GSoC'22 Project Proposal

Sugar Labs: Sugarizer Vue.js UI

Saumya Kushwaha

Basic Details

Full Name:

Saumya Kushwaha

Emails and Contacts:

Primary email: saumyakus@gmail.com,
Secondary Email: saumya.kushwaha.cd.cse20@itbhu.ac.in
Github: S-kus
Matrix Username: Saumya Kushwaha
LinkedIn Profile: Saumya Kushwaha
Phone: (+91) 7380969660
Resume: Resume link

Your First Language:

My first language is Hindi but I am proficient in speaking, reading, writing, and understanding English.

Location and Timezone:

Location: Varanasi, Uttar Pradesh, India
Timezone: Indian Standard Time (UTC+5:30)

Communication:

The time I will be comfortable working with:

- UTC 04:00 - UTC 07:00 (IST 09:30 - IST 12:30)
- UTC 08:30 - UTC 14:00 (IST 14:00 - IST 19:30)
- UTC 16:00 - UTC 19:30 (IST 21:30 - IST 01:00)

I can start my day 2 hours early or late if it helps to communicate with other developers and mentors, and I will be reachable anytime through my Mobile No. and Email.

Education Details

I am a Computer Science and Engineering undergraduate student at the **Indian Institute of Technology Varanasi** (BHU) pursuing a Bachelor of Technology in my second year. I was introduced to the world of programming and software development in my first year. Since then, I have explored various fields such as Web Development, Data Structures, Algorithms, Computer Architecture, and Operating Systems. I also have been contributing to open-source since my 2nd year. For most of my programming journey, I have worked primarily with Web-based technologies and C/C++, Python programs, and I have recently been learning Machine Learning as well.

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

I got introduced to open-source projects and community in the second year of my undergraduate studies. Since then I kept myself involved in developing and learning about software and technologies.

I have been contributing to **Sugar Labs** for the past **4 months.** During this time, I have contributed to many repositories and fixed documentation, bugs, UI changes, enhancements, and updated versions and packages. These past four months have been a great learning experience. These are my contributions to **Sugar Labs**:

Pull request link	Description	Status
#347	Ported the official website from bootstrap version 3 to 4	Merged
#344	Updated SOAS hyperlink and added Trisquel	Merged

#3018	Added scrollbar to block-window in musicblocks	Merged
#3016	Updated "help window carousel" with the title of blocks	Merged
#3009	Enhancing "take a tour" and "usage-guide" modal	Merged
#997	Vanilla and Vue.js activity development Tutorial Improvement	Merged
#104	Fixed the bug related to arrow functionality in Exerciser Activity	Merged
#102	Improved the 'buttons' for exercises	Merged
#8	Created readme.md	Merged
#472	Added scrollbar to block-window in turtle blocks	Merged
#470	Corrected "mouse icon SVG" to "turtle icon SVG"	Merged
#11	Updated readme	Merged
#350	Enhanced Profiles Page	Merged
#371	Fixed mySlides error in console	Merged
#361	Updated 'Sugarlabs' to 'Sugar Labs'	Merged
#356	Removed jsquery.nicescroll plugin and added custom styles	Merged
#355	Removed unused 'modernizr' plugin and fixed related errors	Merged
#1056	Corrected overlapping of back-button in some activities	Open

I have made over **37 commits** till now in Sugar Labs. All of these merged commits and issues can be found here.
Out of these contributions in Sugar Labs, the most significant one is Porting the official website from Bootstrap 3 to Bootstrap version 4. In this pull request, I fixed the

broken UI elements and also corrected the div alignment problem. This problem happened because of the flexbox class in the bootstrap version 4. This porting will be helpful for migration to the latest version because in bootstrap version 4 flexbox is introduced for the first time and hence significant changes were done. In this task, I have added 13,061 and removed 228 lines.

I worked on some of my personal projects for learning purposes and I have also participated in HactoberFest'21. Other than this I also contributed to many open-source projects. Following are some of my open-source contributions to different organizations.

- https://github.com/LesFruitsDefendus/saskatoon-ng/pull/179
- https://github.com/LesFruitsDefendus/saskatoon-ng/pull/187
- https://github.com/COPS-IITBHU/sdg-site/pull/
- https://github.com/COPS-IITBHU/sdg-site/pull/37
- https://github.com/COPS-IITBHU/cops-django-youtube/pull/9

Details of my other contributions can be found on my Github profile here.

Convince us that you will be a good fit for this project, by sharing links to your contribution to Sugar Labs

I am an active contributor of **Sugar Labs** since **Dec 2021** with over **37 commits** merged in various repositories. Spending this much time with the codebase helped me to understand it in a better way.

