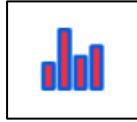




sugarlabs



Sugarizer Word Puzzle and Chart activities

Google Summer of Code 2023 – Project Proposal

About Me

What is your name?

My name is **Vedant Sharma**, I am a 3rd year computer science undergraduate student at Galgotias University, Greater Noida India.

Emails and Contacts:

Primary Email: vedants2003@gmail.com

Secondary Email: 11vedantsharma@gmail.com

Github: [VedantSharma11](https://github.com/VedantSharma11)

Linkedin: [vedantsharma13](https://www.linkedin.com/in/vedantsharma13)

Matrix IRC: VedantSharma11

Phone: (+91) 8742967471

(I will be reachable anytime through my Mobile No. and Email)

What is your first language? (We have mentors who speak multiple languages and can match you with one of them if you'd prefer.)

My first language is Hindi but I am proficient in speaking, reading, writing, and understanding English, I can effectively communicate and express my ideas with clarity and precision in English.

Where are you located, and what hours (UTC) do you tend to work? (We also try to match mentors by general time zone if possible.)

I am located in New Delhi, India and my time zone is Indian Standard Time (UTC + 5:30). I am planning to work from 6:00 to 12:00 (UTC) but my timings are flexible. I'm very excited to work on this project during the summer and I can surely manage my time and be active when the mentors are available.

Have you participated in an open-source project before? If so, please send us URLs to your profile pages for those projects or some other demonstration of the work that you have done in open-source. If not, why do you want to work on an open-source project this summer?

I have experience in Open Source. I love to build projects in collaboration with developers and the community. I am actively contributing to open-source projects. Notably I recently took part in Hacktoberfest, where I was able to make valuable contributions. You can see them here:

- <https://github.com/Zack-Dx/Twitter-Landing-Page/pull/50>
- <https://github.com/K0DEL/HacktoberFest-2022/pull/14>
- <https://github.com/pranjay-poddar/Dev-Geeks/pull/647>
- <https://github.com/pranjay-poddar/Dev-Geeks/pull/713>

Details of my other contributions can be found on my Github profile [here](#).

I have been contributing to **Sugar Labs** for the past month. During this time, I have contributed to the **Sugarizer** repository and fixed bugs, UI changes, enhancements, and ported libraries & packages. This last month was a great learning experience. I contributed to **Sugar Labs** by finding and fixing some issues as follows:

These are all contributions from my side on **sugarizer**:

Pull request link	Description	Status
#1242	Ported Dollar activity from Bootstrap tour to IntroJS tour	Merged
#1243	Ported Flip activity from Bootstrap tour to IntroJS tour	Merged
#1327	Mapped keys for changing pages in Video-viewer activity	Merged
#1307	Fixed entire Introjs tour's bugged UI in Measure activity	Merged
#1247	Fixed Falbracman activity's intro popup Header text & cross button UI issue	Merged
#1300	Fixed activity dropdown's UI in stopwatch activity	Merged
#1310	Excluded bootstrap tour css file in curriculum activity to remove bloat from the codebase	Merged
#1311	Refactored the codebase by removing redundant jQuery file to optimize the codebase	Merged

I have made over [28 commits](#) till now in Sugar Labs. All of these merged commits and issues can be found below

You can view all my open and closed PR's at: [Sugarizer PR](#)

You can view all my open and closed issues at: [Sugarizer Issues](#)

I'm an open-source enthusiast with multiple projects on my GitHub profile. Contributing to open source has helped me learn and grow, and I am excited to apply for the Google Summer of Code program to further

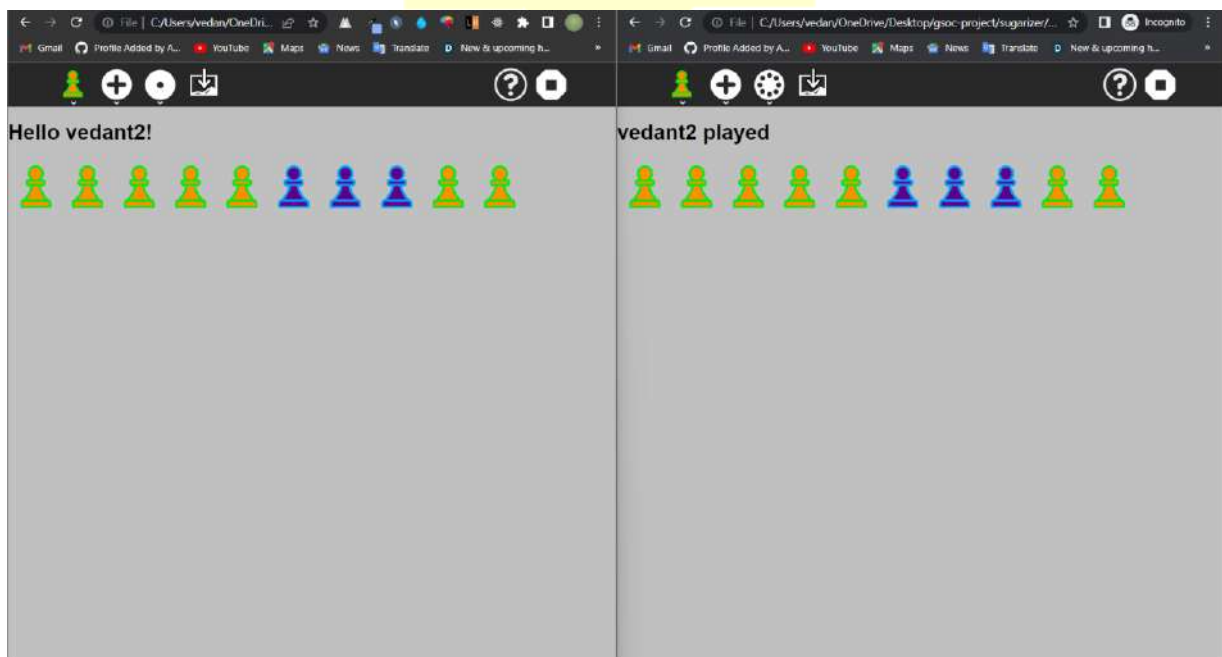
expand my skills and knowledge while also contributing to impactful open-source projects.

Prerequisites for project

As given in the idealist. I already have good working knowledge in HTML5, Javascript, ReactJs, VueJs framework development. Other than this I am already familiar with the Sugarizer codebase and have merged commits into the repository.

I also have completed both **Sugarizer Vue.js activity development tutorial** and **Sugarizer Vanilla JS activity development tutorial**. This helped me understand how the Sugarizer core works and how activities are implemented from scratch. Following are the links to my repository of the implemented tutorials.

- <https://github.com/VedantSharma11/Pawn.activity-VanillaJS>
- <https://github.com/VedantSharma11/Pawn.activity-VueJS>



Learning about the Sugarizer codebase and how to work with Sugar Labs helps me understand how the system works and how different parts of it are connected. This understanding makes it easier for me to see how the system functions and relies on each of its components.

About The Project

What is the name of your project?

The name of my project is **Sugarizer Word Puzzle and Chart activities**.

Describe your project in 10-20 sentences. What are you making? Who are you making it for, and why do they need it? What technologies (programming languages, etc.) will you be using?

