GSoC 2020 Proposal

Sugarizer Knowledge Activity Pack

Personal Information

My name is Dhruv Misra and I'm currently enrolled as an undergraduate student at Guru Gobind Singh Indraprastha University, New Delhi. I'm in my third year pursuing a degree in Computer Science.

Location: New Delhi, India (UTC+05:30)
Languages: English, Hindi (Native)
Preferred working hours: 10:00 AM - 10:00 PM (can be shifted as per the requirement)

Contact Information

E-mail ID: dhruvmisra@live.com
Github: dhruvmisra
LinkedIn: https://www.linkedin.com/in/dhruv-misra-35a96a170/
Resume: Resume
IRC Nick: dhruvmisra or dhruv_misra

Programming Experience

I learned the basics of programming in the final years of high school and spent my freshman year focusing on Competitive Programming. I developed a keen interest in Web Development during the second year of university and learnt the javascript framework Vue.js. I have worked on multiple projects, some personal and some as interns for different organisations using Vue.js and other web technologies.

As my love for the framework grew, I created my own open-source NPM package for Vue.js called Vue-event-card. Apart from this, I have worked on various open source projects, the details of which can be found later in the proposal.

I started contributing to Sugar Labs by fixing bugs and as I became familiar with the codebase, I started implementing functionalities for different activities. I developed

fully-functional activities for Sugarizer and helped in porting others from Sugar. The details of these contributions are listed below.

Contributions to Sugar Labs

I have been contributing to the Sugar Labs' repositories since February and have solved a lot of bugs and added enhancements. Here is a complete list of Pull Requests:

GitHub Link	Title	Status
#652	Added Contents button in toolbar for Ebooks	Merged
#656	Fixed erase button overflow	Merged
#659	Chess Activity for GSoC 2020	Challenge
#662	Fixed text not being visible in Search bar	Merged
#667	Fixed text overflow in Shared Notes	Merged
#672	Fixed transparent background of dialog box	Merged
#674	Fixed opening links in Markdown	Merged
#683	Fraction Bounce Activity Port	Under Review
#690	Fixed the element selection in password	Merged
#234	Added and fixed some Hindi translations	Merged
#236	Fixed scroll on reordering activities	Merged
#239	Fixed dropbox overflow	Merged
#241	Fixed selected text localization	Under Review
#242	Fixed query retention after language switch	Under Review

Here are the issues I raised:

GitHub Link	Title	Status
#651	Add an option to navigate back to Contents in	Fixed

	Ebook Reader	
#655	Video erase button hidden in Record Activity	Fixed
#661	Text not visible in search bar of Calligra Activity	Fixed
#671	Dialog box has no background in Markdown	Fixed
#673	Markdown link opens in frame which crashes the output	Fixed
#689	Password Tutorial targets the wrong element	Fixed
#235	Activity list doesn't scroll while reordering, causes multiple server calls	Open
#238	Overflow on opening any dropbox	Fixed
#240	Selected option's text isn't localized	Open

I will keep updating this list as I make more contributions.

Other open-source contributions

Vue-event-card

A unique and interactive way to showcase your events with smooth animations and flexible designs. This is an NPM package based on Vue.js.

GitHub: https://github.com/dhruvmisra/vue-event-card

InfoXpression 2019

Created the official website for InfoXpresison which is the annual techno-cultural fest of University School of Information and Communication Technology. Created on Vue.js using multiple other javascript libraries.

Website Link: https://infoxpression.in/

GitHub contributions: https://github.com/techspaceusict/infox19/graphs/contributors

Work Samples

Enfeed

Enfeed is a real-time audience engagement platform for events. The web application features multiple functionalities like Live Q&As, polls, feedback, chat with fellow attendees, etc. I was responsible for creating this product as an intern under Xploryo. I learned to work against deadlines and production level workflows handling multiple users at this firm.

Built on Nuxt.js, a framework based on Vue.js and Google's Firebase for backend.
Website Link: https://www.enfeed.in/

Webapp Link: https://app.enfeed.in/

German Leprosy Relief Association India

GLRA-India in a Non-Profit Organisation which has been serving the cause of leprosy for more than 50 years. This was a freelancing project I took in winters where we had to create the entire website for the NGO displaying the various areas of intervention and porting the existing payment portal.

Built on Vue.js and hosted on cpanel.
Website Link: https://www.glraindia.org/

Grynow

Grynow is an influencer marketing company working with influencers on all major platforms including Youtube, Instagram, TikTok, etc. I worked here as a summer intern and was responsible for creating the main website for the company which showcases their exclusive influencers and the various promotions and case studies. I also worked on some other projects for the founders. I learned about SEO and project management here.

Built on pure HTML, CSS and JavaScript using minimal external libraries.
Website Link: https://www.grynow.in/

Motivation for GSoC

I believe the only constant in my life is the process of learning. I have been fortunate to have had good teachers and mentors in my life and have worked on multiple projects and internships but the knowledge and experience that I would gain from the Google Summer of Code will be unparalleled. It will be one of the most unique learning opportunities with a chance to interact with the most experienced mentors.

Why choose Sugar Labs?

It has been my desire to do something for the society and reform the standard educational ideologies to make for a better future. Sugar Labs, from its core, helps children learn better and faster through their platform. Therefore contributing to and working with Sugar Labs would help me fulfil this goal.

About the Idea

I would be working on the Knowledge Pack which includes the following two activities:

Curriculum

The Curriculum activity will be a way for a student to self check his/her skills in a set of knowledge categories and provide multimedia elements to demonstrate these skills. The Curriculum activity will have the following features:

- It will display a hierarchical set of skills grouped by categories then let the user explore the tree.

- On each skill the user should be able to validate (i.e. acquire skill) and provide multimedia elements (pictures or sounds coming from Journal) to demonstrate the skill.
- The activity will provide a settings mode to edit the set of skills: Create/Update/Delete/Sort skills or categories.
- A category should have a title and a color.
- A skill should have a title and an image.
- It should be possible to generate a Word/ODT document with all skills and dated multimedia elements.
- It should be possible to share its skills on the network.

Vote

The Vote activity will allow easy building of a poll system. The Vote activity will have the following features:

- The user can create a poll then share it on the network so any user could vote in real-time.
- Types of polls:
  - Yes/No
  - Choose value in a list
  - Enter a value
- At the end of the vote, a screen will sum up results of the vote:
  - Statistics
  - Who vote for what
  - Who vote first
- The design of the Vote activity could be inspired by the Exerciser activity:
  - Home screen allow to quickly choose a poll template and run the vote
  - Setting screen allow user to create new poll or customize a poll
- A poll is a label (question) and could integrate a multimedia element (image, audio, video, speech to text).

