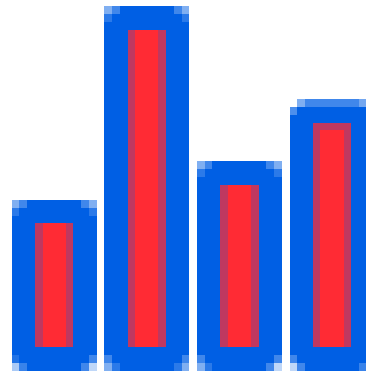
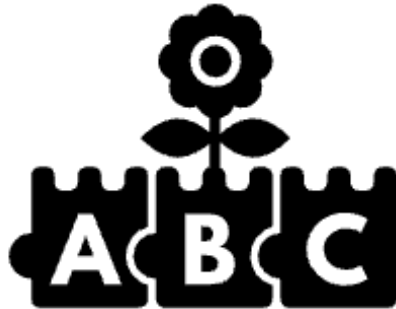




sugarlabs



Sugarizer Word Puzzle and Chart activities

Google Summer Of Code 2023 - Project Proposal

Harshit Maurya

rockharshitmaurya@gmail.com

Table of Contents

Section	Page Number
Introduction	3
Motivation	5
Project Details	6
Word Puzzle	8
Chart Activity	22
Timeline	32
Expernce and Project	34
Activity Tutorial Video Link	link

About You

What is your name?

My name is Harshit Maurya. I am a third-year undergraduate computer science student at Galgotias University, Greater Noida, India.

What are your email address and Contacts?

Primary Email Address rockharshitmaurya@gmail.com.

Secondary Email Address harshit.21scse1010102@galgotiasuniversity.edu.in

My Github username is [rockharshitmaurya](#)

Linkedin Profile [Harshit Maurya](#)

What is your first language?

I am fluent in both English and Hindi, with English being my primary language of communication. I am comfortable expressing myself fluently in both languages, whether it is through verbal or written means.

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 Greater Noida, India, and operate on the Indian Standard Time (UTC + 5:30) time zone. While I typically work from 6:00 to 12:00 (UTC), I remain flexible and can adjust my schedule to meet the needs of the project. I strongly believe that communication is a key component of project success, and I am committed to keeping my mentors informed of my availability and work progress. You can count on my dedication to the project, and I will remain actively engaged and responsive whenever my mentor requires my assistance.

Education Details

I am currently a third-year undergraduate student pursuing a **Bachelor of Technology** in **Computer Science and Engineering** at Galgotias University. Like many others in my field, I was introduced to the exciting world of programming and software development during my first year. Since then, I have eagerly delved into various fields such as **Web Development**, [Data Structures](#), [Algorithms](#), Computer Architecture, and Operating Systems. Contributing to **open-source** projects has been a rewarding experience for me since my second year. Throughout my programming journey, I have worked primarily with Web-based technologies(**MERN**), in addition to utilizing programming languages such as Java, C/C++, and Python. Lately, I have also been exploring the fascinating world of Web3 Technologies to expand my knowledge and skills in this rapidly evolving field

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 am excited to say that I have extensive experience working on open-source projects, and I am passionate about developing copyright-free software in collaboration with other developers and the wider community.

In addition to contributing to several open-source projects, such as Sugar Labs' [Sugarizer](#), [Sugarizer-server](#), [ExerciserReact](#), and [www](#).

Share links, if any, of your previous work on open-source projects:

I have been dedicated to contributing to **Sugar Labs** for the past few months, where I have made valuable contributions to various repositories, including documentation updates, bug fixes, UI changes, enhancements, and version/package updates. These past few months have provided me with an excellent learning opportunity, and I am pleased to share my contributions to Sugar Labs with you.

Pull request link	Description	Status
#1244	Ported Tour of XOEditor.activity from bootstrap to Intro.js	Merged
#156	Improved: Appearance of Exerciser Activity Title and Description Input Fields	Merged
#1276	Solved: Text Pointer Appearing Instead of Mouse Pointer on Scrollbar in write Activity	Merged
#395	Ported Whole Sugarizer server's Tour : from Bootstrap to IntroJS	Open
#1316	Optimizing Sugarizer Codebase : Removing Unnecessary CSS Code for Improved Performance	Merged
#1312	Remove unused jQuery library from the project to reduce file size	Merged
#1308	Refactor: Remove Unused Code for Improved Efficiency and Maintainability	Merged
#400	Fixed: Error in Installation Instructions for Sugar Labs Project	Merged
#1296	Fix Text Pointer Appearing Instead of Mouse Pointer on Scrollbar Across 55 Sugarizer Activities	Open

I have find over **24 issues** till now in Sugar Labs. All of these issues can be found [here](#).

About Project

What is the name of your project?

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

Can you provide an overview of your project in 10-20 sentences, including its purpose, target audience, and the technologies involved?

Sugarizer Word Puzzle and Chart activities are a project that aims to create interactive educational activity for children using the SugarLabs Learning Platforms. The project could involve creating word puzzles template and charts activity that are engaging and age-appropriate, designed to improve children's vocabulary and critical thinking skills.

The project may be designed for young children in primary school who are just starting to learn how to read and write.

Sugarizer is a versatile tool for students and teachers that works as both a mobile application and a web application, offering users more options. With a wide variety of activities, the goal is to promote interactive learning between students and teachers within Sugarizer

Technologies:

JavaScript/HTML5 development.

Vue.js framework development.

React.js framework development.

Sugarizer Core architecture.

Aim of Project:

The aim of your project is to develop new educational activities for children that help them learn and practice concepts in a visual and interactive way. The project goals are as follows:

1. Create a new template **Word Puzzle** for Exerciser activity that allows teachers or anyone with access to generate custom word puzzles for their students. The exercise generator will include various **customization options** such as background image, fill characters, and **showing the searched words**.
2. Develop a **Chart activity** for children to enable them to learn about visual representations of data such as **pie chart, bar chart, vertical bar chart, horizontal bar chart, and line chart**.
3. Provide an **interactive play area** for students to practice the puzzles generated by the exercise generator.
4. Improve **children's learning** experience by providing them with engaging educational activities.

Overall, the project aims to provide children with an interactive learning environment that helps them understand and retain complex concepts more easily.

Project Goals:

The goals are all aimed at achieving the overall aim of developing new educational activities for children that help them learn and practice concepts in a visual and interactive way.

- Develop a new Chart activity that includes pie chart, bar chart, vertical bar chart, horizontal bar chart, and line chart.
- Create a new template Word Puzzle for Exerciser activity that allows teachers or anyone with access to generate custom word puzzles for their students.
- Provide an interactive play area for students to practice the puzzles generated by the exercise generator.
- Improve children's learning experience by providing them with engaging educational activities.

Word Puzzle Template

Introduction:

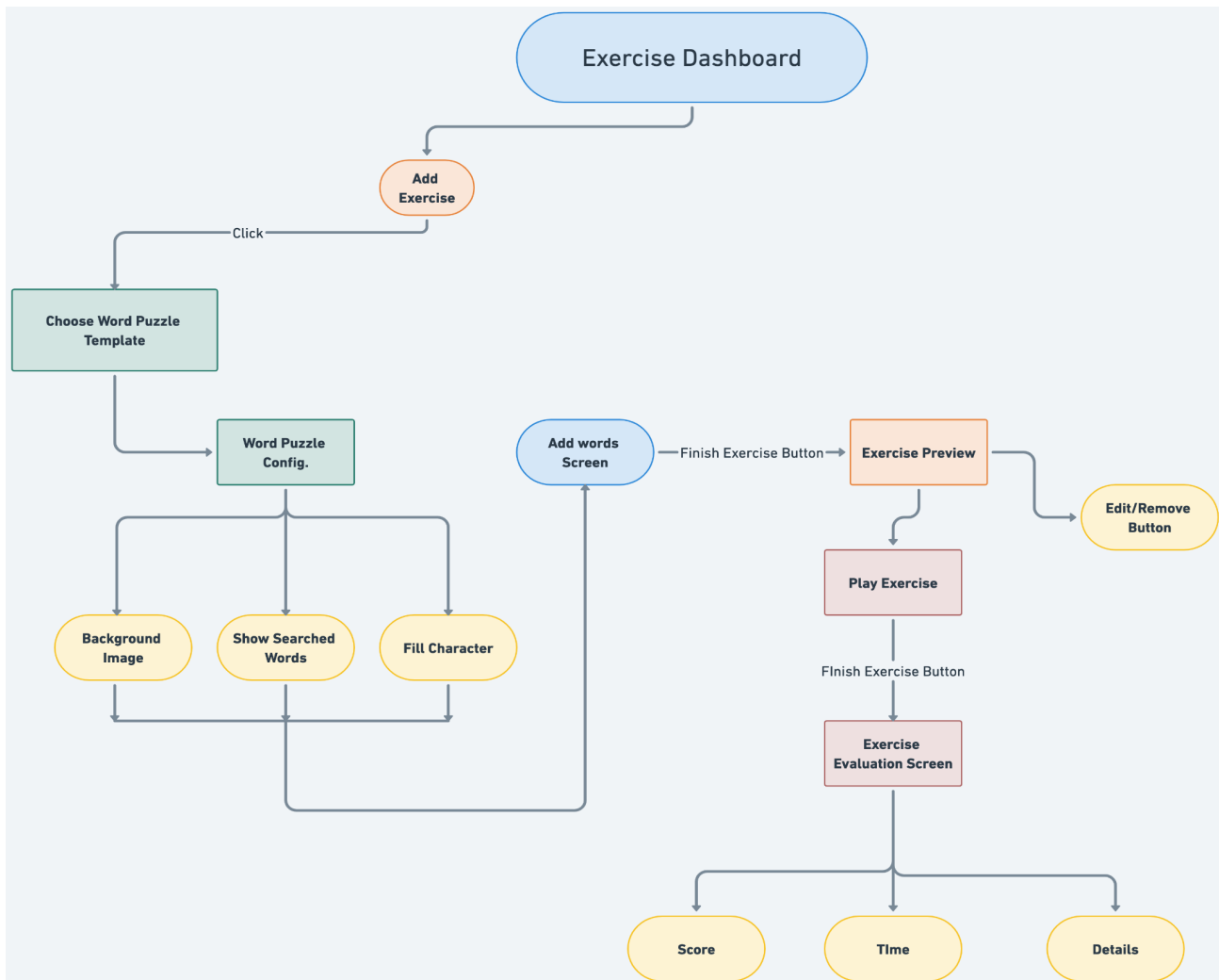
This project aims to provide teachers or anyone with access to an exercise generator that allows them to create custom word puzzles for their students. The exercise generator includes several customization options, such as a background image, fill characters, and showing searched words. Once the puzzle is generated, it will be added to the play exercise screen for students to complete. The project will be built using React.js, HTML/CSS, JavaScript, and react-redux.