Sugarizer is a valuable educational tool. It can work on the web as a webapp and also be used as a mobile app, thus giving Sugar Users a wide range of options. However, there is currently no activity in Sugarizer that provides users or students with a way to learn and understand data visualization. The proposed project aims to integrate a **Chart activity** similar to the Sugar chart activity into Sugarizer activities. The project also aims to include a **Word puzzle** template in the Exerciser activity. The project will utilize technologies such as HTML5, JavaScript, ReactJS, and VueJS.

Project Objective:

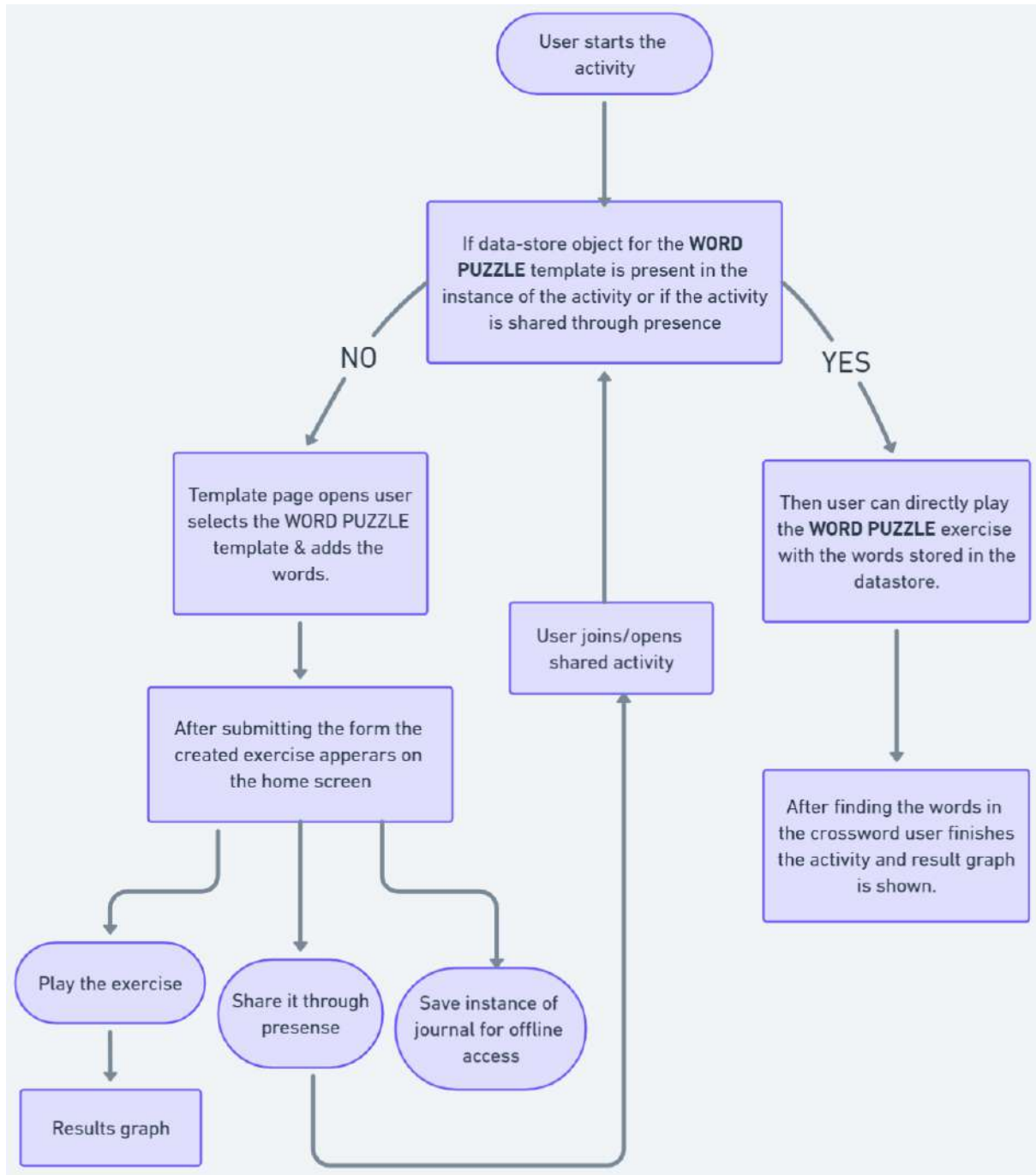
The objective of this project is to develop a chart activity that resembles the Sugar Chart activity and can be used by students/users to gain a better understanding of data visualization. Additionally, a word-puzzle template will be included in the Exerciser Activity, which is an educational activity used by educators to create curriculum-based exercises for their students.

Project Goals:

The goal of this project are:

- To implement a Chart activity in Vue.js inside sugarizer that resembles the sugar chart activity.
- To implement a Word puzzle template within the Exerciser activity.

WORD PUZZLE TEMPLATE IMPLEMENTATION



Explanation:

Once the user opens a new instance of the exerciser activity he would see the default **Word-Puzzle** exercise listed on the screen if its present in the datastore, he can either play it from there or he can choose to make a new one by going to editor button and then add exercise this will open all the templates there are along with the **word-puzzle template**.

After choosing the **word-Puzzle template**, a form will open, the user fills in the words for the word-puzzle, user will have option to finish or test the exercise or to add more words.

After adding all the words the user clicks on the **finish exercise** and gets redirected to the screen where other exercises are listed along with the new word-puzzle exercise he just created. From here user can either play the activity or share it on network.

Modifying 'template.js' to add new word puzzle template:

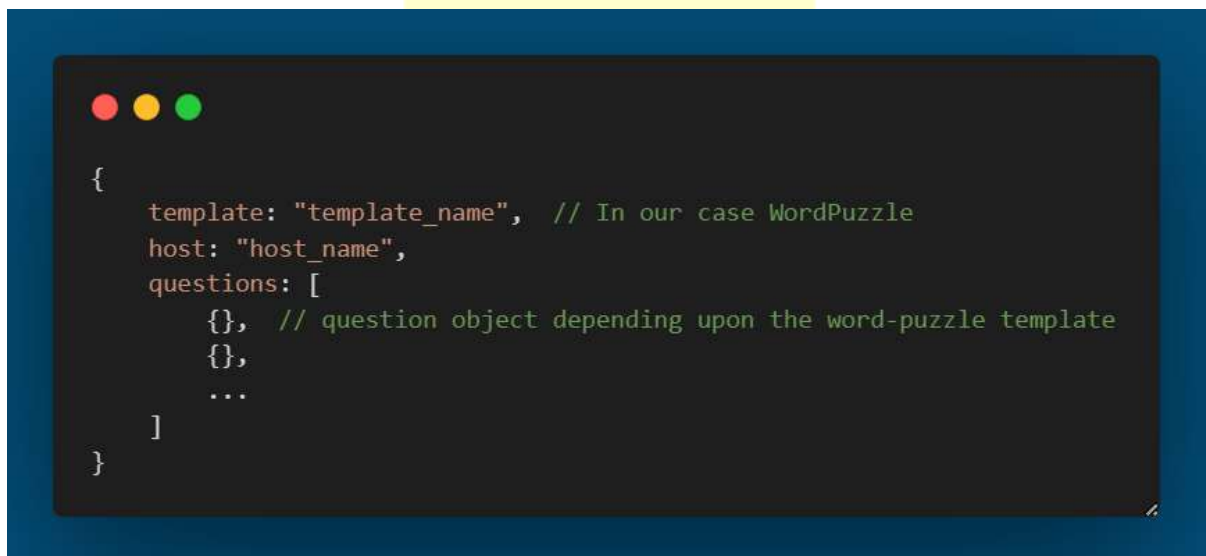
```
1 <div onClick={wordPuzzleSelected.bind(null, props.history)}>
2   // Rest of the code comes here
3
4   <button onClick={wordPuzzleSelected.bind(null, props.history)}>
5     <FormattedMessage id={CHOOSE} />
6   </button>
7 </div>
8
```

- This will add a new word- puzzle card on the add exercise page from where the user can click on it and open the **word-puzzle form** to generate the exercise.