- Commits: 37 (13,304++, 669--)
- Issues: 13 (10 closed, 3 open)
- Pull Request: (17 closed, 1 open)

I am applying to Sugar Labs for Google Summer of Code 2022, because of the community relations and the work carried out by this organization. Here everyone discusses and develops as a team and everyone's opinion matters. It really makes you feel at the peak when you know that your work is going to contribute to the social cause and will impact society in a positive manner.

Prerequisites for Project:

As given in Idealist. I already have experience in HTML5, Javascript, and Vue.js framework development. Other than this I am already familiar with the Sugarizer codebase and have merged commits to the repository.

I am a full-stack **MEVN developer**. I have already worked on many projects in the Vue.js framework and also HTML5/Javascript. Following are some of my major projects related to the Vue.js framework:

- **Project Labz**
  Project Labz is a website for managing and sharing projects. In this, a user can make a particular type of project list (exa. ML, Dev, DSA related) and can add, edit and delete projects from the Project List made by them.
  Technologies Used: Vue3 framework, CSS, HTML, Javascript, Firebase for database and authentication functionality.
- **SDG-site**
  Being part of the **Club Of ProgrammerS (COPS) IITBHU**, I contributed to the official website of its Software Development Group. Contributed to making the Official website SEO-optimized and user-accessible by adding animation and improving UI.
  Technologies Used: Vue/Nuxt.js framework, SCSS, Javascript.
- **Technex'22 Website**
  Technex is Techno-management Fest organized by IIT BHU, India. Being a part of its Tech Team, I was responsible for making the official website. Added the events page and created cards from scratch.
  Technologies Used: Vue/Nuxt.js framework for frontend, Django Rest Framework in the backend, and firebase for storing data as technologies.

I have been doing web development for the last 1.5 years. This is not an exhaustive list of all the projects. I have also participated in "The Digital Alpha's SEC Filing Analyzer for SaaS Companies" competition for the UI team at Inter IIT Tech Meet. My other project work can be found on my Github profile here.

I have already contributed to the Sugarizer project. These are some of my merged pull requests related to **Sugarizer and its activities**:

- https://github.com/llaske/ExerciserReact/pull/102 (merged)
  Improved the 'buttons' UI of exercises
- https://github.com/llaske/ExerciserReact/pull/104 (merged)
  Fixes the bug related to toggle arrow in toolbar
- https://github.com/llaske/sugarizer/pull/997 (merged)
  Vanilla and Vue.js activity development Tutorial Improvement
- https://github.com/llaske/sugarizer/pull/1056 (open)
  Fixes the bug of back-button overlapping in Tangram and Curriculum activity

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/S-kus/Pawn-Activity-Vue.js
- https://github.com/S-kus/Pawn-Activity-VanillaJs

All of these make me familiar with the Sugarizer codebase and comfortable working with Sugar Labs. It also helps me to understand how things actually work internally and how various components are connected.

Project Details

What are you making?

The aim of this project is **to create a framework of Vue.js UI components matching the Sugar UI** by replacing old **EnyoJS, a deprecated framework initially developed for WebOS.** So, I will be:

- Rewriting the required listed (down below) files in Vue.js by replacing the EnyoJs framework. (**Required**)
- I will be replacing the bootstrap tour library with the new one which will be easy to use and maintain. It will also be matching with the current UI of the tutorial popup. (**Required**)
- Implement basic UI components for exa: button, Entryfield, checkbox, etc in the Vue.js framework. (**Optional**)

- Updating the "Sugarizer Vue.js activity development tutorial" in Vue3 from Vue2. (**Optional**)

For this, I will be rewriting the following files of 'js/' directory:

- icon.js
- searchfield.js
- iconbutton.js
- popup.js
- selectbox.js
- palette.js
- password.js
- audio.js
- dialog.js,

and tutorial.js from the 'lib/' directory.

Other than this I will also be migrating Button, Entryfield, Checkbox etc and other small files which are currently based on EnyoJs to the VueJs UI framework.

Details of these files and how the changes will be written are described below:

➢ **Icon (`js/icon.js`):**

It's a class for Sugar icons based on the existing kind, `enyo.control` in EnyoJs. It's used to render SVG icons of buttons, activities, menus, neighborhoods, etc. Now in VueJs:

- For the template of icon component as it's using mainly two components in EnyoJs of namely: `icon` and `disable`. So there will be two `div`s with the class of `web-activity-icon` and `web-activity-disable` with icons ids.
- These icons will be using two main functions that will come under `methods` of VueJs for event `v-on:mouseover` and `v-on:mouseleave` named: `popupShowTimer` and `popupHideTimer` which popup `itemlist` component of **popup.js** i.e. description including activity link for it.
- For iOS and chrome android we can use `v-on:touchstart` and `v-on:touchend` in place of mouseover/mouseleave.

