */
  y: number;
  /** width of the object */
  width: number;
  /** height of the object */
  height: number;
};

```

The [/src/brick](#) contains TypeScript files defining the fundamental components of the Music Blocks project, establishing their structure, behavior, and inheritance

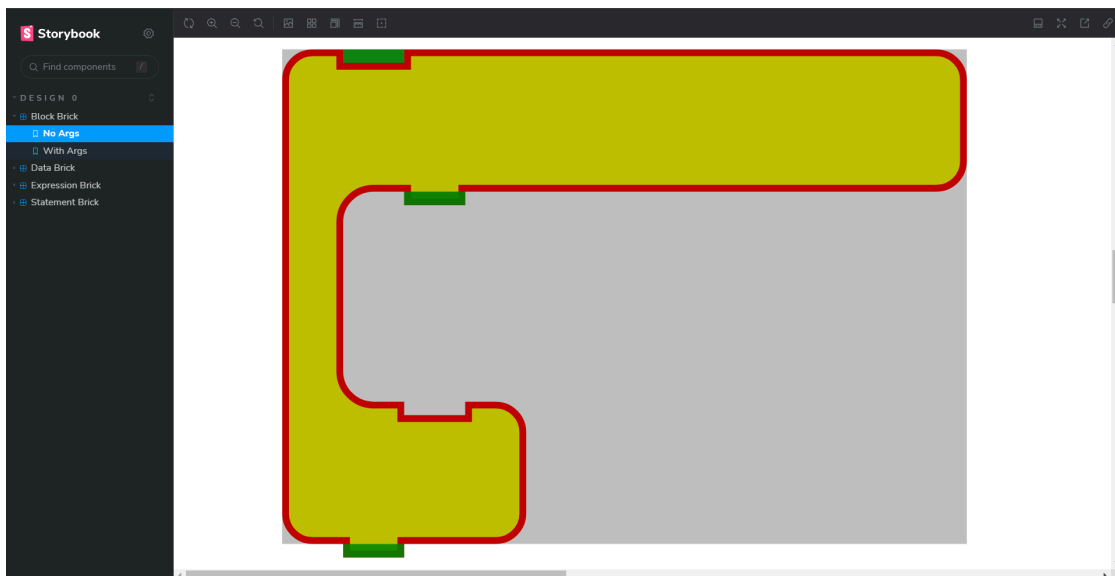
relationships.



For example, the code for [/components/BrickData.tsx](#), renders an SVG representation of a brick using provided data and properties, facilitating visual rendering and manipulation within the Music Blocks project's workspace.

The enhancement of constructor arguments, setters, getters, and the creation of final subclasses for brick types were also executed, emphasizing a separation between state management and visual rendering for improved flexibility.

The [/design0/stories](#) folder contains story files written in TypeScript using a tool, Storybook. These stories serve as interactive documentation showcasing different design variations and use cases for brick components within the Music Blocks project. Each story file demonstrates a specific aspect of brick design, allowing developers to visualize and interact with different brick configurations and behaviors. This is an example of a brick after running it locally on my machine:



The [/src/brick/design0/utils](#) folder contains two essential files:

1. **path.ts**: This TypeScript file contains utility functions that define the shape and structure of a path, used for creating SVG representations of design elements within the Music Blocks project's bricks. These functions define the various sections of the path, including arcs, lines, and notches, along with their dimensions and positions. Additionally, the file includes private helper functions for generating specific parts of the path and public functions for generating the complete SVG path string along with information about bounding boxes.

2. **path.spec.ts**: This TypeScript test file is dedicated to testing the functions defined in `path.ts`. It contains test cases to ensure that the utility functions work correctly and reliably handle various scenarios related to paths. These tests validate the functionality of the utility functions and help maintain the quality and reliability of the codebase.

Together, these files play a crucial role in the design aspect of bricks within the Music Blocks project. They provide utilities for managing paths, ensuring their functionality is tested and validated.

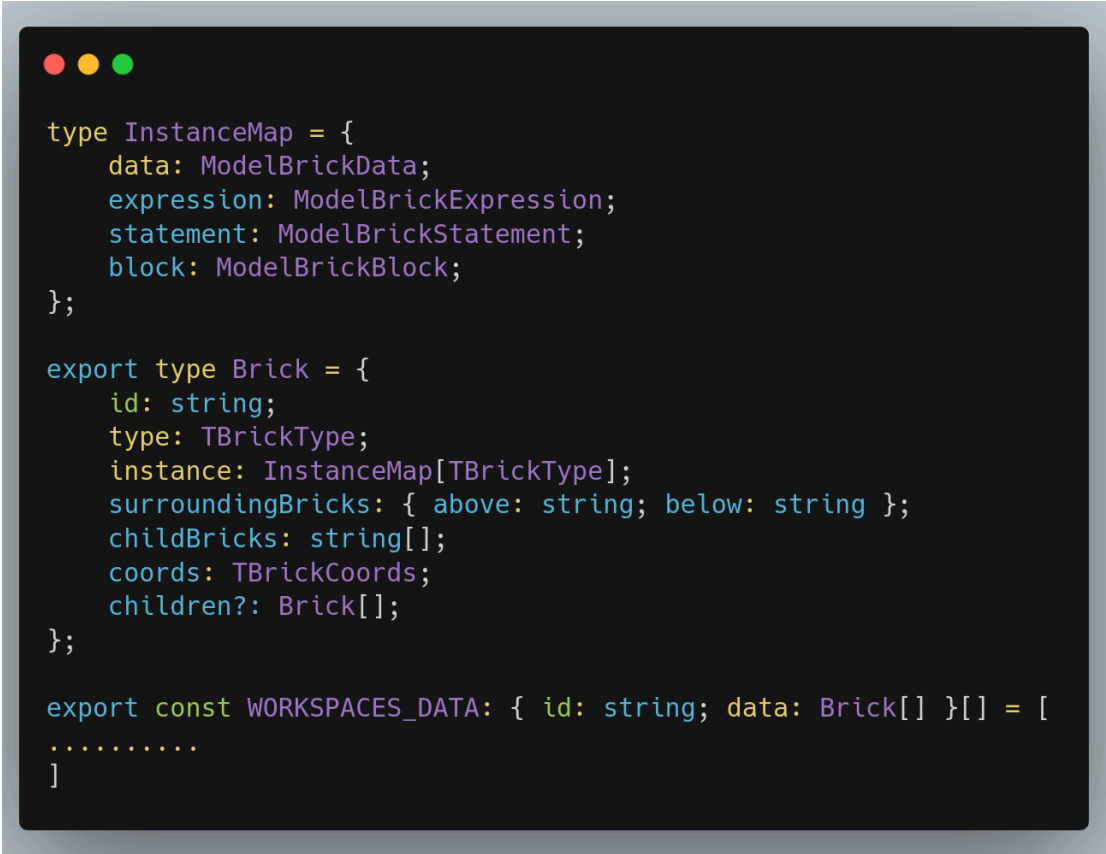
The [/brick/stories/components/BrickBlock.tsx](#) defines a React component that generates a visual representation of a brick element, including its shape and bounding boxes for debugging. It uses provided parameters to create an instance of the brick prototype and renders the brick component along with visual indicators for its bounding boxes. Here is an example code:

```
const VisualIndicators = () => (  
  <>  
    { /* Overall Bounding Box of the Brick */ }  
    <rect  
      x={instance.bBoxBrick.coords.x}  
      y={instance.bBoxBrick.coords.y}  
      height={instance.bBoxBrick.extent.height}  
      width={instance.bBoxBrick.extent.width}  
      fill="black"  
      opacity={0.25}  
    />  
  )
```

The [/src/brick/model.ts](#) file defines abstract classes that serve as the data model for different types of bricks in the project. It encapsulates properties and methods common to various brick types, facilitating their implementation and ensuring consistency in data representation. This model forms the backbone for managing and manipulating bricks within the application's architecture, enhancing code organization and maintainability.

The [/code-builder/playground](#) directory contains files related to a coding playground within the project, providing users with an interactive environment to experiment with code. It includes components, utilities, and configurations necessary for setting up and running the code playground, offering users a sandboxed space to test and iterate on their code in a controlled environment.

The [/playground/pages/Workspace/data.ts](#) contains an initial tree state to maintain the state. It also contains data about each brick and some other metadata. Also, each brick can contain children which will be an Array of different bricks.

A screenshot of a code editor with a dark background and light-colored text. The code defines TypeScript types for bricks. It starts with a type definition for 'InstanceMap' which has four properties: 'data' of type 'ModelBrickData', 'expression' of type 'ModelBrickExpression', 'statement' of type 'ModelBrickStatement', and 'block' of type 'ModelBrickBlock'. Then it defines an exported type 'Brick' with properties: 'id' (string), 'type' (TBrickType), 'instance' (InstanceMap[TBrickType]), 'surroundingBricks' (an object with 'above' and 'below' strings), 'childBricks' (string[]), 'coords' (TBrickCoords), and 'children?' (Brick[]). Finally, it defines a constant 'WORKSPACES_DATA' as an array of objects, each with 'id' (string) and 'data' (Brick[]).

```
type InstanceMap = {
  data: ModelBrickData;
  expression: ModelBrickExpression;
  statement: ModelBrickStatement;
  block: ModelBrickBlock;
};

export type Brick = {
  id: string;
  type: TBrickType;
  instance: InstanceMap[TBrickType];
  surroundingBricks: { above: string; below: string };
  childBricks: string[];
  coords: TBrickCoords;
  children?: Brick[];
};

export const WORKSPACES_DATA: { id: string; data: Brick[] }[] = [
  .....
]
```

In [/Workspace/utils.ts](#) the `getBelowBricksIds` utility function that uses the `recursiveSearch` function recursively to find all the IDs of the bricks whose positions needed to be updated when we drag any specific brick, and it returns the Array of IDs of those bricks.

