Google Summer of Code 2024

Add real-time collaboration to Music Blocks

Ajeet Pratap Singh

Section 1: About You

My name is Ajeet Pratap Singh. I am a second-year undergraduate student at the Chhatrapati Shahu Ji Maharaj University, Kanpur, pursuing a Bachelor's in Computer Science as my major.

What project are you applying for?

Add real-time collaboration to Music Blocks.

Why are you interested in working with Sugar Labs? And how this project will impact Sugar Labs?

I started contributing to Sugar Labs in November 2023. I started contributing to Sugar Labs because I wanted to explore the open-source community and contribute to projects. Contributing to Sugar Labs helped me understand its codebase and whenever I got stuck mentors were readily available for help.

Other than contributions, Playing with Music Blocks is fun. Before this, I had not experienced such an environment where we could learn concepts of music and programming and play altogether. I have also made a project in Music Blocks named Elephant Ring Song.

But while playing with Music Blocks, One thing I missed was my friends. It would be a lot of fun if I'd be able to connect with my friends and play with them.

So, Following this, I want to use my familiarity with the codebase and love to play with Music Blocks to add **Real-time collaboration to Music Blocks**. This project will impact Music Blocks in the following ways:-

1. **Enhanced Learning Experience**: Real-time collaboration allows learners to work together on Music Blocks projects, fostering collaboration and teamwork. This can lead to a richer learning experience, as students can share ideas, learn from each other, and create music collaboratively.
2. **Increased Engagement**: Collaborative features can make Music Blocks more engaging for students. The ability to work together in real time can make learning more interactive and fun, encouraging students to spend more time exploring and creating music.
3. **Broader Reach**: By enabling real-time collaboration, Music Blocks can potentially reach a larger audience. Students and teachers from around the world can collaborate on projects, regardless of their location, leading to a more diverse and inclusive learning environment.
4. **Feedback and Improvement**: Real-time collaboration can also facilitate feedback and improvement. Teachers can easily monitor students' progress, provide feedback, and offer guidance, helping students improve their musical skills and understanding of programming concepts.

Overall, adding real-time collaboration to Music Blocks aligns with Sugar Labs' mission of providing innovative educational tools and fostering a collaborative learning community. It can help enhance the effectiveness of the Sugar Learning Platform and contribute to a more engaging and interactive learning experience for students worldwide.

Prior Experience

I've been doing web development for the past two years, and I'm a web developer who predominantly uses JavaScript, TypeScript, React, Tailwind, and its other libraries and frameworks. For the past 4 to 5 months, I've been contributing to Sugar Labs as a contributor in Music Blocks. Some of my contributions are mentioned below.

Links to Pull Requests

- #3461 Implemented responsive Tooltips in JavaScript Editor.
- #3580 Update the HTML file with the README file
- #3608 Optimized code Using a Function
- #3507 Fixed Creation of Duplicate Phrases in Phrase Maker.
- #3627 Fixed the inappropriate position of the cursor in the Input box
- #3728 Fixed the responsiveness of the Top-right corner Buttons.
- #3731 Implemented Ctrl+z shortcut for Undo functionality
- #3468 Fixed the appearance of the Trashcan by repositioning it.
- #3440 Fixed Typos in the README file

A complete list of my PRs can be found here.

Also, I have opened 18 issues and fixed them which can be found here.

Academic Experience and Projects

As part of my Academic learning, I have made the following projects Projects.

- **IndiaPay(An online payment app)** - This app is inspired by paytm.com that lets users create accounts and send money online securely.
  The front end is developed using React and Tailwind CSS, while the back end is built with JavaScript and Node.js. Credentials security is implemented using bcryptjs and JSON web token, with MongoDB and the Mongoose library utilized for the database. Express.js serves as the server framework, Zod for input validation, and Axios for request handling.

- **GoFoods** - The website enables users to browse food items online, add them to a cart, and place orders. To develop this platform, JavaScript and Node.js are utilized for the back end, React for the front end, MongoDB for the database, and Express.js for the server.

Project Size

I am applying for a large project (~350 hours).

Project Timeframe

01 May 2024 to 03 September 2024

Contact info and timezone(s)

Primary Email: ajeetpratap517@gmail.com

Secondary Email: ajeetpratapsingh351@gmail.com

Contact Number: +91 6386097859

Github: @apsinghdev

Matrix Id: @ajeit2023:matrix.org

Language: Hindi (Native), English (Fluent)

Location: Kanpur, India

Time Zone: IST (GMT+5:30)

Preferred mode of Communication: Email, Google Meet, Jitsi, Matrix

Time Commitment

I am having summer vacation from 5th May to 26th June. In this time frame, I would be able to devote **~40-45 hrs/week**. After that, I would be able to devote **~20-30 hrs/week.** which may increase if the need arises. I am working on the GSoC project from 27th May to 26th August timeframe (**Note:** can be extended if the need arises).

S. No	Dates	Days (Total)	Time Commitment
1.	27th May - 30th June	Mon-Sun (7)	6 hr/day (Mon-Sun)
2.	30th June - 26th Aug	Mon-Sun (7)	3 hr/day (Mon-Fri) 5 hr/day (Sat-Sun)

Estimated Total Working Days: 90

Estimated Hours: 350 hours (This may change as per requirements).

To report my progress, I will provide detailed progress updates every week, outlining the tasks completed, challenges faced, and plans for the upcoming week. These updates will be shared on the project's mailing list or designated communication channel.

Technical Requirements

According to the discussion on Music Blocks's Matrix (Element) server, the expected libraries to use for real-time collaboration include:

1. Socket.io
2. Y.js.

Socket.io can't handle conflicts directly. To implement conflict management in collaboration, we can use Conflicts-Free Replicated Data Types (CRDT). I made a small **Prototype** that uses WebSockets for real-time collaboration.
A chat feature can also be implemented using the same libraries. **Here** is an example of socket.io's GitHub page implementing the same.