Implementation

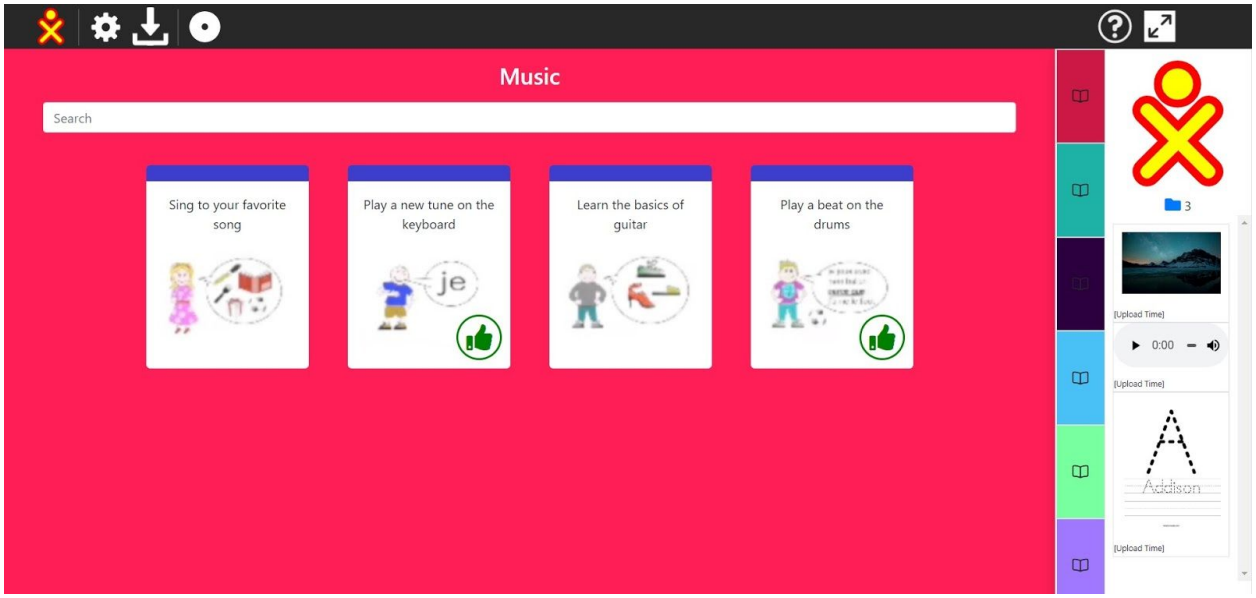
Curriculum

❖ I created a working prototype on Vue.js which can be found here. (GitHub Repository)

Existing libraries to be used to create this project:

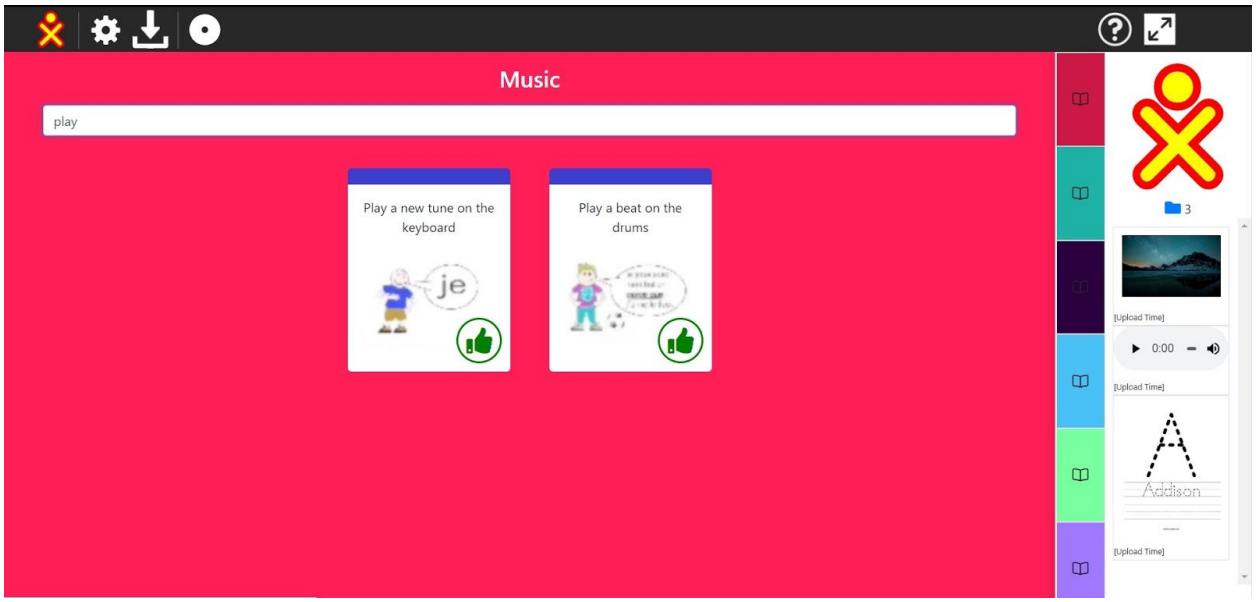
- Html2canvas - A JavaScript library to convert HTML and CSS into a canvas and take a screenshot of it. Used to export to PDF.
- jsPDF - The leading HTML5 client solution for generating PDFs. Used along with html2canvas to generate PDF while exporting.

❖ The main UI of the activity:

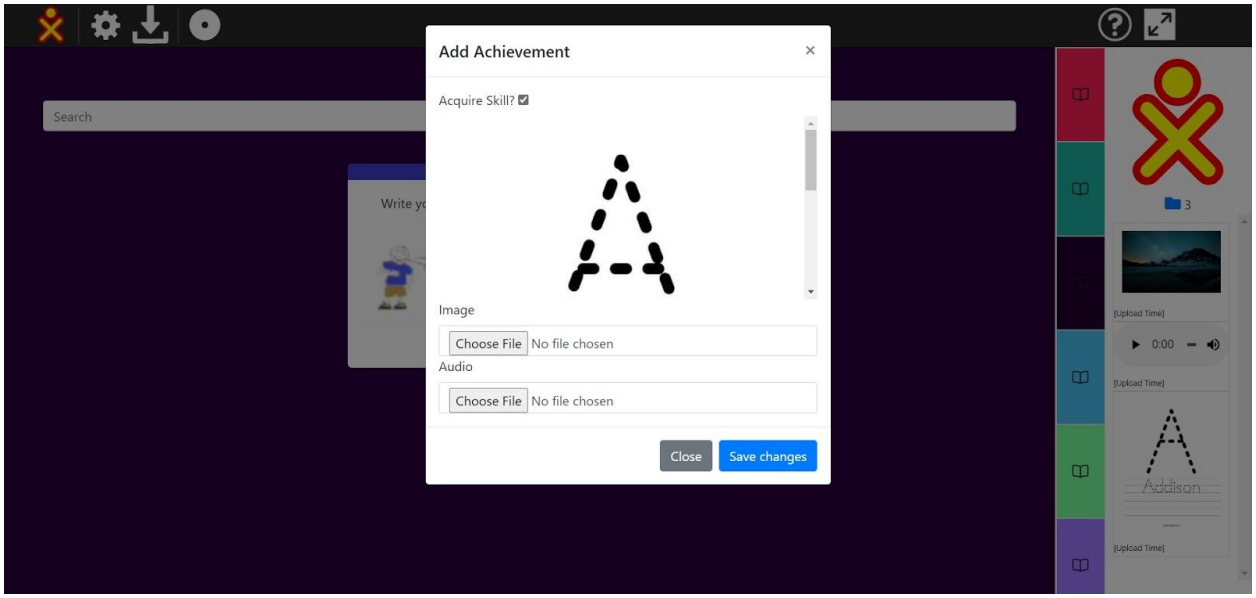


Toolbar buttons: Settings Mode, Download, Network

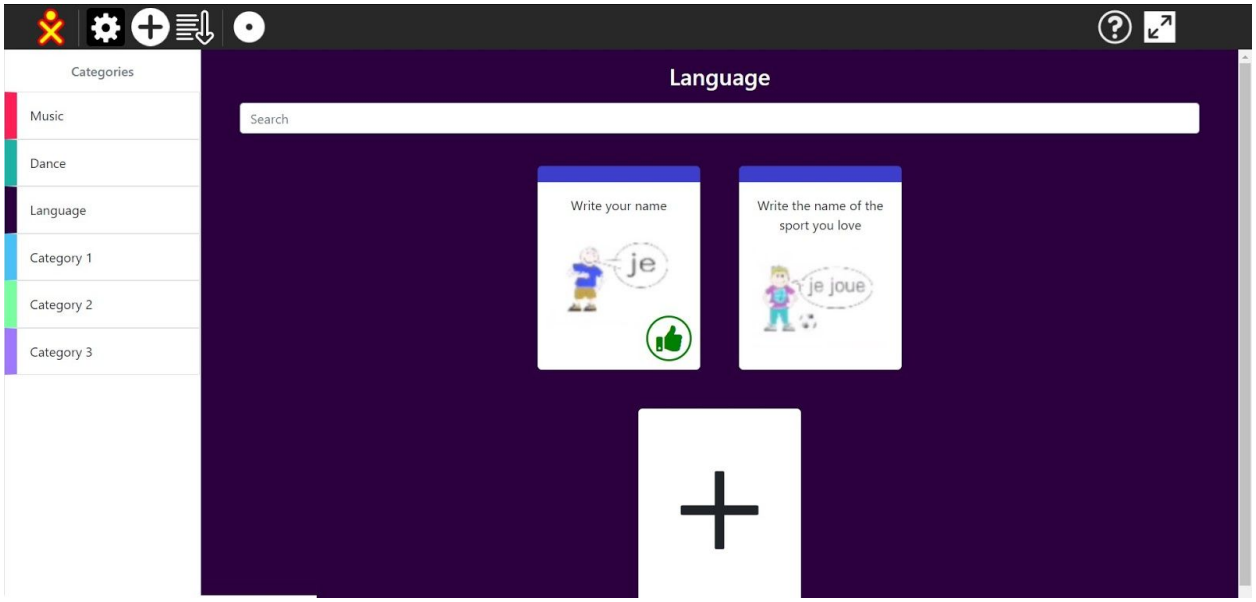
The main screen of the activity will be colourful and will contain user friendly cards for each skill having a title and an image. The acquired skills will have a thumbs up icon on the bottom-right. The panel on the right side provides navigation for all categories. It also shows all the uploaded documents.



The skills can be searched inside a category



Whenever a skill is clicked, a popup will appear showing any uploaded documents for that skill and options to mark the skill as required, to upload image/audio. (**NOTE**: The upload buttons will be linked to Journal and have a different UI, this is just a demonstration on the web)



Toolbar buttons: Settings Mode, Add Category, Reorder Categories, Network

In the Settings mode, the user will be able to create/update/delete/sort categories and skills using drag-and-drop. Add Categories will open a popup to add the title, background colour and icon from the Journal. Add Skill will have a similar popup specifying title and image from Journal.

❖ Observing the nature of this project, I believe it can be decomposed into multiple Vue.js components which can easily be reused dynamically.

The major data members that need to be persisted are:

- Categories[]: Stores all the categories and the skills in it
- Student{}: Stores the information about the student

Some details of the student, like name and buddy colours can be taken from the Journal and others can be inputted on the first run of the activity.

Categories and student information will be saved in the **Journal** whenever these data members are updated to have a persisted state and prevent loss of changes.

Sample object of categories array:

```
{
  id: 0,
  title: "Category 1",
  bg: '#4AC1F6',
```

```
  icon: 'book',
  skills: [
    {
      id: 0,
      title: "Skill 1",
      image: "d1.jpg"
    },
    {
      id: 1,
      title: "Skill 2",
      image: "d2.jpg"
    }
  ]
}
```

Each category has a title, background color and an icon. There is a unique ‘id’ for each category. Category contains a ‘skills’ array to store all skills under that category. Each skill has a title, an image and a unique ‘id’. (Note that skill ‘id’ has to be unique inside a category only, not necessarily among all skills from all categories)

Sample data in student object:

```
{
  name: 'ABC',
  buddyColors: {
    fill: "#FFFF00",
    stroke: '#FF0000'
  },
  age: 10,
  rollNo: 0,
  acquired: {
    0: {
      0: {
        acquired: true,
        image: null,
        audio: null
      },
      1: {
        acquired: false,
        image: null,
        audio: null
      },
    },
    1: {
      0: {
        acquired: false,
```

```
        image: null,
        audio: null
      },
    }
  }
}
```

The student object will have basic information like the name, age, rollNo, etc. It will also have an important acquired property which would be an object of skills acquired by the user. The keys of this object denote the category IDs. The inner object has skill IDs for keys and another special object as the value.

The advantages of using such a structure for the acquired property:

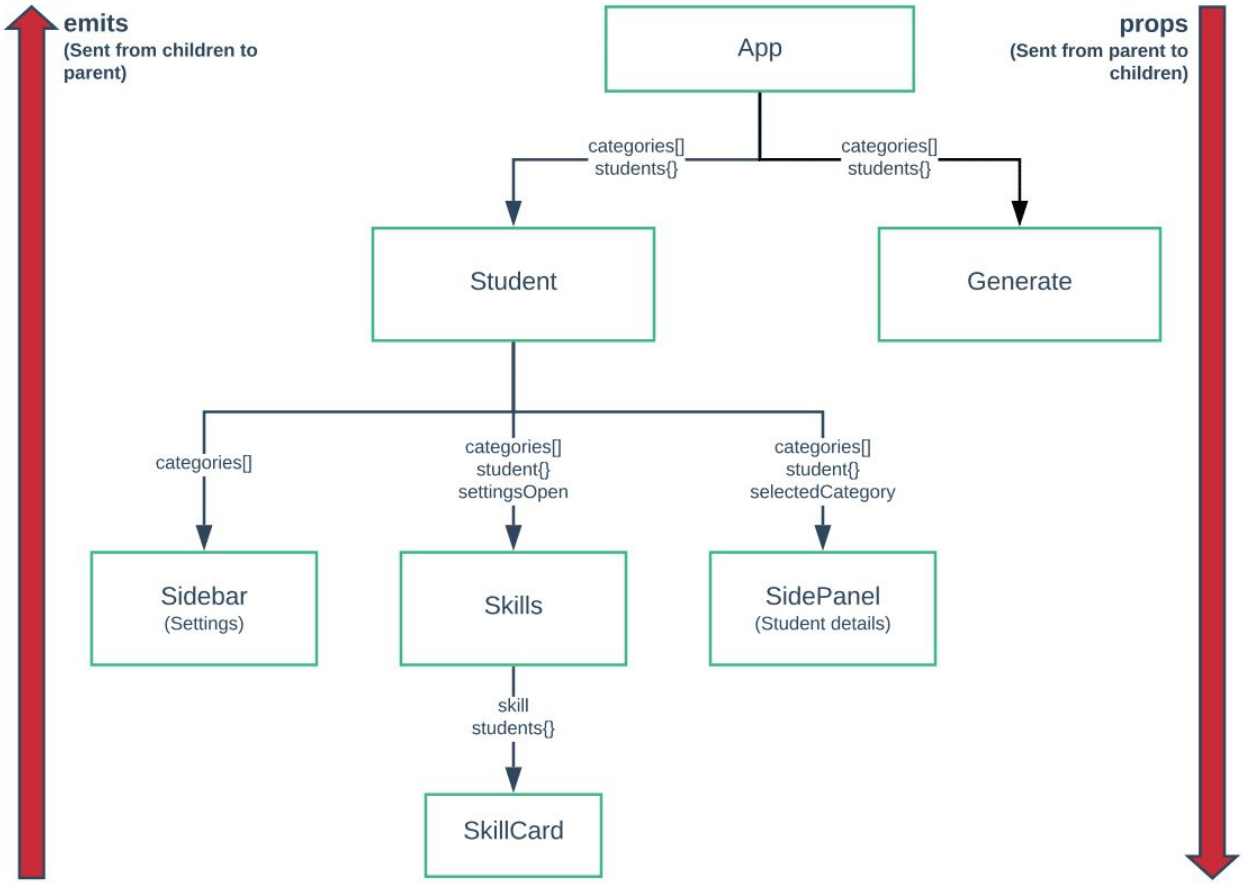
- Reduces redundancy as only the category and skill IDs have to be saved.
- To check if a skill if acquired, a constant time check can be made using the category ID and skill ID:

  student.acquired[**categoryId**][**skillId**].acquired //returns a Boolean

- The remaining image and audio properties can be used to store the uploaded multimedia elements as objects.
  - Images can be converted to base64 data URL using readAsDataURL() function of FileReader and stored as strings.
  - Audio files can be converted to objectURL strings using URL.createObjectURL() and be stored and used easily.
- As all data being handled can be represented by a JavaScript object, it can be converted to JSON and saved in the Journal.

These main data members will reside at the root component and can be sent to the various components using props and the children components can communicate with the parent using emit statements.

❖ The system design and the hierarchy of components can easily be visualized through the following diagram:



The text on arrows denote the props being passed from parent to child

Description of the various components:

Component Name	Description
App	Root component which manages the app and contains main data members and all other system components like the toolbar.
Student	Manages the skills' grid, side panel for student and the sidebar for settings mode. Handles addComponent, addSkill and other emits from child components. Switches between sidebar and side panel according to the value of settings mode.
Generate	This component is required to export the skills of the user to a Word/PDF document.
Sidebar	Visible in settings mode, shows the order of categories and can be used to add/delete/re-order categories. It is visible only in the

	settings mode.
Skills	This component shows a grid of skill cards along with the search bar and category title in normal view. It is reused in the settings mode to show rearrangeable skill cards with an ‘Add Skill’ option card.
SidePanel	This panel shows the user buddy icon and acts as a navigation panel for the various categories. It also shows the number and list of uploaded multimedia elements. Clicking on the uploaded item will navigate to the skill for which the item was uploaded. It is visible only in normal view.
SkillCard	A card component to control the behaviour of the skill card throughout the activity.

- Major workflows:
  - **Adding/deleting Categories**

    The add button in the toolbar opens a modal in the Sidebar component and after filling the required details, the component emits the action of ‘categoryAdded’ all the way up to App where the new category is pushed to the categories array. An entry corresponding to the new category ID is also made in the student.acquired object. As the Vue objects are reactive, the props are updated automatically in all components.

    Deleting a category follows a similar pattern with an emit being fired up to the root App and the final updation taking place there. In this case the category with the specified ID is spliced from the array.

  - **Adding/deleting Skills**

    The add card in Skills component opens a modal to fill the required details. After completion, the details are sent as an object all the up to App where the new skill object is pushed to the categories[categoryID].skills array. The student.acquired[categoryID] is also updated to have an initial object:

```
{
  acquired: false,
  image: null,
  audio: null
}
```

Deleting a skill also follows the same pattern where the skill matching the skill ID is spliced from categories[categoryID].

- **Adding multimedia elements**

  Clicking on a skill in normal view will open a modal which will allow the user to set a skill as ‘acquired’ and also upload an image or audio to support the skill. The uploaded image/audio is converted to base64 string or object URL and then saved along with the upload time to student.acquired[categoryID][skillID].[‘image’/’audio’].

  The modal also checks for any previous uploads by checking the same location and displays the content using a dynamic img tag or an audio tag.

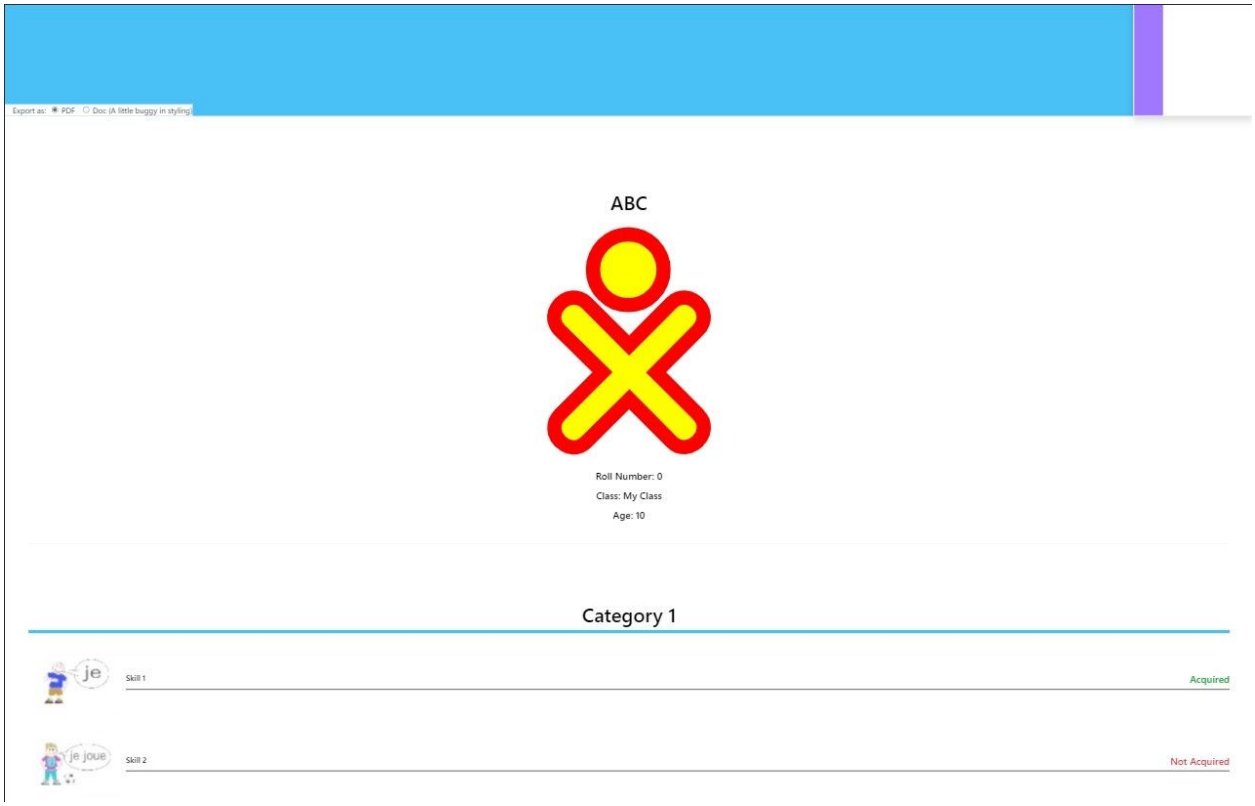
  A thumbs-up icon is shown on the acquired skills.

- **Displaying uploaded documents and linking**

  In the SidePanel, a computed property called ‘documents’ is created which updates whenever the student object is updated. It loops through all acquired activities and saves their object along with categoryID and skillID to an array which is returned in the end. This ‘documents’ property is used to render a list of all uploaded elements using a combination of v-for and v-if statements. A click listener is added to each element using v-on:click to call a method to navigate to the corresponding skill.