```

export function getBelowBricksIds(arr: Brick[], item: string): string[] {
  let result: string[] = [];

  function recursiveSearch(arr: Brick[], item: string) {
    arr.forEach((element, index) => {
      if (element.id === item) {
        arr.slice(index + 1, arr.length).map((el) => {
          result = result.concat(el.childBricks);
          result = result.concat(el.id);
        });
        return;
      }
      if (Array.isArray(element.children)) {
        recursiveSearch(element.children, item);
      }
    });
  }

  recursiveSearch(arr, item);
  return result;
}

```

For moving the Bricks and updating the position and state properly, [Zustand](#) was used. The coordinates of each brick were stored respectively to their IDs. The method `setCoords` is used to update the coords value for each brick and `getCoords` to get the coords value of each brick. The code can be found at [/Workspace/BricksCoordsStore.ts](#).

```

setCoords: (brickId: string, coords: { x: number; y: number }) =>
  set(
    (state: {
      allCoords: {
        brickId: string;
        coords: {
          x: number;
          y: number;
        };
      };
    }) => ({
      allCoords: state.allCoords.map((item) =>
        item.brickId === brickId ? { brickId, coords } : item,
      ),
    }),
  ),
  getCoords: (brickId: string) =>
    get().allCoords.find((item) => item.brickId === brickId)?.coords,

```

The [/Workspace/BrickFactory.tsx](#) component serves as a dynamic renderer for different types of bricks based on the provided data. It facilitates interactive manipulation through drag-and-drop functionality, which is achieved using the `useMove` hook from [`react-aria`](#).

The component efficiently manages brick coordinates using the `useBricksCoords` hook, allowing seamless updates during drag events. This ensures smooth repositioning of bricks within the workspace.

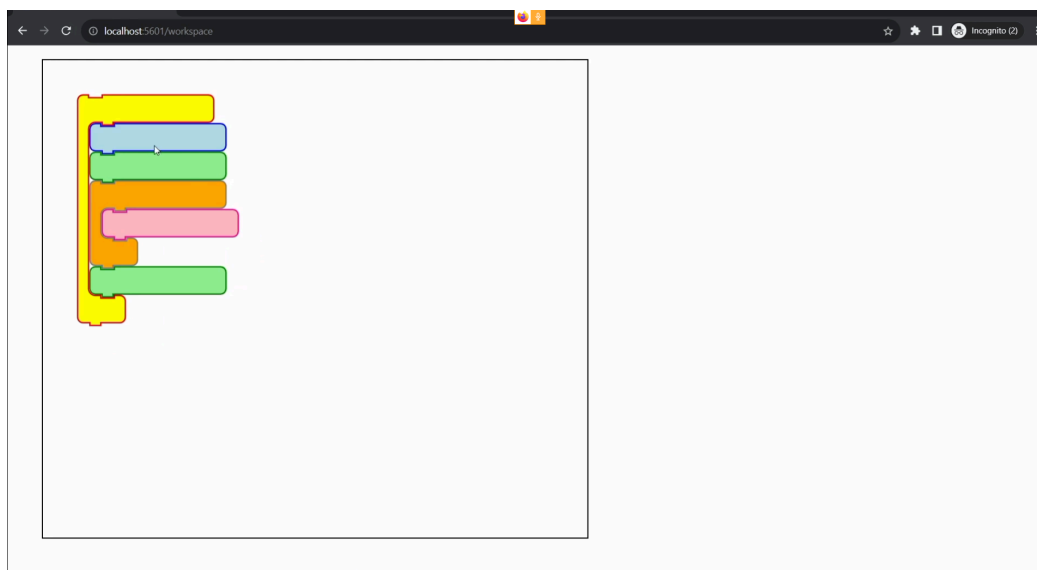
Additionally, visual indicators such as bounding boxes are rendered to provide users with a clear understanding of brick boundaries and positioning. These indicators dynamically adjust based on the size and structure of each brick type, enhancing the user experience and usability of the workspace.

```
const { moveProps } = useMove({
  onMoveStart(e) {
    console.log(`move start with pointerType = ${e.pointerType}`);
    setColor('white');
  },
  onMove(e) {
    const newX = brickCoords.x + e.deltaX;
    const newY = brickCoords.y + e.deltaY;
    setCoords(brickData.id, { x: clampX(newX), y: clampY(newY) });

    brickData.childBricks.forEach((childBrick) => {
      const childBrickCoords = getCoords(childBrick!);
      setCoords(childBrick, {
        x: childBrickCoords.x + e.deltaX,
        y: childBrickCoords.y + e.deltaY,
      });
    });
  });