Benefits of using Socket.io

- **Real-time Communication:** Socket.io enables real-time, bidirectional communication between clients and servers, crucial for collaborative applications.
- **Event-Based Architecture:** Its event-driven model simplifies handling various user interactions and updates.
- **Cross-Browser Compatibility:** Works across different browsers, ensuring a consistent experience for all users.
- **Scalability:** Supports scaling to accommodate a large number of concurrent users or connections.
- **Error Handling:** Provides robust error handling, ensuring reliable communication even in challenging conditions.
- **Room Support:** Allows grouping clients into rooms, facilitating targeted messaging and collaboration.
- **Low Latency:** Minimizes latency, providing a more responsive and engaging user experience.

- **Ease of Use:** Provides a simple API that abstracts the complexities of WebSokets, making it easy to integrate into the application.
- **Versatility:** This can be used in various types of applications, from our chat application to real-time project sharing/editing.

Benefits of using Y.js

- **Conflict Resolution:** Y.js uses CRDTs to handle concurrent edits, ensuring that all users see a consistent view of the document without conflicts.
- **Offline Editing:** Users can edit the document even when offline, and changes will be synced automatically when the connection is restored.
- **Scalability:** Y.js is designed to scale to a large number of users collaborating on a document simultaneously.
- **Efficiency:** Y.js uses efficient data structures and algorithms to minimize bandwidth and processing requirements, making it suitable for low-latency applications.
- **Customizable:** Y.js provides a flexible API that allows developers to customize and extend its functionality to meet specific requirements.
- **Conflict Resolution Strategies:** Y.js offers different conflict resolution strategies, allowing developers to choose the most suitable approach based on their application requirements. Strategies include last-write-wins, highest-priority-wins, and more.

In a few of the previous discussions, mentors mentioned Jabber as the technology for collaboration. So, whether to use WebSockets or Jabber or rewrite something similar to that for Music Blocks is still under consideration. The final technology to be used will be decided after discussions with mentors.

Other Summer Obligations,

I have no commitments in the summer. I'll be staying back home for most of it. I have mentioned my typical working hours above and on average will be able to spend 40-45 hours per week on the project.

Communication Channels

I am active on Emails and the Matrix (Element) app. I can work with whatever platform my mentor prefers. Meetings can be held every week to discuss progress in the project.

Section 2: Proposal Details

Problem Statement

Final Target	**Add real-time collaboration to Music Blocks**
Target Audience	● Music Blocks Users
Core User Need	● **Real-time Collaboration**: Users need the ability to collaborate on Music Blocks projects with others in real-time, sharing code stacks and graphical output instantly. ● **Synchronization**: Users require a high degree of synchronization between their projects across different browsers, ensuring that changes made by one user are immediately reflected for all collaborators with low latency.

Section 2.1: WHAT (Key Milestones)

1. **Invite friends to join the project.**
    a. Start a session
    b. Generate a shareable link to invite
2. **Create a common space to collaborate.**
    a. Give a unique ID to the user (the session creator)
    b. Create an interface for collaboration
    c. The interface can have the following elements:
        i. A **list** of friends who've joined
        ii. A **chat section** for the users
        iii. **Common space** for collaboration
        iv. Names of the users on their interfaces
        v. A **stop** button
3. **Connect friends to the common space.**
    a. Send the friends to the common space when they click on the link
    b. Give each friend a unique ID
    c. Give the friends a way to set their display name (Username)
    d. Show the new joiners in the **list**
4. **Share the same state of the project among all the friends.**
    a. Share the project among all in the common space
    b. Share the position and movement of the cursor (with its username) among all the friends
5. **Collaborate in real-time with low latency.**
    a. Enable updates to the project for all the friends when changes are made by one friend
    b. Handle the possible conflicts
6. **Save the final changes to the original project.**

a. Save the final project when all the users quit the session
b. Handle edge cases
c. Terminate the session

Section 2.2: HOW

I have divided this project into three parts.

1. **Implementation of User Management and Collaboration Setup**
2. **Implementation of Real-Time Collaboration Interface and Activity**
3. **Data Management and Finalization**

Part 1: Implementation of User Management and Collaboration Setup.

User Management and Collaboration Setup can be implemented by following these steps.

- **Setup route that serves the common room**

  For this, I will have to define a new route on the server side that will host the common space. That route will look something similar to this:

```
https://musicblocks.sugarlabs.org//#room=${room_id},${user_id}
```

  Where **`room_id`** will be the unique ID of the room where users will collaborate and **`user_id`** will be the unique ID given to the creator of the session.

- **Implement the functionality to start a new session and create a common space for collaboration.**

  For the collaboration, the standard practice is to start a session so that the changes in the project can be tracked. Also, a session works as a flag for the collaboration that can be used to start/end certain activities when the collaboration starts/ends.

  To implement this functionality, I'll emit a **`startSession`** event when the 'Start a Session' button is clicked. The code snippets below are the overview of this functionality.

  Server-side:

```
const wss = new WebSocket.Server({ port: 3000 });
const rooms = new Map();
wss.on('connection', (ws) => {
  ws.on('message', (message) => {
    const data = JSON.parse(message);
    if (data.type === 'startSession') {
      const roomId = uuidv4();
      const userId = uuidv4();
      ws.send(JSON.stringify({ type: 'sessionCreated', roomId, userId
}));
    }});
});
```

  Client-side

```
const ws = new WebSocket('ws://localhost:3000');
let isSessionOn = false;
let shareableLink;
startASession.addeventlistener('click', ()=>{
  ws.send(JSON.stringify({ type: 'startSession' }));
```

```
})
ws.onmessage = (event) => {
  const data = JSON.parse(event.data);
  if (data.type === 'sessionCreated') {
    const roomId = data.roomId;
    const userId = data.userId;
    shareableLink = createRoomLink(roomId, userId);
    isSessionOn = true;
  }
};
```

- **Implement functionality to generate a shareable link that can be sent to friends to invite them to join the session.**

  A function **`createRoomLink`** will be defined for it and it will return the link that we'll share to invite users for collaboration.

```
const createRoomLink = (roomId, userId) => {
  const link =
`https://musicblocks.sugarlabs.com/#room=${roomId},${userId}`;
  return link;
};
const roomLink = createRoomLink();
```

Note: We can use the UUID module to generate unique ID's for users and sessions.