❖ **Exporting data to a Word/PDF document**

The Generate component is responsible for generating the required file. It renders the entire document formatted using CSS and data being filled dynamically using Vue.js props. Then this DOM element is used to create the required file. The necessity for such an approach is that the various methods and libraries need the HTML element to be visible at the time of conversion.



The output being file being rendered below the main activity

I was faced with a challenge here to somehow hide the entire DOM element from the user but still keep it visible to the generating functions. I solved this problem using some intelligent CSS and HTML tags. I wrapped the entire DOM element in a wrapper container div and set styles for this wrapper, specifically:

```
.doc-container {
  height: 0;
  overflow: hidden;
}
```

This way the entire element is hidden from the user due to height: 0 but still rendered (visible) for the generating functions to do their job.

JavaScript libraries html2canvas and jsPDF were used to create and download PDF files. Here multiple screenshots are taken of the DOM element according to page size of the PDF and placed into the PDF file. The file is saved in the end.

The Word output is generated using a set of special attributes for the html tag and converting the entire document into a Blob with type: 'application/msword'. This is then encoded into a URL and downloaded for the user.

❖ Network functionality using **Presence** will be added where the user will be able to share his/her skills with the network. The entire data used for rendering skills is in a js object, JSON version of which can be sent through the network efficiently.

The following presence actions can be defined:

- init: Connect the users so the host knows who all will receive the skills
- sendSkills: Send the skills JSON to all/some connected members

The host can select which users shall receive the skills. Once received, the skills will automatically be downloaded on the connected user's device.

❖ Some sort of rewards (stars or number rating) can be given to the students based on the number of skills acquired. This would instill a feeling of competition among the students and would help them learn quicker.

Vote

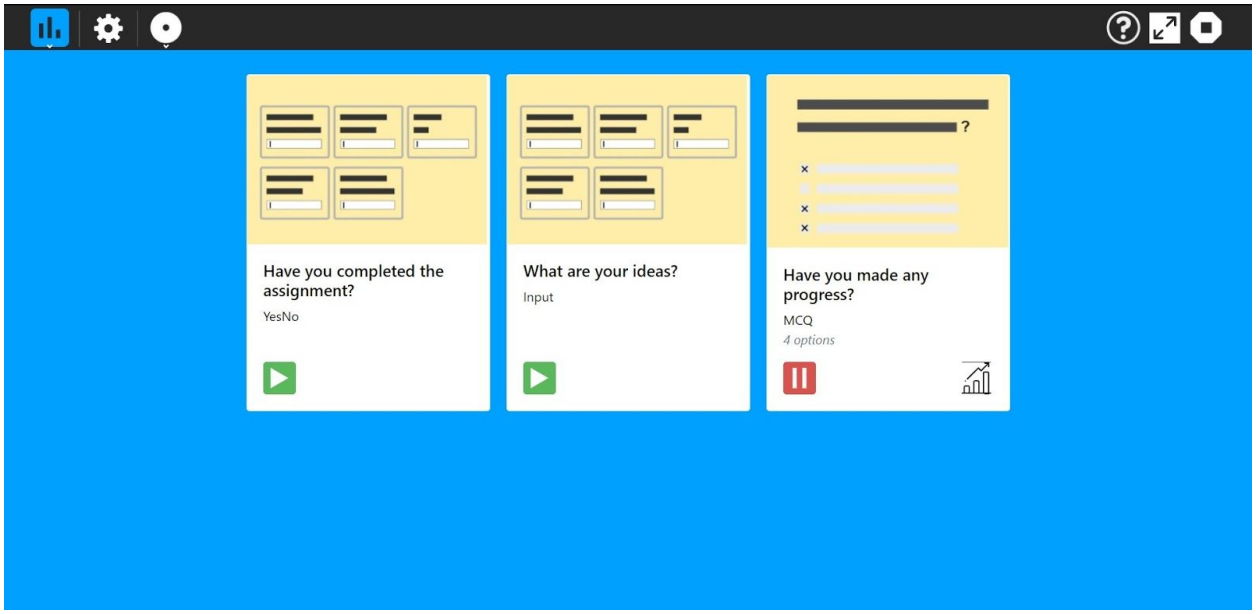
❖ I created a very basic implementation of this activity in Sugarizer using inline Vue.js which can be found here on GitHub.

Libraries which will be utilised to create this activity:

- Chart.js: Simple yet flexible JavaScript library to create numerous types of charts. This will be used to visualize the statistics. It has almost all types of graphs to show data interactively. I will be using this to show bar graphs or pie charts in questions with options.

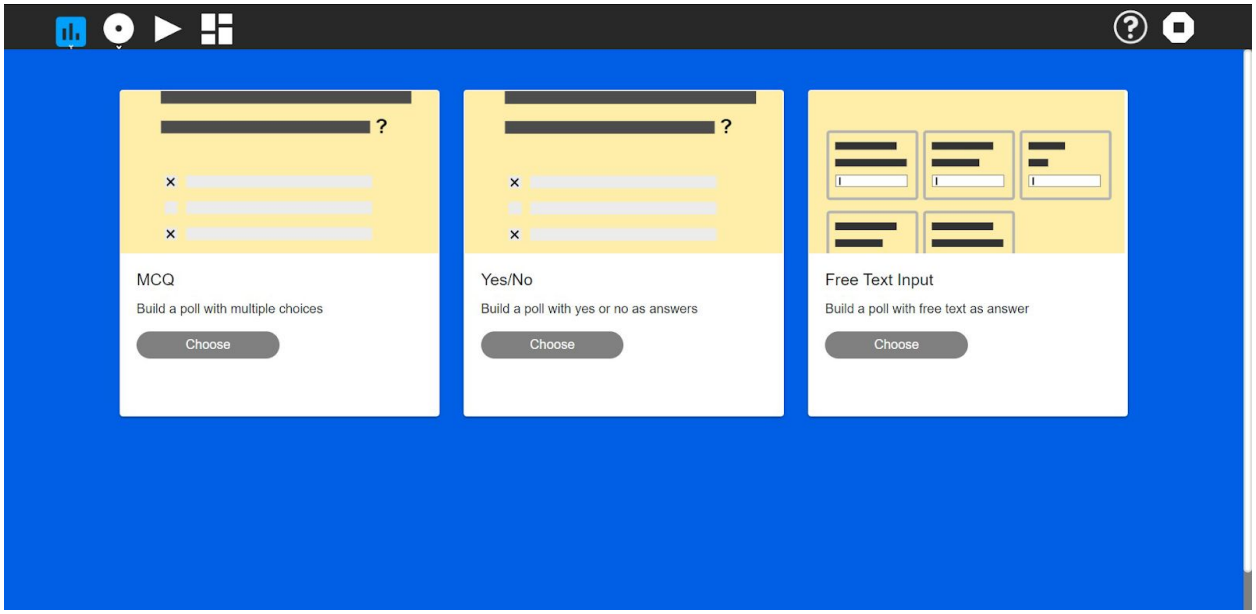
- Export-to-csv: Helper library to quickly and easily create a CSV file in the browser. It will be used to export the poll results to CSV files for external use.