Creating a new file 'WordPuzzleForm.js' inside (src/containers/Builders) :

- A new 'WordPuzzleForm.js' file will be created including all the features and implementations for the word-Puzzle form.

Datastore object for all templates:



```
{
  template: "template_name", // In our case WordPuzzle
  host: "host_name",
  questions: [
    {}, // question object depending upon the word-puzzle template
    {},
    ...
  ]
}
```

- Sharing:

The object with words array in the above datastore object is shared whenever the user will share the word-Puzzle exercise. The object is sent from the **host** to all other users on the network using **presence**.

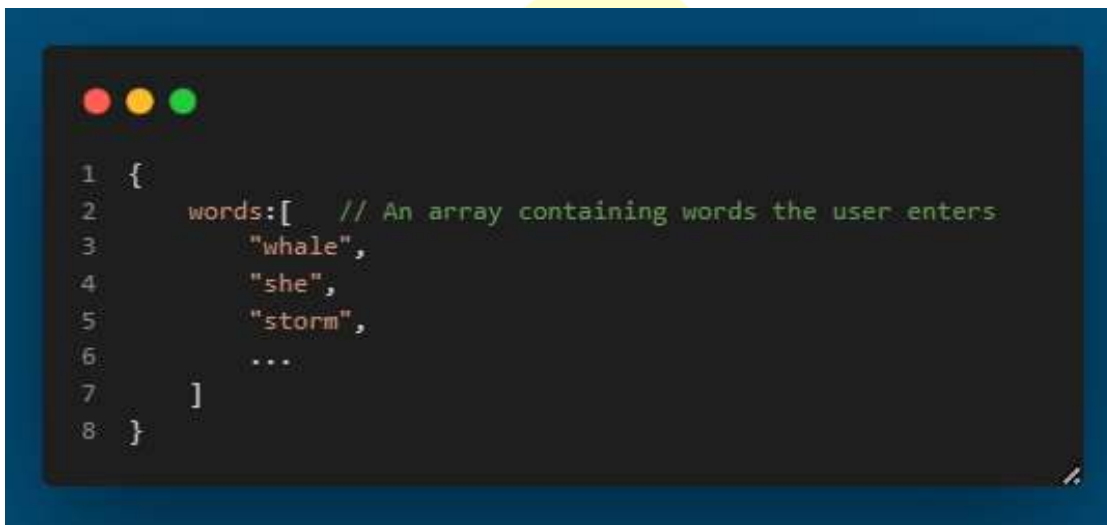
- Locally:

Once the user builds the exercise, the object with words array is stored in the datastore. If the instance of the activity is copied locally through the journal. Then, when a user opens the activity the datastore object will contain the questions array and hence the

Word-Puzzle exercise will show on the home page of listed exercises.

Word-Puzzle Template:

For the word-puzzle template, the question object for the word-puzzle template will contain an array containing all the words that the user enters.



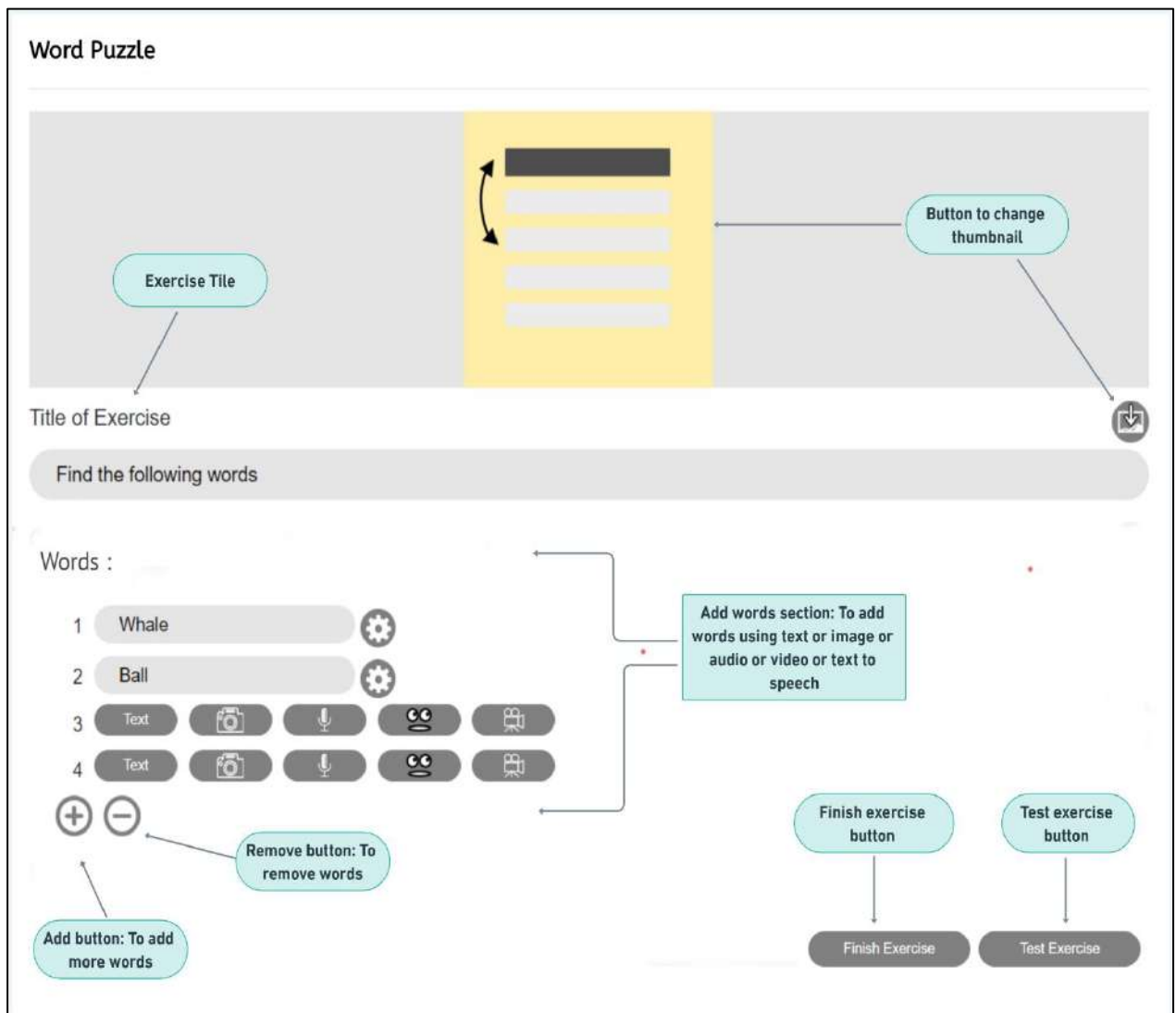
```
1 {
2   words: [ // An array containing words the user enters
3     "whale",
4     "she",
5     "storm",
6     ...
7   ]
8 }
```

This object containing words array will then get stored in the datastore which will afterwards be used for generating the crossword grid. More features like **Hints** for each word can be added to the object later.