Deliverables:

1. Create a new "Word Puzzle" template card in the Exerciser activity.
2. Allow teachers or anyone with access to create custom **word puzzle** for their students.
3. Provide options for **customization** such as **background image**, **fill characters**, and **showing the searched words**.
4. Ability to add words for the word puzzle.
5. Allow editing and removal of the word puzzle by the creator.
6. Implement the ability for students to play the word puzzle.
7. Display **evaluation** details, such as **score, time, and details** (number of questions attempted).

I have meticulously listed all the essential **steps** required to create a fully functional word puzzle template and have also illustrated each step and feature of the project in the **diagram below**.

Workflow of new exercise template:



Implementation

➤ Add Exercise (Builders/WordPuzzleForm.js):

The `WordPuzzleForm.js` form for the word puzzle exercise will allow users to set various options before they begin adding questions, such as

- selecting a banner image,
- setting a background image for the word puzzle,
- toggling the display of the searched words(show/hide),
- Fill custom character for the unused cells.

Once users have configured their exercise to their liking, they can **start adding**

content to the exercise, such as **text, images, sounds, speech, or videos**. After adding the desired content, users can complete the exercise setup process and submit the form. The completed exercise will then be added to the activity's homepage along with other exercises.

Below are some methods that will be available for configuring the word puzzle exercise:

changeBannerImage () :

This feature enables the user who is configuring the puzzle to set the banner image for the exercise.

Following code snippets are provided as an example only, and the actual implementation may differ.

```

1  const changeBannerImage = (imageUrl) => {
2    // Dispatch an action to update the background image in Redux store
3    dispatch(updateBannerImage(imageUrl));
4  }
5

```

changeCrosswordBackgroundImage () :

This method will help the user to set a background image for the word puzzle.

```

1  const changeCrosswordBackgroundImage = (image) => {
2    return {
3      type: 'CHANGE_BACKGROUND_IMAGE',
4      payload: image
5    }
6  }
7
8
9
10

```

viewBackgroundImage () :

This method displays the current background image of the word puzzle

```

1  const viewBackgroundImage = () => {
2    const backgroundImage = useSelector(state => state.backgroundImage);
3    return (
4      <img src={backgroundImage} alt="Background" />
5    );
6  }
7
8
9

```

ShowSearchedWord() :

This function provides a checkbox that toggles the display of the searched words in the word puzzle.

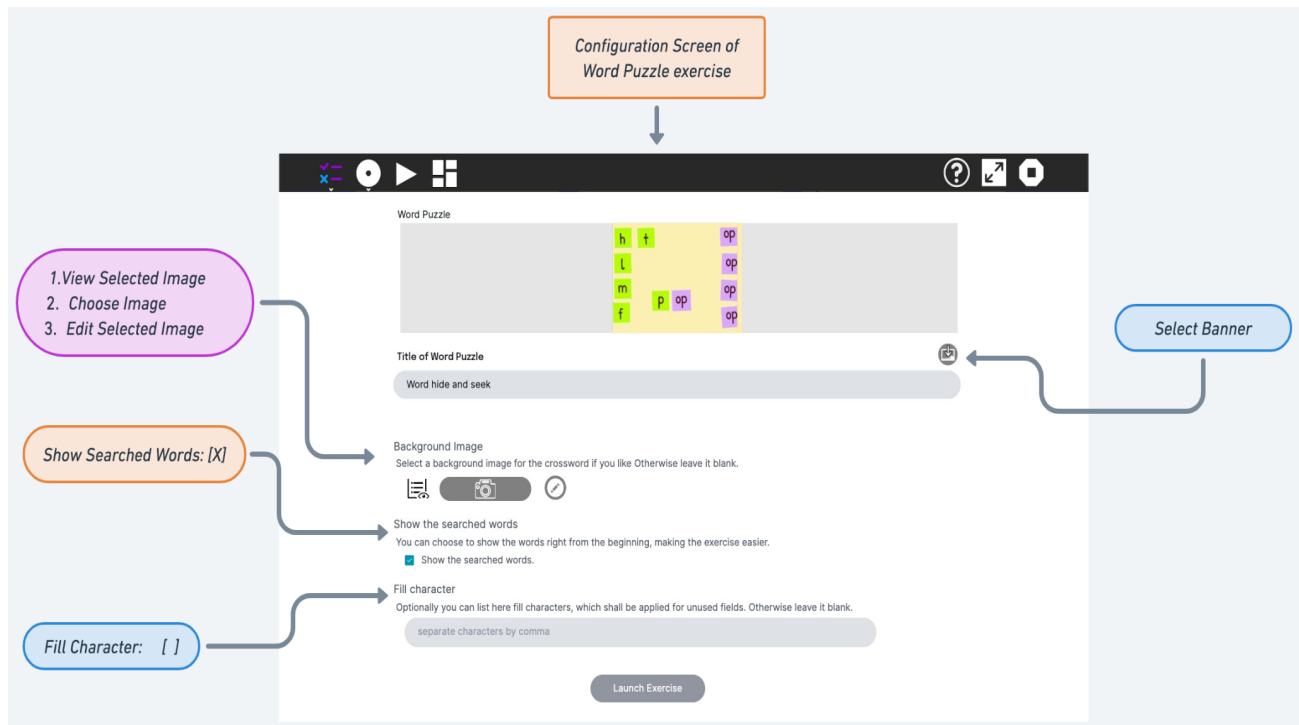
```
1  const ShowSearchedWord = () => {
2    const showWords = useSelector(state => state.showWords);
3    return (
4      <div>
5        <input type="checkbox" checked={showWords} onChange={() => dispatch({ type: 'TOGGLE_SHOW_WORDS' })} />
6        <label>Show Words</label>
7      </div>
8    );
9  }
10
11
12
13
```

FillCharacterAtUnusedPlace() :

This function provides a form for setting the fill character that will be used for unused cells in the word puzzle.

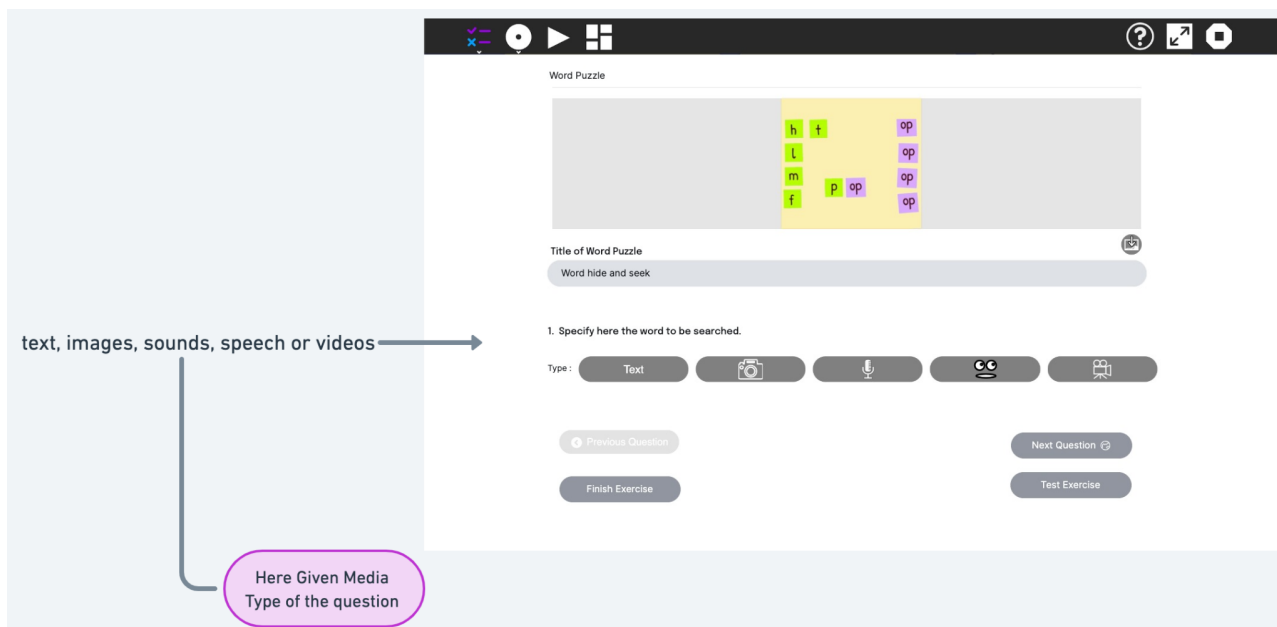
```
1  const FillCharacterAtUnusedPlace = () => {
2    const fillChar = useSelector(state => state.fillChar);
3    return (
4      <div>
5        <input type="text" value={fillChar} onChange={(e) => dispatch({ type: 'SET_FILL_CHAR', payload: e.target.value })} />
6        <label>Fill Character</label>
7      </div>
8    );
9  }
10
11
12
13
14
```

➤ **Word Puzzle Form will look like this:**



When the user clicks on the "**launch exercise**" button, they will be taken to [next screen](#), On this screen, where they can add words, clues, and other content to build their puzzle. Once they have finished creating their puzzle, they can save it to be used later or start solving it right away.