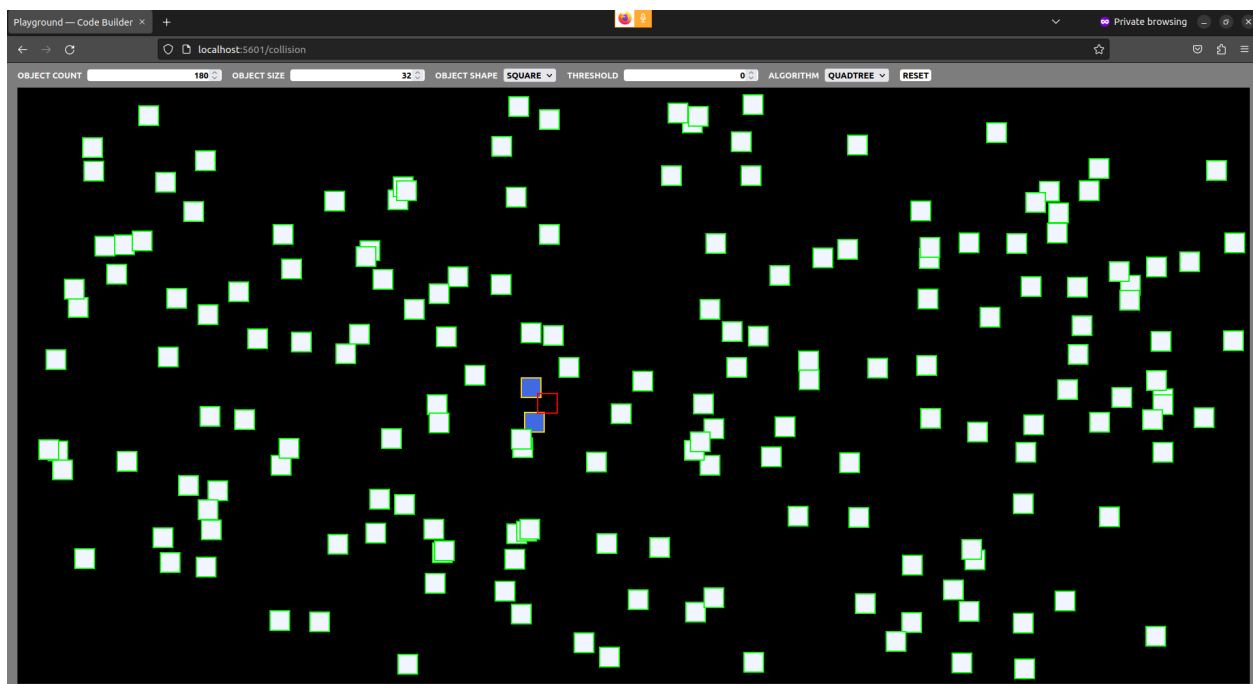
  const belowBrickIds = getBelowBricksIds(WORKSPACES_DATA[0].data, brickData.id);
  belowBrickIds.forEach((belowBrickId) => {
    const belowBrickCoords = getCoords(belowBrickId!);
    setCoords(belowBrickId, {
      x: belowBrickCoords.x + e.deltaX,
      y: belowBrickCoords.y + e.deltaY,
    });
  });
});
```

This is how it looks when I run it locally:



The [index.ts](#) file in [pages/collision](#) renders a playground for collision detection. It manages states for object count, size, type, and collision parameters. Using React hooks, it initializes objects and collision space models. Mouse movement triggers collision detection, visually highlighting collisions. User controls allow adjusting parameters and resetting the playground. It employs two collision detection algorithms: **brute force** and **quadtree**. Users can toggle between these algorithms to observe their performance differences. Brute force algorithm checks collisions between all pairs of objects, while the quadtree algorithm divides space into hierarchical regions for efficient collision checks. This allows for optimized collision detection, particularly when dealing with a large number of objects.

When run locally, It looks like this:



The [/code-builder/src/collision](#) contains code for the algorithms.

Detection of the collision between the different bricks after the drag end is the toughest and trickiest part. First, the bricks will go through some functions for us to find out if the brick we have dragged is colliding. If those functions return true, we then have to do calculations based on that and update the parent block and the global store based on the latest data. We need to inject the brick to the exact position, and to the exact slot where it can be fitted only.

The **Quadtree** is generally *better for collision detection in scenarios with a large number of objects* due to its hierarchical spatial partitioning. It reduces the number of collision checks [*time complexity of $O(\log N)$*] by organizing objects into nested regions, optimizing performance by focusing checks only on potentially colliding objects. Conversely, **brute force** *compares every object with every other object*, resulting in a quadratic time complexity and making it less efficient [*time complexity of $O(N^2)$*] especially as the number of objects increases. Therefore, quadtree offers superior efficiency and scalability in collision detection for complex scenarios.

Deliverables:

As I have already added the detailed breakdown of the code-builder module, here are the major objectives on which I want to work as a part of Google Summer of Code 2024.

1. Creation of a Design Document for the WIP: 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. The project's entire progress can be found [here](#). Outline a development roadmap with milestones, timelines, and task breakdowns to guide the implementation process and ensure progress tracking.

2. Implementing, Improving the collision detection and resizing the Bricks(SVGs) based on distance between bricks: Detect collision between bricks using those detected collision points and update (connect or disconnect bricks) the state of the workspace accordingly and resize the SVGs.

I believe the Quadtree algorithm should be employed, especially as the number of objects increases. Otherwise, the screen may experience lag, similar to what occurs in Music Blocks v-3, resulting in slow simulation or frame drops. You can see the reasons why here in this [gist](#).

We can also take reference from [DropZones/DropZones.ts](#) of the prototype for integrating the collision with the bricks workspace is responsible for managing horizontal and vertical drop zones using Quadtree data structures for collision detection. It provides methods for retrieving colliding drop zones and removing drop zones by ID. Additionally, it implements the singleton pattern to ensure a single instance throughout the application.