The Word-Puzzle form will look something like this:

- The word-puzzle form will allow users to create the exercise it will have a mandatory title field for the title of the exercise.
- Then there will be a Add-words section to add the words for generating the word-puzzle grid, users can add or remove words by using the '+' or '-' buttons.
- Users can add the words using different MULTIMEDIA types like text, audio, image, video, text to speech using '**utils.js**'.

- It will have 2 buttons 'test exercise' & 'finish exercise' the test button will allow users to test the exercise before submitting/generating it. Finally the finish button will generate the exercise and add it to the exercise List on the home screen and the data will be stored in the datastore.



Form options:

1) Title field:

The user will be able to add title for the exercise here. The title field will be associated with a **onChange** attribute that will be linked to a function. The function will check if the title field is empty or not if its empty then user will not be able to submit the form.

```
1 ChangeTitle = (e) => {
2   if (e.target.value === "") {
3     // form cannot be submitted
4   }
5   this.setState(
6     // rest code will go here
7   );
8 };
```

2) Remove word:

This will only remove a word from the words array if the current length of the array is greater than one. This is to ensure that at least **one word** is available for generating the word-puzzle. The button will be assigned an onClick attribute associated with a removeOption function.

```
1 RemoveOption = () => {
2   let { words } = this.state
3   if (words.length > 1) {
4     words.pop();
5     this.setState(
6       // rest code will go here
7     );
8   }
9 };
```

3) Add new word:

This will enable users to include additional words in the word-puzzle, it allows the addition of new words into the words array for the word-puzzle template. An onClick attribute associated with the AddOption function is assigned to the button.

```
1 AddOption = () => {
2   const { currentword } = this.state;
3   this.setState(
4     {
5       // Adding the word .
6     },
7   );
8 };
9
```

4) Finish/ Test Exercise:

The finish exercise button will submit the form and save the data in the datastore while performing validations on the form. After saving the form data the user gets redirected to the home screen from where he can play the exercise, while test exercise button saves the form data locally to allow users to test the exercise before submitting it.

```
1 FinishExercise = () => {
2   const { title, wordArr } = this.state;
3
4   // Save the form data in the datastore
5   saveToDatastore({
6     title: title,
7     words: wordArr
8   });
9
10  // Redirect to the home page where the exercise
11  //will be displayed
12  };

```

This is just a rough implementation and not a rigid one. The functions and implementations may change once I start working on the project and start developing the 'wordpuzzleform.js' .

Creating a new file 'WordPuzzlePlayer.js' inside (src/containers/Players):

- Once the user fills the words and details in the word-puzzle form, which gets saved in the word-puzzle template in the datastore and the user gets redirected to the listed exercise section. To enable the user to play the exercise, the '**WordpuzzlePlayer.js**' will be utilized for implementation.
- This file will also contain the logic necessary to generate the crossword grid for the word-puzzle exercise using the array of words that the user entered in the form. The player will be able to interact with the puzzle, finding the words and seeing if they are correct or not.

Implementation of Crossword Grid:

- Determine the dimensions of the grid:
The first step will be to determine the dimensions of the grid based on the length of the longest word in the array. For example, if the longest word is 10 letters long, we will create a 12 x 12 grid to provide some extra space.

```
1 function generateCrossword(words) {
2   let longestWordLength = findLongestWord(words);
3   let gridLength = longestWordLength + 2;
4   let gridWidth = longestWordLength + 2;
5
6   // initializing the grid
7
8   //... Rest code will go here
9 }
```

- Initialize the grid:

Once you know the dimensions of the grid, you can initialize it as an array of arrays using a loop. For example, if your grid is 12 x 12, we might initialize it like this:

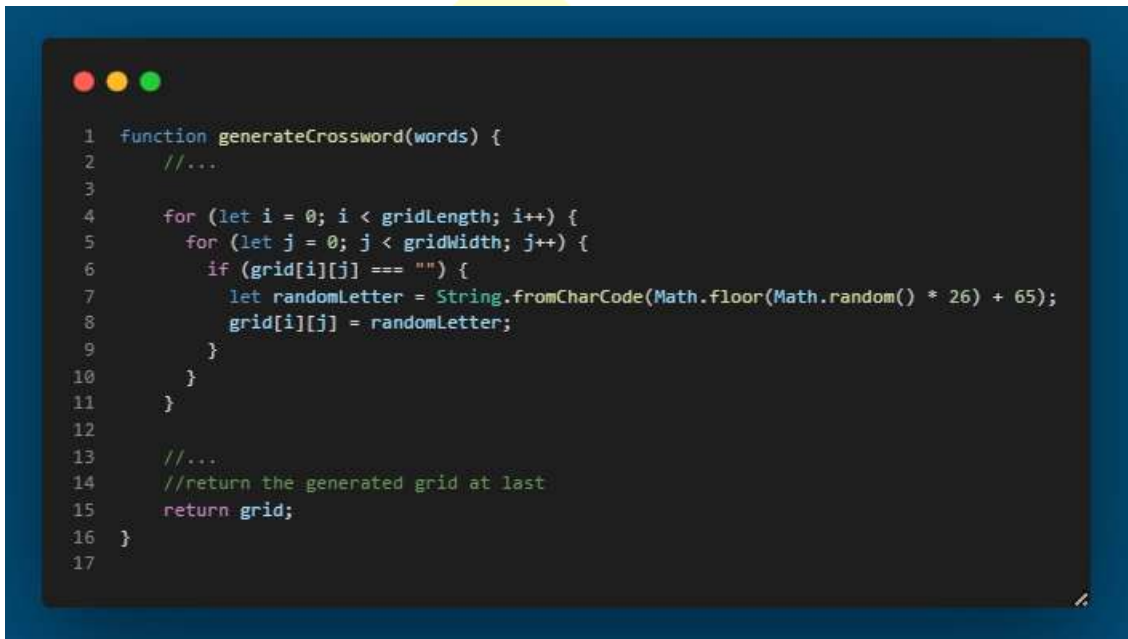
```
1 // initializing the grid
2 let grid = [];
3 for (let i = 0; i < gridLength; i++) {
4   grid[i] = new Array(gridWidth).fill("");
5 }
6
7 PlaceWordsInGrid(words, gridLength, gridWidth);
8 // Call function to place words in the grid
9 //... Rest code will go here
```

- Place the words in the grid:

Now that you have a blank grid, you can start placing the words in it. Begin by selecting a word from the array at random. Then, choose a random starting position for the word on the grid. You can choose whether the word will be placed horizontally or vertically at random as well. Once you have determined the starting position and orientation of the word, you can place it in the grid using a loop.

```
1
2 function PlaceWordsInGrid(words, gridLength, gridWidth) {
3   for (let i = 0; i < words.length; i++) {
4     const wordLength = words[i].length;
5     const x = Math.floor(Math.random() * (gridLength - wordLength - 2)) + 1;
6     const y = Math.floor(Math.random() * (gridWidth - wordLength - 2)) + 1;
7     const orientation = Math.floor(Math.random() * 2);
8
9     // Placing the word in the loop horizontally or vertically
10    // depending on the orientation
11  }
12 }
```

- Fill in the empty spaces:
After placing the words in the grid, there will be some empty spaces left. You can fill these spaces with random letters to make the puzzle more challenging.
- Return the completed grid:
Once the grid is complete, you can return it to be displayed to the user.