- As in EnyoJs, Rendered is called when a component is ready to be placed into the DOM, So we can use the `mounted()` hook for touch events and functions to get the colorized color of the icon.
- For the 2nd `div` with a class of `web-activity-disable`, we will be using methods and dynamic styling to change in the background dynamically on the basis of whether `div` is active or not.
- At last, this component will rely on direct SVG rendering. For this, we will be using a similar method as in `Xmas Lights activity`. So, for this firstly we will create a list of the CSS class to match all possible XO buddy colors using custom properties `--stroke-color` and `--fill-color`. Similar to this,

  Now, We need to use the `use` tag and the `xlink:href` attribute. To include an SVG file as if it was directly in the code.
- Other than these we will be using other simple functions in `methods` like checking if the cursor is inside the `icon` or not, also to check related to Native things.

➢ **IconButton (`js/iconbutton.js`):**

It's a class for a Sugar button with an icon and a text under it and based on the existing `kind`, `enyo.control`. Use in dialog screen, first screen, and for the empty journal.

- It's using two components under the components of EnyoJs, namely: `icon` and `text` and class of `icon-button`. Therefore, for the template of this component, we will be using a `div` and a `p` inside the parent `div`. The inside `div` will be using the template of the **icon.js** file for rendering icons by importing it as a parent component, then for text a simple `p` tag with the class `icon-button-text`.

- There will be a condition to check a language direction So that we can easily add `rtl-10` class if the `l10n.language.direction=="rtl"` condition is true.
- Then at last there is a function for colorizing the icon.

➢ **Searchfield (`js/searchfield.js`):**

A class for search with blur and focus feature of icons on the basis of relevance and non-relevance. It's also based on the existing kind, `enyo.control` in EnyoJs.

- For the template in VueJs, it's using three `div`s. In which
  - A magnifier icon with the class of `search-field-iconsearch`, then a field for text-input of class `search-field-input` with the features of `onfocus` and `onblur` to disable or inable the icon as required.
  - An icon of cancel button with a class of `search-field-iconcancel` will show only when `inputString.length >0`.
- In the input text part, we will be using the `Computed` property of VueJs, this will help to monitor input text so that we can modify icons accordingly by adding and removing `search-field-border-nofocus` and `search-field-border-focus` classes as required i.e. relevant and have similar sub-string as the title of activity or icons.
- There will also be a click event listener function in `methods` for cancel icon `div`, to set input string length back to 0.

➢ **Popup (`js/popup.js`):**

It's a class for a Sugar popup menu with a header and class `home-activity-popup` and based on the existing `kind`, `enyo.control`.

- Its template will basically consist of three parts:
  - Firstly Header, with an icon on the left of fixed size and based on already defined kind `Sugar.Icon` i.e. based on **icon.js** component. It also has a click event listener for further actions.

- Other than this there are two more parts in the header: name and title tags on the right side with a class name of `popup-name-text` and `popup-title-text` respectively.
- Then an optional component for list items. It has a class of `popup-items` and is based on the kind named `Sugar.DesktopPopupListView`. It's also based on a scrollable `div` in EnyoJs. So, there will be another child component in VueJs for ListView. Its template is the list of items with icons and name tags, with the class named `item-list-item`.
- Then a footer `div` of the name `footerlist`, which is currently based on the kind named `Sugar.DesktopPopupListView`.

- There is a timer function in `methods` for this popUp to appear and hide, according to the cursor over the icon.
- At last in the ListView child component, we add a loop for `item in itemList`, So different works of that activity will appear on the popup screen.

➢ **Select box (`js/selectbox.js`):**

A class for Sugar select box, which is based on `enyo.control`. Also currently used only to select a language in language settings from the dialog. It have the class named `selectbox-border`.

- In the template, Inside a list of options, each item consists of two parts, the first icon on the left and based on the already defined kind `Sugar.Icon` i.e. based on **icon.js** component. The second is the text or title of the option.
- This component is based on `Sugar.Popup` i.e. **popup.js** file. It will also contain a function in `methods` for click or touch event, to start a timer and show the select box popup window.
- It will receive an items array as props for rendering the options. The selected item (or default selected item) will appear at the top by setting its `index` to 0.

➢ **Password (`js/password.js`):**

This file contains two classes, `Sugar.Password` for password and `Sugar.Emoji` for emojis code in EnyoJs. It is a class for password field in

Sugar and based on `enyo.control`. It is a specific type of box that takes input as emojis of letters and numbers and contains a class named `password-class`.

