Sugarizer Knowledge Activity Pack (GSoC 2020 Proposal)

Personal Details

Name: Utkarsh Raj Singh
E-Mail Id: u.rajsingh2503@gmail.com
GitHub Profile: https://www.github.com/utkarsh-raj
LinkedIn Profile: https://www.linkedin.com/in/utkarsh-raj25
First Language: English, Hindi (Native)
Location: India
Timezone: GMT +5:30

Open Source Contributions

Contributions to Sugarlabs

I have been an active member of the Sugarlabs community for more than a year. Started engaging and contributing in February 2019, and since then it has been a learning enriched journey. Mentored for Google Code In with Sugarlabs for Sugarizer Projects in December 2019. I am comfortable with the Sugarizer architecture and have built several patches:

- **#706**
  - **Built the Chess Activity as a part of the GSoC 2020 task**
  - **A full activity integrated in Sugarizer with Sugar Web, Bootstrap 3 and Presence**
- #386
  - Address the issue of non standard implementation of presencepalette.js in Memorize Activity
- #283
  - Fixed the issue of vanishing recent marks
  - The issue occured in the Stopwatch Activity for the recorded times vanished after a few recordings

- ➔ #282
    - ◆ Fixed the issue of responsiveness in the Memorize Activity
    - ◆ Tested for all modern viewport dimensions
- ➔ #280
    - ◆ Related to PR #283, discussed with the maintainers the justification of the issue
- ➔ #279
    - ◆ Fixed unexpected token in JSON Error and stopping the Activity from hanging
    - ◆ Applied data cleaning before parsing
- ➔ #278
    - ◆ Identified the vulnerabilities of RegEx DoS due to deprecated dependencies in the Developer Electron JS build for Sugarizer
- ➔ #277
    - ◆ Added the Click and Drag Feature after discussion with the maintainers
- ➔ #276
    - ◆ Discussed with the maintainers and safely closed the mild issue of extent of zooming out in the Color My World Activity
- ➔ #304
    - ◆ Proposed the Chopsticks Activity for Sugarizer
    - ◆ Worked on the JS version of the proposed implementation and submitted for review by the maintainers

Contributions to other Open Source Projects

I have been engaging with the Open Source community in general and contributing in various forms. Some of them are:

- **GirlScript Summer of Code** (My contributions) - Currently serving as a mentor for WebTech , we are guiding students to develop and maintain a Web Application for Testing the Tech Stack of websites.
- **freeCodeCamp** (#25829 , #25978) - Deliberated over the best programming practices with the developers for the guides in FCC.
- **Appwrite.io** (#107 , #116) - Fixed CSS bug in Hacktoberfest 2019, member of the organisation since then.
- **Mozilla Developer Network** (#26) - Fixed a bug in the tutorial code of the Web Speech API.

Open Source/Personal Projects

In the journey of learning, I also have developed several projects to implement the theory in practice:

- **Vue.js Activity Template for Sugarizer Activities** - A template inspired by the Ebook Reader Activity. Provides the basic functionality for the setting of Sugarizer Application on the web.
- **Vue.js practice** - A collection of mini projects and exercises to learn Vue.js.
  - **Monster Slayer** - A simple player vs. monster game, practice the basics of Vue.js
  - **Wonderful Quotes** - Components and slots practice
  - **Components - 1** - Practicing components fundamentals
  - **Components - 2** - Communication through components
  - **Directives** - Creating the directives to handle the click event
  - **Dynamic Components** - Switching between components with the component tag
- **Rich Text Editor** - A RTEditor developed as a prototype for the previous GSoC Application
  - Incorporates features like Font size, style, format, subscript, superscript.
  - Text align, copy paste and font color change and background color change.
  - Supports image uploads in the document.
- **Cook Book** - Bootstrap 3, Node.js, Express.js, MongoDB Atlas based web application with Multi-User Authentication, search and CRUD.
  - Cook Book - LIVE
- **APIs and Microservices** - Developed several services as APIs in JS and Python, including File Metadata, Header Parser, Timestamp and URL Shortener
- **Chat Mini** - WebSocket based online real time Instant Messaging Service, built with JS.
  - Chat Mini - LIVE
- **Chop Game** - The Japanese Chopsticks game written in Vanilla JS
  - Chop Game - LIVE
- **RESTful Blogging** - A Full Stack Web application developed in NodeJS
  - Utilizes MongoDB and has Express Framework and Semantic UI in the Frontend

Project Details

What am I making?

The Sugarizer Knowledge Activity Pack includes the following activities:

Curriculum Activity

The basic objective is to provide the students to self assess their skills and maintain interactive records for the same. A developed prototype for an application with similar use cases can be found **here**

For the development, the following features and use cases are proposed:

- Display a hierarchical set of skills, grouped by categories.
- Will have two main modes -
  - **Display mode** - will display the skills grouped by the categories, an option to select different categories to view the skills associated with each of them.
  - **Settings mode** - will give the user options to Create, Read, Update and Delete (CRUD) skills, their associated multimedia and the categories.
- Each skill card will include -
  - Boolean checkmark for the skill is acquired or not
  - Name and description of the skill
  - Image to demonstrate the skill, default to a predefined stock image
  - Audio to demonstrate the skill, default to a predefined stock audio clip
- Each category will include -
  - A title for the category
  - A color of the category for visual feedback.
- Star rating to every user - will provide the ratio of skills acquired to the total number of skills
- The user can click on the skill card, which will display the name and image initially, to get a popup and get all the data about the skill (READ operation on skills card).
- A **Progress Card** feature will help the user **export** his data, like the skills acquired, the multimedia saved, with the dates, his buddy icon and his star rating to **DOC/ODT/PDF/RTF** format and download in his computer.
- The user will be able to add more skills to a category in the **Settings** mode.
- A side panel shows the user his summary - his buddy icon, the skills acquired across all the categories, the multimedia saved (images and audio), and his star rating
- The skills can be shared in a read only mode on the network so that other users can see the progress.
- Fullscreen Button.
- Local Database Storage (Journal Integration).

- Interactive tutorial and documentation.
- Sugar Web UI, Buddy Colors when possible.
- Multi device support and responsiveness on all screen sizes.
- Localisation with the webL10n library.

Vote Activity

The basic objective is to allow the user to easily build a poll system from pre-defined templates and share in the network. The developed prototype of for an application with similar use cases is given **here** (GitHub)

For the development, the following features and use cases are proposed:

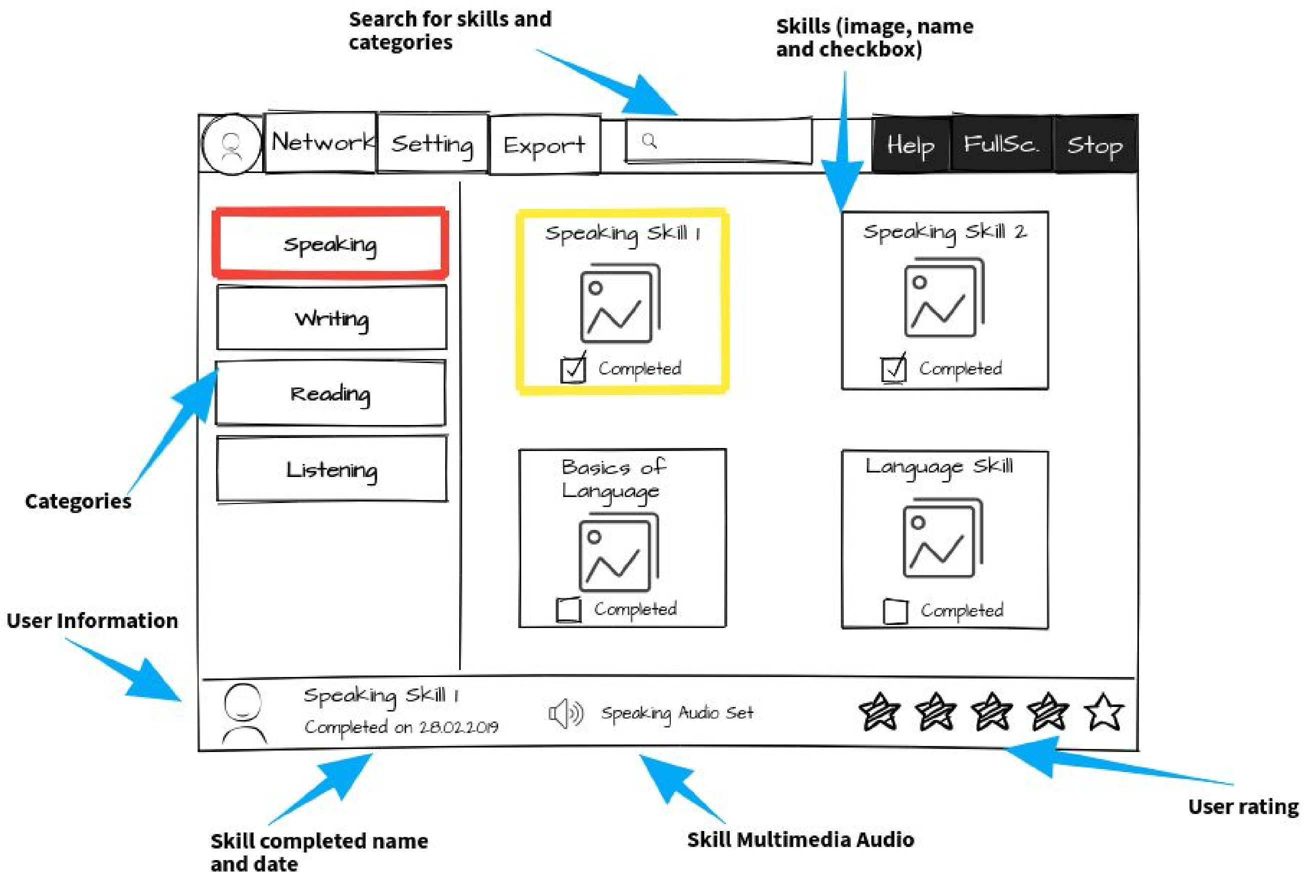
- The user can easily build a poll of the following three templates:
    - Yes/No Voting (Binary Voting)
    - Multiple Choice Questions (4 Options for the current prototype)
    - Free text answers
- The user can create a new poll and share it with the network to allow the other users to join and vote.
- The users will be able to see the results and the statistics of the vote:
    - In real time, with the host
    - After the poll ends
    - As set in the template configurations by the host
- The home screen will allow the user to quickly start a poll from the set templates present in the Journal, or from previous templates designed by the user.
- The settings screen will give the user options to create a new poll, or customise an existing poll template.
- The Poll results can be exported in the PDF/CSV formats.
- Search feature for existing polls, in Home Page and Settings Page.
- Timer Integration
- Local Database Storage (Journal Integration).
- Interactive tutorial and documentation.
- Sugar Web UI, Buddy Colors when possible.
- Multi device support and responsiveness on all screen sizes.
- Localisation with the webL10n library.
- Fullscreen Button.