```

private constructor() {
  const vw = Math.max(document.documentElement.clientWidth || 0, window.innerWidth || 0);
  // will have to look into height as the canvas is scrollable
  const vh = Math.max(document.documentElement.clientHeight || 0, window.innerHeight || 0);
  this.quadtreeHorizontal = new Quadtree({ width: vw, height: vh });
  this.quadtreeVertical = new Quadtree({ width: vw, height: vh });
}

public get flow(): Quadtree<IDropZoneFlow> {
  return this.quadtreeHorizontal;
}

public get arg(): Quadtree<IDropZoneArg> {
  return this.quadtreeVertical;
}

public getCollidingFlowZones(
  x: number,
  y: number,
  width: number,
  height: number,
): IDropZoneFlow[] {
  return this.quadtreeHorizontal.colliding({ x, y, width, height });
}

```

[/DropZones/DropZonesController.ts](#) serves as a higher-level abstraction for managing various types of drop zones for blocks in a UI. It offers methods for adding drop zones below blocks, for child blocks, and for block arguments, calculating their positions and dimensions based on block configurations and UI settings.

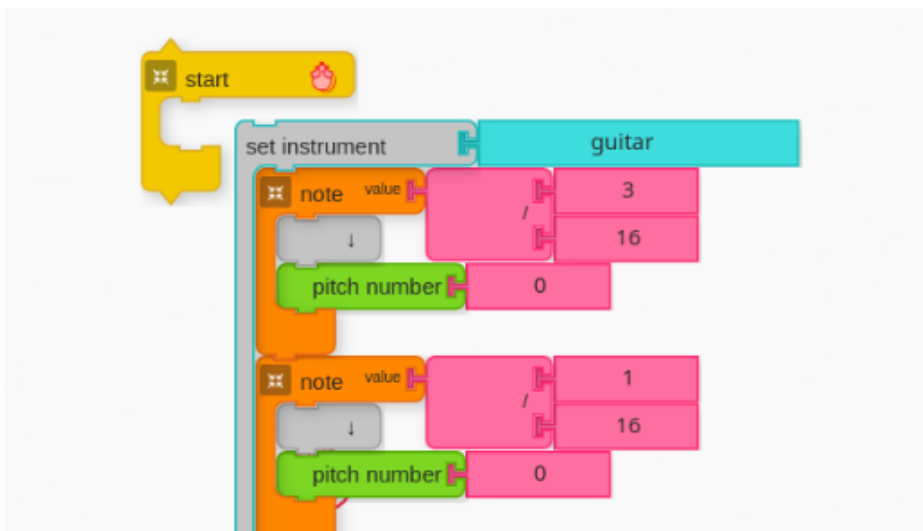
```