- Currently, It contains 6 categories of emojis in which each of them has 10 emojis. At a particular time, a single category is displayed completely on the password box. At the sidebar of this box, there are three categories of emoji to go up or down to change the category. These emojis arrays are currency stored in the **constant.js** file.
- For the template of the box,
  - First, there will be a parent `div` with a class named `password-line`. Inside it, there will be three `div`s. First is a text label of the class `password-label`. Then a `div` for inputting the password also contains a cancel icon. At last, the set of 10 emojis and a sidebar for the category of emojis.
  - Each emoji block is based on a `Sugar.Emoji` kind in EnyoJs. So, we will be creating a child component for it. Its template contains the main `div` of class `emoji`, then inside it there will be two `div`s, one for emoji-icon and another one for emoji-letter.
- We will have functions in `methods` for clicking any category of the sidebar. Pseudocode for categories selection:

```
Category0clicked: function(){
    if(currentCategory==0)
    return;
    const categoryIndex= category0FirstEmojiInd
    if(category0FirstEmojiIndex-10>=0)
{
    category0FirstEmojiIndex= categoryIndex-10;
    category1FirstEmojiIndex= categoryIndex;
    category1FirstEmojiIndex= categoryIndex+10;
    currentCategory=1;
}
//code for updating emojiIndex of a category
}
```

Similarly, for the 1 and 2 index categories.

- To update the 10 emoji Index that are shown, we will get the current category starting index and update the 10 emojis from that index.
- If the user is clicking emojis to input, then we will be using the function `emojiClicked()` to get the emoji index and then `convertToChar()` to store its value. Similarly, if the user inputs the value in the password box using the keyboard, then by using `convertToEmoji()` we will show the respective emoji in the password box.
- In `Sugar.Emoji` VueJs component, we will be adding a function for animation and flash effect by adding some CSS styles using a timer to show the effect for a small duration when an emoji is clicked.
- We will be using emoji values stored in data to update the index of emoji to a new emoji.

➢ **Palette (`js/palette.js`):**

It's a component for the Sugar palette based on `enyo.control` and used for the filter option in the Journal screen toolbar.

- For template, in parent `div` two classes namely `palette-sugarizer` and `palette-down` will be added. Inside this for options of filter, there will be two `div`s. Firstly an icon on the left which is based on the already defined kind `Sugar.Icon` i.e. based on **icon.js** component and class `palette-icon`. Secondly, a text with class `palette-text`.
- Here I will also be adding a function in `methods` to get the active option id and will update the palette tool icon dynamically to the active option icon.
- To open and hide the palette, we will be using dynamic CSS classes `palette-down` and `palette-up`.

➢ **Audio (`js/audio.js`):**

A file for handling Sugarizer audio stuff and based on HTML5 `audio` element and Cordova. Firstly, this file has a main class, namely `HTML5.Audio` based on `enyo.control` and then a class using it also based on `enyo.control`, namely `Sugar.Audio` in EnyoJs. For the Vue component:

- In this, we are writing functions for handling volume buttons on Android by listening to event `volumeupbutton` and `volumedownbutton`.

- In the main component, the two main functions will be for the events: on ending of sound and for updating the timestamp.
- Smaller functions will also be there for loop, mute, pause etc, which will be using inbuilt methods and functions for action.
- For the play button on the web HTML5, `audio` can be used but for Android and iOS, Cordova `Media` Constructor will be useful.
- In the child component, there will be two main functions `broadcastEnd` and `broadcastUpdate` for updating sound length. It will be according to whether the sound has ended or update the new timestamp if it has changed.

➢ **Dialog (`js/dialog.js`):**

This class is used for the settings dialog, named `Sugar.DialogSettings` and based on `enyo.Popup` kind in EnyoJs. It has a class named `settings-dialog`.

- For the template of it in VueJs:
  - It will contain two `div`s. First a toolbar in which a search bar based on `Sugar.SearchField` kind i.e. **searchfield.js** file and a cancel button for closing the dialog based on `Button` kind. Then a second `div` which will have 8 components for different types of settings namely: `Aboutme`, `about my computer`, `aboutserver`, `security`, `privacy`, `language`, `androidSettings`, and `resetLauncher`.
  - `androidSettings` and `resetLauncher` are for android and iOS.
- In the main class i.e. `settings-dialog`, in `mounted()` hook we will forbid the resizing of home when dialog popup is shown. By code: `app.noresize = true;`
- There will be a function in `methods` to filter the dialog components, i.e. to show only those types of settings icons whose title is matching with the search input string.
- Other than this, there will be a click event listener at each type of setting. On clicking their icon, that particular component popup dialog will open.
- For the **About me** dialog,