Implementation Details

Curriculum Activity

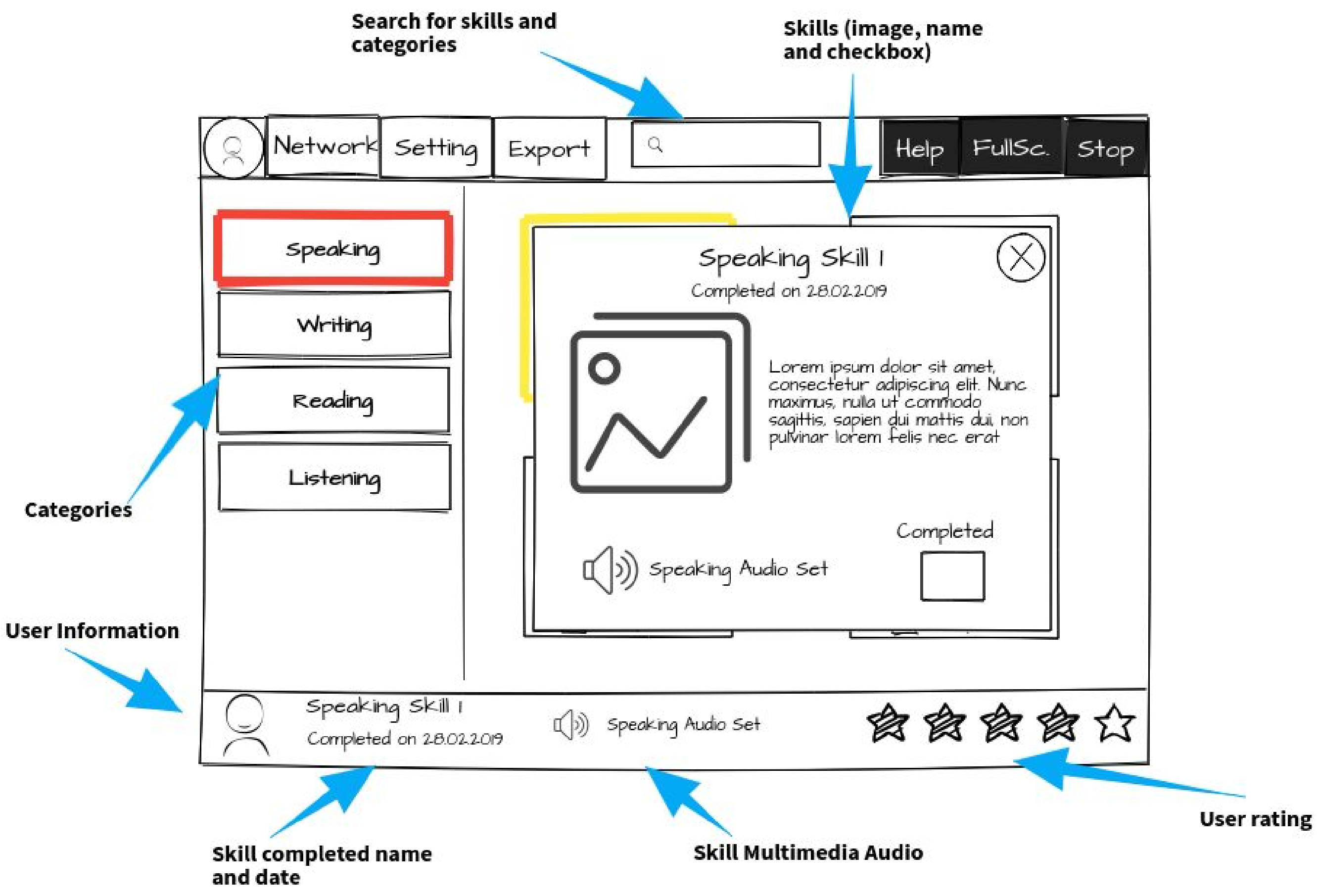
The features proposed are discussed in the implementation level:

- Play Mode
  - UI and flows
    - The proposed UI for the Play Mode will be



    - The categories will be in the left, stacked top to bottom. Each category tile will have a **custom background color selected by the user**, and the name of the category. Upon clicking, the category will be selected and the associated skills will be displayed on the right hand side.
    - The skills will be in the right, in the form of tiles. Each skill tile will have the name of the skill, the image and a checkbox if it is acquired by the user.

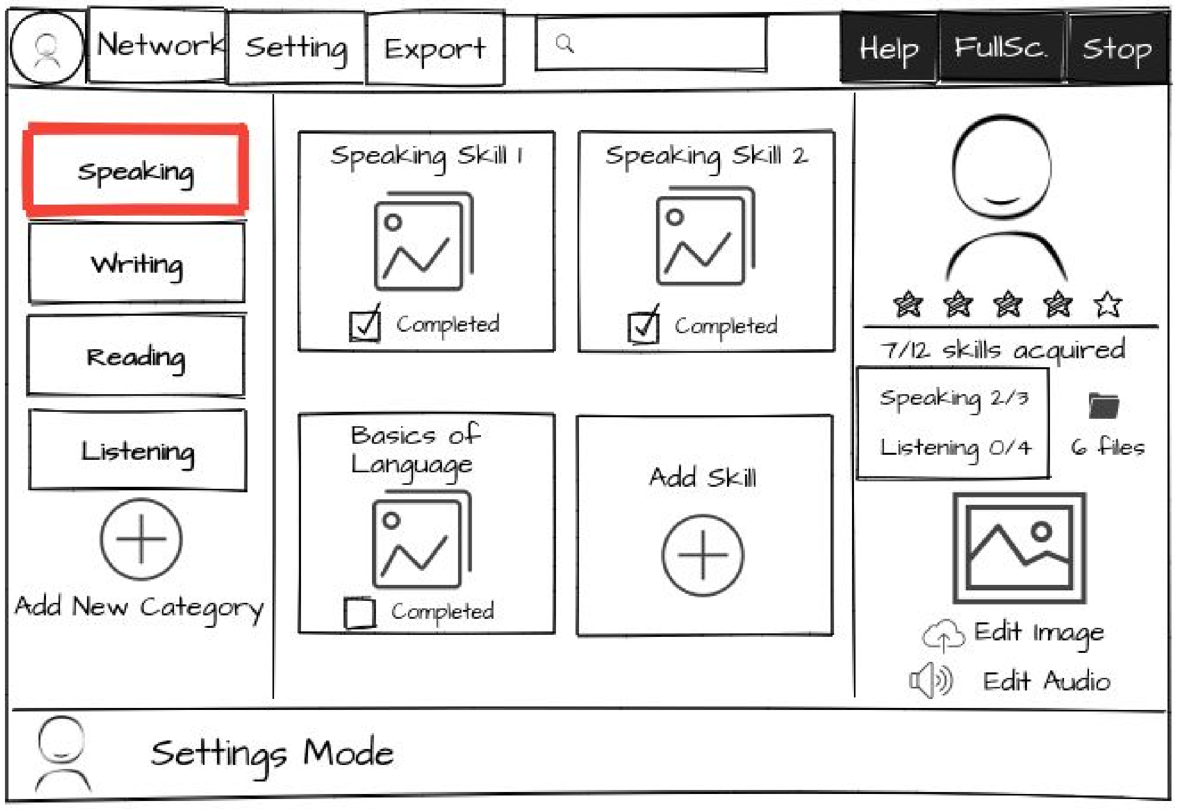
- Upon clicking on the skill, a popup will open which will give the name, description, multimedia (audio, image), a checkbox to tick if the skill is acquired by the user.
- The user can **Update** the multimedia and the checkbox (skill acquisition status) from the popup. An option to upload the multimedia from the journal will be given in the popup.
- The sample of the popup is shown


Search for skills and categories
Skills (image, name and checkbox)
Network
Setting
Export
Help
FullSc.
Stop
Speaking
Writing
Reading
Listening
Speaking Skill 1
Completed on 28.02.2019
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc maximus, nulla ut commodo sagittis, sapien dui mattis dui, non pulvinar lorem felis nec erat
Completed
Speaking Audio Set
Categories
User Information
Speaking Skill 1
Completed on 28.02.2019
Speaking Audio Set
User rating
Skill completed name and date
Skill Multimedia Audio

- The bottom of the page will show the user details and the **date on which the skill was acquired**. We can also play the audio from the bottom panel. The user rating will also be visible in the right.
- There will be a button on the right side of the screen, which **when clicked will open the Side Panel**. This can also be done by removing the button and opening **as soon as the user takes his mouse pointer to the rightmost part of the screen**. Another

approach will be **to make the side panel visible only in the settings modes**. The detailed implementation of how to activate the side panel will have to be discussed with the mentor.
        - The UI will have Buddy colors, and the selected category and the skill will be highlighted.