A screenshot of a code editor with a dark background and blue border. The code is written in JavaScript and defines a function 'generateCrossword' that iterates through a grid and fills empty spaces with random letters. The code is as follows:

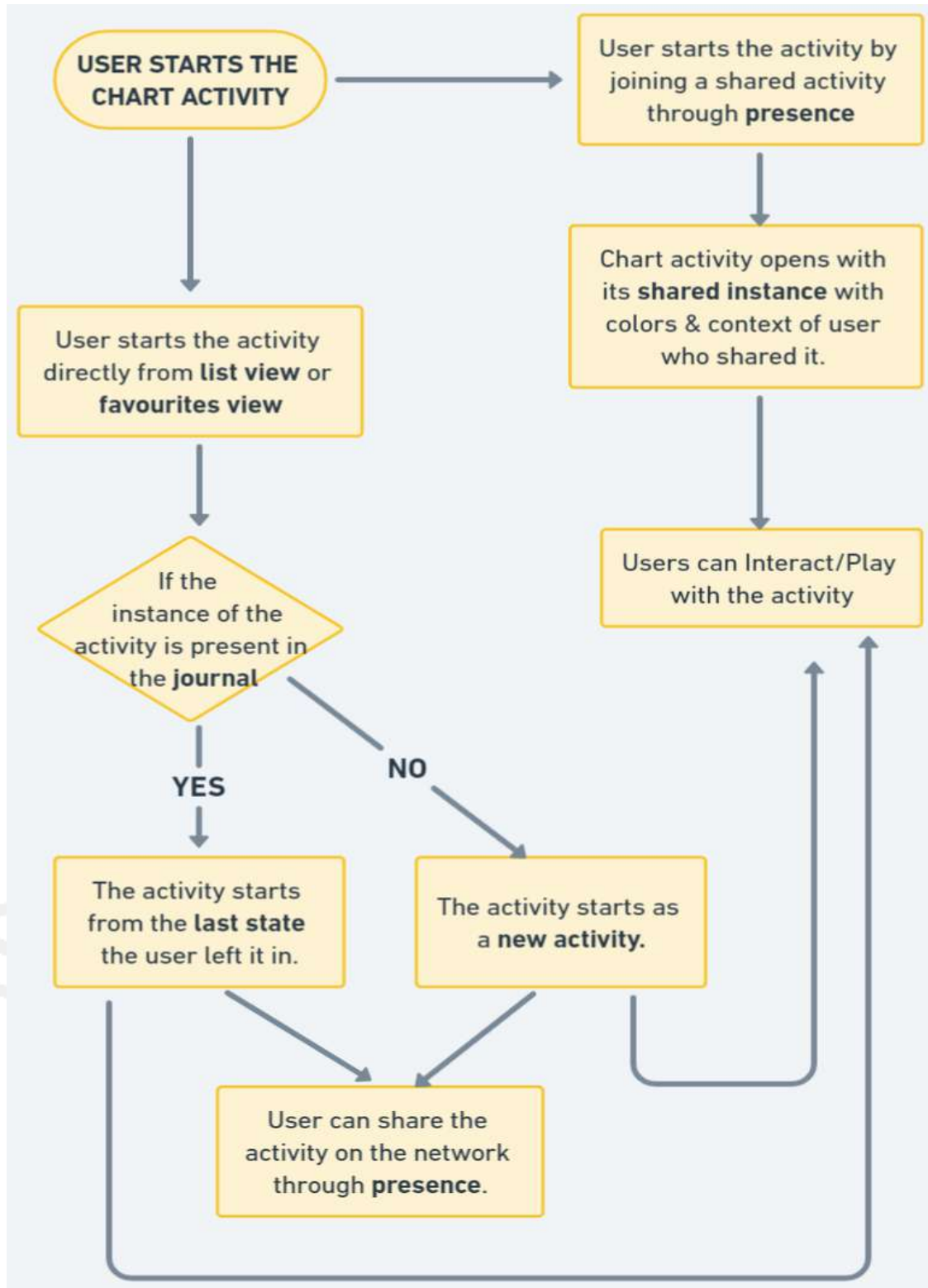
```
1 function generateCrossword(words) {
2   //...
3
4   for (let i = 0; i < gridLength; i++) {
5     for (let j = 0; j < gridWidth; j++) {
6       if (grid[i][j] === "") {
7         let randomLetter = String.fromCharCode(Math.floor(Math.random() * 26) + 65);
8         grid[i][j] = randomLetter;
9       }
10    }
11  }
12
13  //...
14  //return the generated grid at last
15  return grid;
16 }
17
```

Results after submitting the exercise:

- Once the user submits the exercise his results will be shown using customized charts displaying users score in percentage, his/her time in minutes and finally a details section to show how many words the used found in the grid. For displaying the charts we will use '**react-chartjs-2**' library which is already being used for all other exercises in the activity.

Again This is just a rough implementation and not a rigid one. The functions and implementations may change once I start working on the project and start developing the 'WordPuzzlePlayer.js' .

CHART ACTIVITY IMPLEMENTATION



Explanation:

When starting the activity:

- The user has two options to open the activity: from the list / favourites view or from a shared instance of the activity on the neighbourhood view.
- If the user joins /opens a **shared instance**, the activity will start with the colours and context of the user who shared it on network.
- If the user opens the activity from the list/favourites view, two scenarios are possible:
 1. If the instance of the activity is already present in the **journal** (which is stored in the datastore), the activity will resume from the last state the user left it in.
 2. If the instance of the activity is not present in the **journal**, the activity will start as a new activity.

After starting the activity:

- The user can interact with the activity, using the various features and functions within it.
- Alternatively, the user can share the activity on the network, using **presence** to let others join.
- When the user shares the activity, others can join in and interact with it, while also bringing their own colours and context to the activity.

Summer of Code

This is how I will try to implement the chart activity:



Sugarizer already comes with a template for the Vue.js activities that includes all the sugar-web components for implementing standard sugarizer features:

- SugarToolbar.js
- SugarJournal.js
- SugarIcon.js
- SugarPresence.js
- SugarPopup.js
- SugarActivity.js
- SugarL10n.js

And some other components, using all these components we are going to implement the chart activity in sugarizer.

Before using any resources for the activity, we will add the **sugar-activity component** with **v-on: initialized directive** bind-ed with the initialized: function() defined in methods in the Vue app.

```

// add the component in index.html
<sugar-activity ref="SugarActivity" v-on:initialized="initialized"></sugar-activity>

//activity.js
...
methods: {
  initialized: function () {
    // Sugarizer initialized
    var environment = this.$refs.SugarActivity.getEnvironment();
    ...
  }
}
...

```

Activity toolbar:

- The toolbar for chart activity will have various buttons for implementing different functionalities, such as selecting the chart type, Adding or removing the data, editing the axis labels, exporting the chart as image, and more.
- This can easily be done by using the **SugarToolbar.js component** which sets up the toolbar at the top of the screen. This component basically defines 2 components: **sugar-toolbar** and **sugar-toolitem** (buttons).
- The component can be added to the html file and we can add all the the buttons here:

```

1 <sugar-toolbar ref="SugarToolbar">
2   <sugar-toolitem id="activity-button"></sugar-toolitem>
3
4   <!-- Add more buttons here -->
5   <sugar-toolitem id="add-button" title="Add label"></sugar-toolitem>
6
7   <!-- Toolitems with class="pull-right" will be right aligned -->
8   <sugar-toolitem id="stop-button" title="Stop" class="pull-right"></sugar-toolitem>
9 </sugar-toolbar>

```

1. Add and Remove button:

- The add button will enable users to include new labels along with their values to be displayed on the charts on the screen whereas the remove button will remove the selected label and its value (key:value pair) from the **List object** that will be defined for the labels and values which I will be explaining later.
- The button will be assigned with the **v-on:click directives** to trigger specific functions to handle the events.



```

methods: {
  ...
  onAddClick: function () {
    const labelInput = document.getElementById('label');
    const valueInput = document.getElementById('value');
    const label = labelInput.value;
    const value = parseFloat(valueInput.value);
    // pushing the pair in the array
    data.push({ label: label, value: value });
  },

  onRemoveClick: function () {
    const label = //getting hold of the selected label
    //looping in the list array to find the label to remove it
    for (let i = 0; i < data.length; i++) {
      if (data[i].label === label) {
        data.splice(i, 1);
        break;
      }
    }
  },
}

```

2. Buttons to change the chart type:

- These four buttons will enable the user to change the chart type to a vertical bar chart, horizontal bar chart, line chart or a pie chart.