- **Implement logic to connect friends to the common space when they click on the shareable link.**
- **Assign each friend a unique ID and manage their connection to the session.**

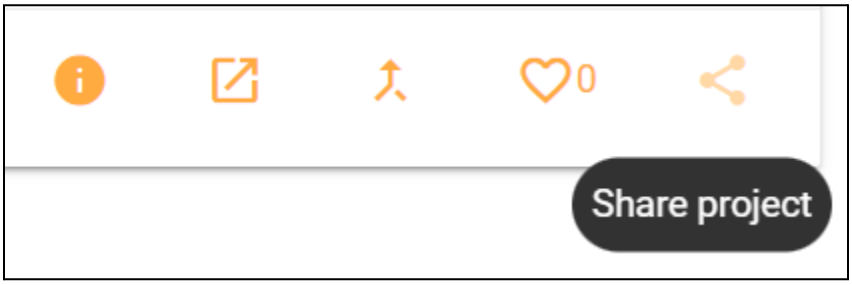
When a user clicks on that shared link, the collaboration interface will be opened for them and a unique ID will be assigned. We'll store their unique IDs and joining orders in the `rooms` object for future use cases.

Example:

```
wss.on('connection', (ws) => {
  ws.on('message', (message) => {
    const data = JSON.parse(message);
    if (data.type === 'join') {
      const roomId = data.roomId;
      const userId = uuidv4();
 // Store the user ID and WebSocket connection in the room
      if (!rooms.has(roomId)) {
        rooms.set(roomId, new Map());
      }
      const room = rooms.get(roomId);
      const joiningOrder = room.size + 1;
      room.set(userId, { ws, joiningOrder });
// Broadcast to all users in the room that a new user has joined
      rooms.get(roomId).forEach((userWs, id) => {
        if (id !== userId) {
          userWs.send(JSON.stringify({ type: 'userJoined', userId }));
        }
      });
    }
  });
});
```

- **Add UI for "Collaborate", "Start a Session" and "Send Link" functionalities:**

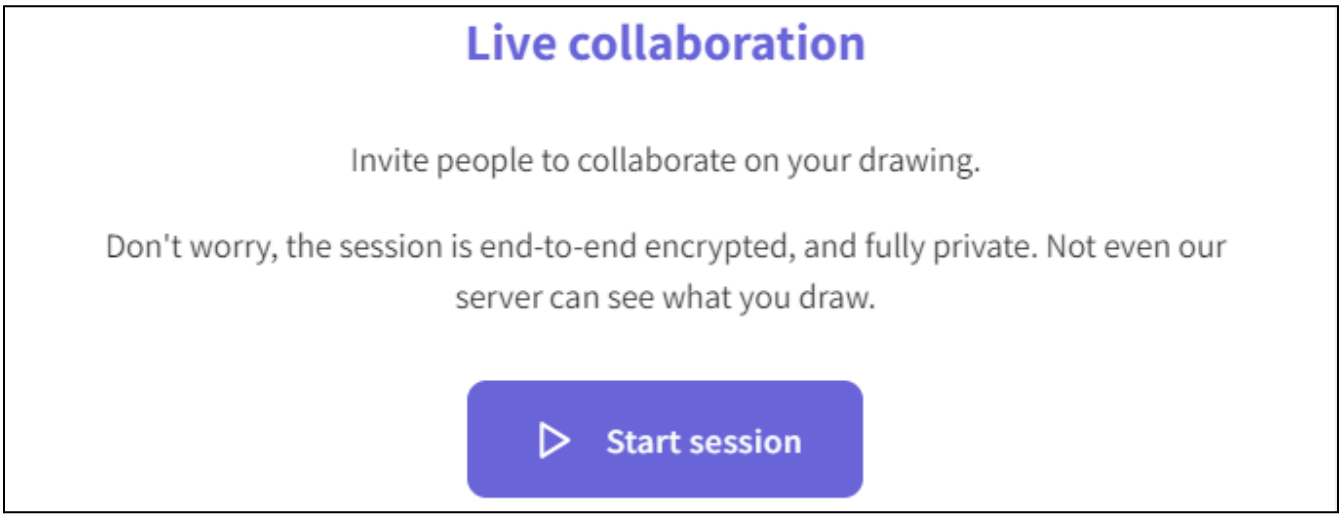
  Collaborate button UI (similar to this)



As we already have buttons like “Share Project”, “Open in Music Blocks”, and “Merge”, Besides these buttons, I’ll implement a similar one for “Collaborate” by the following practice.

```
<a class="project-icon tooltipped" data-position="bottom"
data-delay="50" data-tooltip="${_("Collaborate")}"
id="global-project-collaborate-{ID}"><i
class="material-icons">share</i></a>
```

Start a session card UI (similar to this)