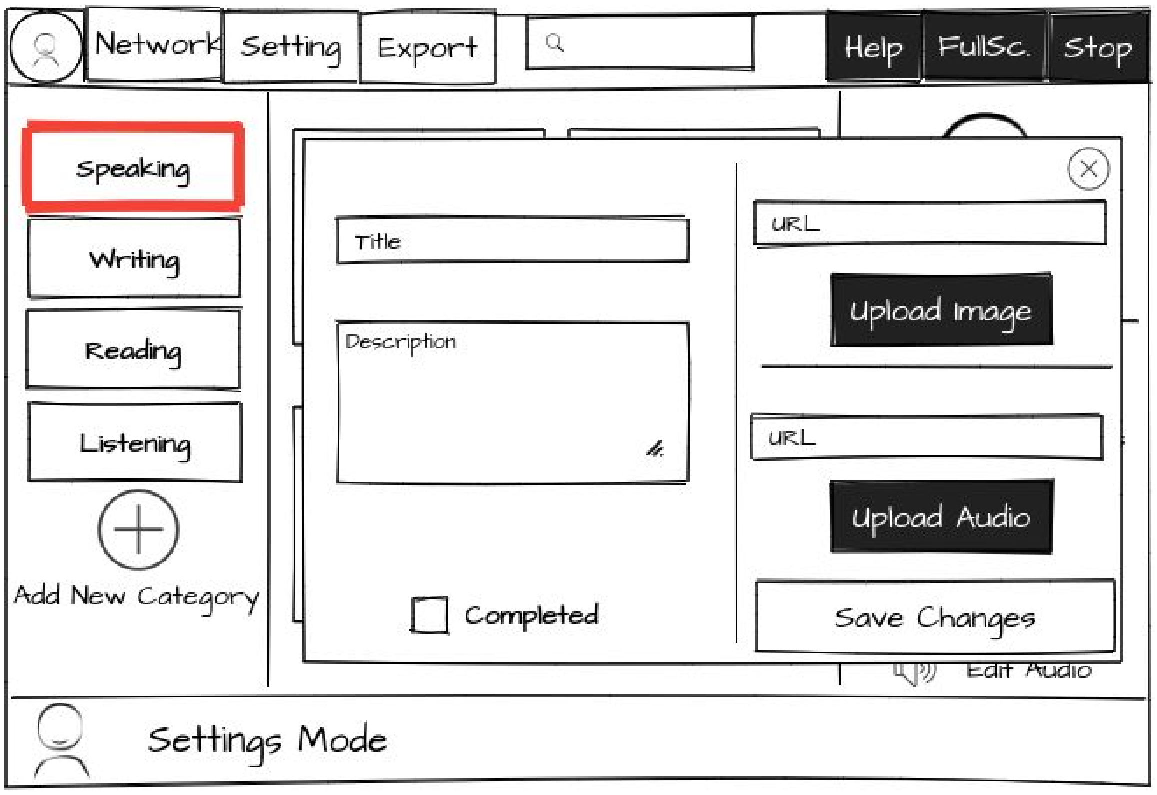
- Side Panel and Bottom Panel
    - UI and flows



        - The Side Panel will have the buddy icon and all the statistics related to the user.
        - The **number of skills acquired** and the **total number of skills**
        - Multimedia related to the selected skills
        - Total multimedia instances uploaded
        - The categories in which the user has acquired the most number of skills, that is, **the strongest category**. Similarly, we can show the category which requires the most improvement by the user.

- Settings Mode
    - CRUD skills
        - Create, Update Skills

- The add skill will be the last tile in the skills area in the settings mode.
- Upon clicking, the user will get a popup, similar in UI to the one in the play mode, where he will be able to add the title, description, image, audio.

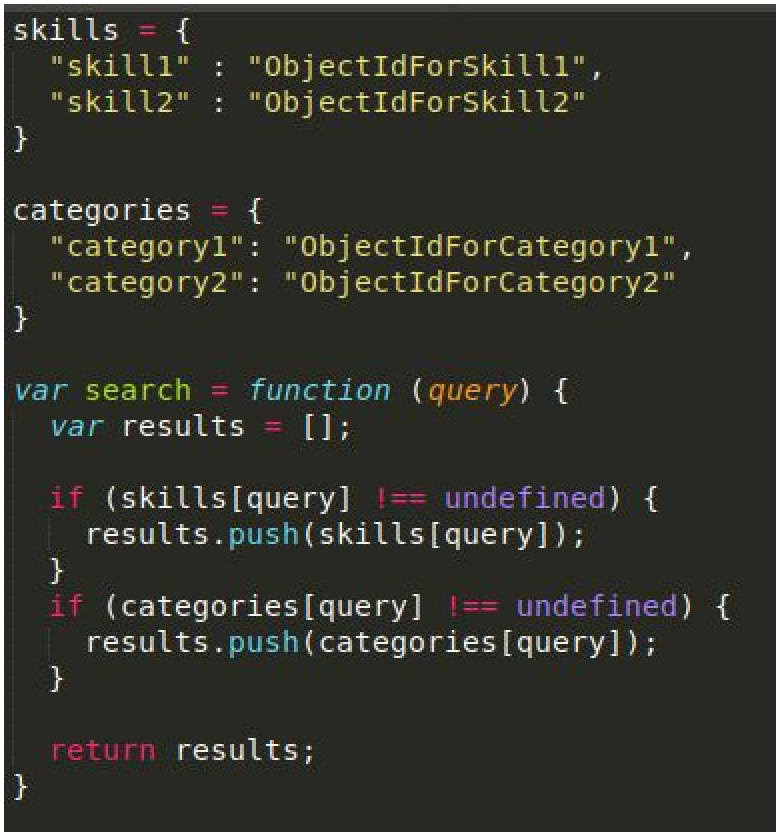


- When save changes is clicked, the **skillAdded** action is emitted right up to the **App Component**, where the data manipulation is carried. Here the following actions take place -
    - A new skill object is created with the stringified multimedia
    - The associated category-id id added to the object
    - The object is added to the skills array
    - The student acquired-skills object is updated with the addition of the {skill-id: false} and since Vue components are reactive, the change is transmitted to all the DOM elements.

■ Delete Skills

        - The deleteSkill is emitted to the App Component and the data manipulation is process
        - The skills array and the student object are updated
        - The reactive nature of Vue reflects the updates in all parts of the DOM elements.
- CRUD categories
    - Create/Update Categories
        - The user will be able to click in the plus icon in the bottom of the left side of the screen.
        - The categories add will set a popup much similar to the one in the skills area.
        - When save changes will be clicked -
            - A new category object is created with the stringified multimedia
            - The object is added to the categories array
            - The Vue components are reactive, the change is transmitted to all the DOM elements.
    - Delete Categories
        - The object is deleted from the categories array.
        - The skills array is traversed and all the skills with the category-id equal to the category deleted are also deleted (If category is deleted, then all the skills under that category are also deleted.)
        - The Student object is updated with the skills.
- CRUD multimedia
    - The skills and the categories will be able to get user uploaded multimedia from the Journal.
    - The user will also update the audio and the image from the settings mode.
    - A **computed** property can be added in the SidePanel which will reflect all the multimedia changes in the data as soon as they are updated.
    - The uploaded multimedia will be converted to base64 strings and saved in the Journal with the time of upload.
    - The history of updates in the media is maintained with the timestamp and displayed with image and audio tags in the DOM.
    - The set of **v-if** and **v-else** statements is used to display the multimedia elements and **v-on:click** to each of the elements to call a method to navigate to the skill associated.