- To implement this I will create a separate chart component using an external library **chart.js** that takes the **type of chart** and the **array of objects** in which the label: value pairs are stored as **Props**, and depending upon which button is pressed the props will be passed and the charts will get changed.


```

1  var chart={
2    <template>
3    <div> <canvas ref="chartCanvas"></canvas> </div>
4    </template>,
5    props: {
6      chartType: String, // chart type as prop
7      chartData: Array // data array as prop
8    },
9    mounted() {
10     const ctx = this.$refs.chartCanvas.getContext('2d');
11     this.chart = new Chart(ctx, {
12       type: this.chartType, // set chart type from prop
13       data: {
14         //pass the data
15       }
16     });
17   },
18   watch: {
19     chartData() {
20       // update chart data when chartData prop changes
21       ...
22       this.chart.update();
23     }
24   }
25 }

```

Summer of Code

3. Customization buttons with custom palettes:

- To create custom palettes sugar-web provides the **palette** library, which is the base class for all palette objects.
- To create a custom palette for the 2 buttons in the chart activity . We will have to create a palette directory in **src** directory of the project.

- There we will add 4 files which will be **Toolbuttonpalette.js**, **Toolbuttonpalette.html** and similarly **Textbuttonpalette.js**, **Textbuttonpalette.html**.
- Now to the both the buttons in the sugar-toolitem component we will specify some attributes such as **palette-file**, **palette-class**, **palette-title**.
- Both the javaScript files of palettes will include the custom functionalities of the palettes and the html files will define the Ui of the palettes.



```
define(["sugar-web/graphics/palette",
"text!activity/palettes/Toolbuttonpalette.html"], function(palette,
template){
  var Toolbuttonpalette= {};
  // the code for generating custom palette functionality will come here.
  ...
  return Toolbuttonpalette;
}
```

- To implement palettes I will refer to the below documentation.
- <https://github.com/llaske/sugarizer/blob/dev/docs/tutorial/VueJS/step8.md>

4. Stop, Tutorial, Fullscreen buttons:

- The "stop" button will be used to stop the activity and at the same time saving its instance in the journal.
- The "Tutorial" button will start a tutorial that provides a detailed explanation of how to use the activity. I will explain the implementation of tutorial later. Additionally, there is a "Full Screen" button that allows the user to maximize the chart canvas and view it on the entire screen.

Journalizing the Chart Activity:

- Journal is used for **saving the instance of the activity** so that whenever the user opens the activity again it starts from the exact state the user left it in.
- To use the journal, Sugar-web provides **sugar-journal component** that needs to be added to the html file inside app element.



```
1 <div id="app">
2   ....
3   ....
4   // adding sugar journal component
5   <sugar-journal ref="SugarJournal"></sugar-journal>
6 </div>
```

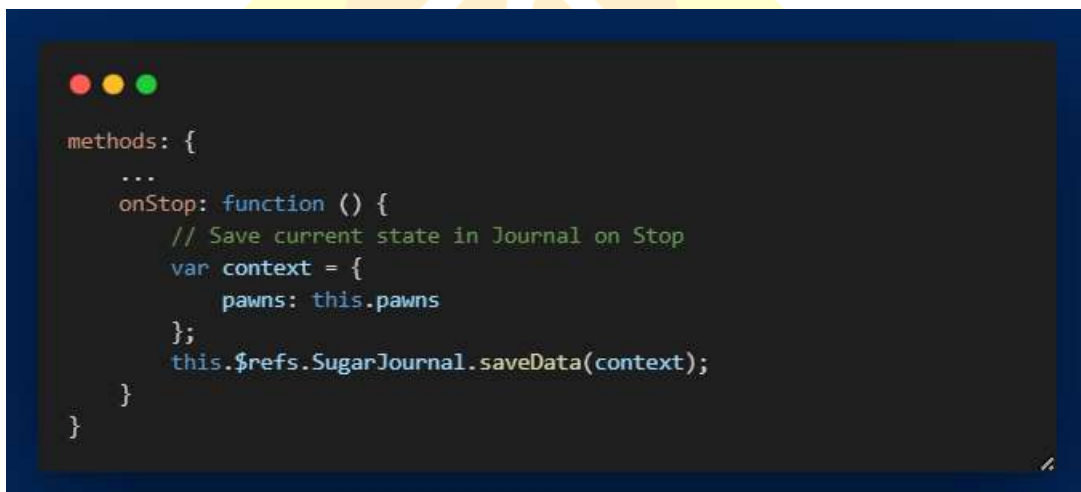
Store context in datastore:

- To store the **context**, we have to handle the **datastore**. The datastore is the place where Sugar stores the Journal. During the activity setup, Sugar-Web automatically initializes the datastore for the activity.
- The **context** in the case of **Chart activity** is the **List of all the label and values** that the user enters , so that whenever the user opens the activity again it opens with similar context that the user left it in.



```
1 const List = [
2   { label: 'January', value: 100 },
3   { label: 'February', value: 150 },
4   ...
5 ];
6
```

- The **saveData()** method allows you to store a JavaScript object into the datastore which can be retrieved later. The data is converted to JSON automatically and stored as a text string.
- The SugarJournal component automatically retrieves the data inside Journal (if any) for the current activity and emits an event **journal-data-loaded**. Then we can handle this event to show the data when the activity is loaded. The data is parsed from the JSON format and returned as a JavaScript object.
- So to save the context we will add a **v-on:click** directive on the stop button so whenever the user closes the activity the current state of the activity will get saved in the journal.



```
methods: {  
  ...  
  onStop: function () {  
    // Save current state in Journal on Stop  
    var context = {  
      pawns: this.pawns  
    };  
    this.$refs.SugarJournal.saveData(context);  
  }  
}
```

Load the saved instance when user starts the activity:

- When the user starts the activity The **Sugar-Journal** component automatically retrieves the data inside Journal (if any) for the current activity and emits an event **journal-data-loaded**.
- To handle this event we need to add a **v-on:journal-data-loaded="onJournalDataLoaded"** directive on the Sugar-Journal component and handle it using a function.


```
methods: {
  ...
  onJournalDataLoaded: function(data, metadata) {
    console.log("Existing instance");
    this.List = data.Lists;
  },
}
```

Share Activity Feature using Presence in Chart Activity:

- Presence is a library provided by sugar-web that enable users to interact with each other on same activity. Using **presence** users can share instances of their activity on the network and can join other users shared activity's instance.
- We will start by including the network button in the toolbar and setting up its palette.
- The **SugarPresence component** handles the integration of the network-button palette. We just need to add these 2 attributes to the component **palette-file="sugar-web/graphics/presencepalette"** and **palette-class="PresencePalette"**.

```
1 // add the component in index.html
2 <div id="app">
3   // add the network-button
4   <sugar-toolitem
5     id="network-button"
6     title="Network"
7     palette-file="sugar-web/graphics/presencepalette"
8     palette-class="PresencePalette"
9   ></sugar-toolitem>
10 ...
11 <sugar-presence ref="SugarPresence"></sugar-presence>
12 ...
13 </div>
```