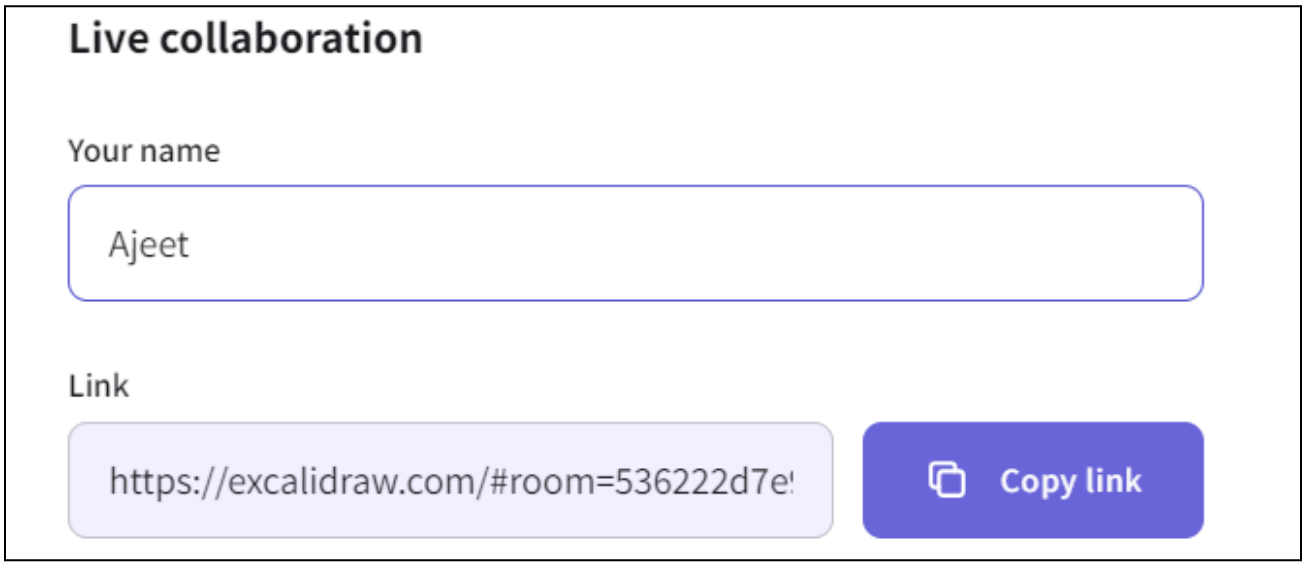


In order to add this UI, I'll create a container div. Then a card div. And after adding the necessary elements I'll add proper styling to make it perfect. The code snippet below is an example of the implementation of this UI.

```
function startSessionCard() {
const container = document.createElement('div');
container.classList.add('card-container');
document.body.appendChild(container);

const card = document.createElement('div');
card.classList.add('card');
container.appendChild(card);

const heading = document.createElement('h2');
heading.textContent = 'Live collaboration';
card.appendChild(heading);

const description = document.createElement('p');
description.textContent = 'Invite your friends to collaborate on your
drawing';
card.appendChild(description);

const button = document.createElement('button');
button.textContent = 'Start session';

card.appendChild(button);

container.style.display = 'flex';
container.style.justifyContent = 'center';
container.style.alignItems = 'center';
container.style.height = '100vh';

card.style.padding = '20px';
card.style.border = '1px solid #ccc';
card.style.borderRadius = '8px';
card.style.textAlign = 'center';
```

```
  return button;
}
```

Send Link card UI (similar to this) -




```
function inviteFriends() {
  const container2 = document.createElement('div');
  container2.classList.add('card-container');
  document.body.appendChild(container2);

  const card2 = document.createElement('div');
  card2.classList.add('card');
  container2.appendChild(card2);

  const closeButton = document.createElement('button');
  closeButton.textContent = '×';
  closeButton.classList.add('close-button');
```

```
card2.appendChild(closeButton);

closeButton.addEventListener('click', () => {
  container2.remove();
});

const heading2 = document.createElement('h2');
heading2.textContent = 'Live collaboration';
card2.appendChild(heading2);

const nameLabel = document.createElement('label');
nameLabel.textContent = 'Your name:';
card2.appendChild(nameLabel);

const nameInput = document.createElement('input');
nameInput.setAttribute('type', 'text');
card2.appendChild(nameInput);

const linkContainer = document.createElement('div');
card2.appendChild(linkContainer);

const linkSpan = document.createElement('span');
linkSpan.textContent = 'Link: ';
linkContainer.appendChild(linkSpan);

const linkInput = document.createElement('input');
linkInput.setAttribute('type', 'text');
linkInput.value = shareableLink;
linkInput.readOnly = true;
linkContainer.appendChild(linkInput);

const copyButton = document.createElement('button');
copyButton.textContent = 'Copy link';
copyButton.addEventListener('click', () => {
  linkInput.select();
  document.execCommand('copy');
});
```

```
  linkContainer.appendChild(copyButton);
}
```

- **Integrate UI with its respective functionality.**

  For the integration of “Collaborate” I would have to catch the button with ID **`global-project-collaborate-${this.id}`** and when it gets clicked, It would render the UI of “Start a Session” card and then I would access the “Start a Session” button from **`startSessionCard()`** function. When this button gets clicked it executes the **`collaborate`** method that would be defined in **`SessionStarter.`**

```
frag.getElementById(`global-project-collaborate-${this.id}`).addEventListe
ner("click", () => {
  const button = startSessionCard();
   button.addEventListener("click", () => {
    inviteFriends();
    Planet.GlobalPlanet.SessionStarter.collaborate(this.id);
  });
});
```

Part 2: Implementation of Real-Time Collaboration Interface and Activity.

After the part 1, I will implement Real-Time collaboration Interface and Activities in the following steps:

- **Develop the frontend interface that could have the following elements**

  1. A list of joined friends

  For this, I would have to extract the friends' names from the **`room`** object I mentioned above and then show them on the screen.

```
function getJoinedUsers() {
 let joinedUsers = [];
 rooms.forEach((room) => {
    room.users.forEach((user) => {
    const userName = user.username;
    joinedUsers.push(userName);
     });
 });
 return joinedUsers
}
```

2. A chat section

Adding a chat feature will be pretty straightforward. To implement this, I would have to write code that fires **`chat`** event and **`messages`** from the client side and when the server catches the **`chat`** event it will emit the **`messages`** to all the peers connected in the collaboration room.