➤ **The Next Screen will look like this:**



On the following screen, the user or teacher can add previously created questions with their previously configured settings. They have the option to select various media types to form the question, including :

1. Text
2. Images
3. Sounds
4. Speech
5. Videos

Depending on the chosen media type, a corresponding input field will open for the user to input their question.

The purpose of offering a variety of media types is to improve the visual appeal and engagement for children using the platform. For example, if an image is selected as the media type and an image of an elephant is uploaded, the players will see the image and need to locate the word "elephant" in the word puzzle.

➤ ***Here is an Example Screen if the user selects text as input:***

The screenshot shows a 'Word Puzzle' interface. At the top, there is a navigation bar with icons for home, play, and settings. Below the title 'Word Puzzle', there is a grid of letters. The letters 'h', 't', 'l', 'm', 'f', 'p', and 'op' are visible in a grid. Below the grid, there is a text input field for the 'Title of Word Puzzle' with the text 'Word hide and seek'. Below that, there is a section for '1. Specify here word to be searched.' with a text input field containing 'Apple'. At the bottom, there are four buttons: 'Previous Question', 'Next Question', 'Finish Exercise', and 'Test Exercise'. An orange callout box on the left side of the screen contains the text 'Text Type Input of Question' with an arrow pointing to the 'Specify here word to be searched.' input field.

If the user selects "text" as the input media type as above shown, the screen will display an input field where they can type in their question-word. Once the "question word" is entered, the user can proceed to the next question in their puzzle.

After the user has added a sufficient number of questions to their exercise, they can finish creating the exercise. Once the exercise is completed, it will be automatically added to the home page of the exerciser activity and will be ready to play. From the home page, other users or students can access and engage with the.

➤ **Play Exercise (Players/WordPuzzlePlayer.js):**

exerciseWordPuzzlePlayer.js file is intended to handle the functionality related to playing the word puzzle exercise once it has been created. Here I am going to explain some of the **possible functions** for the **WordPuzzlePlayer.js** file:

Here are some of the possible states of this **WordPuzzlePlayer** file going to hold:

```

1 // Define the states
2 const [words, setWords] = useState([]);
3 const [grid, setGrid] = useState([]);
4 const [selectedWord, setSelectedWord] = useState('');
5 const [selectedCells, setSelectedCells] = useState([]);
6 const [wordsFound, setWordsFound] = useState([]);

```

createEmptyGrid:

The createEmptyGrid function creates an empty 2D array to hold the words for the word search puzzle. It calculates the maximum length of the words in the input array, and then creates the empty grid with dimensions of maxLength by words.length. The function returns the empty grid array.

```

1 // Create an empty grid to fit all the words
2 const createEmptyGrid = (words) => {
3   // Get the maximum length of the words
4   const maxLength = words.reduce((maxLength, word) => {
5     return Math.max(maxLength, word.length);
6   }, 0);
7
8   // Create the grid
9   const grid = [];
10  for (let i = 0; i < maxLength; i++) {
11    grid.push([]);
12    for (let j = 0; j < words.length; j++) {
13      grid[i].push('');
14    }
15  }
16
17  return grid;
18 };

```

fillGridWithWords:

This function fills the empty 2D array created by the createEmptyGrid function with the input words for the word search puzzle. It takes in two arguments: the empty grid array and an array of words to place in the grid. Overall, the fillGridWithWords function is responsible for placing the words in the grid in a random and non-overlapping way.

```

1 // Create an empty grid to fit all the words
2 const createEmptyGrid = (words) => {
3   // Get the maximum length of the words
4   const maxLength = words.reduce((maxLength, word) => {
5     return Math.max(maxLength, word.length);
6   }, 0);
7
8   // Create the grid
9   const grid = [];
10  for (let i = 0; i < maxLength; i++) {
11    grid.push([]);
12    for (let j = 0; j < words.length; j++) {
13      grid[i].push('');
14    }
15  }
16
17  return grid;
18 };

```

tryPlaceWord:

This code defines a React component called TryPlaceWord. The component takes in five props: grid, word, row, col, and direction. These props are used to calculate the ending row and column of the word based on its direction and to check if the word can fit in the grid.

If the word doesn't fit in the grid, the component returns false. If the word does fit in the grid, the component checks if the cells in the grid are available for each letter in the word. If any

of the cells are already occupied by a different letter, the component returns false. If all the cells are available, the component places the word in the grid by iterating through each letter of the word and assigning it to the corresponding cell in the grid. The component returns true if the word was successfully placed in the grid.

```

1 // Try to place the word in the grid in the given direction
2 const tryPlaceWord = (grid, word, row, col, direction) => {
3   // Calculate the ending row and column based on the word and the direction
4   const deltaRow = [-1, -1, -1, 0, 0, 1, 1, 1][direction];
5   const deltaCol = [-1, 0, 1, -1, 1, -1, 0, 1][direction];
6   const endRow = row + (word.length - 1) * deltaRow;
7   const endCol = col + (word.length - 1) * deltaCol;
8
9   // Check if the word fits in the grid
10  if (endRow < 0 || endRow >= grid.length || endCol < 0 || endCol >= grid[0].length) {
11    return false;
12  }
13  //rest code will go here ...
14 }

```

generatePuzzle:

This function generates a word search puzzle. It takes an array of words as its parameter and performs the following steps:

1. It converts all the words to uppercase and sorts them alphabetically.
2. It sets the words in the state using the setWords function.
3. It creates an empty grid using the createEmptyGrid function, which takes the words as its parameter and returns a two-dimensional array representing the grid.
4. It fills the grid with the words using the fillGridWithWords function, which takes the grid and words as its parameters and randomly places the words in the grid in different directions.
5. It fills the remaining cells of the grid with random letters using the fillGridWithRandomLetters function, which takes the grid as its parameter and replaces any empty cells with random letters.
6. It sets the grid in the state using the setGrid function.

Overall, this function generates a complete word search puzzle by taking the input words, creating a grid to fit all the words, filling the grid with the words in random directions, and then filling the remaining cells with random letters.

```

1 // Generate the puzzle
2 const generatePuzzle = (words) => {
3   // Convert words to upper case and sort them alphabetically
4   words = words.map((word) => word.toUpperCase()).sort();
5
6   // Set the words in state
7   setWords(words);
8
9   // Create a grid to fit all the words
10  const grid = createEmptyGrid(words);
11
12  // Fill the grid with the words
13  fillGridWithWords(grid, words);
14
15  // Fill the remaining cells of the grid with random letters
16  fillGridWithRandomLetters(grid);
17
18  // Set the grid in state
19  setGrid(grid);
20 };

```

➤ *Here is an example of what the word puzzle exercise may look like once it has been added to the home screen:*

The screenshot shows a home screen with a purple background and a black top bar containing navigation icons. There are four exercise cards arranged in a 2x2 grid:

- Conjugate "to be"**: Cloze Text, 10 Questions, Best Score: 0/10
- Capitals of the World**: MCQ, 8 Questions, Best Score: 0/8
- Animal sounds**: Matching Pair, 4 Questions, Best Score: 0/4. It features a cartoon illustration of a dog.
- Word Puzzle**: Word Puzzle, 5 Questions, Best Score: 0/8. It features a 10x10 grid of letters:

V	Q	D	G	I	P	I	Z	R	I			
T	Y	I	N	K	J	K	H	L	W	O	S	A
C	R	R	O	B	L	L	E	C	E	V	I	Z
F	D	K	D	S	A	T	U	R	D	A	Y	P
P	S	Z	S	G	T	T	M	O	N	D	A	Y
B	D	N	J	G	K	I	S	R	E	B	Y	D
T	B	A	N	I	Z	H	K	M	S	K	R	S
I	S	T	Z	C	O	S	H	N	D	A	Y	I

An orange callout box on the right side of the screen contains the text: "After adding 'word puzzle exercise', it will be displayed in the following manner". An arrow points from this box to the 'Word Puzzle' card.