- In the template, firstly there will be the parent `div` of class `module-dialog`. Then inside it, there will be two `div`s. Firstly for the toolbar, there will be:
    - A module-language icon based on `Sugar.Icon` i.e. based on **icon.js** component.
    - A title
    - A cancel button for closing the dialog
    - An OK button for saving the current settings and closing.
- Now in the second `div`, there will be two `div`s and a `p`. In the first `div`, there will be 5 owner-icons. The first two i.e. `psicon` and `nsicon` are used to set the current stroke color from the middle owner-icon (which will be storing current changes) and different fill color combinations. The last two `pficon` and `nficon` are used to set the current fill color from the middle owner-icon and different stroke color combinations. The second `div` is for changing the current user name.
- To set the color combination of `psicon`, `nsicon`, `pficon` and `nficon`, we get the middle or current fill and stroke color and then get their combination randomly from the data. Here we will be ensuring that for a particular owner-icon, the fill and stroke color combination didn't match.
- In the OK function in the `methods`, we will update the owner-icon combination, save it and render it on the home screen.
- For updating the username, after getting the input, we will trim the extra space by using `inputUsername.trim()` and then save it.

- **My Security** dialog is used to change the login image password, but firstly, one has to enter the current password and then the new one, and once it's updated a final message for success will be received. This dialog will be present in the main popup settings only if the user is connected to the server.
    - In the template inside the parent `div` of the class `module-dialog`, there will be two `div`s. First for the toolbar, which will be similar to others, and the other for content.

        - In content, there is an instruction message and then the password box based on `Sugar.Password` i.e. **password.js** file. Other than this, the spinner `div` for loading state, buttons, and warning message in case of error will be present.
    - Now for the `next` button, after getting the password from the input, we will set the loading to be true and meanwhile, we will check from the server if the login password and the input password are same or not. After that, we will set the loading to be false and if they are same, then we will render the next screen for changing the password else will show the warning text popup.
    - For a new password, after input, we will set the loading to be true then we will check whether it's completing the required length size or not, and if it's correct we will update the new password and set the loading to be false. In case of any error, the warning message will be displayed.
- **Language Dialog** is used to choose the language and it depends on `Sugar.SelectBox` i.e. **selectbox.js** file.
    - In this component's template, there will be a parent `div` of class `module-dialog`, then inside it, there will be two `div`s. First for the toolbar, there will be:
        - A module-language icon based on `Sugar.Icon` i.e. based on **icon.js** component.
        - A title
        - A cancel button for closing the dialog
        - An OK button for saving the current settings and closing.

      Then the second `div` is for the content. It firstly contains a `p` for instruction and secondly the select box based on the **selectbox.js** component.
    - In this component, we will be storing data of languages options and sending this array of languages as props to the selctbox.js file.
    - If the language selected from the box is different from the current language, then we will update the language to the index 0 and

restart the app by rendering it to the home screen. Else on clicking OK we just close the dialog popup.

- In the **About My Computer** dialog, which shows about the current device and its details about copyright and license.
  - In the template inside the parent `div`, firstly we will have a `div` for toolbar similar to others, then a warning component `div` which is based on `Sugar.DialogSettingsWarningBox` i.e. warningbox component. And then there will be a `div` for all software and copyright details.
  - In the content `div`, we will have details about the Sugarizer version, computer-client type, computer browser as well as its version and computer storage. After that a checkbox for Reinit journal and settings. Then after an `hr`, information about copyright and also a link to see all contributors.
  - We will dynamically set the data about the computer and in `reinitcheck` function, if the checkbox is checked we will clear the stored data in the journal and an alert message at the bottom will be shown. So, after clicking `OK` on the toolbar, it will restart the app with these changes.
- In the **About My Server** dialog, there is a setting using which we can connect to the server and can be in shared mode and interact and play multiplayer games with other users connected to that particular server.
  - In the template, after the toolbar similar to other dialogs, there is a checkbox. If it's checked or we are already connected to the server then the `div` about server name, address, description, and our username will be shown. Else only the checkbox will render and if the user has checked it, then only other components for taking input will render, i.e. user will have to enter the server name he wants to connect to. Then for connection user have to enter its image and password. Other than this, there is a warning box `div`, which will be rendered after changing the checkbox value and clicking the OK button. It has two buttons: **Restart Now** to apply changes and **Cancel Changes** to cancel changes. Currently, at the time of input, there is also a `div` for the QR code.