Server-side

```
const server = http.createServer();
const io = new Server(server);

io.on('connection', (socket) => {
    socket.on('chat', (message) => {
    socket.broadcast.emit('chat', message);
  });

  socket.on('disconnect', () => {
    console.log('A user disconnected');
  });
});
```

Client-side (Browser)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Music Blocks Chat</title>
</head>
<body>
  <ul id="messages"></ul>
  <form id="form">
    <input id="input" autocomplete="off" />
    <button id="send">Send</button>
  </form>

  <script
src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/4.3.2/socket.io.
js"></script>
  <script>
    const socket = io();
    const form = document.querySelector('form');
    const input = document.getElementById('input');
    const messages = document.getElementById('messages');
    form.addEventListener('submit', (e) => {
      e.preventDefault();
      if (input.value) {
        socket.emit('chat', input.value);
        input.value = '';
      }
    });

    socket.on('chat', (message) => {
      const item = document.createElement('li');
      item.textContent = message;
      messages.appendChild(item);
      window.scrollTo(0, document.body.scrollHeight);
```

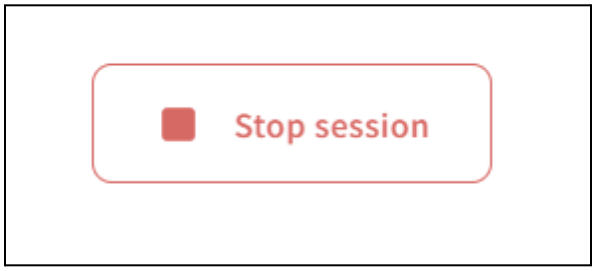
```
    });
  </script>
</body>
</html>
```

3. Common space for collaboration

This will be the most significant part of the interface, containing the project on which users will collaborate. As far as I know, we won't need to add any extra UI for this part, as it is supposed to be like a container that will contain the project. However, I'll discuss with mentors how this common space should look and implement it accordingly.

4. A button to save/Quit/Exit the project

After collaborating on a project, The users would need to stop-the-session/save-the-project. For that, I'd add a button similar to this:



When this button gets clicked, It will trigger the **`handleStopSession()`** function, and that function will execute the logic to exit the user from the session and save a local copy of the project with the last synced changes.

Also as per the discussions with mentors, we won't have to add "ownership" for a project so we'll save the changes of the project when the last user leaves the room or there is no one in the room. A simple prototype of the above functionality is presented below.

```
const stopSessionButton = document.createElement('button');
stopSessionButton.textContent = 'Stop Session';
stopSessionButton.addEventListener('click', handleStopSession);

document.body.appendChild(stopSessionButton);
function handleStopSession() {
    console.log('Session stopped');
}

function handleStopSession() {
   mergeChanges(projectData, lastSyncedChanges);
   saveProjectToLocal(projectData);
  // notify others of this exit
}
function mergeChanges(projectData, changes) {
   //We'll apply CRDT merge strategies here
  Object.assign(projectData, changes);
}
function saveProjectToLocal(projectData) {
  localStorage.setItem('projectData', JSON.stringify(projectData));
}
```

Note: The final list of what elements are needed for the collaboration interface is yet to be discussed with mentors and after the discussion, I will finalize and implement them.

- **Implement functionality to share the project state among all friends in the**

common space.

After the user starts/joins the session, we have to make the ‘common space’ and the ‘project’ (on which the user has initiated the collaboration) available to the user. To achieve this, On a high level, I would create a function **`startCollaboration()`** that will serve two main purposes:

1. Render the “Collaboration Room” UI for the user that includes
    a. A display to show the list of friends who joined
    b. A chat section to chat with them
    c. A “Common space” to collaborate
    d. An exit/save button
2. Load that project from the server and Render it in the “Common space”.

Example:

```
function startSession() {
  setInterface(); // Create the user interface
  setupProject(); // Load and render the project
}

function setInterface() {
  // Here the Interface components I mentioned above will be called
}
function setupProject() {
  const projectId = '1710428168192027';
  const url =
`https://musicblocks.sugarlabs.org/index.html?id=${projectId}&run=True`
;

  fetch(url)
    .then(response => response.text())
    .then(html => {
     const commonSpace = document.getElementById('common-space');
```

```
      commonSpace.innerHTML = html;
    })
    .catch(error => {
      console.error('Error loading project:', error);
    });
}
```

- **Handle real-time updates to the project when changes are made by one of the friends.**

  As discussed with mentors, we are going to use **client-server architecture** for the collaboration. The overview of its implementation would be like this:

  1. First, I'll initialize the Yjs and access the WebSocket server.
  2. After that, I'll have to create a shared data type (e.g., Y.Map, Y.Array) representing the document being collaborated on.
  3. When the user makes a change to the document, those changes will be applied to the local Yjs data type.
  4. As Yjs automatically generates CRDT operations to represent the change, It will emit an update event.
  5. After this, those CRDT operations will be sent over the WebSocket connections to the server.
  6. Now we'll have to Listen for incoming CRDT operations from other clients over the WebSocket connection.
  7. And apply these operations to the local Yjs data type to reflect changes made by other users.
  8. Finally, As Yjs handles the merging of concurrent changes automatically based on the CRDT operations, All clients will receive and apply the same set of operations, ensuring consistent document states across all clients.

The following snippets are the prototype of the above steps:

Client-side