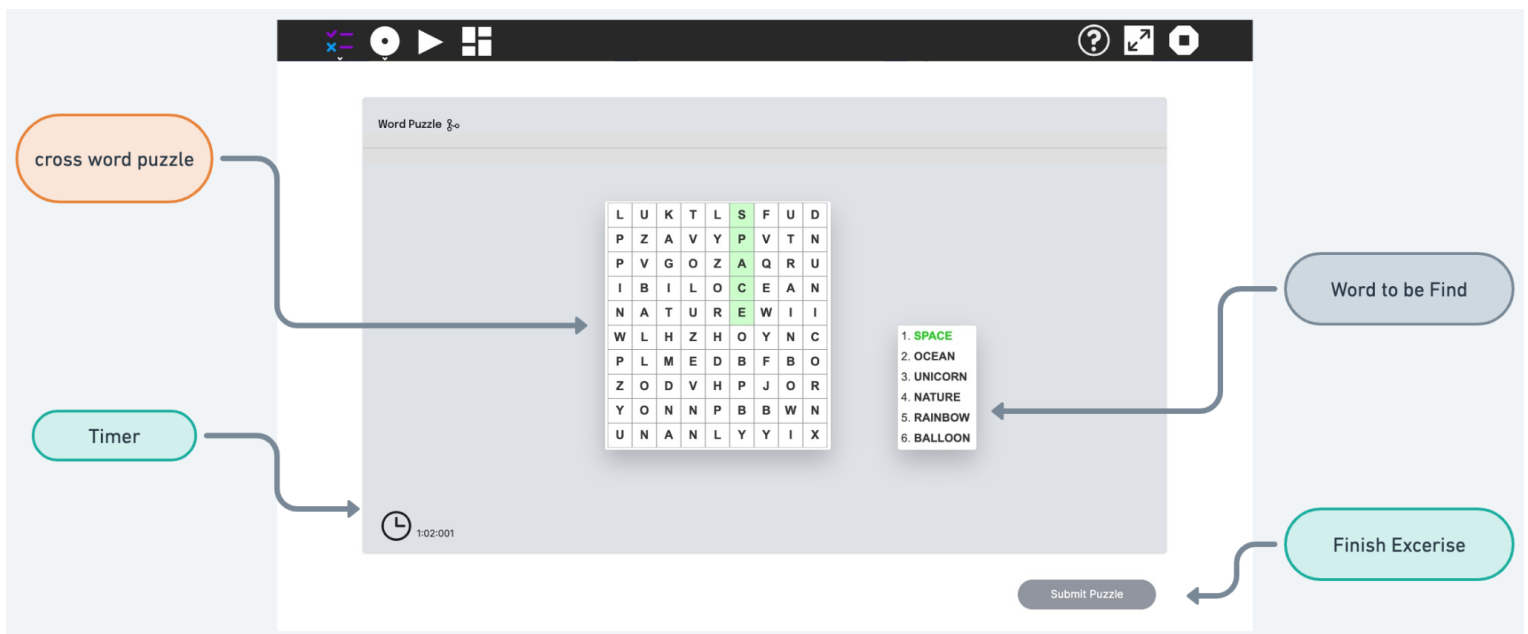
On the home screen, the exercise will be displayed with its title, best score, no of questions, and a thumbnail image. Users or students can click on the exercise to start playing and solving the puzzle.

On the word puzzle card displayed on the home screen, there are several different functionalities that users can perform. These include:

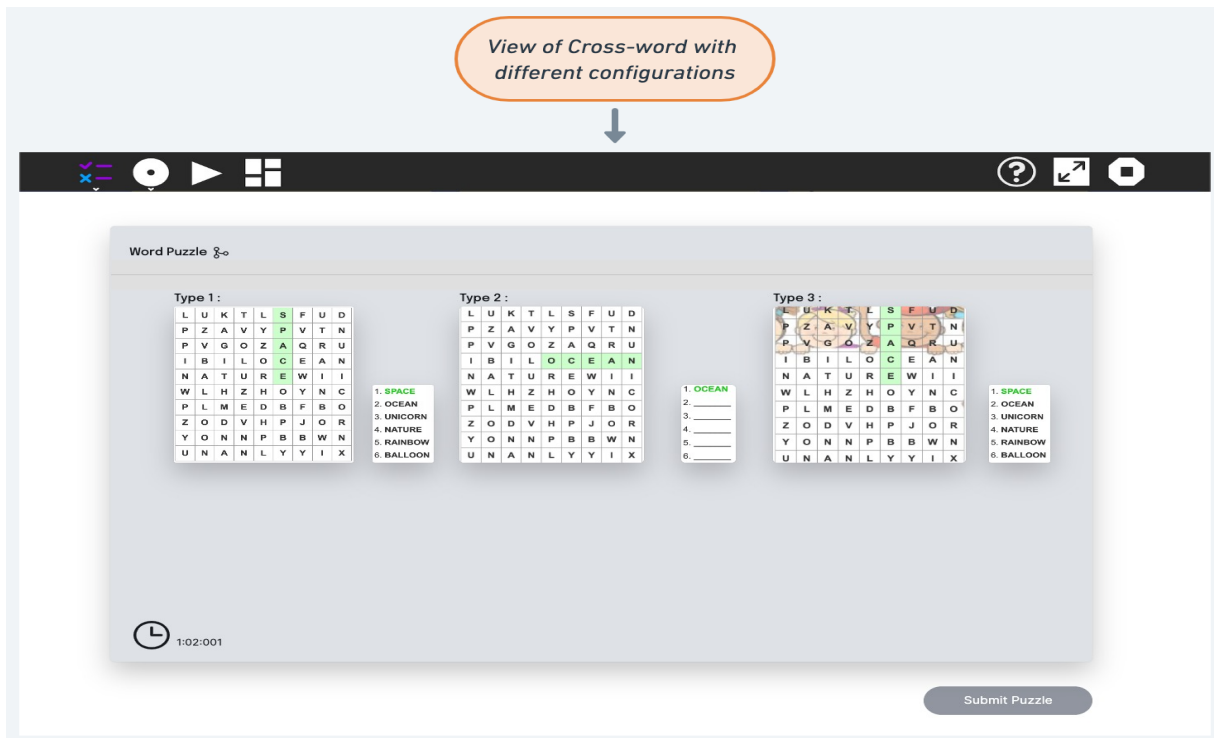
- **Play**: This button allows users to start playing the word puzzle exercise.
- **Edit Exercise**: Users can edit the exercise and modify its content or settings by clicking on this button.
- **Remove Exercise**: This button enables users to delete the exercise from the home screen if desired.

Here are some visuals of the exercise in play mode:

Screen 1:



Screen 2:



In the Above Screen 2 displays a visual representation of the word puzzle showcasing how it will appear with various configurations. There are three types of displays:

- **Type 1:** When the `show searched word` option is selected on the [configurations screen](#).
- **Type 2:** When the `show searched word` option is not selected on the [configurations screen](#).
- **Type 3:** When the user has `selected a background image` for the word puzzle on [configurations screen](#), which will be reflected in the puzzle's appearance.

There will 3 Options for evaluation of the game result:

- **Score:**
 - The user will have the ability to view their score in comparison to the highest score, and also visualize their score using a bar chart.
- **Time:**
 - The game score evaluation feature I've implemented allows users to easily visualize their performance over time, making it easier to track progress and identify areas for improvement.

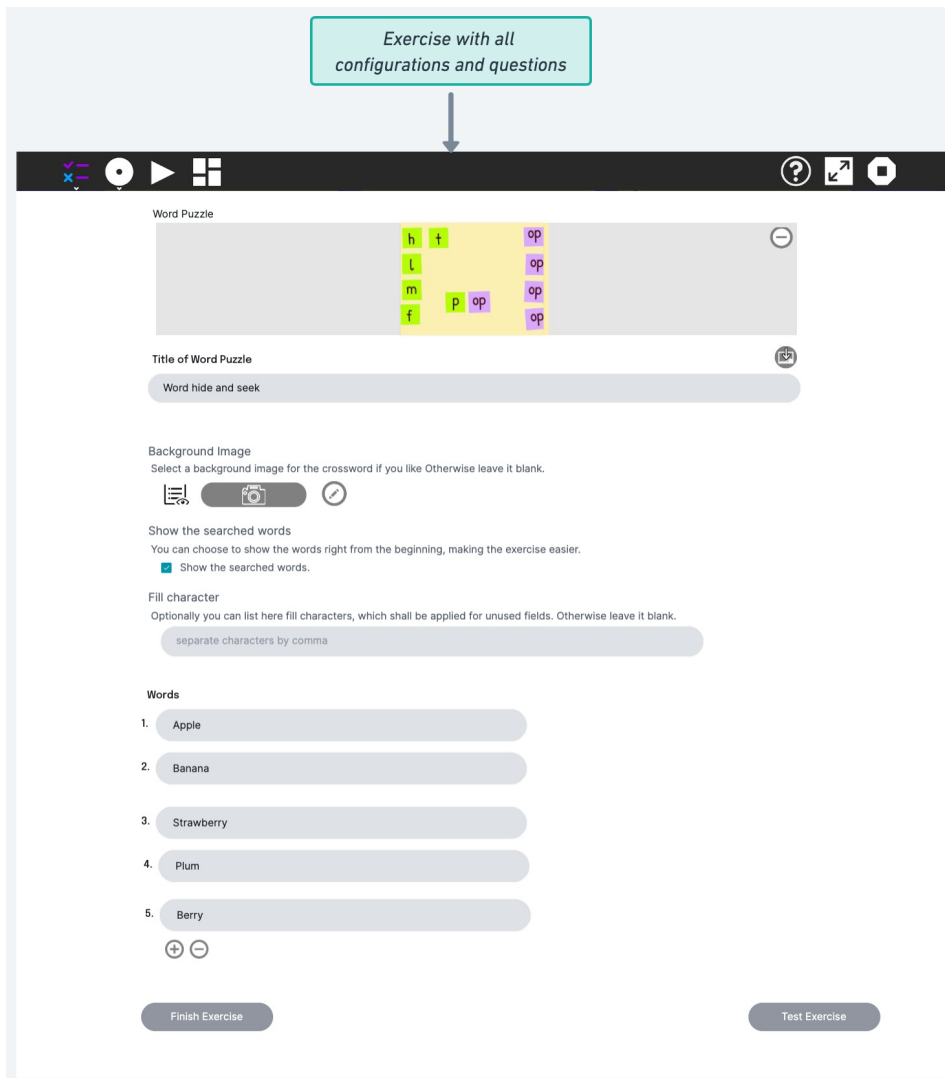
➤ **Details:**

- This Score Evaluation feature provides users with additional information about their score and performance in the game. Specifically, it shows how many words the user was able to find in the puzzle and how many they were not able to find. This can help the user to identify.