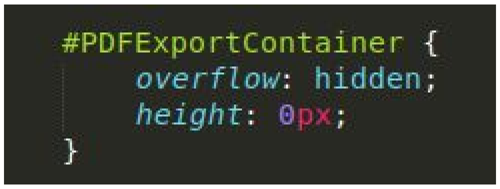
- Search feature
    - A search area will be present in the toolbar, which will **allow the user to search for skills and categories by their name.**
    - All the matching skills and categories will be displayed which are associated with the search query.
    - For implementing this feature, we will **maintain a dictionary of all the skill and category names**, with their object Id as the value. An almost O(1) lookup can be designed and we can get very fast search results.
    - The results can be rendered with the **v-for** directive and **v-else** and **v-if** directives in Vue.js in the DOM



```
skills = {
  "skill1" : "ObjectIdForSkill1",
  "skill2" : "ObjectIdForSkill2"
}

categories = {
  "category1": "ObjectIdForCategory1",
  "category2": "ObjectIdForCategory2"
}

var search = function (query) {
  var results = [];

  if (skills[query] !== undefined) {
    results.push(skills[query]);
  }
  if (categories[query] !== undefined) {
    results.push(categories[query]);
  }

  return results;
}
```

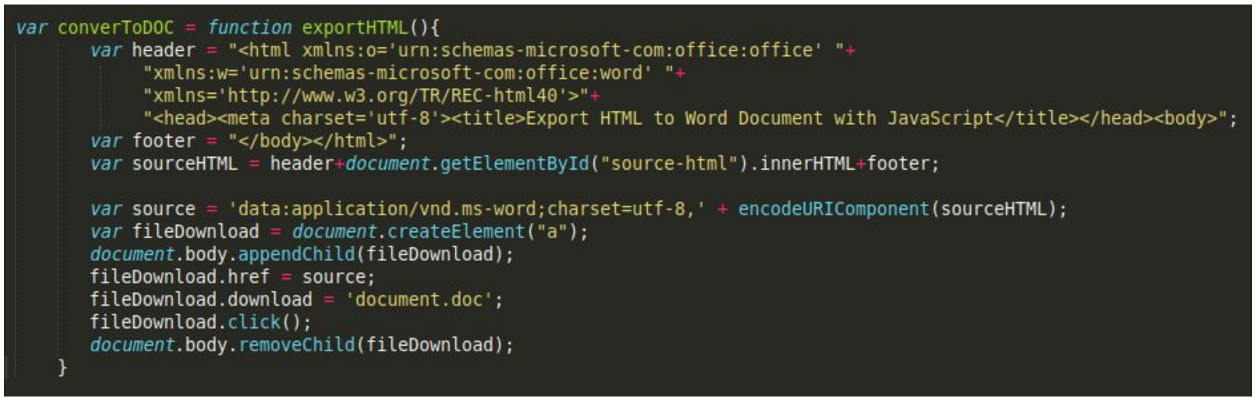
- Progress Report (export)
    - The Report Component will be responsible for the User Data to be made into the PDF/DOC/RTF/ODF files. The Export button in the toolbar will expose to the user a palette which will have 4 buttons for the four possible export options mentioned earlier.
    - The entire document will be **rendered with CSS and data will be provided by the props in Vue.js**. This will create a DOM element which will be given the **#expor**t id.
    - This DOM element will be used to provide content in the various approaches discussed to export the document.

- The approaches in most instances require the DOM element to be actually present in the viewport. In other words, it must be visible to the libraries to be able to take snapshots of the content.
  This can be tackled in two ways. They are discussed as -
    - We need to have custom CSS styled DOM content to export to PDF, which can be done just as soon as the user clicks the export to PDF button. In other words, **we will make the DOM element come into existence and the viewport for as long as the PDF library runs**, and then we can delete the element from the page.
    - Another approach will be to always let the custom CSS style DOM element exist in the page, but **wrap it in a container and set its height property to 0**. This will make the Progress Report visible to the APIs, but not to the user.



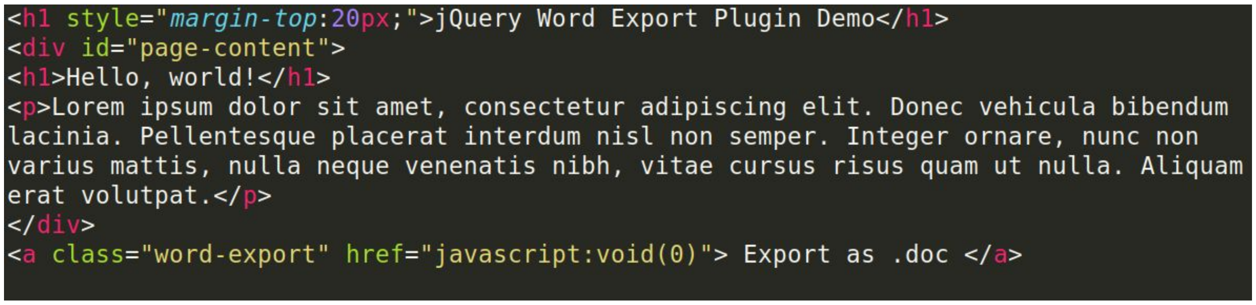
```
#PDFExportContainer {
    overflow: hidden;
    height: 0px;
}
```

- Export to PDF
    - **html2canvas** - The script allows us to take "screenshots" of webpages, directly on the user's browser. The screenshot is based on the DOM and will be used to generate the PDF.
    - **jsPDF** - Utilised with html2canvas to generate PDF documents.
- Export to ODT
    - We can use **the approach in the Write Activity** for the generation of ODT Document. We convert all the data to XML and append to a string with the requisite headers.
    - **Odt.js** - A JavaScript Library to convert from HTML to ODT and reverse.
- Export to DOC
    - We can add the requisite header and footer to the html to export to the DOC Format, as described in the following process

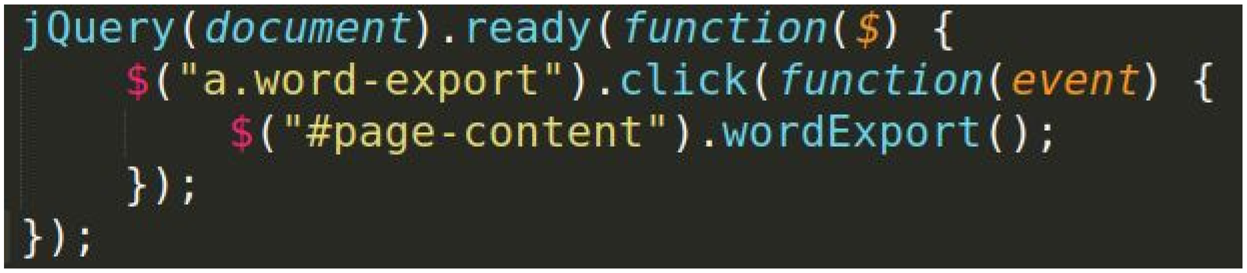


```
var converToDOC = function exportHTML(){
        var header = "<html xmlns:o='urn:schemas-microsoft-com:office:office' "+
             "xmlns:w='urn:schemas-microsoft-com:office:word' "+
             "xmlns='http://www.w3.org/TR/REC-html40'>"+
             "<head><meta charset='utf-8'><title>Export HTML to Word Document with JavaScript</title></head><body>";
        var footer = "</body></html>";
        var sourceHTML = header+document.getElementById("source-html").innerHTML+footer;

        var source = 'data:application/vnd.ms-word;charset=utf-8,' + encodeURIComponent(sourceHTML);
        var fileDownload = document.createElement("a");
        document.body.appendChild(fileDownload);
        fileDownload.href = source;
        fileDownload.download = 'document.doc';
        fileDownload.click();
        document.body.removeChild(fileDownload);
    }
```

- jQuery Word Export Plugin - The following approach can be used to export the file to DOC Format utilising the **wordExport()** function

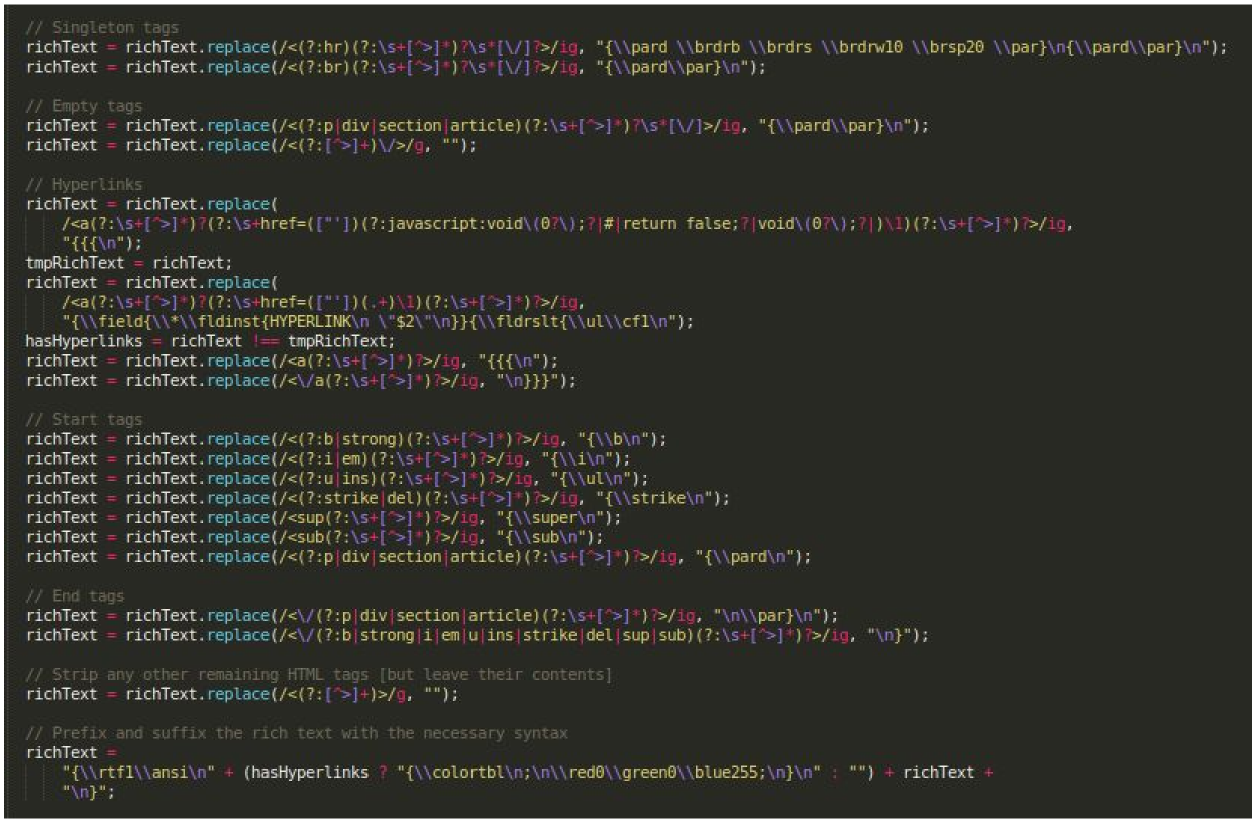


```
<h1 style="margin-top:20px;">jQuery Word Export Plugin Demo</h1>
<div id="page-content">
<h1>Hello, world!</h1>
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec vehicula bibendum
lacinia. Pellentesque placerat interdum nisl non semper. Integer ornare, nunc non
varius mattis, nulla neque venenatis nibh, vitae cursus risus quam ut nulla. Aliquam
erat volutpat.</p>
</div>
<a class="word-export" href="javascript:void(0)"> Export as .doc </a>
```