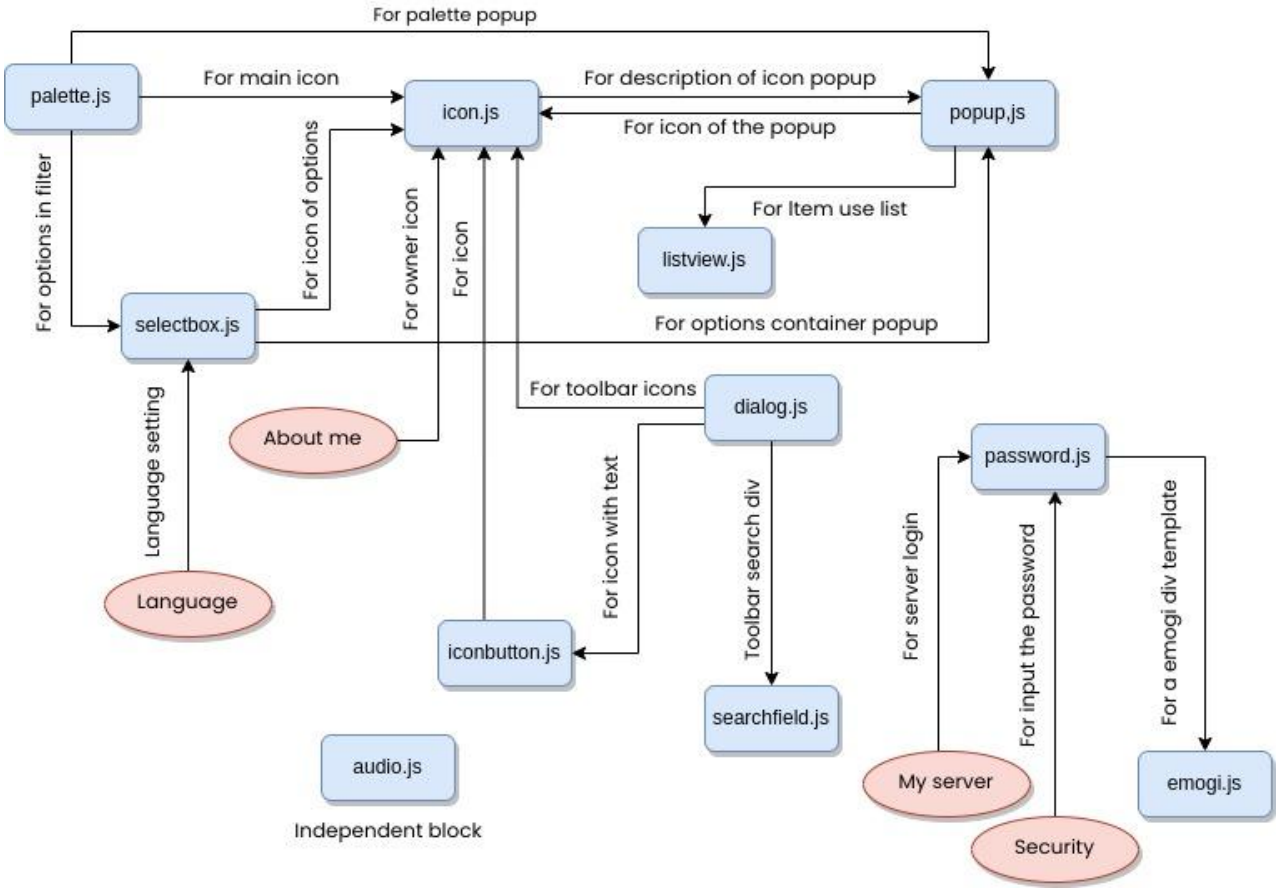
    - We will be using all these components in steps:
        - In step 0, we show the `PleaseConnectMessage` i.e. to connect to the server.
        - In step 1, input the server URL.
        - In step 2, input the password image.
        - In step 3, after clicking the OK button we will render a warning message to restart the application.
    - For connecting the user to the server by using the username and image password, we get its color, its private and shared journal data from the server and then update the app and render the home screen with all this information.
- For **My Privacy** dialog, which is used for some personal settings related to sharing activity work to the server and also for deleting the local or server account.
    - In the template of this dialog, there will be a toolbar and warning box similar to other dialogs. After that in the content `div` we will have three checkboxes, in which two are related to the server so will be displayed only when the user is connected to the server. The third one is about deleting the account, so it will show option to only delete the local account if the user is not connected to the server. Else user can see both buttons i.e. to delete the local or server account.
    - For deleting an account there will be a warning message to restart or cancel changes. But for the other two checkboxes privacy will be just updated after clicking the OK button.
    - For deleting the local account, we just need to clean the local datastore by setting it to null, but for deleting the server account, after the function of deletion from the server, we will also have conditions to catch any server-related error if the function doesn't perform perfectly.
- Now for **Launcher** dialog, which is for android and iOS:
    - In the template, there will be a parent `div` of class `module-dialog`, then inside it, there will be three `div`s present. First for the toolbar, which contains:

        - a launcher icon based on `Sugar.Icon` i.e. based on **icon.js** component
        - A title
        - A cancel button for closing the dialog
        - An OK button for saving the current settings and closing will be present.

      Now, 2nd `div` is for the warning box, and the third is for `launcher-message` and a nested `div` for `launcher-icons` class, which are again based on the **icon.js** component.