Edit Exercise:

This feature will become available to the user in the play menu when they open the exercise in edit mode from the top bar. Once in the edit screen, the user can view all the previous configurations and questions they have set for the exercise. They can then modify any setting they have previously set and test the exercise to ensure that everything is working properly. If everything works as expected, the user can finish editing the exercise.

Visual Screen From Edit mode of puzzle Exercise:



- Please find the link below to access all the visuals that I have created for this word puzzle activity. Any feedback or suggestions from the maintainers are always welcome. Kindly click on the link to view the visuals :

Link: <https://whimsical.com/word-puzzle-Tq8s7hvzkCNYGyip5XdCpT>

Additional Features:

1. Word Puzzle containing weekdays and months:

- The game can include a set of words related to weekdays and months, such as Monday, Tuesday, January, February, etc.
- This can be a fun and educational way for children to learn and memorize the names of the days and months of the year.

2. Word Puzzle containing animal or color name:

- Another set of words that can be included in the game are names of animals or colors, such as cat, dog, green, blue, etc.
- This can provide a fun way for children to learn and memorize the names of different animals and colors.

Conclusion:

With the addition of these extra features, the Word Puzzle game can become more diverse and engaging for children. By including sets of words related to different themes, such as weekdays and months or animals and colors, the game can offer a fun and educational way for children to learn and improve their vocabulary. These features can enhance the overall experience of the game and make it more appealing for children to play.

Chart Activity

Introduction:

The Sugar Labs organization has a project called Sugarizer, which aims to provide a free/libre and open-source learning platform for children. One of the key features of Sugarizer is its collection of activities that provide interactive learning experiences. One such activity is the Chart Activity, which is currently only available in Sugar OS and written in Python. The purpose of this proposal is to convert the Chart Activity into a Vue.js component so that it can be used in Sugarizer applications.

Objective:

The main objective of this project is to convert the existing Chart Activity into a reusable Vue.js component that can be easily integrated into Sugarizer applications. The specific goals of the project are:

- Convert the existing Python code to JavaScript and Vue.js syntax
- Implement the chart rendering functionality using the Vue Chart.js library
- Allow the Chart component to be easily customized and configured by passing props to the component
- Implement the ability to share activities with other users on the network so that they can explore activities together
- Implement the ability to export the chart as an image

Scope:

The scope of this project is limited to converting the Chart Activity into a Vue.js component using the Vue Chart.js library and implementing the two additional features. The conversion process will involve rewriting the existing Python code into JavaScript and Vue.js syntax and implementing the chart rendering functionality using the Vue Chart.js library. The Chart component will be designed to be easily customizable and configurable by passing props to the component. The two additional features will allow users to share activities with other users on the network and export charts as images.

Methodology:

The project will follow an iterative development process, with each iteration consisting of the following steps:

Design: In this phase, we will gather the requirements for the Chart component and the two additional features along with other existing features, and design a high-level architecture for the component. This will include deciding on the props to be passed to the component, the expected behavior of the component, and the implementation details for the two additional features.

Implementation: In this phase, we will implement the Chart component in Vue.js using the Vue Chart.js library. We will also implement existing features along with the two additional features, allowing users to share activities with other users on the network and export charts as images. We will ensure that the Chart component is customizable and configurable by passing props to the component.

Testing: In this phase, we will test the Chart component and the two additional features to ensure that they work as expected and are compatible with Sugarizer's requirements. We will use automated testing frameworks like Jest and Cypress to test the component, as well as manual testing to ensure that the component is intuitive and easy to use.

Feedback and Refinement: In this phase, we will gather feedback from users and stakeholders, and use it to refine the Chart component and the two additional features. We will iterate on the design, implementation, and testing phases as necessary to ensure that the Chart component and the two additional features meet the needs of Sugarizer's users.

Deliverables:

1. A fully-functional Chart Activity implemented using Vue.js and the Vue-Chartjs library.
2. Features implemented to improve the functionality of the activity, including adding and removing data, changing the graph title, changing the type of graph, configuring graph settings, formatting text, sharing activity, exporting chart as an image, and full-screen mode.
3. Regular progress reports and communication with the Sugar Labs community.

Before Starting the **plan**, I would like to express my gratitude to our project mentor, [Mr. Lionel Laské](#), for his invaluable [tutorial](#) on building activities in Vue.js from scratch. The tutorial helped me gain a better understanding of the code structure and how things work in Sugarizer.

Plan Strategy:

Assuming that I have set up the basic template provided by Sugarizer to create a new Vue activity, I plan to implement the Chart Activity by following these steps:

1. Create a layout for the activity that includes a **toolbar**, a chart area, and a data list area. The toolbar will have buttons for adding data, removing data, changing the chart type, accessing settings, entering full-screen mode, exporting the chart as an image, and **saving the state** of the activity.
2. Implement a method in **Vue.js** to push new data to the chart data array when the user clicks the add data button. When the user clicks the remove data button, another method will be used to splice the selected data from the array.
3. Add functionality to the chart type button to allow the user to switch

between Vertical Bars, Horizontal Bars, Lines, and Pie chart types. When a new chart type is selected, the chart data will be updated accordingly.

4. Implement a settings button that opens a toolbar with options to change the chart title, horizontal and vertical labels and color scheme. This will be achieved using an internal library provided by Sugarizer, which includes files such as `exportPalette.js`, `fontPalette.js`, `ForegroundColorPalette.js`, `formatTextPalette.js`, and `sizePalette.js`.
5. Allow the user to format text, including changing font style, increasing/decreasing font size, and changing text color.
6. Enable the user to export the chart as an image, PDF, or other file formats. This will be done using the libraries like `html2canvas` and `jspdf` Vue.js `toDataURL` method.
7. Implement a method to save the state of the activity, so the user can pick up where they left off if they close and reopen the activity. This can be achieved using the built-in `localStorage` object in Vue.js to store the state of the chart data array and other activity variables.
8. Lastly, add a feature to allow the user to share the activity with others on the network. This can be done using Sugarizer's built-in networking capabilities to allow other users to collaborate on the same chart activity.

I believe that implementing these features will result in a user-friendly and comprehensive Chart Activity that will provide an excellent user experience for Sugarizer users.

Let's Explore Each Implementation Step in Detail:

Rendering Chart:

To display a chart on the screen, we will be using the external library called `vue-chart.js`. Vue.js uses `Chart.js` and abstracts the basic logic while exposing the Chart.js object to provide maximum flexibility such as changing the chart color to the same as the user theme color of sugarizer using `chartOptions` parameter.

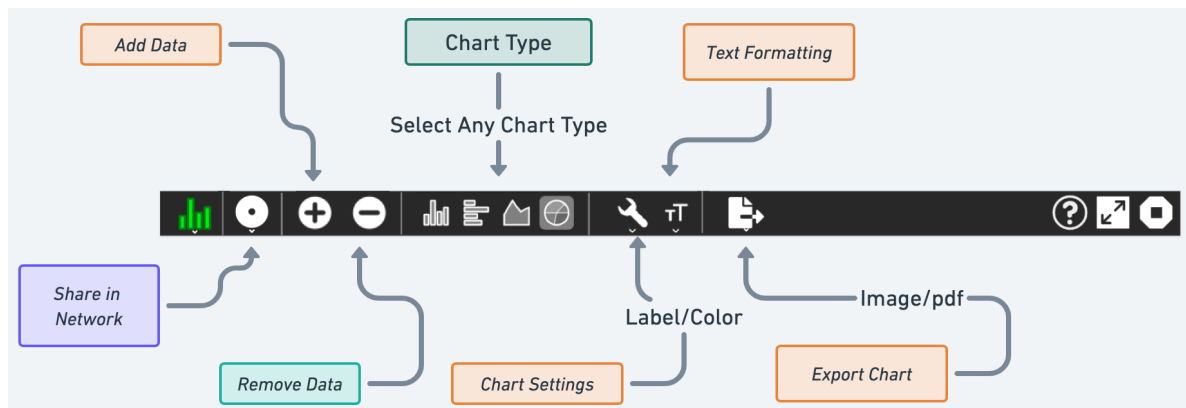
Here is a simple code snippet using `vue-chart.js` that implements a bar chart showing sales data.

```


1 <template>
2 <Bar ref="chart" :data="data" :options="ChartOptions" />
3 </template>
4
5 <script lang="ts">
6 import { Chart as ChartJS, Title, Tooltip, Legend, BarElement, CategoryScale, LinearScale } from 'chart.js'
7 import { Bar } from 'vue-chartjs'
8 import * as chartConfig from './chartConfig.js'
9
10 ChartJS.register(CategoryScale, LinearScale, BarElement, Title, Tooltip, Legend)
11
12 export default {
13   name: 'App',
14   components: {
15     Bar
16   },
17   data() {
18     //chart config contains data set and chart options
19     return chartConfig
20   }
21 }
22 </script>
23

```

Diagram of ToolBar with features:



Add Data

The "Add data" feature allows users to add data to the chart by clicking on a " button. To implement this feature, we will create a method in Vue.js that will push the new data to the existing chart data array.