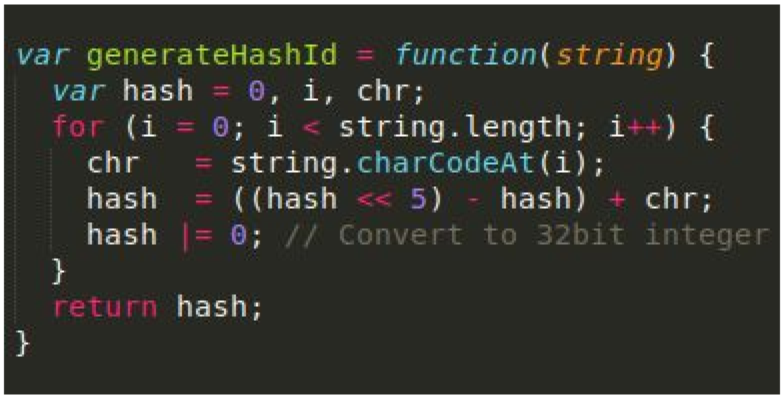
```
jQuery(document).ready(function($) {
    $("a.word-export").click(function(event) {
        $("#page-content").wordExport();
    });
});
```

- Export to RTF
    - FileSaver.js - Implements the FileSaver API for all major Web Browsers
    - The following approach also demonstrates a technique to format the source in the RTF Format. The approach is based on Regex



```
// Singleton tags
richText = richText.replace(/<(?:hr)(?:\s+[^>]*)?\s*[\/]?>/ig, "{\\pard \\brdrb \\brdrs \\brdrw10 \\brsp20 \\par}\n{\\pard\\par}\n");
richText = richText.replace(/<(?:br)(?:\s+[^>]*)?\s*[\/]?>/ig, "{\\pard\\par}\n");

// Empty tags
richText = richText.replace(/<(?:p|div|section|article)(?:\s+[^>]*)?\s*[\/]>/ig, "{\\pard\\par}\n");
richText = richText.replace(/<(?:[^>]+)\/>/g, "");

// Hyperlinks
richText = richText.replace(
    /<a(?:\s+[^>]*)?(?:\s+href=(["'])(?:javascript:void\(0?\);?|#|return false;?|void\(0?\);?|)\1)(?:\s+[^>]*)?>/ig,
    "{{{\n");
tmpRichText = richText;
richText = richText.replace(
    /<a(?:\s+[^>]*)?(?:\s+href=(["'])(.+)\1)(?:\s+[^>]*)?>/ig,
    "{\\field{\\*\\fldinst{HYPERLINK\n \"$2\"\n}}{\\fldrslt{\\ul\\cf1\n");
hasHyperlinks = richText !== tmpRichText;
richText = richText.replace(/<a(?:\s+[^>]*)?>/ig, "{{{\n");
richText = richText.replace(/<\/a(?:\s+[^>]*)?>/ig, "\n}}}");

// Start tags
richText = richText.replace(/<(?:b|strong)(?:\s+[^>]*)?>/ig, "{\\b\n");
richText = richText.replace(/<(?:i|em)(?:\s+[^>]*)?>/ig, "{\\i\n");
richText = richText.replace(/<(?:u|ins)(?:\s+[^>]*)?>/ig, "{\\ul\n");
richText = richText.replace(/<(?:strike|del)(?:\s+[^>]*)?>/ig, "{\\strike\n");
richText = richText.replace(/<sup(?:\s+[^>]*)?>/ig, "{\\super\n");
richText = richText.replace(/<sub(?:\s+[^>]*)?>/ig, "{\\sub\n");
richText = richText.replace(/<(?:p|div|section|article)(?:\s+[^>]*)?>/ig, "{\\pard\n");

// End tags
richText = richText.replace(/<\/(?:p|div|section|article)(?:\s+[^>]*)?>/ig, "\n\\par}\n");
richText = richText.replace(/<\/(?:b|strong|i|em|u|ins|strike|del|sup|sub)(?:\s+[^>]*)?>/ig, "\n}");

// Strip any other remaining HTML tags [but leave their contents]
richText = richText.replace(/<(?:[^>]+)>/g, "");

// Prefix and suffix the rich text with the necessary syntax
richText =
    "{\\rtf1\\ansi\n" + (hasHyperlinks ? "{\\colortbl\n;\n\\red0\\green0\\blue255;\n}\n" : "") + richText +
    "\n}";
```

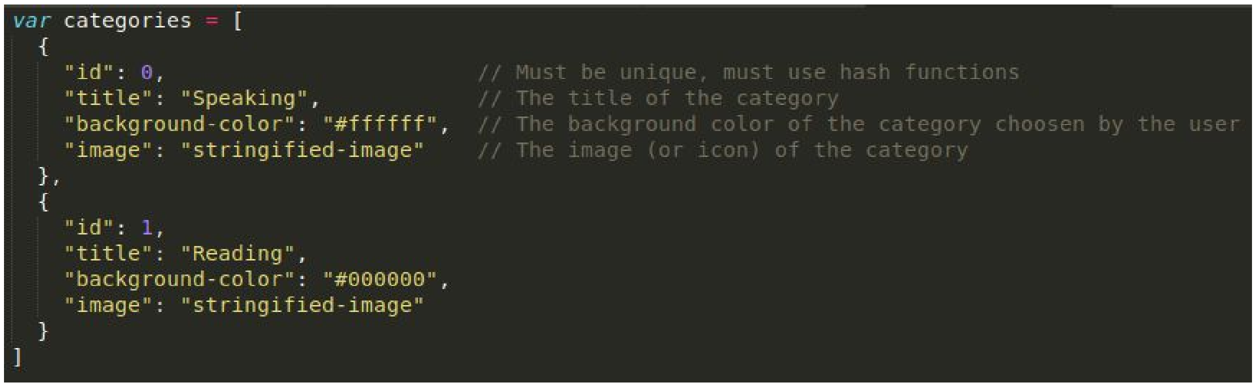
- Journal Data Object
    - We will make associations among the database tables to -
        - Make the storage more efficient
        - Lessen the lookup times
    - We will have to make sure that all the Ids of the objects created are unique. For the same motive we will utilise the approach to make the strings into a Hash Value.



```
var generateHashId = function(string) {
  var hash = 0, i, chr;
  for (i = 0; i < string.length; i++) {
    chr   = string.charCodeAt(i);
    hash  = ((hash << 5) - hash) + chr;
    hash |= 0; // Convert to 32bit integer
  }
  return hash;
}
```

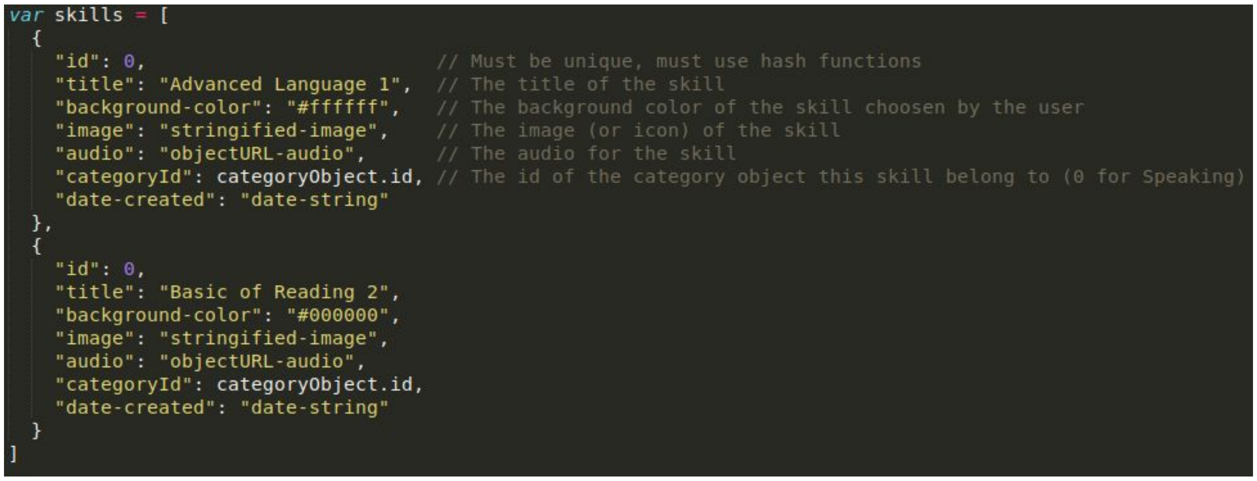
        - The hash value of any length string will be a fixed length string
        - The hash value of two different strings will never be equal
    - This will ensure that we have unique IDs for all the objects.