```
const ydoc = new Y.Doc();
const provider = new WebsocketProvider('ws://localhost:3000',
'my-room', ydoc);

const ymap = ydoc.getMap('project');

ymap.observe((event)=>{
  console.log(event)
})

ymap.set('key', 'value');

// Send changes over WebSocket
provider.on('update', (update) => {
  sendUpdateToServer(update);
});

// Receive changes from WebSocket
provider.on('message', (message) => {
  applyUpdateFromServer(message);
});
```

Server-side

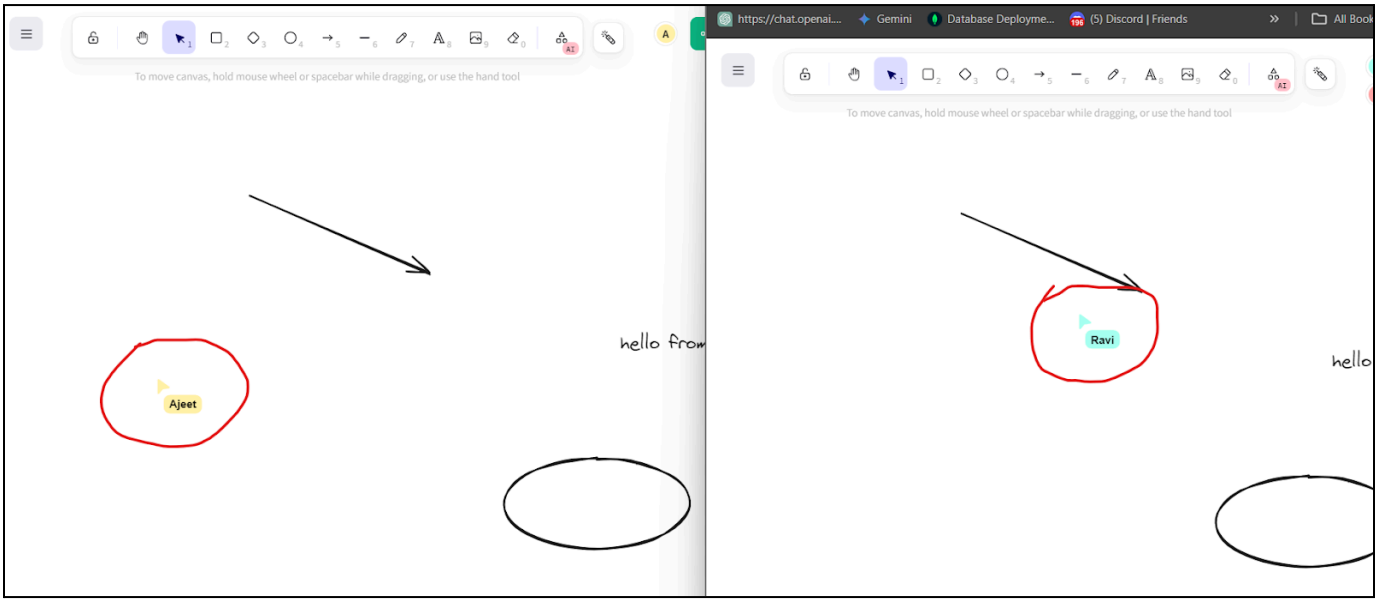
```
const WebSocket = require('ws');
const { Y, WebsocketProvider } = require('yjs');
const ydoc = new Y.Doc();
const wss = new WebSocket.Server({ port: 3000});
```

```
wss.on('connection', (ws) => {
  const provider = new WebsocketProvider(ws, ydoc, 'my-room');
    provider.on('update', (update) => {
       Y.applyUpdate(ydoc, update);
    // Broadcast the update to all other clients
    wss.clients.forEach((client) => {
      if (client !== ws && client.readyState === WebSocket.OPEN) {
        client.send(JSON.stringify(update));
      }
    });
  });

  // Handle disconnection
  ws.on('close', () => { // cleanup code });
});
```

- **Implement functionality to share the position and movement of the cursor among all friends in real time.**

  As of now, we'll already have the users stored with their unique IDs and usernames, now I'll implement a web socket event to share the position of the cursor and username of a particular user among all the peers connected on a project. As clarified by the mentors, It will be helpful for the peers to check where their friends are heading without explicitly asking them in the chat. It will make the collaboration smoother and joyful.

  This functionality will look similar to this:



I've created a demo video of the live sharing of cursors that can be watched [here]

Client-side

```
function handleIncomingMessage(message) {
  const { type, data } = message;
  if (type === 'cursorMove') {
    updateCursor(data.userId, data.cursorPosition);
    updateCursorUI();
  }
}
function sendCursorMove(cursorPosition) {
  const message = {
    type: 'cursorMove',
    data: { cursorPosition },
  };
  WebSocket.send(JSON.stringify(message));
}

function updateCursor(userId, cursorPosition) {
  // Update the cursor position for the user with userId
```

```
}
function updateCursorUI() {
  // Update the UI to reflect the cursor positions
}
document.addEventListener('mousemove', (event) => {
  const cursorPosition = { x: event.clientX, y: event.clientY };
  sendCursorMove(cursorPosition);
});
```

Server-side

```
function broadcastCursorMove(roomId, userId, cursorPosition) {
  const message = {
    type: 'cursorMove',
    data: { userId, cursorPosition },
  };
}

WebSocketServer.on('connection', (socket) => {
  socket.on('message', (message) => {
    const { type, data } = message;
    if (type === 'cursorMove') {
      // Broadcast the cursor movement to all clients in the room
      broadcastCursorMove(socket.roomId, socket.userId,
data.cursorPosition);
    }
  });
});
```

Part 3: Data Management and Finalization.

- **Implement logic to handle conflicts that may arise when multiple users try to modify the same part of the project.**

  As we use CRDT techniques to synchronize the changes, it handles the conflicts on its own. But along with this, we would have to add our methods to handle the conflicts.

  For example, Mentor Walter Bender suggested that when applying concurrent changes in the project made by users, we should give priority based on the order in which they have joined.

  Here is the overview of how we can implement it:

  As we have already created the **`rooms`** object to store the information of the user like **`username, unique user ID,`** and **`joinOrder,`** Using the CRDT, I'll add the logic to sort the changes based on the joining order and then apply them.