The method will be **triggered** by an event when the user clicks on the "+" button. The method will take the input values from the user, such as the label and data, and push them to the chart data array. Once the data is added, the chart will be **re-rendered** to display the updated chart with the new data.

```

1   methods: {
2     addData: function() {
3       this.chartData.labels.push('Label');
4       this.chartData.datasets[0].data.push(0);
5     }
6   }
7

```

Remove Data

The "Remove Data" feature allows users to remove data from the chart by selecting the data from a list of data points rendered beside the chart. The selected data will be highlighted, and the user can remove the data by clicking on a  button. To implement this feature, we will create a method in Vue.js that will **splice** the selected data from the existing chart data array.

The method will be triggered by an event when the user clicks on the "-" button. The method will take the selected data point from the list of data points, and splice it from the chart data array. Once the data is removed, the chart will be re-rendered to display the updated chart without the removed data.

```

1  methods: {
2    removeData: function(index) {
3      this.chartData.labels.splice(index, 1);
4      this.chartData.datasets[0].data.splice(index, 1);
5    }
6  }
7
8

```

Change Graph Type:

Currently, for the chart visualization, the toolbar provides users with four distinct options to choose from when visualizing their data: vertical bars, horizontal bars, lines, and pie charts. By utilizing a Vue.js method, users can easily switch between these different **chart types**. Users can change the type of the charts (Vertical Bars, Horizontal Bars, Lines, Pie). This feature will be implemented using a method in Vue.js to update the chart type.

```

1  methods: {
2    changeChartType: function(type) {
3      this.chartType = type;
4    }
5  }
6

```

Config Label

The "Chart Settings" feature in the chart activity allows users to customize their chart by changing the labels of the horizontal and vertical axes. This feature is important because it allows users to accurately represent their data by labeling the axes correctly. Users can also change the color of the labels to make them stand out or match a certain theme. To implement this feature, Vue.js components and **props** will be used to ensure that the changes made by the user are reflected in the chart in real time. By giving users the ability to customize their chart, they will have **more control** over how their data is presented and can make more informed decisions based on the insights they gain from the chart.

```

1  methods: {
2    changeLabels() {
3      // get new label values from input fields
4      let newHorizLabel = this.newHorizLabel;
5      let newVertLabel = this.newVertLabel;
6
7      // update chart options object with new labels
8      this.chartOptions.scales = {
9        xAxes: [{
10         scaleLabel: {
11           display: true,
12           labelString: newHorizLabel
13         }
14       }],
15       yAxes: [{
16         scaleLabel: {
17           display: true,
18           labelString: newVertLabel
19         }
20       }]}
21     };
22
23     // update chart with new options
24     this.updateChart();
25   },
26
27   // method to update chart with new options
28   updateChart() {
29     this.$refs.chart.chartInstance.options = this.chartOptions;
30     this.$refs.chart.chartInstance.update();
31   }
32 }
33

```

Text Formatting:

The formatting option includes various text formatting tools that users can use to customize the appearance of text in the chart activity. These tools include changing the color of the text, increasing or decreasing the size of the text, and changing the font style to one of the pre-defined font styles available in the application. Users can easily select the desired text formatting tool from the formatting toolbar and apply it to the text in the chart activity. This feature will be implemented using the **internal library** provided by Sugarizer, which includes various files such as **fontPalette.js**, **ForegroundColorPalette.js**, **formatTextPalette.js** and **sizePalette.js**. With the help of these files, we will be able to implement text formatting tools in the chart activity.

```

1 <sugar-toolitem id="size-palette" v-bind:title="l10n.stringGridSize" palette-file="js/palettes/sizePalette.js"
2 palette-class="SizePalette" palette-event="grid-size-selected" v-on:grid-size-selected = "onGridSizeChange"></sugar-toolitem>
3
4 <sugar-toolitem id="format-text-button" v-bind:title="l10n.stringFormatText" palette-file="js/palettes/formatTextPalette.js"
5 palette-class="FormatTextPalette" palette-event="format-text-selected" v-on:format-text-selected = "onFormatText" ></sugar-toolitem>
6 <sugar-toolitem id="font-button" v-bind:title="l10n.stringChooseFont" palette-file="js/palettes/fontPalette.js"palette-class="FontPalette"
7 palette-event="font-updated" v-on:font-updated = "onFontChange" ></sugar-toolitem>
8 <sugar-toolitem id="increase-font" v-bind:title="l10n.stringIncreaseFont" @click='increaseFont'></sugar-toolitem>
9 <sugar-toolitem id="decrease-font" v-bind:title="l10n.stringDecreaseFont" @click='decreaseFont'></sugar-toolitem>
10

```

Export Chart As

This feature enables users to export the current chart as an image or a PDF and **save** it to the **Journal**. To accomplish this, we will use Sugarizer's internal component, **SugarJournal**. We will add the **js/components/SugarJournal.js** file to our **index.html** to include SugarJournal. Moreover, we will need to include a custom component, **<sugar-journal>**, in our Vue template to prepare our application for saving data to the Journal. To export the chart as an image, we will need two external libraries, namely **html2canvas** and **jspdf**. Below is an implementation example of how we can export the chart as an image or a PDF.

Export Chart As Image :

To export the current chart as an image, we will use the **html2canvas** library. First, we need to include the library in our **index.html** file. Then, we can add a button or an option in our toolbar for exporting the chart.

we can use the **html2canvas** library to capture the current chart as a canvas. We can then convert this canvas into a data URL using the **toDataURL()** method. This data URL can be used to display the image in a new window or save it to the Journal using the SugarJournal component as explained earlier.

Here's an **example code** snippet for exporting the chart as an image:

```

1 exportChartAsImage() {
2   // Get the chart container element
3   const chartContainer = document.getElementById('chart-container');
4
5   // Use html2canvas to capture the chart as a canvas
6   html2canvas(chartContainer).then(canvas => {
7     // Convert the canvas to a data URL
8     const imgData = canvas.toDataURL();
9
10    // Save the image to the Journal using SugarJournal
11    const metadata = {
12      mimetype: 'image/png',
13      title: 'Chart Image',
14      activity: 'org.example.MyChartActivity',
15      timestamp: new Date().getTime(),
16      creation_time: new Date().getTime(),
17      file_size: 0,
18    };
19    this.$refs.SugarJournal.createEntry(imgData, metadata); //save in sugar journal
20  });
21 }

```


Export Chart As PDF

exporting as PDF is a useful feature that allows users to save the current chart as a PDF document. To implement this feature, we can use an external library called **jsPDF**, which allows us to generate PDF documents in the browser.

Here's how we can implement the export as PDF feature in our chart application:

1. First, we need to include the jsPDF **library** in our project by adding a script tag to our index.html file.
2. Next, we need to create a function that will generate the PDF document. We can use the jsPDF library's **addImage method** to add the chart image to the PDF document. We can also use the setFont and text methods to add any additional text or information to the PDF.
3. We can then create a button or link in our chart application's UI that, when clicked, will call the PDF export function.
4. Finally, we can use the Sugarizer Journal component to save the PDF document to the Journal for future reference.

Overall, the export as PDF feature is a valuable addition to our chart application.

```

1  exportPDF() {
2      // Get the canvas element
3      let canvas = this.$refs.chart.canvas;
4
5      // Create a new jsPDF instance
6      let pdf = new jsPDF('landscape');
7
8      // Convert canvas to an image data url
9      let imgData = canvas.toDataURL('image/png');
10
11     // Add the image to the PDF
12     pdf.addImage(imgData, 'PNG', 10, 10, 250, 150);
13
14     // Save the PDF to the journal
15     let metadata = {
16         mimetype: 'application/pdf',
17         title: 'Chart.pdf',
18         activity: 'org.olpcfrance.MediaViewerActivity',
19         timestamp: new Date().getTime(),
20         creation_time: new Date().getTime(),
21         file_size: 0,
22     };
23     this.$refs.SugarJournal.createEntry(pdf.output('blob'), metadata);
24 }

```

Share Activity

To enable the Share Activity feature in a chart activity, a **palette** can be included. The term "palette" refers to a pop-up menu in the toolbar that displays

items inside, often other buttons. **Sugar-Web** exposes a Palette library and, more precisely, a **PresencePalette**, which provides real-time communication between clients using the **publish/subscribe** pattern.

To implement this feature, first, include the following code in your index.html file:

```

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>
10
11 <!-- Inside app element -->
12 <sugar-presence ref="SugarPresence"></sugar-presence>
13
14 <!-- After script loads -->
15 <script src="js/components/SugarPresence.js"></script>

```

Rest, I'll follow this beautiful article on [Handle multi-user with presence](#) provided by our mentor

Activity Tour :

The Activity UI tutorial is a series of dialog boxes that guide users through the meaning of different UI elements by highlighting them. To implement this feature in our activity, we first need to include the Intro.js external library in our project.

Intro.js is a lightweight JavaScript library that allows us to create powerful step-by-step customer onboarding tours. Additionally, the **sugar-tutorial component** is available, which not only handles the displaying of steps but also has the Sugar UI built-in, so we don't have to worry about setting up Bootstrap tour or styling. This is a convenient and time-saving solution for our activity."

Implementing State Saving in the Application:

In this implementation, we can save the data entered by the user in **Sugar-Journal**, which is a component of Sugarizer. This helps to manage the state of an activity. The idea is that every time we exit the activity, we will store the context of the current activity in the **datastore**. The datastore is the place where Sugar stores the Journal, and **Sugar-Web** automatically initializes it during activity setup.

To store the **context**, we define a context object and add the necessary data to it, such as the chart data in this case. Then, we pass this object to the **saveData()** method of Sugar-Journal to save the state of the activity.