- Categories [] - An array of Category objects. This will be stored in the journal and retrieved at runtime.
    - The category object will have id, title, background-color and the image.



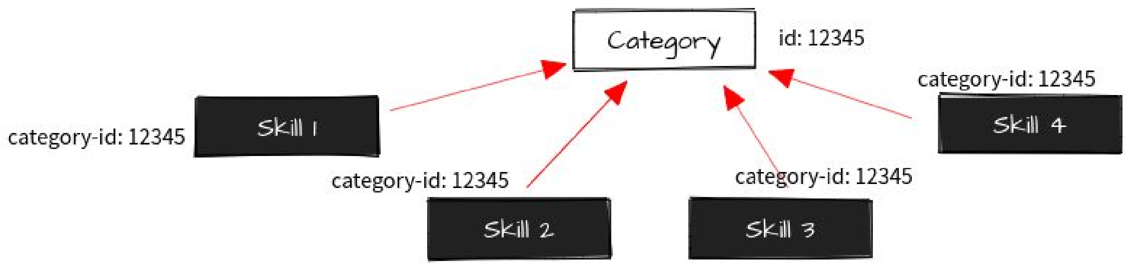
```
var categories = [
  {
    "id": 0,                          // Must be unique, must use hash functions
    "title": "Speaking",              // The title of the category
    "background-color": "#ffffff",    // The background color of the category choosen by the user
    "image": "stringified-image"      // The image (or icon) of the category
  },
  {
    "id": 1,
    "title": "Reading",
    "background-color": "#000000",
    "image": "stringified-image"
  }
]
```

- Skills [] - An array of Skill objects. This will be stored in the journal and retrieved at runtime.
    - The Skill object will have id, title, background-color, image,audio, and the **category-id**.

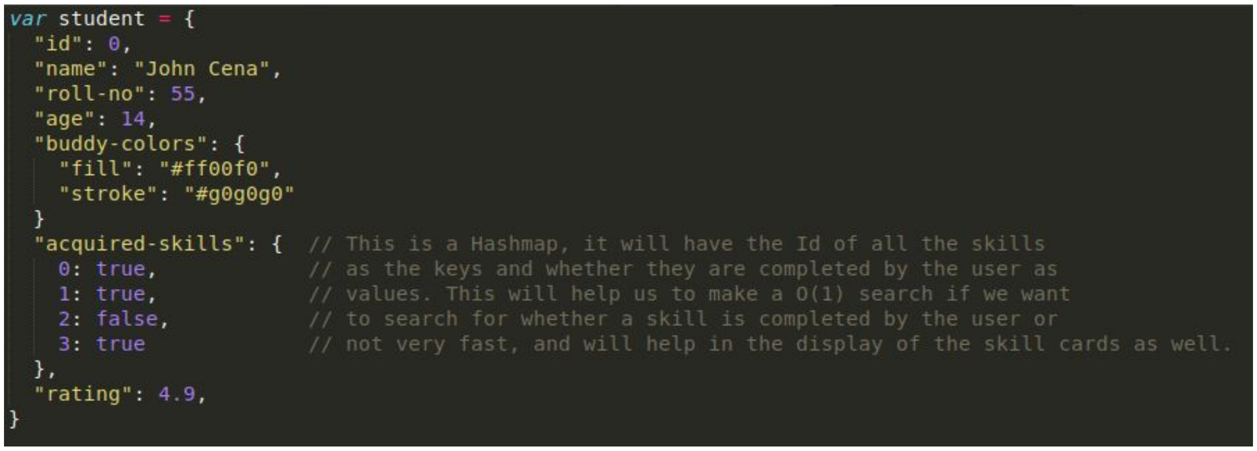


```
var skills = [
  {
    "id": 0,                             // Must be unique, must use hash functions
    "title": "Advanced Language 1",      // The title of the skill
    "background-color": "#ffffff",       // The background color of the skill choosen by the user
    "image": "stringified-image",        // The image (or icon) of the skill
    "audio": "objectURL-audio",          // The audio for the skill
    "categoryId": categoryObject.id,     // The id of the category object this skill belong to (0 for Speaking)
    "date-created": "date-string"
  },
  {
    "id": 0,
    "title": "Basic of Reading 2",
    "background-color": "#000000",
    "image": "stringified-image",
    "audio": "objectURL-audio",
    "categoryId": categoryObject.id,
    "date-created": "date-string"
  }
]
```

    - The **category-id** will set up the many to one relationship between the Skills table and the categories table, as depicted.



- Student {} - An object which will store the information about the student. Will be stored in the journal and retrieved at runtime.
    - Will store the **id, name, roll-no, age, buddy-colors, rating** (the exact properties will best be discussed), and **the acquired-skills object**.
    - The acquired-skills object is essentially a dictionary, which will help us provide an efficient O(1) lookup.

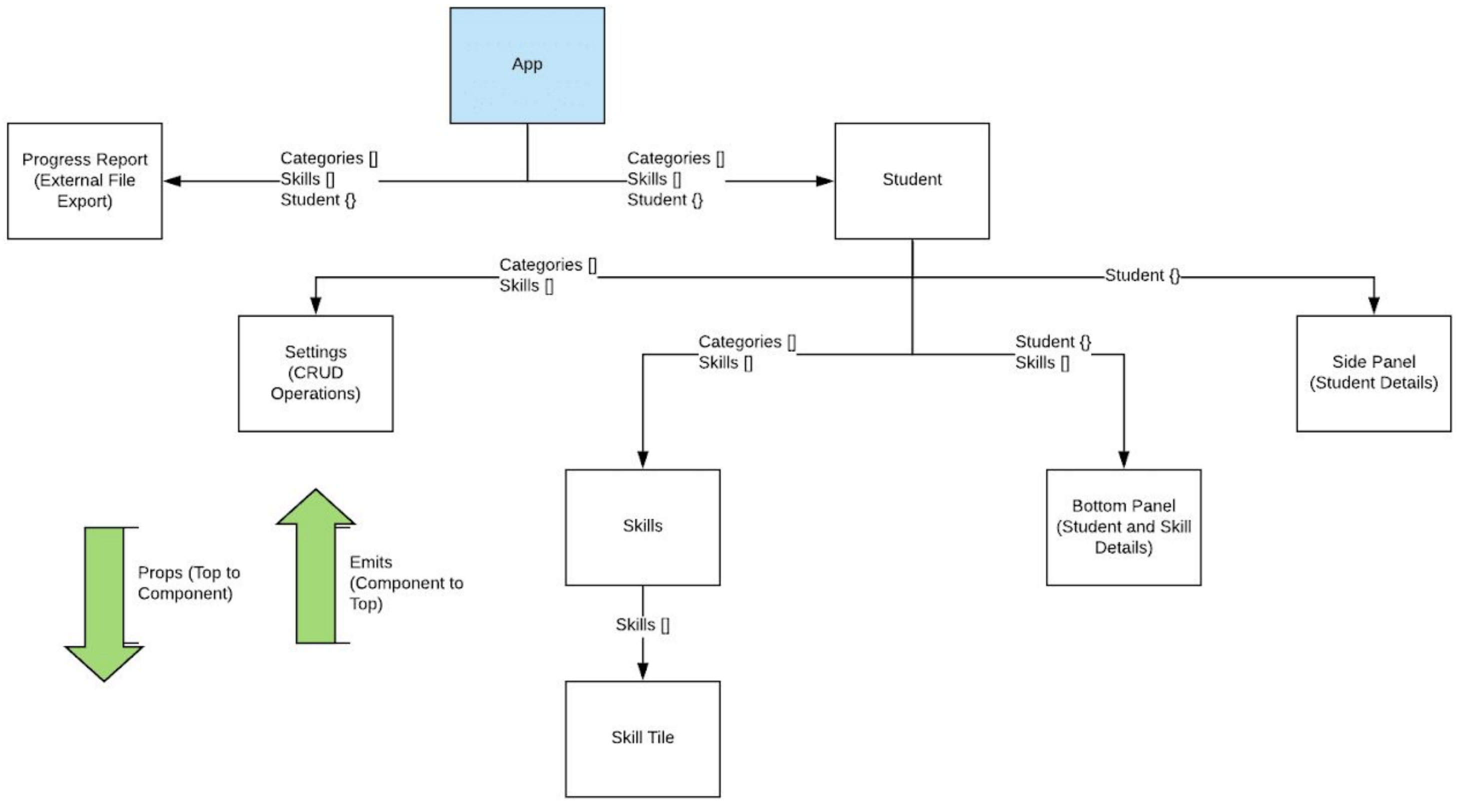


```
var student = {
  "id": 0,
  "name": "John Cena",
  "roll-no": 55,
  "age": 14,
  "buddy-colors": {
    "fill": "#ff00f0",
    "stroke": "#g0g0g0"
  }
  "acquired-skills": {  // This is a Hashmap, it will have the Id of all the skills
    0: true,            // as the keys and whether they are completed by the user as
    1: true,            // values. This will help us to make a O(1) search if we want
    2: false,           // to search for whether a skill is completed by the user or
    3: true             // not very fast, and will help in the display of the skill cards as well.
  },
  "rating": 4.9,
}
```