    - To set the launcher, we will add the class `selected` dynamically on clicking `nativeIcon` or `sugarIcon`.
    - If any error happed, a warning box `div` will display or if everything works fine we will restart with the selected launcher and hide the dialog popup.
  - The template of the **Warning Box** component will have the parent `div` with a class named `settings-warningbox` and contain mainly two `div`. The first one is for Title: "Warning" and for the warning message. The second one contains two buttons, one for "cancel changes" and another one for "restart now".

So these files' interconnection can be summarized in the image given below:



➢ Tutorial (`lib/tutorial.js`):

Intro.js tour library can be used in place of the bootstrap tour. It is a lightweight JavaScript library for creating step-by-step and powerful customer onboarding tours and live demos similar to bootstrap-tour.

Similar to current tutorial.js, we can keep a specific icon button to start the tour of the app, by using `introJs().start();` inside the function which is called on clicking the icon to start the tour.

Comparison of bootstrap-tour and intro.js:

Comparison	bootstrap-tour	intro.js
Stars	4.4k	21.1k
Forks	956	2.6k

Latest Version	30 Sep 2017	5 Apr 2021
Last Commit	4 years ago	11 days ago
Language	CoffeeScript	JavaScript

Other than these files I will also be implementing the following two important features:

1. **An encapsulation for basic UI components: Button, Entryfield, Checkbox, Popup, etc:**
   Currently, `enyo.Button`, `enyo.Checkbox` etc are used in the above files. As above files will be written in VueJs replacing EnyoJs. It's better if we write buttons, checkboxes, etc components that are used in these files in VueJs too so that these files become completely independent from EnyoJs.
   For this, I will be writing UI component files of buttons, Entryfiels, checkboxes, etc.
2. **Sugarizer Vue.js activity development tutorial to vue3 from vue2:**
   Currently VueJs tutorial for the development of Pawn activity is written in Vue2, But Vue3 is the new default version of VueJs. It would be better if the tutorial will be written using the latest version else it might happen in the future that a particular function that is not supported anymore, will break the code.

How will it impact Sugar Labs?

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. Currently, Sugarizer's core UI relies on the EnyoJs framework. It is a very old deprecated Javascript framework for cross-platform mobile, and desktop. It's not updated frequently and its documentation for working is also not beginners friendly. In this project, I will be rewriting some .js files written in EnyoJs to VueJs. This will help Sugarizer to replace EnyoJs in future versions.
VueJs documentation is easy to follow and it's updated from time to time. Vue is easy to maintain and considering future prospects it is necessary to have a codebase easier to understand and contribute, to let new contributors comfortably

come in. Also, it will be a starting step to shift to VueJs from EnyoJs. This framework will also help in the smooth functioning of Sugarizer and its activities.

What technologies (programming languages, etc.) will you be using?

The Major part of the project will involve coding in the Vue.js with this I will also work on some styling changes in CSS to match the new component template with the current Sugar UI. Other than this, for tutorial.js, bootstrap tour library migration will require some frontend work in CSS to match the current tutorial UI and tour box.

Timeline

Break down the entire project into chunks and tell us what will you work on each week.

Days	Tasks
Pre GSoC Mar 7, 2022 to May 20, 2022	➢ Learn the working of EnyoJs and understand the working of files and Sugarizer core architecture ➢ Familiarize myself with the current implementation ➢ Stay connected with the community and contribute to Sugar Labs
Community Bonding Period May 20, 2022 to Jun 12, 2022	➢ Discuss the creation of the testing app for this project and learn about it ➢ Go through documentation of EnyoJs and VueJs for code understanding about the integration to android and iOS platform ➢ Discuss the final tutorial library to implement tutorial.js and learn about it ➢ Also, go through the merged and in-progress PR's to have a better understanding and discuss doubts related to it with mentors. ➢ Explore the tech stack required for the project.
Week 1 Jun 13, 2022 to Jun 19, 2022	➢ Start working to Integrate testing app to check the working of rewritten files ➢ Discuss sequence and set up to write the files and