Here's the code snippet:

```
1  methods: {  
2    onStop: function () {  
3      // Save current pawns in Journal on Stop  
4      var context = {  
5        pawns: this.chartData  
6      };  
7      this.$refs.SugarJournal.saveData(context);  
8    }  
9  }
```

This code should be called at the end of the activity, which we can do by catching the click on the Stop button. We create a new method in `js/activity.js` called `onStop()` to handle this, which includes the above code to save the state. Then, we add an event listener to the Stop button in `index.html` to call this method when the button is clicked:

```
1  <sugar-toolitem id="stop-button" title="Stop" class="pull-right" v-on:click="onStop"></sugar-toolitem>
```

the rest of the process is described [here](#).

Dependencies:

- **Vue-chartjs:** Vue-chartjs is an external library used to render charts in Vue.js applications. It provides an abstraction layer over Chart.js, which allows for greater flexibility and customization.
- **Html2canvas:** Html2canvas is an external library used to capture screenshots of web pages in a canvas element. It allows for easy exporting of web pages as images or PDFs.
- **Jspdf:** Jspdf is an external library used to generate PDFs from HTML content. It provides a simple and easy-to-use interface for creating and customizing PDF documents.

What is the timeline for the 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 canceling projects which are not mostly working by then.

Timeline

Time Period	Plan
Week one/two	<ul style="list-style-type: none"> ➤ Explore more about sugarizer and technologies in which sugarizer activities are build.
May 12 - May 28, 2023,	<ul style="list-style-type: none"> ➤ End-Semester Examination ➤ Discuss Chart Activity and word-puzzle template strategy with the mentor(s) ➤ Get to know about mentor(s) better. ➤ Ask about the actual design and grab more about the features by mentor(s). ➤ Learn more about the internal working of sugarizer from the mentor(s) ➤ Create the initial setup and set up the project in the GitHub repository.
	<p>Coding Period May 29, 2023 - September 28, 2023,</p>
Week One	<ul style="list-style-type: none"> ➤ Setting up the development environment in sugarizer for implementing chart activity, then build some UI

Week Two	<ul style="list-style-type: none"> ➤ Continue Adding features in the toolbar of the chart activity using the sugar-toolbar component, and working on chart container UI.
Week Three	<ul style="list-style-type: none"> ➤ Adding functionality in the toolbar features like adding and removing data to the list which will be rendered beside the chart ➤ resolving errors and bugs.
Week Four	<ul style="list-style-type: none"> ➤ Will be working on the rendering chart using the list data and setting up the level and its color based on the user team color, ➤ Also Will be building all 4 chart types and mapping the data model to mean how it will be interpreted internally.
Week Five	<ul style="list-style-type: none"> ➤ Implementing the text formatting features in the toolbar with full functionality ➤ Implementing features like customizing the label and color of the charts using the color palette
Week Six	<p>Mid Evaluation (July 10, 2023 - July 14, 2023)</p> <ul style="list-style-type: none"> ➤ Implementing features to export chart as any selected media types. ➤ Fixing errors and cleaning up the code by removing the redundant lines & logic

Week Seven	<ul style="list-style-type: none"> ➤ Implementing feature share activity with other users and also implementing, saving the state of the application to not to lost data when leaving
Week Eight	<ul style="list-style-type: none"> ➤ Start working on word puzzle template exerciser activity, build UI for the for a form of word puzzle template ➤ Also adding functionality in the form of word puzzle.

Week Nine	<ul style="list-style-type: none"> ➤ Working on the UI for the exercise in the play mode, ➤ Also implementing the logical part for the word puzzle, and, fixing errors and bugs till current development
Week Ten	<ul style="list-style-type: none"> ➤ Working on the UI Section of the Score evaluation section. ➤ Adding functionality to the score evaluation mode.
Week Eleven	<ul style="list-style-type: none"> ➤ Working on the UI for the word puzzle in edit mode, also implementing the functionality in the edit mode ➤ Also Implementing test exercise mode of the puzzle.
Week Twelve	<ul style="list-style-type: none"> ● Test and resolve bugs then clean up the code. ● Submit final deliverables. ● Create a summary and project report
	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, I am also a coding enthusiast who loves to implement new ideas. I have been **developing** web **applications** for a long time now, with a tech **stack** focused on **MERN**, and I have a solid working knowledge of node.js , **react.js** and **vue.js** . I have previous **experience** working as a web developer for a **startup** called "Doubtfree Learning," where I was responsible for developing their e-commerce platform for gifts. In addition to my professional experience, I am also ranked in the **top 6%** of competitive **coders** on [LeetCode](#) **globally**, which has helped me develop my problem-solving and algorithmic thinking skills. I am confident that my technical skills, passion for coding, and past work experience will

enable me to successfully complete my project within the given timeline

Also I completed both the **Sugarizer Vanilla Javascript activity development tutorial** and the **Sugarizer Vue.js activity development tutorial**. This helps me understand how the Sugarizer core works and how the activity is implemented. Following is the link to my repository of the implemented tutorials.

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

Activity Tutorial Video Link :

<https://discord.com/channels/1078051575580336249/1078054265517506681/1088168362137358477>

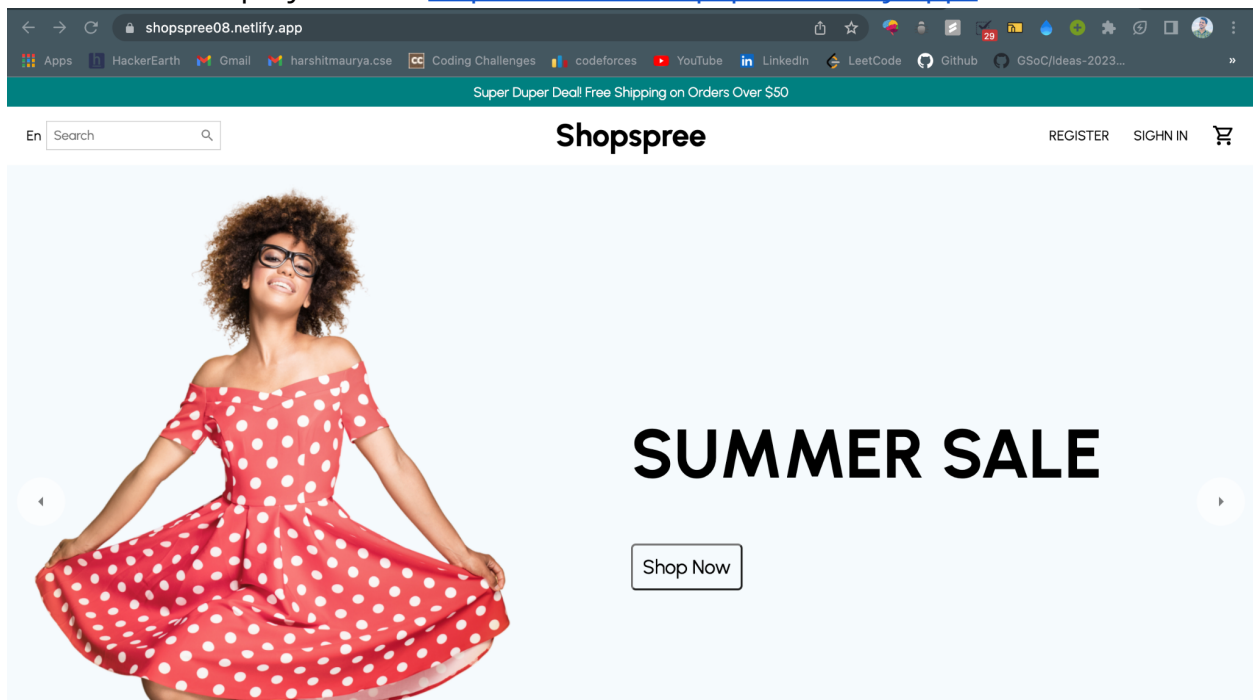
These are some of the major web-based projects that I have built:

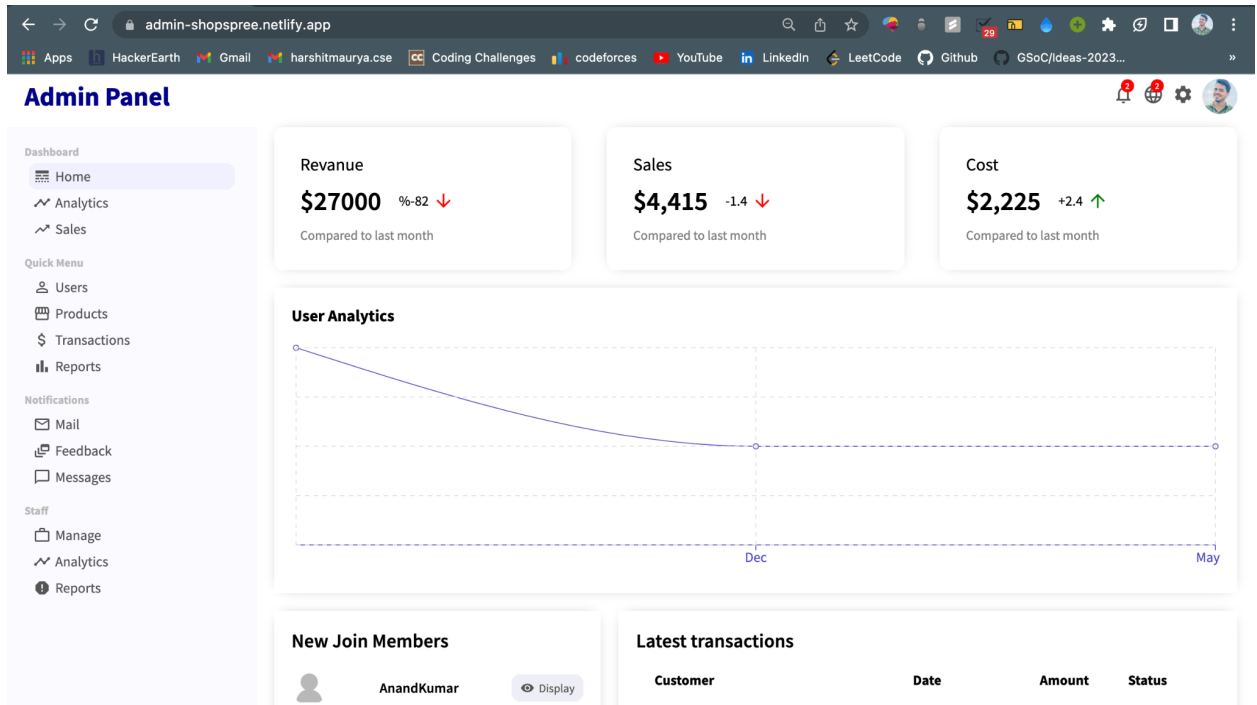
E-Commerce Platform: An e-commerce platform integrated with **payment gateway** made with **React.js**, Tailwind Css, Redux, Stripe, Email Js, Material Ui for the frontend and **Node Js**, Express, MongoDB, and JWT for the backend. It also has an **Admin Panel** to edit and add products on the store. social media platform just like Instagram, which connects people on the web. Built on MERN stack.

Link: <https://github.com/rockharshitmaurya/e-commerce-application>

Deployed Link: <https://shopspre08.netlify.app/>

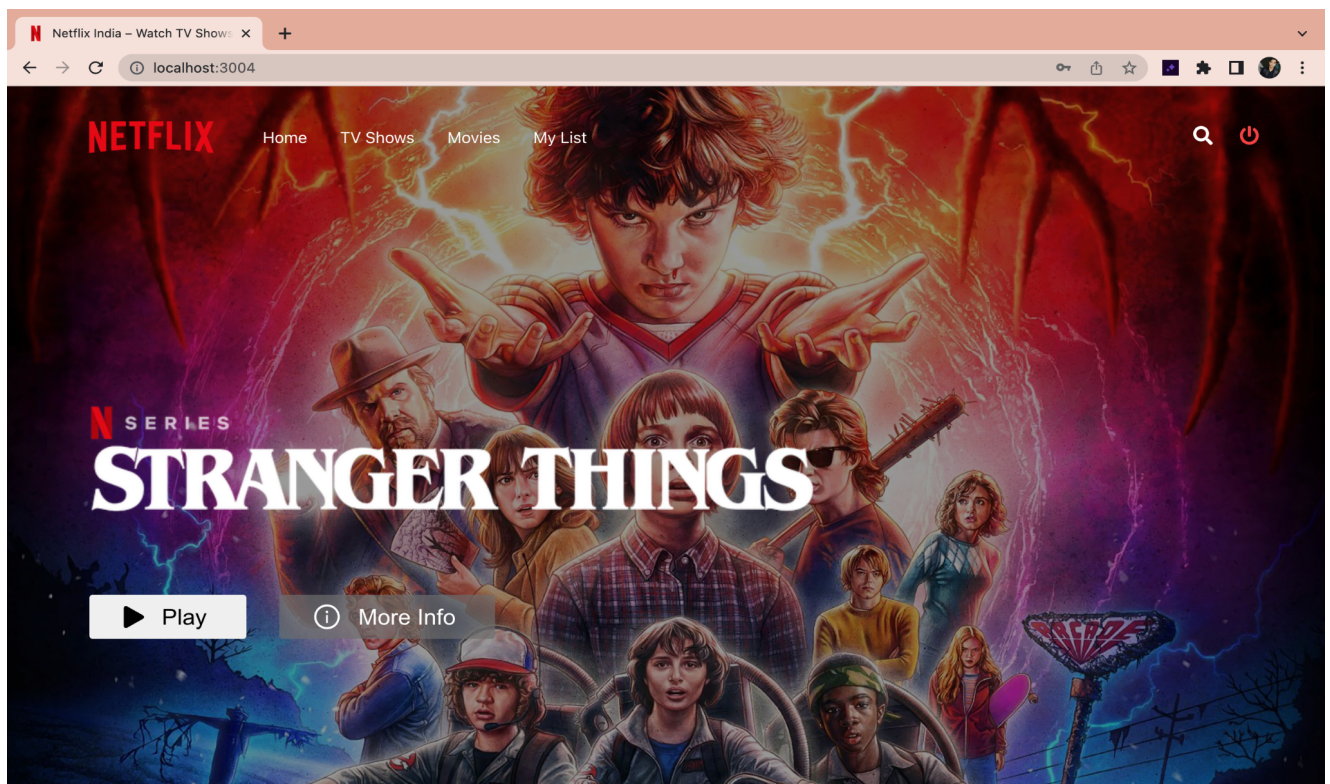
Admin Panel Deployed Link: <https://admin-shopspre08.netlify.app/>

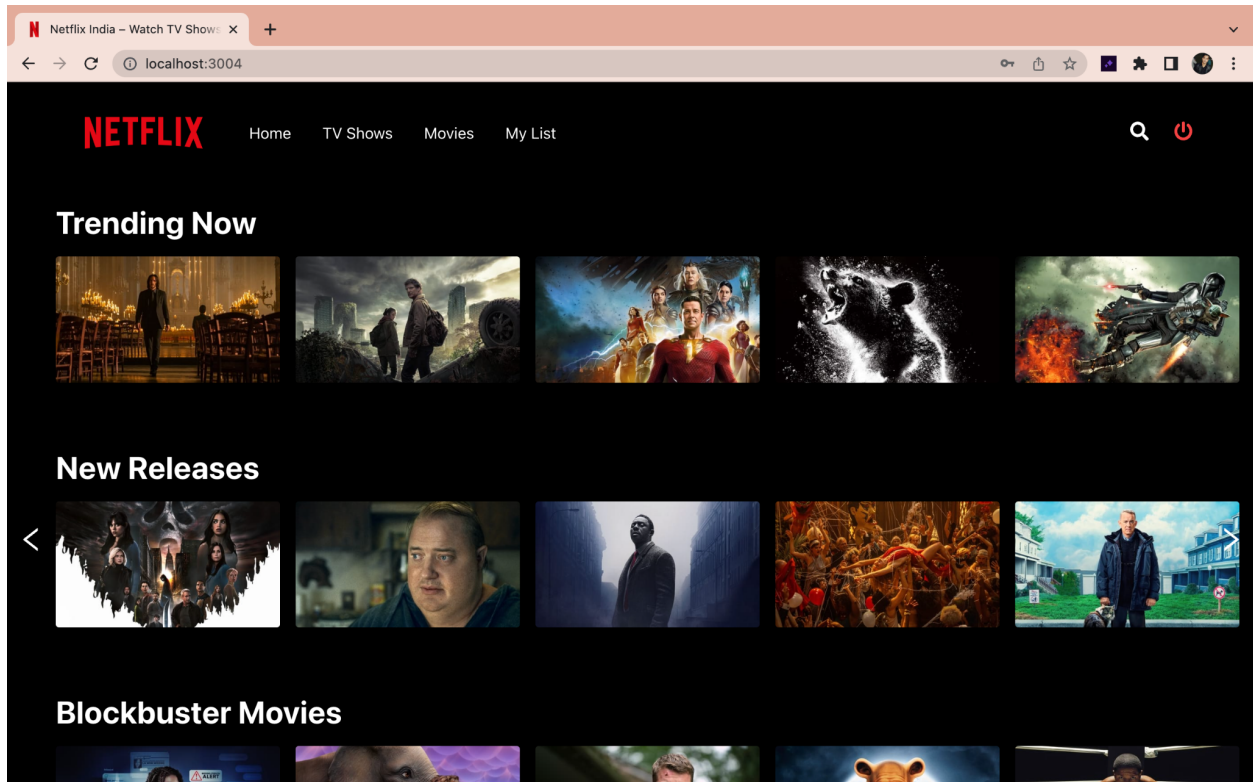




Netflix-Clone : This project is a Netflix UI clone using the MERN stack, enabling users to browse and search for movies and TV shows, view trailers and summaries, and create watchlists..

Link: <https://github.com/rockharshitmaurya/netflix-clone>





You and Community

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

In such a situation, I will first try to solve the problem myself by searching on the web for suitable solutions. If this doesn't work, then I will contact other developers on Element on the Sugar channel. And I experienced and noticed that the members

of the community are quite responsive and I believe that they will help me out. Also, I have some contact with other senior developers in college who have good development experience to help me out. Also, I have contacts with some experienced developers in the industry that I made during my past internship who can help me out.

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 keep a writing blog about the project where I will post updates on progress, obstacles faced, and their solutions., I will make regular requests to pull on the sugarizer, so that anyone at org can watch my progress. For problems or questions, I will use Element (IRC) as I have been using it for a long time during the suggestion and keep in touch with my advisors and text them in Element.

FAQs

- **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 20th May - to 30th May 2023.

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

I will be working 6 hours on weekdays from 9 am to 3 pm (IST) and 6 - 7 hours on Saturday from 9 am - 4 pm (IST). In total, ~50 hours per week.

---END OF PROPOSAL---