Share the Chart Activity instance:

- We will keep a data variable as a reference to the component instance as it will be used multiple times and we will add a `mounted: function ()` in the Vue main app.

```

1 data: {
2   SugarPresence: null,
3   ...
4 },
5 mounted: function () {
6   this.SugarPresence = this.$refs.SugarPresence;
7   ...
8 },

```

- Now we will update the `sugar-toolitem` for `network-button` to integrate presence we will first handle the click on the share button for this we will need to add few more attributes to `network-button` `sugar-toolitem` **`palette-event="shared"`** , **`v-if="SugarPresence"`**, **`v-on:shared="SugarPresence.onShared"`**.

```

1 <sugar-toolitem
2   id="network-button"
3   title="Network"
4   palette-file="sugar-web/graphics/presencepalette"
5   palette-class="PresencePalette"
6   palette-event="shared"
7   v-on:shared="SugarPresence.onShared"
8   v-if="SugarPresence"
9 ></sugar-toolitem>

```

- Now when user clicks on the share button, the shared event defined by **`palette-event`** gets triggered & the activity will get shared and will be visible to others on the neighbourhood view. This will be handled by the **`onShared()`** method of the component.
- The **`v-if`** directive will render this button only if `SugarPresence` component exists.

Now to handle the event when a user joins a shared activity I will use 2 events provided by the **sugar-presence component**.

- data-received
- user-changed

```
methods:{
  ...
  onNetworkDataReceived: function(msg){
    //handling data recieved event
  },
  onNetworkUserChanged: function(msg){
    //handling user changed event
  }
}
```

For rest of the implementation, I will refer to this resource:

<https://github.com/llaske/sugarizer/blob/dev/docs/tutorial/VueJS/step6.md>

Implementation of Export chart as image feature:

- In order to export/save charts as image I will add a button in the toolbar with a **v-on:click** directive that will trigger a function to save the chart as image in the journal

```
1 <sugar-toolitem
2   id="capture-image-button"
3   v-bind:title="l10n.stringCaptureImage"
4   v-on:click="captureImage"
5 ></sugar-toolitem>
```

- The capture-image function() binded to the sugar-toolitem will save image in the journal and create its entry with its mimetype, title, timestamp, creation type etc.

```

1  captureImage: function() {
2    var mimetype = 'image/jpeg';
3    var inputData = this.canvas.toDataURL(mimetype, 1);
4    var metadata = {
5      //defines the details of the image
6    };
7    var vm = this;
8    vm.$root.$refs.SugarJournal.createEntry(inputData, metadata)
9      .then(() => {
10     //save and creates the entry of image in journal
11     });
12 }

```

Localization of the Chart Activity:

- Localizing the chart activity will allow the activity to run in various languages.
- To localize the activity we will yet again use another sugar-web component called **sugarL110n**.
- To localize all the strings in the activity we will modify the **locale.ini** file where we will write the translations for all the strings in different languages.

```

1  //adding inside app element
2  <sugar-localization ref="SugarL10n"></sugar-localization>
3
4  //locale.ini file structure
5  [*]
6  ...
7  Hello=Hello {{name}}!
8
9  [en]
10 ...
11
12 [fr]
13 ...
14
15 [es]
16 ...

```

Integrating tutorial:

- To implement the tutorial tour in the activity we will use the `intro.js` library along with the **sugar-tutorial** component provided by `sugar-web`.
- We will add a button using the `sugar-toolitem` component and then assign **v-on:click** directive to it which triggers a **onHelp** function.



```

1  onHelp: function () {
2    var steps = [
3      {},
4      {},
5      .
6      .
7    ];
8    this.$refs.SugarTutorial.show(steps);
9  },

```

- The function contains a array of objects that gets passed on to the **show()** method of the `sugar-tutorial` component.
- Localization for the tutorial is done in the **locale.ini** file. I will refer to this documentation for integrating the tutorial.
<https://github.com/llaske/sugarizer/blob/dev/docs/tutorial/VueJS/step9.md>

This is just my rough implementation for the chart activity it may change when I actually start implementing the activity according to the project needs.

Dependencies:

- **Bootstrap:** Library to help build responsive design. The UI will be capable to running on desktops to phones.
<https://getbootstrap.com/>
- **React-chartjs-2(for word-puzzle template results):** Library will be used to show results using charts so that the student can visualize his results. <https://react-chartjs-2.js.org/>

- **Vue-Chart.js (for chart activity):** Library to integrate different types of charts in the activity with custom data. <https://vue-chartjs.org/>

Deliverables:

By the end of my GSoC period I plan to deliver the following features in Sugarizer.

- Implementation of word-puzzle template in the exerciser activity
- Implementation of new chart activity in Vue.js with following features:
 1. Localization
 2. Integration with journal
 3. Share activity feature using presence
 4. Export chart as image functionality
 5. Responsiveness
 6. Intro.js tutorial integration

What is the timeline for development of your project? The Summer of Code work period is from mid-May to mid-August; tell us what you will be working on each week. (As the summer goes on, you and your mentor will adjust your schedule, but it's good to have a plan at the beginning so you have an idea of where you're headed.) Note that you should probably plan to have something "working and 90% done" by the midterm evaluation (end of June); the last steps always take longer than you think, and we will consider cancelling projects which are not mostly working by then.

This is the timeline I plan to follow:

Time Period	Plan
April 4 – May 4, 2023	<ul style="list-style-type: none">• Application review period

Week One/Two	<ul style="list-style-type: none"> • Explore more about sugarizer and its architecture. • Learn some new technologies, in which sugarizer activities are built.
Week Three/Four	<ul style="list-style-type: none"> • Gain good understanding of require.js and vue.js frameworks • Will learn how sugar components work.
May 4 – May 28, 2023	<ul style="list-style-type: none"> • Community bonding period • Discuss the implementation of the activities with mentor. • Go through the documentation of vue.js to understand code integration • explore other tech stack required for the project.
May 29, 2023	<ul style="list-style-type: none"> • Coding period starts
Week one & two (May 28 – June 11, 2023)	<ul style="list-style-type: none"> • Setting up the environment for adding word-puzzle template and then build some UI. • Start building the Word-puzzle form.
Week three & four (June 12 – June 25, 2023)	<ul style="list-style-type: none"> • Start implementing the play functionality for the exercise. • Implement the edit and delete exercise feature.
Week five & six (June 26 – July 9, 2023)	<ul style="list-style-type: none"> • Implement the crossword grid logic to generate grid. • Implement the results section using react-chartjs-2 library. • Implement the share exercise functionality.
July 10 – July 14, 2023	<ul style="list-style-type: none"> • Midterm evaluation

Week seven & eight (July 15 – July 28, 2023)	<ul style="list-style-type: none"> • Setting up the environment for the chart activity in sugarizer. • Start by building up the chart activity toolbar with different buttons and functionalities.
Week nine & ten (July 29 – August 11, 2023)	<ul style="list-style-type: none"> • Build the list section and the chart section below the toolbar. • Implement Vue-chart.js to integrate charts in the exercise.
Week eleven & twelve (August 12 – August 25, 2023)	<ul style="list-style-type: none"> • Handle presence in the activity using sugar-presence. • Handle the datastore and journal for the activity. • Add the export chart as image feature in the toolbar.
August 28 – September 4, 2023	<ul style="list-style-type: none"> • Mentors submit final GSoC contributor evaluation • Coding period ends

Convince us, in 5-15 sentences, that you will be able to successfully complete your project in the timeline you have described. This is usually where people describe their past experiences, credentials, prior projects, schoolwork, and that sort of thing, but be creative. Link to prior work or other resources as relevant.