- This technique is efficient as -
    - Only the associated Ids are stored and redundancy is minimised.
    - If we want to get the status(acquired or not) of a skill of a user, we just need to have the skill-id

Student.acquired-skills[skill-id] // returns true if the skill is completed

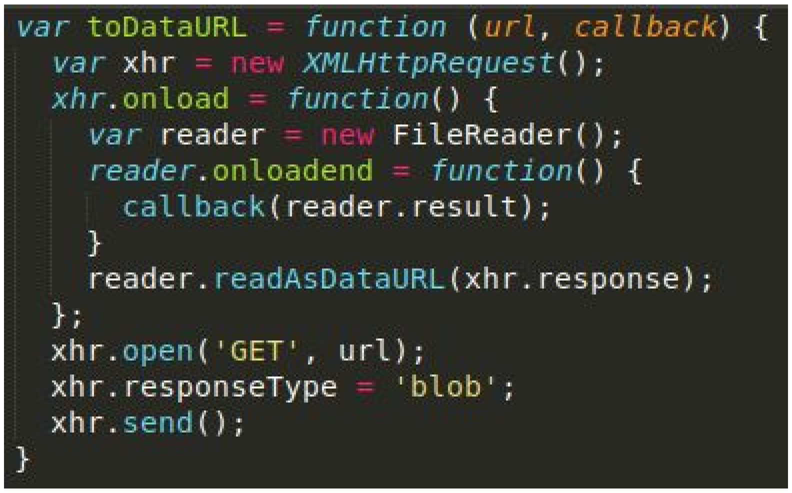
- Vue.js Components
    - Design of the components
        - The Activity can easily be fragmented into reusable Vue components
        - The state variables will be -
            - Categories [] - Will store all the category objects
            - Skills [] - Will store all the skill objects
            - Student {} - Will store the information about the student
    - Hierarchy of the components - props and emit
        - The diagram will give a general idea about the hierarchy in the components



- Description of the components
    - App - Contains all the data structures, and the Sugar components like the toolbar, search input
    - Progress Report - Contains the logic and the templates to generate the PDF/ODT/DOC/RTF files
    - Student - Manages the Settings, the skills panel, the skill tile and the side and bottom panel. This will also be the handler for all the Updation emits from the child components
    - Settings - Facilitates the Create, Read, Update and Delete functions on the categories and skills
    - Skills - Shows the tiles with the skills
    - Skill Tile - Shows the data associated with the skill and the option to add and update the multimedia
    - Bottom Panel - Shows the statistics about the user and the skill selected
    - Side Panel - Shows the data and statistics about the user, the categories, the multimedia and the user ratings, with an option to

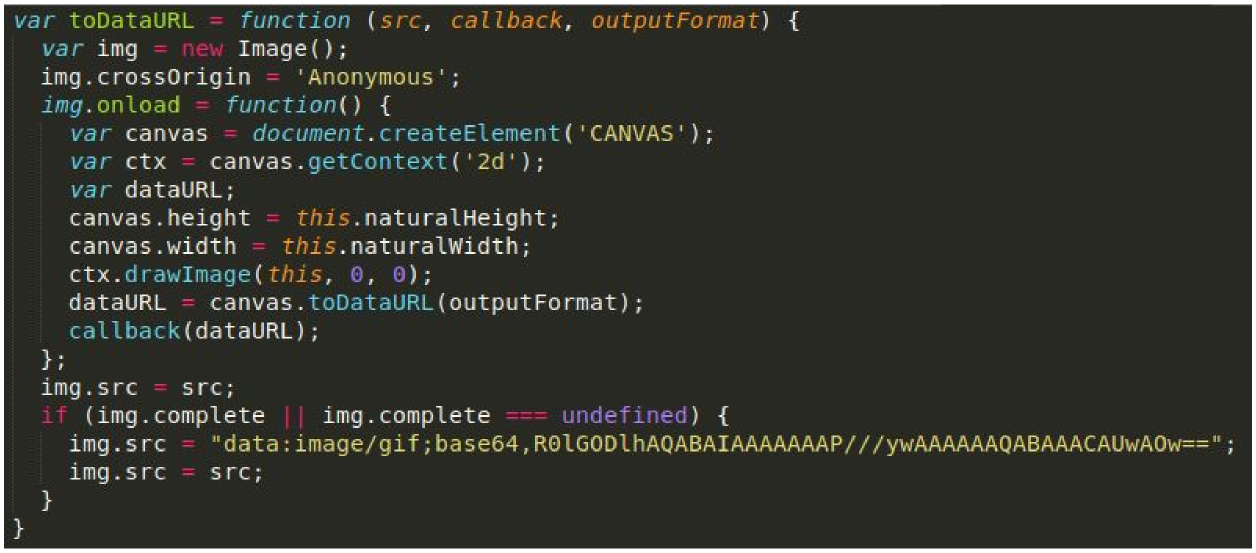
Update them. Will be visible only when clicked or in the settings mode.

- Image
    - Approach 1 - Images can be converted to base64 data URL using the **readDataAsURL()** method of **FileReader**, and could then be stored as strings.



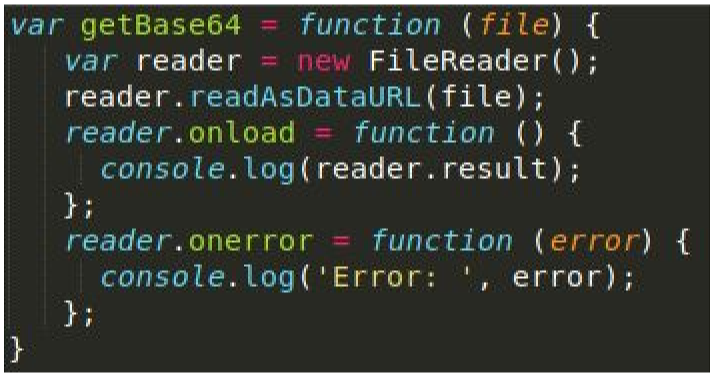
```
var toDataURL = function (url, callback) {
  var xhr = new XMLHttpRequest();
  xhr.onload = function() {
    var reader = new FileReader();
    reader.onloadend = function() {
      callback(reader.result);
    }
    reader.readAsDataURL(xhr.response);
  };
  xhr.open('GET', url);
  xhr.responseType = 'blob';
  xhr.send();
}
```

    - Approach 2 -Loading the image into and Image-Object and painting to a non tainted canvas and the converting the canvas to the dataURL



```
var toDataURL = function (src, callback, outputFormat) {
  var img = new Image();
  img.crossOrigin = 'Anonymous';
  img.onload = function() {
    var canvas = document.createElement('CANVAS');
    var ctx = canvas.getContext('2d');
    var dataURL;
    canvas.height = this.naturalHeight;
    canvas.width = this.naturalWidth;
    ctx.drawImage(this, 0, 0);
    dataURL = canvas.toDataURL(outputFormat);
    callback(dataURL);
  };
  img.src = src;
  if (img.complete || img.complete === undefined) {
    img.src = "data:image/gif;base64,R0lGODlhAQABAIAAAAAAAP///ywAAAAAAQABAAACAUwAOw==";
    img.src = src;
  }
}
```