// adds drop zone for child - Applicable for Clamp Blocks
ChildDropZone(
  groupRef: React.RefObject<HTMLDivElement>,
  block: Block,
  dropZones: IDropZones,
  UIConfig: {
    BLOCK_SIZE: number;
    STEM_WIDTH: number;
  },
) {
  const area = groupRef!.current!.getBoundingClientRect();
  const dropZone: IDropZoneFlow = {
    x: area.left + UIConfig.STEM_WIDTH * UIConfig.BLOCK_SIZE,
    y: area.top + (1 - 0.15) * UIConfig.BLOCK_SIZE,
    id: `${block.id}-child`,
    width: (block.defaultBlockWidthLines / 2) * UIConfig.BLOCK_SIZE,
    height: 0.3 * UIConfig.BLOCK_SIZE,
  };
  dropZones.flow.push(dropZone);
}

```

Implementation:

1. Develop the logic to render a preview of the brick's collision while it's being dragged. This might involve dynamically creating a transparent version of the brick being dragged and positioning it relative to the cursor's position or the slot it's potentially colliding with.
2. Integrate the collision detection code that uses the Quadtree algorithm into the system. This involves ensuring that the Quadtree is properly constructed and updated to represent the positions and sizes of the blocks and slots on the canvas.
3. Modify the preview rendering logic to utilize the collision detection results obtained from the Quadtree algorithm. Continuously update the preview as the brick is being dragged, providing users with real-time feedback on potential collisions and fitting into slots.
4. The basic implementation of collision of brick factory using the Quadtree is ,
Gist-<https://gist.github.com/Karan-Palan/bbbc2146069451d55f8f31428019246f>
5. The SVGs should dynamically resize based on distance between bricks.
Gist-<https://gist.github.com/Karan-Palan/df1393cd793aa2e3af82cbda3ff6e790>
6. A reference from older version of Music Blocks:



Some possible edge-cases about collision detection - [Gist](#).

3.Render the remaining bricks(if any) and make sure the stack of bricks are properly draggable:

In our project, the management of groups of bricks is orchestrated through a combination of React hooks and state management libraries like Zustand. The global state, which includes information about brick positions, sizes, and other attributes, is centralized within a dedicated store file, BricksCoordsStore. This centralized store ensures that state changes are synchronized across all components, facilitating coordinated movement and updates of SVG elements within the group.

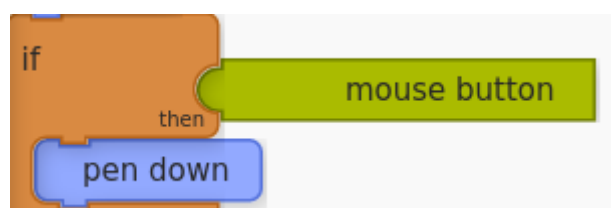
To achieve synchronized movement, we utilize libraries like `react-aria` and `useMove` from React-aria, which provide hooks for handling user interactions such as dragging and resizing with accessibility support. These libraries enable us to implement smooth and responsive interactions for manipulating groups of bricks.

Here is a detailed explanation of the Before and After a Brick Connection - [Gist](#).

This functionality works smoothly as of now in the code-builder module. As I described above, it uses React-aria open source library for drag and drop functionality. As I make the workspace infinite, and as the number of bricks increases on the screen, it will have some bugs which I will resolve during that time.

4.Add text to the SVGs and improve the Brick Ui on collision: Incorporate text into SVGs in React components by adding a `<text>` element within the SVG and setting attributes for positioning and styling.

The aim is to add text to the SVGs to make it look like this :



Implementation:

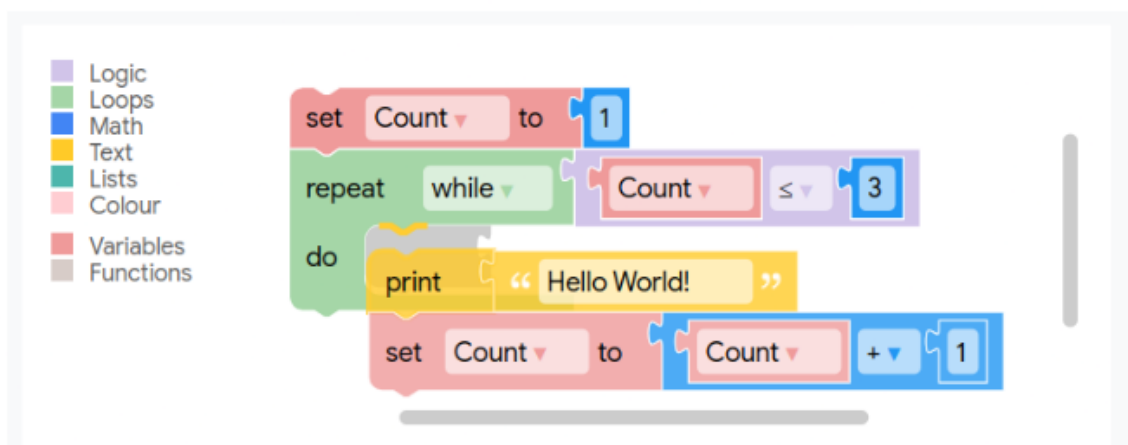
1. Inside the SVG element, add a `<text>` element. This element will represent the text to display. Set its attributes such as `x` and `y` to position the text within the SVG canvas. Additionally, use attributes like `fontSize` and `fill` to style the text according to the design requirements.
2. Within the `<text>` element, add the actual text content to display. This content will be visible to users when they interact with the React component.

3. We can also refer to this [youtube video](#) and these [stack-overflow solutions](#).

```
<svg
  xmlns="http://www.w3.org/2000/svg"
  x="0px"
  y="0px"
  width="64"
  height="64"
  viewBox="0 0 172 172"
  style="fill: #000000"
>
  <!-- Existing SVG elements -->