❖ The main UI of the activity:

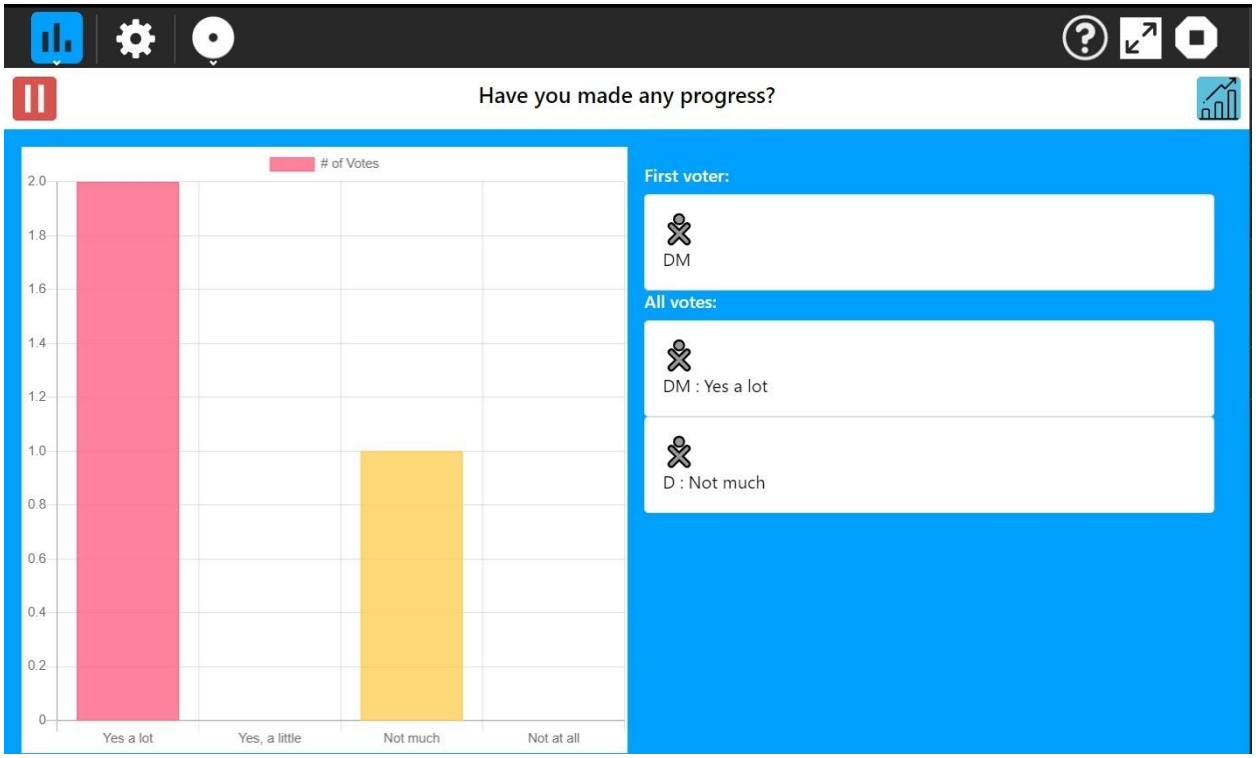


Toolbar buttons: Settings Mode, Network

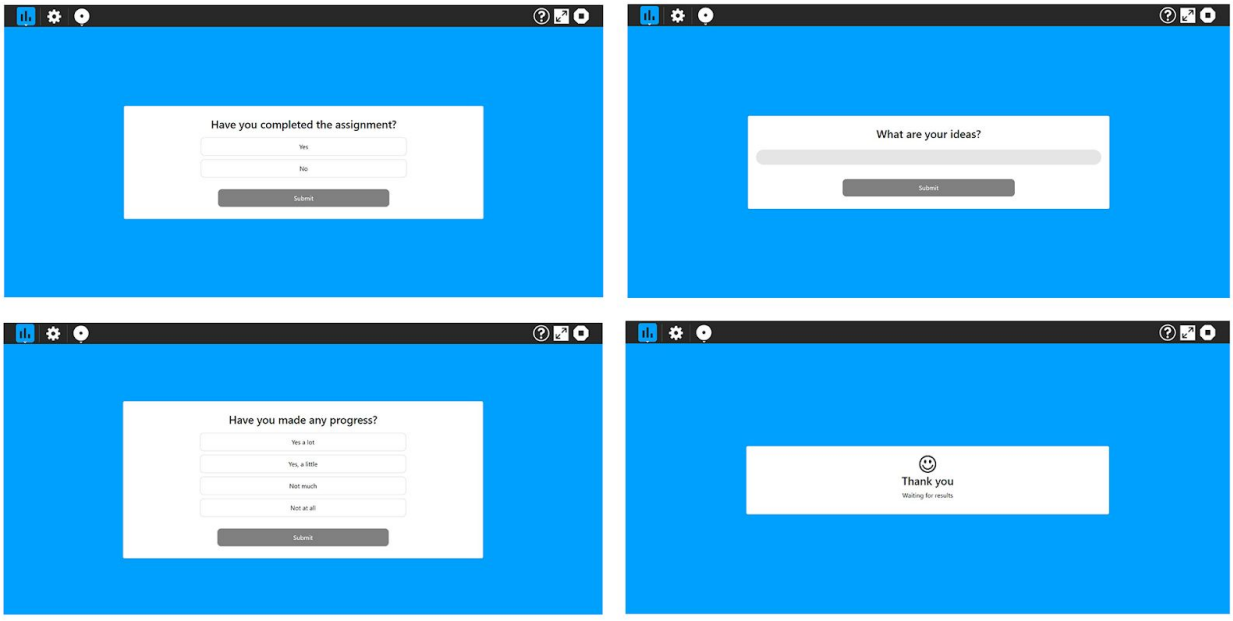
The polls can be started/stopped using the bottom-left icon. Stats can be displayed using the bottom-right icon in the card.



The add poll page will be similar to the Exerciser activity.



The stats will have the bar graph and lists showing votes. (NOTE: The buddy icons will be coloured in that user's colours, in contrast to the way shown in the image)



Voting screen will be adapted to the running poll. It will display a thank you message after submitting a response.

❖ The major data members are:

- Polls[]: Stores all the polls shown on the Home screen
- Answers[]: Stores all the answers obtained during an active poll
- RunningPoll Stores the information about the poll currently running
- Submitted Stores whether or not the user has answered the runningPoll

Polls array can be saved in the **Journal** whenever it is updated to have a persisted state and prevent loss of changes.

Sample polls array:

```
polls: [
  {
    id: 0,
    type: 'YesNo',
    question: 'Have you completed the assignment?'
  },
  {
    id: 1,
    type: 'Text',
    question: 'What are your ideas?'
  },
  {
    id: 2,
    type: 'MCQ',
    question: 'Have you made any progress?',
    answer: 0,
    options: [
      'Yes a lot',
      'Yes, a little',
      'Not much',
      'Not at all'
    ]
  },
],
```

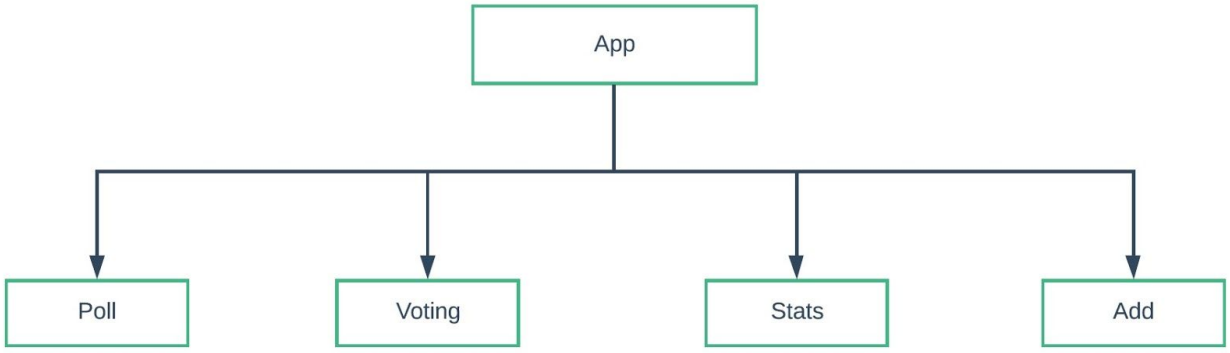
Each poll has a unique id and a type. The type property is used throughout the system to handle the various elements of the poll. Types:

- YesNo Contains only a question with yes or no as allowed answers
- Text Contains only a question with a free text answer
- MCQ Contains a question and an array of options

Images, audios and other multimedia content can also be added as a property to the poll object. Any media can be converted to a string representation which can be stored in the object and parsed as JSON to be sent over the network using **Presence**.