```
const ydoc = new Y.Doc();
const provider = new WebsocketProvider('ws://localhost:3000',
'my-room', ydoc);

let joinOrder = [// user's joining order along with their id ];

function applyChanges(changes) {
  changes.sort((a, b) => joinOrder.indexOf(a.userId) -
joinOrder.indexOf(b.userId));

  for (let change of changes) {
    applyChange(change);
  }
```

```
}
function applyChange(change) {
  // add logic to apply the change
}
provider.on('update', (update) => {
  applyChanges(update);
});
```

- **Implement functionality to save the final changes to the original project when all users quit the session.**

  When users are done with the collaboration, they'll have to save the final project. For this, I'll implement the functionality in the following manner:

  1. By now we'd have already determined the criteria (quit/save/exit) that users have finished collaborating, we would use this as a signal to finalize the project.
  2. After this, we'll make sure that the project state in all the clients' browsers is in syn.
  3. Then, we collect the project data from one of the users (most preferably from the one who initiated the collaboration).
  4. Finally, we will make an API call to store the project on the server.
  5. Lastly, A notification will be sent to all the peers that "Project saved".

- **Provide a way to terminate the common space and end the session.**

  As I mentioned above, we'll provide the users a button to "Stop the session". If

a user stops the session from his end, he will have a local copy of the project with the last synced changes at the point he stopped the session.

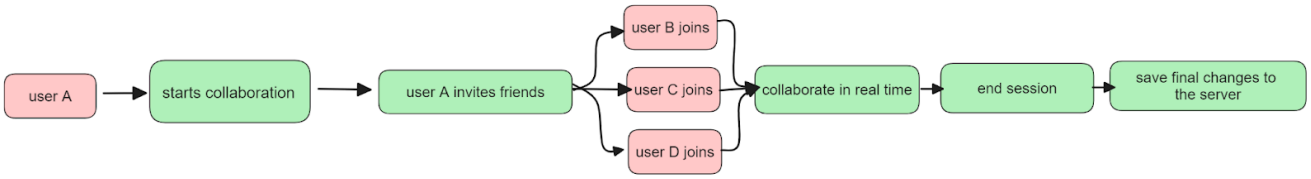
Lastly, We'll save the final changes when the last person leaves the room or when there's no one in the room. Here is a simple prototype of this:

```
const WebSocket = require('ws');
const Y = require('yjs');
require('y-memory')(Y);

const wss = new WebSocket.Server({ port: 8080 });
const ydoc = new Y.Doc();
const room = ydoc.getMap('room');
const users = ydoc.getMap('users');

wss.on('connection', (ws) => {
  const userId = uuidv4();
  users.set(userId, { connected: true });
  ws.send(JSON.stringify({ type: 'userId', userId }));

  ws.on('message', (message) => {
    const data = JSON.parse(message);

    switch (data.type) {
      case 'disconnect':
        users.set(userId, { connected: false });
        checkEmptyRoom();
        break;
        }
  });

  ws.on('close', () => {
       users.set(userId, { connected: false });
    checkEmptyRoom();
  });
```

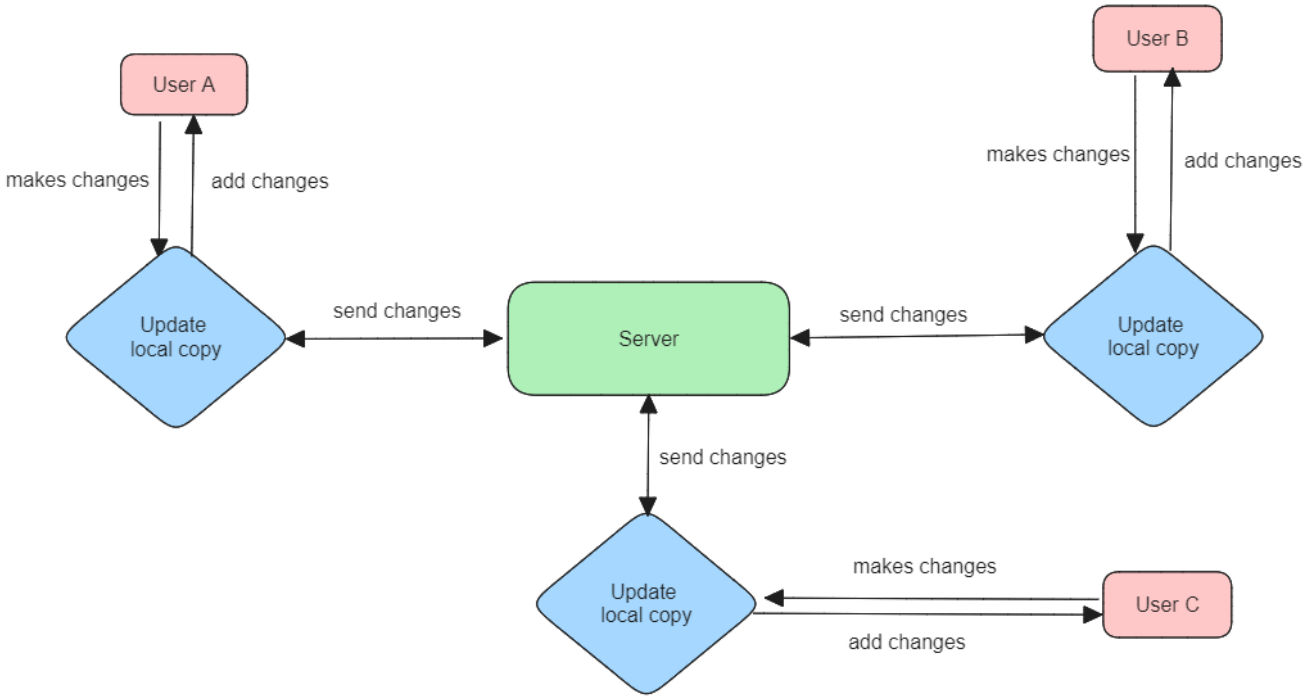
```
});