	commit the changes
Week 2 Jun 20, 2022 to Jun 26, 2022	➢ **Start writing icon.js file** As most of the components depend on it, it's important to start with this file frist
Week 3 Jun 27, 2022 to Jul 3, 2022	➢ **Implement direct SVG rendering feature** As this icon.js component will rely on direct SVG rendering, with this icon.js file will be completed
Week 4 Jul 4, 2022 to Jul 10, 2022	➢ **Complete iconButton.js** The iconButton.js component is based on icon.js, so after completing icon.js, it would be easier complete it
Week 5 Jul 11, 2022 to Jul 17, 2022	➢ **Make new Listview.js to use in the popup.js** To start popup.js file, first the content component i.e. child component needed to be completed
Week 6 Jul 18, 2022 to Jul 24, 2022	➢ **Start working on popup.js** Popup.js file depends on icon.js and listview.js, both had been done, so popup.js can be completed this week
Phase 1 Evaluation Jul 25, 2022 to Jul 29, 2022	➢ **Completed popup.js and icon.js** Icons rendering with popup timer function and the popup will be ready by this time
Week 7 Jul 25, 2022 to Jul 31, 2022	➢ **Start working on selectbox.js** As popup.js and icon.js both are completed, selectbox.js component can integrated
Week 8 Aug 1, 2022 to Aug 7, 2022	➢ **Complete palette.js** palette.js depends on icon.js, selectbox.js and popup.js, so it can also done now
Week 9 Aug 8, 2022 to Aug 14, 2022	➢ **Start working on emogi.js** Password.js will be depend on emoji.js for particular block of emoji with letter and image
Week 10 Aug 15, 2022 to Aug 21, 2022	➢ **Complete password.js file with emoji.js** As emoji.js completed, now we can complete password.js component
Week 11	➢ **Start Searchdfield.js**

Aug 22, 2022 to Aug 28, 2022	To use in the dialog.js toolbar it's should complete before dialog.js
Week 12 Aug 29, 2022 to Sep 4, 2022	➢ **Started working on dialog.js** ➢ **Complete warningbox component** This will be the most lengthy file, but can be started now as mostly dependent component are covered
Week 13 Sep 5, 2022 to Sep 11, 2022	➢ **Complete about me and language component** These child components of dialog.js for different settings, can be completed as icon.js and slectebox.js are covered
Week 14 Sep 12, 2022 to Sep 18, 2022	➢ **Complete My server component** Password.js is used here for server login, which is completed, so it can also be done easily
Week 15 Sep 19, 2022 to Sep 25, 2022	➢ **Complete my security component** It completely depends on password.js, so can be completed in a week
Week 16 Sep 26, 2022 to Oct 2, 2022	➢ **Complete about my computer component** An independent component but an important part of dialog.js
Week 17 Oct 3, 2022 to Oct 9, 2022	➢ **Complete my privacy component** Can be completed as only depends on icon.js and warning-box component
Week 18 Oct 3, 2022 to Oct 9, 2022	➢ **Complete my launcher and android setting components** Complete these two specific android settings icon in dialog.js
Week 19 Oct 10, 2022 to Oct 16, 2022	➢ **Complete dialog.js using all child components** Integrate all the small components related to dialog.js together in a file
Week 20 Oct 17, 2022 to Oct 23, 2022	➢ **Implement audio.js file** With a good knowledge of Cordova, now rewriting of this file would be easy
Week 21	➢ **Tutorial.js file**

Oct 24, 2022 to Oct 30, 2022	Integration of this file using the new decided library for tutorials and instructions
Week 22 Oct 31, 2022 to Nov 6, 2022	➢ **Complete integration of tutorial.js file** Finally completing this file with the required CSS styling to make it similar as Sugar UI currently
Week 23 Nov 7, 2022 to Nov 13, 2022	➢ **Start working on extra UI components** Implement basic UI components like Button, Checkbox, Popupbasic in the VueJs framework
Week 24 Nov 14, 2022 to Nov 21, 2022	➢ Cleanup and wrap up the work. ➢ Tutorial migration for development of tutorial in Vue from Vue version 2 to 3

How many hours will you spend each week on your project?

My college summer vacations start from May 12 to July 20. In this period I can give about 45-50 hours per week and after college starts, I can give about 35-40 hours per week. I have no other commitments for the summer vacation, so I can devote most of my time to GSoC.

How will you report progress between evaluations?

I will be active on GitHub as I will make regular pull requests to the Sugarizer while interacting with the mentors, so anyone in the org can view my progress. Thus, my progress will always be reported thoroughly on GitHub. I am also planning to write weekly or fortnightly blogs in which I will post updates about my progress, obstacles being faced, and their solutions. I will be reachable anytime through matrix IRC, Email, or a planned video session.

Discuss your post-GSoC plans. Will you continue contributing to Sugar Labs after GSOC ends?

I am planning to continue working on the project after the program ends. After this project, the VueJs framework will help to replace EnyoJs in future versions. So, there will still be many files and small UI components to complete. I am also planning to update the versions of Vue in many activities written in VueJs. I am amazed by the community relations and the work carried out by this organization. I vision to hone

my skills further and put them to use to give back to the community. I aim to develop mentorship skills and the ability to guide others and try to give back to the community by mentoring and guiding others. I hope to mentor future GSoC and GCI students.

I am looking forward to contributing to Sugar Labs this summer season.
Kind Regards.