I am not just a developer, but also a coding enthusiast who enjoys implementing various projects. For past 4-5 months, I have been working on developing Node.js, React.js & Vue.js applications. Over the last month, I have been engaging with the community to gain insights into the organization's needs and the required features for this project. Since I have no other commitments during my college summer break, I can fully dedicate myself to GSoC, and I am confident that I can achieve the project goals within the given timeframe without any hindrances.

I am an active contributor of Sugar Labs for past month with over **28 commits** merged, spending this much time with the codebase helped

me understand it in a better way. --Issues (6 open, 4 closed) --Pull Request (17 closed, 0 open).

I have gained experience working collaboratively in different hackathons, which has helped me develop a strong understanding of teamwork.

Recently, my team and I won a hackathon where we designed a web application focused on alerting the users during natural disasters.

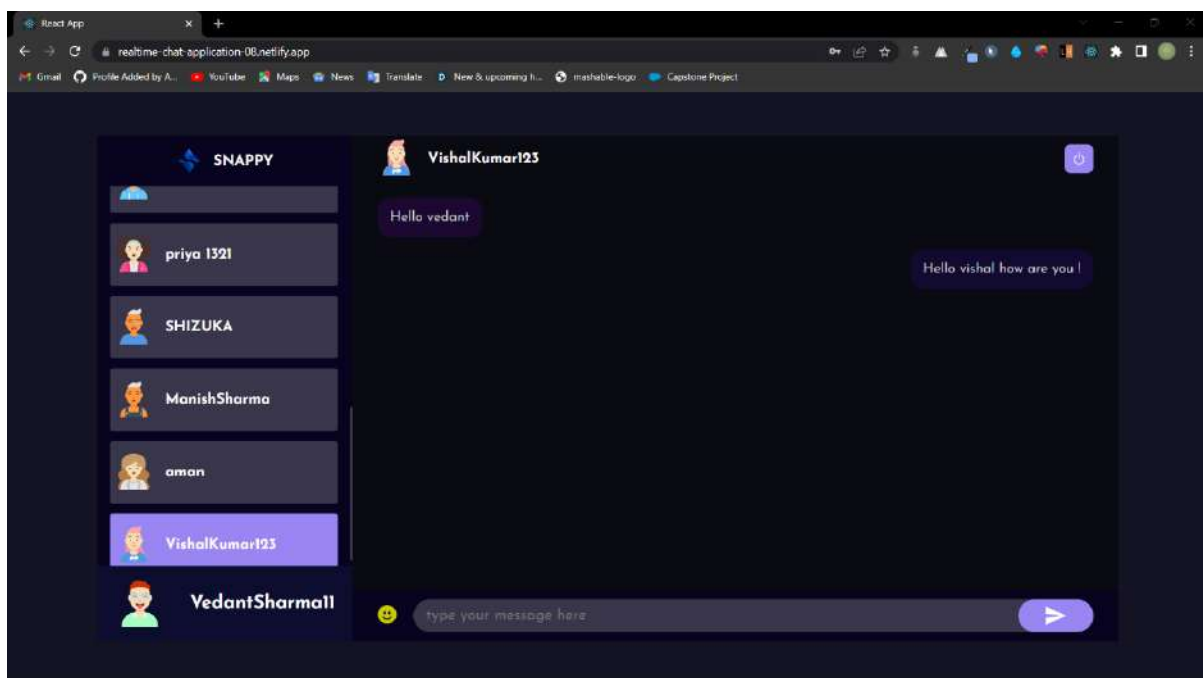
- <https://drive.google.com/vedant/file/d/15INhUa/view?usp=sharing>

I possess a good understanding of Sugarizer's core architecture and have hands-on experience in creating activities from scratch using both **VanillaJS** and **VueJS** frameworks. Check them below

- <https://github.com/VedantSharma11/Pawn.activity-VanillaJS>
- <https://github.com/VedantSharma11/Pawn.activity-VueJS>

These are some of the projects that I have built:

MERN Chat Application:



This is a real-time chat application built using React, styled-components, Node/Express, MongoDB, and Socket.io. Users can register/login via email and password, update their profile, and generate random avatars using DiceBear API. The app allows real-time chatting to users.

<https://github.com/VedantSharma11/MERN-chat-app> .

Vue.JS Weather Application:



This is a weather app web application that allows users to check the weather conditions of a specific location. The app uses OpenWeatherMap API to get the weather data for the user's location or a specified location. The app also provides users with a search bar where they can enter the name of any location to get the current weather conditions for that location. <https://github.com/VedantSharma11/Vue-weather-app> .

You and the community

If your project is successfully completed, what will its impact be on the Sugar Labs community?

Sugarizer is a great tool with many features. It can be widely adopted especially with the advent of smartphones and internet access all around the world. However there is no activity in sugarizer that teaches data visualization to students, so with the implementation of chart activity students will be able to better understand data visualization using various types of charts. Alongside this with the addition of word puzzle template in Exerciser activity teachers will be able to build word puzzles related with lessons learned in class and students can solve the puzzles and see their results. Finally, all this will greatly enhance the potential to use Sugarizer as a learning platform.

What will you do if you get stuck on your project and your mentor isn't around?

If I encounter a problem, my initial step is to search for suitable solutions online on the web. If I am unable to find a suitable solution, I will reach out to other developers on the Sugar's Element channel or mailing list. Based on my past experience, I have observed that community members are highly responsive and supportive, I believe that they will surely help me out. Moreover, I maintain contact with fellow college students and some seniors who have development experience and can assist me in resolving any issues that I may encounter.

How do you propose you will be keeping the community informed of your progress and any problems or questions you might have over the course of the project?

I plan to maintain an active presence on GitHub by frequently submitting pull requests to Sugarizer and interacting with mentors, allowing anyone in the organization to track my progress. Additionally, I intend to write weekly or biweekly blog posts to update others on my progress, any obstacles I face, and the solutions I develop. I will remain easily accessible through Matrix IRC and email at all times.

Describe a great learning experience you had as a child.

When I was in school, I used to actively participate in art-related competitions. Once, my teacher told me about an inter-school art competition 1 day before the final date of submission, and even though it was quite late to start working on my art piece from scratch and finishing it, I still didn't give up and tried my best to create something unique and

beautiful. I successfully submitted my artwork before the deadline. Even though I didn't win that competition, it taught me how to stay calm and focused under stressful situations. It also showed me that with dedication and hard work, I could achieve something that I thought was impossible.

FAQ's

- **Will you have any other time commitments, such as school work, another job, planned vacation etc during the duration of the program**

I have my end semester exams from **JUNE 1 – JUNE 15**

- **What will be your typical working hours and how many hours per week will you dedicate to this project?**

I will be working 5-6 hours on weekdays from 10 am to 4 pm (IST) and 6-7 hours on Saturday from 10 am to 5 pm (IST). In total (~50-55 hours per week). But again these timings are flexible and if the project needs more work they can be changed.

~~ END OF PROPOSAL ~~

Google
Summer of Code