function checkEmptyRoom() {
  if (Array.from(users.values()).some(user => user.connected)) {
    return;
  }
  wss.clients.forEach(client => {
    if (client.readyState === WebSocket.OPEN) {
      client.close();
    }
  });
}
```

Collaboration Flowchart

1- Overall Workflow of the collaboration



2- Real-time collaboration Workflow



Possible Edge Cases

- **What if multiple users make changes at the same time?**

  For this, we are already using the CRDT so it will be handled by it. If the need arises, I'll consider implementing custom solutions for it like setting up priority rules, using timestamps and version vectors, etc.

- **What if a peer lost the network connection unexpectedly?**

  To deal with this scenario, when a peer loses its network, we will make attempts to reconnect the peer to the network. This can be done using the WebSockets reconnection mechanism.

  If reconnection attempts fail, we can switch the peer to an offline mode where local changes are still allowed but not synchronized with other peers.

Once the network connection is restored, we can attempt to synchronize the changes.

Until the peer reconnects to the network, we can buffer the changes made by the peer locally and Once the connection is back, the buffered changes can be synchronized with other peers.

If the peer makes conflicting changes while offline, we can use conflict resolution strategies to resolve conflicts once the connection is restored.

Also when the peer loses the network, a message about network loss will be displayed on its screen.

If the network connection cannot be restored after multiple attempts, we may choose to terminate the collaboration session and notify other peers about the disconnected peer.

- **What if a peer joins late and a lot of changes have been made in the original project?**

  As per the discussions with mentors, the ideal thing to do in this scenario would be to show the current state of the project to the new joiner irrespective of when they have joined in the duration of the session.

- **What should be the limit of peers to join the common space?**

  Once the real-time collaboration is implemented, We'll experiment with the limits of this, and with its insights and discussion with the mentors I will implement it as per the conclusion.

- **What if the user exits from the common space and joins after some time?**

  In such a case when the user exits in the middle of the collaboration and wants to join after some time, We can show them the current state of the project. As of now, the way to rejoin the session is only to click on that

shared link again. When they click the link they will be joined as a new collaborator.

- **What if the user closes the window by mistake?**

  If the user closes the window by mistake, they will be disconnected from the room. As we don't store the user information permanently, So a simple and easy way to rejoin the session can be to click on that link and join as a new collaborator as described above.

These are the few edge cases I have in my mind. I will discover and discuss more edge cases with mentors and will consider the implementation for them. Also, some edge cases may arise when we implement the functionalities mentioned above. Those edge cases can be considered at the same time of implementation.

Note: While making this proposal, I have taken the reference from socket.io and Y.js docs. Along with this, I've used the proposal template provided by the Sugar Labs.

Implementation Plan

GSoC is around about 12 weeks in duration, with about 25 days of Community Bonding Period in Addition.
I will be spending **80% of the time on implementing the functionalities** in this project,**10% of the time on fixing the bugs** left out in the current version of the project, and the **remaining 10% of the time on testing the app and preparing the Wiki and writing documentation for the project**.
The detailed timeline is linked below.

Timeline	Start Date	End Date	Task
Community Bonding	1 May	26 May	Further requirements gathering, reading docs, and getting familiar with the codebase
	27 May	29 May	Setting up the route for the collaboration room
	30 May	3 June	Implementing the functionality to start a session and create a common space for collaboration
	4 June	8 June	Implementing ID assignments and managing their connection to the session
	9 June	15 June	Implementing the logic to connect with friends to the common room when they click on the shared link
	16 June	22 June	Adding UI for “Collaborate”, “Start a Session”, and “Send Link” functionalities
	23 June	27 June	Integrating UI with its respective functionalities
	28 June	5 July	Developing the interface for the collaboration room that includes a common space, a chat section, a display of users who joined, and a stop button
Phase 1 Evaluation	8 July	12 July	
	6 July	12 July	Implementing the functionality to share the project state among all the users connected in the common space
	13 July	27 July	Implementing the logic to handle the

			real-time update to the project when changes are made by one of the connected users
	28 July	1 August	Adding the logic to share the position and movement of the cursor among all the users
	2 August	8 August	Implementing the functionality to handle the conflicts that may arise during the collaboration.
	9 August	15 August	Implementing the functionality to save the final changes to the original project when finalization gets triggered
	16 August	20 August	Integration of the session termination and afterward activities
Phase 2 Evaluation	19 August	26 August	
	21 August	26 August	Preparing Documentation, Wiki and FAQs, and a Webcast on the Final Product.

Future Work

In the future, I am going to work on

1. improving the current real-time collaboration
2. implementing other functionalities like maintaining the history of the changes and adding a section where users would be able to search what rooms are active to collaborate.
3. handling some other edge cases that may arise when actual users use this

and provide feedback.

Also, I am very interested in the Music Blocks V4 project so I will work on it as well.

I can assure you that if I get selected to work at Sugar Labs this summer, I definitely will do my best to make this project successful and would love to continue working with Sugar Lab's other projects even after the summer.

Also for some reason, if I am not selected this year even then I'll contribute to this and other projects as much as possible and retry again next year.

Looking forward to working with you.
Thanks, And Regards
Ajeet Pratap Singh