The answers array would consist of two fields, user and answer. User field stores the user's name, networkId and color value. The answer field stores the answer given by the user (differs according to the type of poll).

- The hierarchy of component templates can easily be visualized through the following diagram:



Templates

The communication of data between the components will take place through props and emit statements.

Description of the various components:

Component Name	Description
App	Root component which manages the app and contains main data members and all other system components like the toolbar.
Poll	Contains a grid of all the poll templates generated by the user and manages the switching of polls to active/inactive. It is reused in the settings mode to show edit/delete buttons on the user templates.
Voting	This component is rendered when a shared instance of the activity is opened. It shows the running poll and handles submission of votes from the user.
Stats	This component has all chart.js elements and other forms of statistics regarding the current poll. The host can view the

	statistics anytime but the connected user will only see the results when the host allows it or when the voting ends.
Add	This component can only be viewed from the settings mode and is used to create custom poll templates similar to the Exerciser activity. It also handles editing of poll templates.

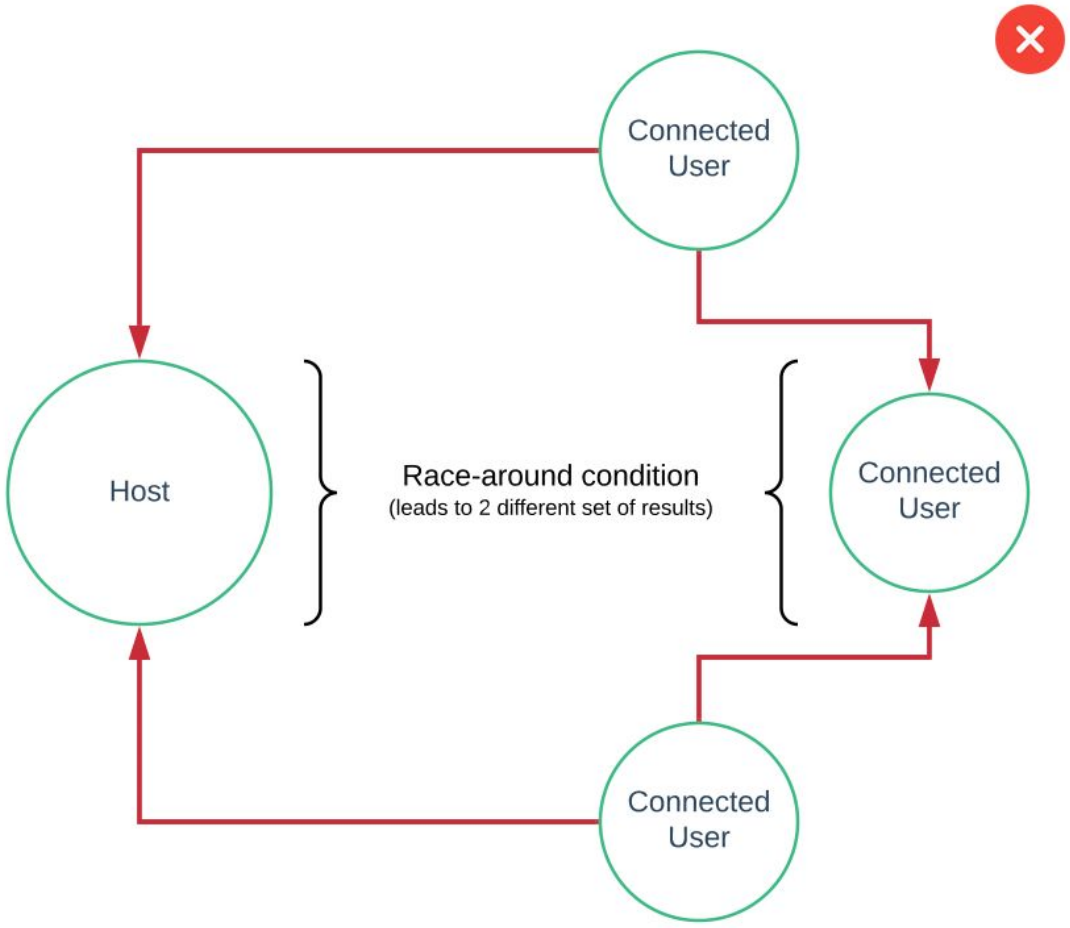
- This activity is purely based on the network hence it will use Presence extensively. The host will share a poll to the network and all users joining the shared activity will receive the poll information and will send their answers to the host. In the end, the results (answers array) will be shared with everyone in the network to show them the results as well. The host will have an option to show real-time results to everyone, which will send the answers array to everyone whenever a new answer is received.

  The following presence actions can be defined:

  - init: Connect the new user and send them the runningPoll (if any) or the results
  - startPoll: Send the selected poll to all users to start voting
  - vote: Send the selected answer to the host
  - showResults Send the answers array to all users to display the results

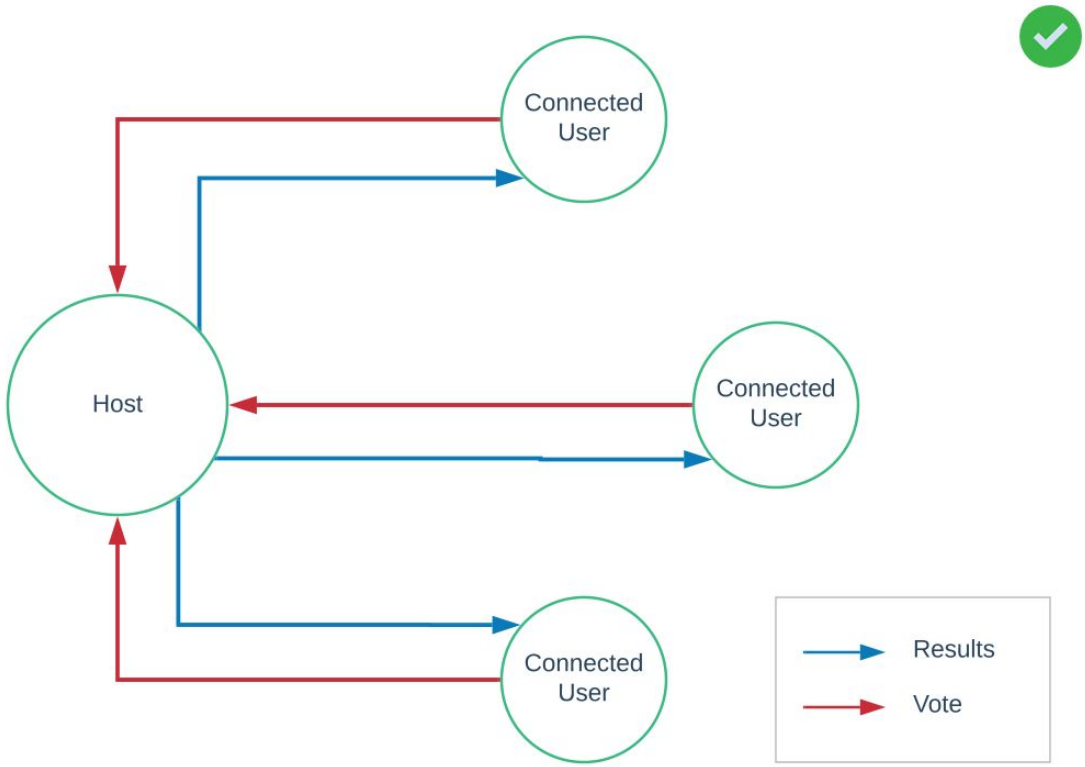
- There can be 2 approaches to maintaining results:

1. **Sending votes to all users in the network to update all answers array**



This can lead to a race-around condition. The vote from User 1 might reach before User 3's vote to the host but the other way around to User 2. This leads to two different answers and leads to inconsistency in results.

2. **Sending votes only to the host and results are sent from the host**



This allows for a consistent result as only a single answers array is maintained and is used to update all users' results.

❖ Major workflows:

- **Adding/editing a poll**

  A new poll can be added by going into settings mode and selecting the add button. This displays the Add component which will be similar to the add section of Exerciser activity. There will be a grid containing types of polls, clicking which will lead to a form to input the required information and multimedia elements for the poll. The add button in the end will emit the poll information contained in an object back to the App where it will be pushed into the polls array.

  Editing a poll will open a similar form containing the poll information. The save button will emit the new poll object and App will replace the old object by the new one using pollId.

  Deleting a poll will splice the poll object from the polls array using the pollId.

- **Starting a poll**

  Clicking on the play button on any poll on the Home screen will set the runningPoll to the selected one and share the activity automatically to the network. The shared and isHost properties are set true. Component displayed is Poll.

- **Voting**

  Once a user joins the network, the shared property is set true and the Voting component is displayed. The runningPoll is received along with the ‘init’ action and is set to the runningPoll. Once answered, the answer is sent using the ‘vote’ action back to the host and the submitted property is set true. The component is changed to Stats once the results are received.

- **Showing results**

  The answers array in the host is updated whenever a user votes. This array can be sent by the host over the network once the polling is stopped or at any time using an option in the toolbar. The ‘showResults’ action will be received by connected users which will set answers to the received array and make statsOpen true. This opens the Stats component.

- **Displaying stats**

  The Stats component is shown when the statsOpen property is true. This property can be triggered by the host on the Home screen or through the ‘showResults’ action. The stats component contains two sections - The graph and the lists.

  The graph section contains a Chart.js canvas to display a bar graph of all the votes. The graph is only visible in polls containing options. It is updated whenever a new answer is received using the watch property in Vue.js.

  The list section has different types of information like who vote first, who vote for what, etc. The list contains the name of the user along with the buddy icon in that user’s colors. It also shows the answer given by that user.

The Stats section can be further improved by providing pie chart and line graph options to visualize the data. Also more information like total votes, winning option or word cloud(in case of free text) can be shown.

- Another good feature will be **exporting** the results in a CSV file for use in external environments. This can be achieved by creating the required array of strings and using export-to-csv library to convert it into a CSV file.

How will it impact Sugar Labs?

The knowledge pack would help students understand two of the most essential qualities, skill acquisition and equality. Acquiring skills is the fundamental goal of education. Understanding that everyone is equal and has an equal contribution is enabled by the voting. Addition of these two activities would prove to be an enhancement to the learning experience and would also help in increasing student engagement with the platform.

The Curriculum activity helps the student be aware of the skills they possess and need to work on. It is a fun activity to track the progress of the student regularly and easily share it with others in the network. It gives them a reason to learn new skills and help them understand the importance of introspection.

The Vote activity ensures that everyone in a group has a say in decisions. It enables teachers to quickly take feedback from students. Voting can be used to create quick quizzes and it also acts as a conflict resolution technique in large groups.

Availability during GSoC Project

My semester was expected to end on **May 27** but due to the COVID-19 pandemic the timeline may be volatile. Considering May 27 as the end, I will need a break for 9 days which I will cover up by putting in extra work hours everyday after my exams.

I will devote at least 40-50 hours a week on my project and can extend my involvement if the need persists. My aim is to stick to the timeline and I'm confident that the project will be completed in a timely manner.

I will be committed to this project throughout my GSoC tenure as I have no other internships or projects to be distracted by.

Timeline

Time Period	Task
28 April - 18 May	Exploring various technologies and libraries. Discussing the implementation approaches with mentors and the community. Development environment setup and structuring the project.
18 May - 27 May	Break for semester examinations. Staying in touch with mentors to finalize the feature list.
27 May - 2 June	Start creating the Curriculum activity. Finalize UI design, icons and colours. Implement basic workflow. Create Student component and all its underlying components Store and retrieve context from the Journal.
2 June - 8 June	Add create/update/delete/sort for categories and skills. Integrate uploading image/audio from the Journal and saving them in required skills. Add the uploaded documents list in the side panel.
8 June - 14 June	Linking the documents in the side panel to the skill it was uploaded for. Create the Generate component with the design template of the exported file. Adding Presence support to connect with other users.
15 June - 19 June	**Phase 1 Evaluations** (Progress: Functional activity with categories/skills addition, image/audio uploads

	from Journal, displaying & linking uploaded documents, basic generate design and Presence)
20 June - 26 June	Finalize the export feature. Adding rewards for various skills levels. Add a tutorial for the activity. Make the required changes after evaluation. Test activity on different platforms. Start working on the Vote activity. Finalizing the Presence actions through discussions with mentors.
27 June - 3 July	Create the Home screen with poll cards. Create Settings mode to add/edit/delete polls. Add form UI to enter information. Add Journal support to store polls.
4 July - 12 July	Add multimedia element uploads from Journal. Initialize Presence module. Add all network actions to share the activity. Create the Voting component.
13 July - 17 July	**Phase 2 Evaluations** (Progress: Home screen, Settings mode, Journal support, Multimedia elements upload, running polls on the network)
18 July - 24 July	Create Stats component. Add graphs using Chart.js and lists. Add option to show real-time stats to all users. Add export to CSV feature.
25 July - 31 July	Finalize the export implementations. Discuss with mentors and add functionality to other types of graphs and stats.

1 Aug - 9 Aug	Add a tutorial to the activity. Fix any bugs in both activities and perform various types of unit and integration testing.
10 Aug - 17 Aug	Buffer week to enhance the look and feel of the activities along with beta testing.
17 Aug - 24 Aug	**Final Evaluations** (Progress: Statistics for voting, real-time options, export to CSV, both activities completed)

Plans post GSoC period

After my tenure at GSoC, I will continue to maintain the activities and fix any bugs that show up. I would also continue to work as an active community member and help new developers and contributors get familiar with the platform. I hope to keep working towards the growth of this organisation.

At last, I should mention that it has been a great learning experience for me while contributing to Sugar Labs and the healthy community really helped me work my way through the platform to utilize my skills. I am very enthusiastic towards working with Sugar Labs in the Google Summer of Code 2020 and make contributions to help the community even further. I am determined to make this project successful.

Thank You,
Dhruv Misra
Undergraduate, Third Year
Guru Gobind Singh Indraprastha University, New Delhi