  <!-- Adding text -->
  <text x="20" y="150" font-size="16" fill="red">Sample Text Musicblocks</text>
</svg>
```

Currently, our prototype does not provide a real-time preview of block collisions. As a result, users are unable to determine which brick is colliding with the current brick and fitting into the slot. The current system requires dragging and dropping the block to fit it into the other brick's slot, without any preview. To improve the user experience, we can show a preview of the fitting while the block is being dragged, without actually dropping it. After the drop, the brick will be fitted to the preview slot as usual. To implement this feature, we can refer to [Blockly Game](#) and learn from their approach.



5. Creation of a Wrapper Component: Develop a wrapper component that acts as a bridge between the Project Builder framework and the music blocks-v4 platform. The wrapper component should handle communication between the Project Builder and other components of music blocks-v4.

Implementation:

1. The Wrapper Component will basically return two Components which will be a palette and Workspace Component. Each of these components will be a Children that will be rendered along with the existing components of MusicBlock.
2. Test the wrapper component to ensure that it functions correctly and that it can communicate with the Project Builder framework and other components of music blocks-v4
3. This Wrapper component will render two children components which will be a palette (will be used to drag and drop the required block from the palette to workspace) and the workspace components (will contain the block stack to be executed).

Gist - <https://gist.github.com/Karan-Palan/222c3cba7883d0ce00013150d8796b96>

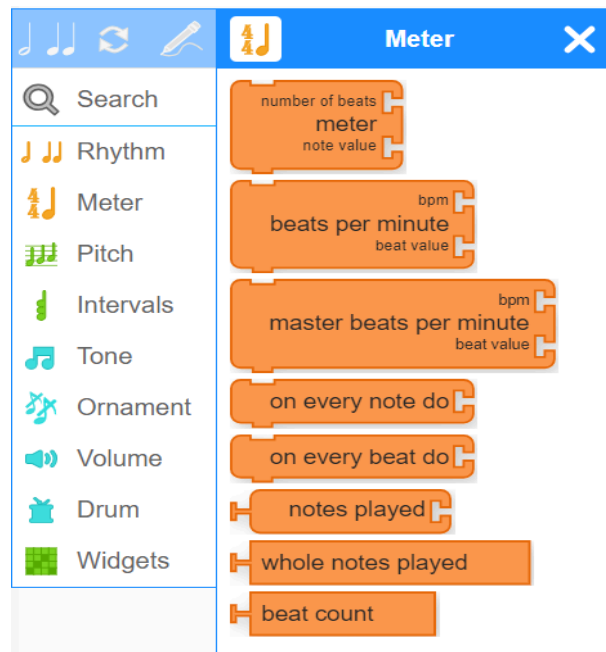
6. Palette Feature and Infinite Scroll: Introducing a palette feature and implementing infinite scroll functionality to enhance usability, allowing users to access a wider range of music blocks and scroll through them effortlessly.

Gist - <https://gist.github.com/Karan-Palan/9522bce7a429d0ef4b0bcdcbf8d0bf1b>

Implementation:

We can load some hard coded blocks 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.

1. Develop a user interface that displays the available Music Blocks, categorized into sections such as Music, Flow, and Graphic. This component serves as a visual representation of the bricks that users can add to their programs. I will be taking inspiration from the original palette:



2. Introduce a drag and drop feature to the Palette component, allowing users to select and add Music Blocks to their programs effortlessly. Users can drag blocks from the Palette and drop them into the program workspace. Additionally, organize blocks into tabs within the Palette to facilitate easy navigation across different categories of components.
3. Implement the palette component inside a Draggable Overlay which will enable us to drag a component from a children component to another (as both the palette and workspace are children components to be present inside of the parent component). And we will be needed to use a useDraggable component inside of the workspace component to enable us to drop blocks from the palette.
4. Thoroughly test the Palette component to verify its correct functioning and user-friendliness. Test scenarios should include verifying the drag and drop functionality, ensuring that users can easily select and add Music Blocks to their programs from the Palette.

7. Adding utilities to the Builder framework to communicate with Specification and Syntax Tree APIs of the Programming Framework:

In addition to refining the brick rendering process, we'll also develop a Wrapper Framework and integrate utilities into the Builder framework. These utilities will facilitate seamless communication with the Specification and Syntax Tree APIs of the Programming Framework, enhancing the overall functionality and interoperability of the system.

Gist - <https://gist.github.com/Karan-Palan/89d18ffb5760a8ca02022c3c2a32e821>

Implementation

1. This should be a two-way communication, where we can generate the array containing multiple blocks with their details to render to the UI, from the Programming Framework's APIs. As well as we should be also able to generate the exact JS syntax structure, from the array of Blocks, that we need for communication with the APIs and methods
2. Develop the necessary utilities within the wrapper component by leveraging the existing codebase of the Programming Framework. This involves adapting existing components, creating new ones, or integrating external libraries and APIs to facilitate seamless communication between the Project Builder and the Syntax Tree APIs.
3. Enable the execution of the MusicBlocks program present in the workspace (Brick Stack) by accessing bricks stored in the global Zustand store and executing them through the APIs provided in MusicBlocks-v4-lib. This execution process should be initiated by a handler triggered upon the user clicking the "Run" button. Utilize the Syntax Tree API to create a syntax tree based on the blocks present in the workspace, ensuring that any changes in block state trigger the recreation of the syntax tree.
4. Implement a handler to rearrange blocks if the syntax tree changes, possibly due to modifications made in a different editor. Monitor the current syntax tree provided by the Syntax Tree API and trigger the block rearrangement process whenever a change occurs. This will involve updating the global state of blocks, resulting in a corresponding rearrangement of the UI elements.
5. References - </specification/index.spec.ts> , </syntax/tree/index.spec.ts>

8. Integration of Builder Project: Integrating the entire code-builder into the Music Blocks v-4, consolidating the development efforts and ensuring seamless functionality across all components.

Implementation:

1. Begin the integration process by incorporating the Project Builder framework into the music blocks-v4 platform. Ensure that the integration is seamless and that all necessary dependencies are properly included and configured within the music blocks-v4 environment.
2. Verify that the integrated framework functions correctly within the music blocks-v4 platform. Test all aspects of functionality to confirm that the Project Builder features operate as expected and that there are no conflicts or compatibility issues with existing components of music blocks-v4.
3. Conduct thorough testing of the integration to validate its reliability and effectiveness. This includes both unit testing and system testing to identify and

address any potential bugs, errors, or performance issues that may arise during the integration process.

Other objectives on which I'll work as a part of GSoC 2024 are:

1. Bug Fixes: Addressing any lingering bugs or issues from previous years to ensure the stability and reliability of the music blocks application.

Through these proposed features and enhancements, the project aims to elevate the music blocks application, making it a more comprehensive and user-friendly tool for music composition, while also addressing any existing issues to ensure a smooth and reliable user experience.

2. Add Tests:

I'll add the tests for the components and the methods simultaneously. We have Jest and Cypress already set up on the v4 project for testing purposes. Jest is used for Unit Testing and Cypress is used for End to End testing. Also, the CI has been already configured with GitHub Actions to run the tests on the pipeline. I'll try to improve the test coverage by adding the necessary tests where needed

Impact on Sugar Labs:

Sugar Labs aims to promote learning, and Music Blocks is one of their most significant projects. We are transitioning from the old VanillaJs application to the modern v4 one and need to bring all the essential features of the old project into the new one. The "Project Builder" is one of the most important features of the application as it provides a block-based visual approach to generate music, making it easier for children and beginners to learn.

Completing the "Project Builder" feature for the v4 application is a significant project for this year's Google Summer of Code. It will have a positive impact on Music Blocks' users and the entire Sugar Labs community by making it easier for them to use the application.

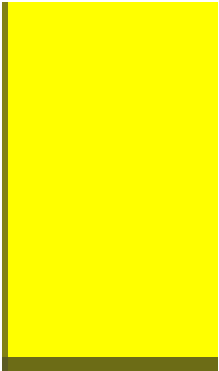
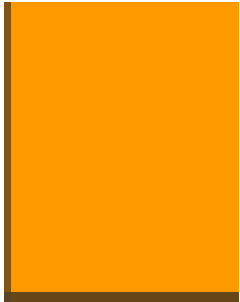
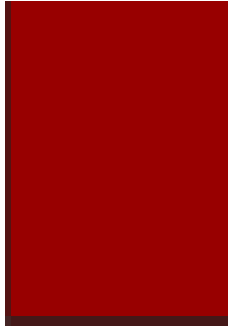

As an experienced developer and open-source contributor, I am eager to take on the responsibility of integrating this feature for the v4 application and completing it. I am confident that I can help the entire Sugar Labs community get started with this feature and contribute to their mission of promoting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

Timeline:

Since the project is a Monorepo, I will be working on an existing/new branch and making 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.

Legend: Importance and Time Devoted:  >  >  > 

Time Frame	Tasks	
Pre GSoC period March 18 - April 30	<ul style="list-style-type: none">Take a deeper dive into the work done till now and the code-builder to get an even firm understanding of the project.Learn even more about the newer libraries implemented like Zustand, React-aria etc and contribute more.	
Community Bonding Period May 1 - May 26	<ul style="list-style-type: none">Much time won't be wasted as I'm aware of most parts of the codebase and know the mentors well.Decide the best approach to handle different events and other methods.Create a milestone to keep a proper track.	
Coding Officially Begins ! May 27 - June 17 (3 weeks)	<ul style="list-style-type: none">Creating the Design Document which outlines all aspects of the projectStart to work on implementing the collision detection between the bricksDiscussing the process with the mentors and improving the collision detection functionality	
June 18 - July 8 (3 weeks)	<ul style="list-style-type: none">Making sure that the Stack of bricks are rendered correctly and work smoothly. If not, solve all bugs regarding that aspectAdding text to the SVGs and working on the resizing of SVGs based on the	

	distance between bricks.	
Phase 1 Evaluation July 8 - July 12	<ul style="list-style-type: none"> Completing pending work (if any). Fixing bugs. 	
July 13 - July 20	<ul style="list-style-type: none"> Completing Everything about the bricks i.e collision detection, adding text and change in size etc. Starting to work on the palette component 	
July 21 - August 11 (3 weeks)	<ul style="list-style-type: none"> Completing the work on palette component Making the workspace infinite* Starting the integration of the whole thing 	
August 12 - August 19	<ul style="list-style-type: none"> Documentation for the Project Builder, Edit block and Palette . Completion of integration. 	
Phase 2 Evaluation Till August 26	<ul style="list-style-type: none"> Completing whatever tasks left. Adding tests for the remaining parts. 	
August 27 - November 11	<ul style="list-style-type: none"> Asking the mentors for more work Solving issues from past projects/work ex. Fixing recording etc. 	

Availability:

I plan to dedicate 35-50 hours per week to the project and will be most active between Thursday and Sunday from 8 AM to 7 PM IST. I will have my End-semester exams between 1 May - 18 May i.e. the 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 every 2-3 weeks on <https://musicblocks.net/> about my progress and share it on my social media 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.

The Music Blocks v3 projects are saved on the Music Blocks Planet. Once I complete this project, *I will figure out how to save newer Music Blocks v4 projects into the planet.* I am committed to maintaining Music Blocks v4 in the long term and becoming a lifelong member of the Sugar Labs community to help others in the field of open-source.

Along with my work on Music Blocks v4, I also plan to explore and contribute to other Sugar Labs projects, including the programming framework of Music Blocks v4, Sugarizer, Sugar, and Music Blocks v3, 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 2024, 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, Redux, 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.