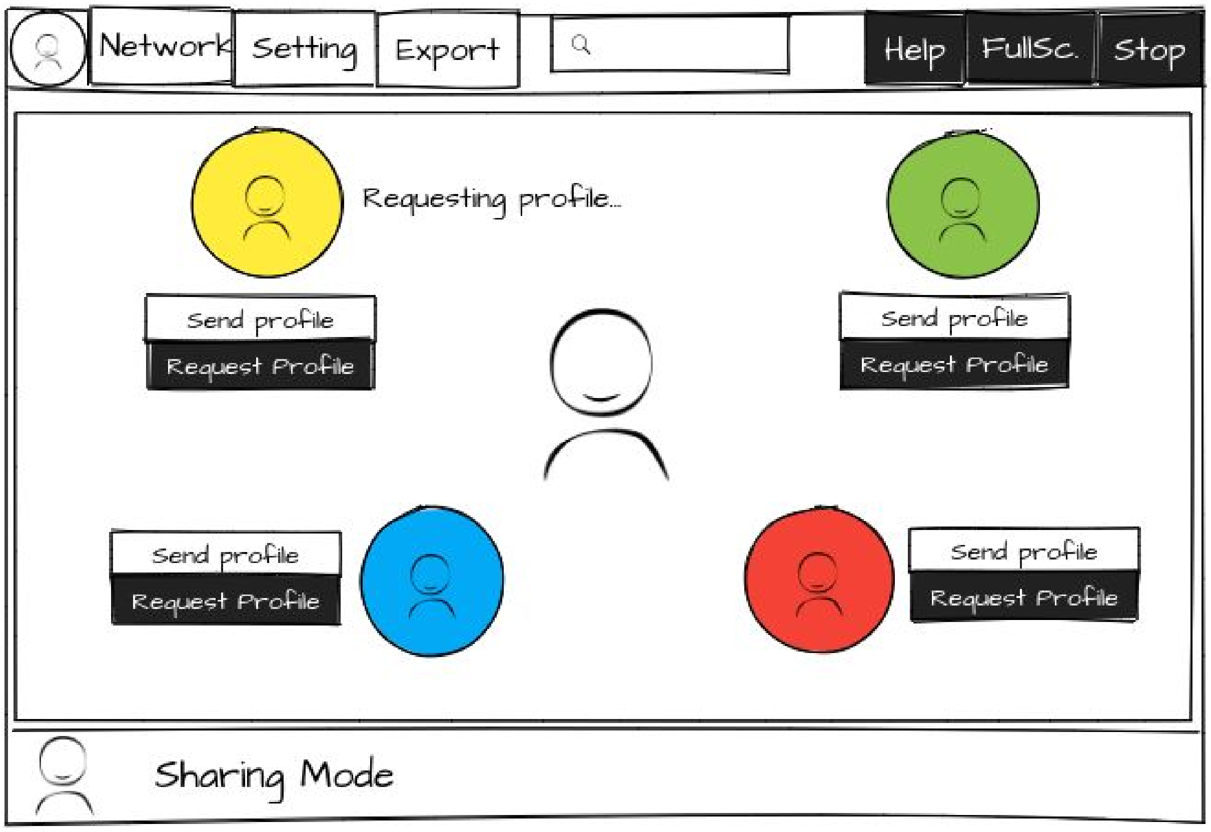
- Audio
    - Approach 1 - Audio can be converted into ObjectURL strings using the **URL.createObjectURL()** method, and then be stored as strings.
    - Approach 2 - We can get the base64 strings by the FileReader methods as depicted.



```
var getBase64 = function (file) {
    var reader = new FileReader();
    reader.readAsDataURL(file);
    reader.onload = function () {
      console.log(reader.result);
    };
    reader.onerror = function (error) {
      console.log('Error: ', error);
    };
}
```

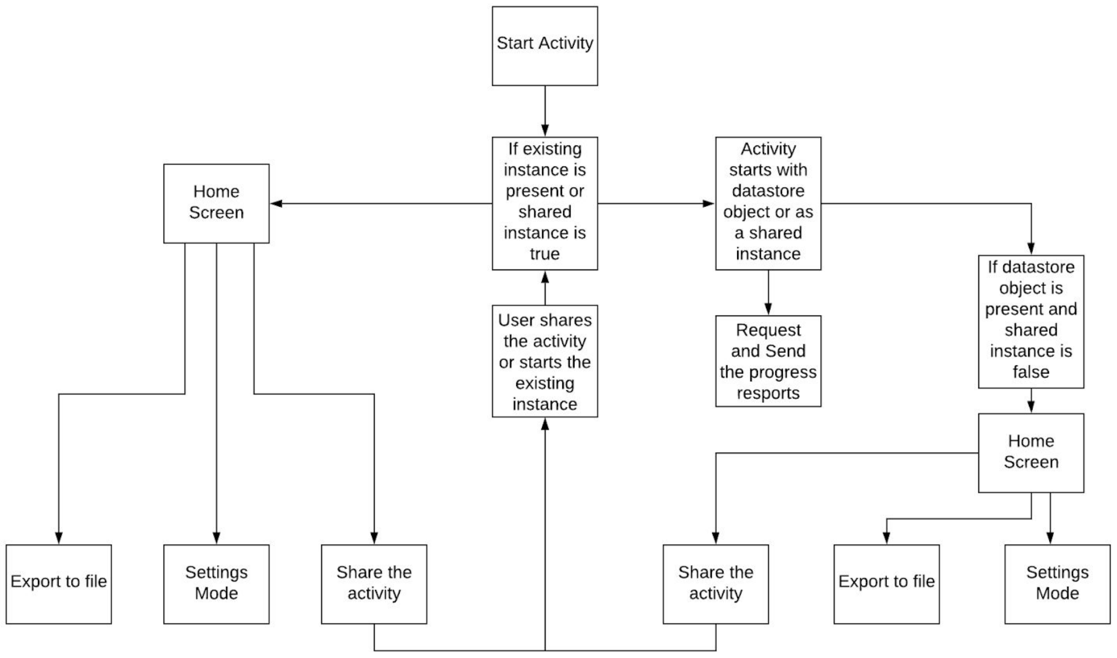
The data structures will be present in the root component and can be passed to the children in the form of props. The children component can communicate with the updates in the form of emit statements.

- User ratings
    - The user will have a rating, which will depend on
        - The ratio of the skills acquired to the total number of skills
        - The time taken to acquire the skills (additional)
    - A star system will be implemented to give the user an overall sense of accomplishment when using the activity.
- Network Sharing
    - UI and flows -
        - The user will be able to share his skills with the network. Once he shares the instance, he can see all the users who have joined the activity.
        - Two options will be visible with the buddy icon of each user -
            - Send profile - The user will send his profile and the receiver computer will open a popup to show the basic details of the user and an option to download in PDF/ODT/DOC/RTF Format the progress report.
            - Request profile - The user can also request users on the network to send their profile. The buddy icon will show a help text to let the user know which user has requested for his profile. He can send the profile if he wants to them with the send function.

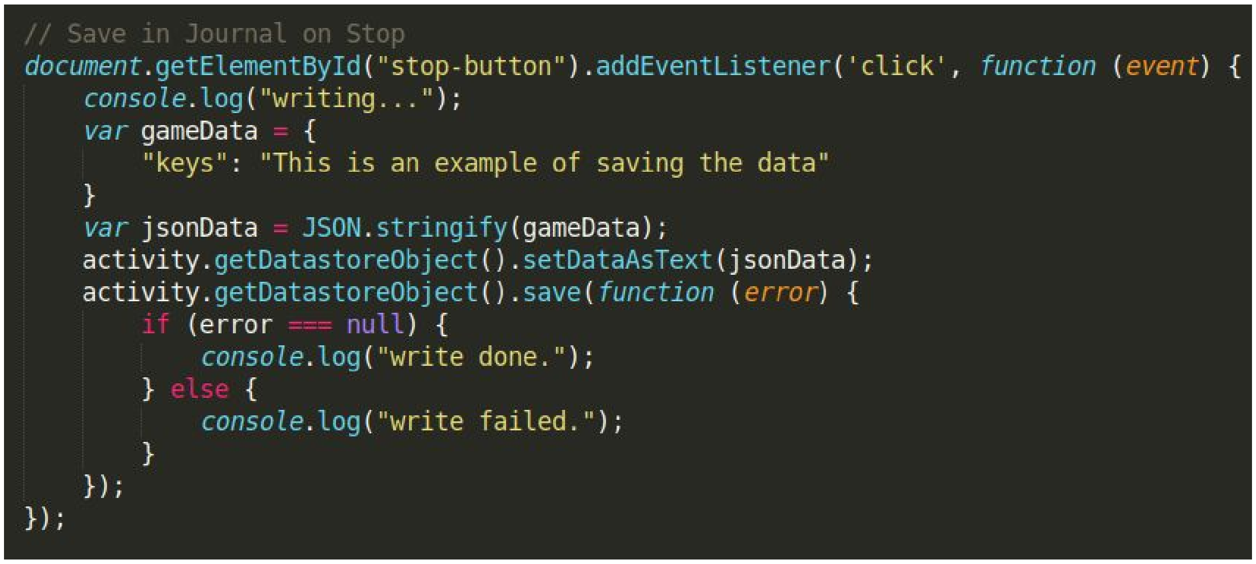


- Presence actions definition
    - The following Presence actions can be defined -
        - Init - Connect the user to the network as the host to be able to view the others in the lobby
        - sendReport - Send the data in JSON stringified form to the user.
        - requestReport - Request the report from a user present in the network. (Note: The user can send his report without waiting for the request also.)
    - The Progress Report component will handle the export if the user wants to generate the file.

- The flow diagram can be -


Start Activity
If existing instance is present or shared instance is true
Home Screen
Activity starts with datastore object or as a shared instance
If datastore object is present and shared instance is false
User shares the activity or starts the existing instance
Request and Send the progress resports
Home Screen
Export to file
Settings Mode
Share the activity
Share the activity
Export to file
Settings Mode

- Journal Integration
  - We can use the **LZ based compression algorithm for JavaScript** to compress the string before storing them in the Journal, taking inspiration from the Exerciser Activity.
  - The current game variables of game state will be taken as the data.
  - The state variables will be converted to a JSON object.
  - The JSON object will be stringified and stored in the Journal in the standard manner
  - According to the features, we can discuss and decide as to exactly which variables and data structures should be stored in the object.



```
// Save in Journal on Stop
document.getElementById("stop-button").addEventListener('click', function (event) {
    console.log("writing...");
    var gameData = {
        "keys": "This is an example of saving the data"
    }
    var jsonData = JSON.stringify(gameData);
    activity.getDatastoreObject().setDataAsText(jsonData);
    activity.getDatastoreObject().save(function (error) {
        if (error === null) {
            console.log("write done.");
        } else {
            console.log("write failed.");
        }
    });
});
```

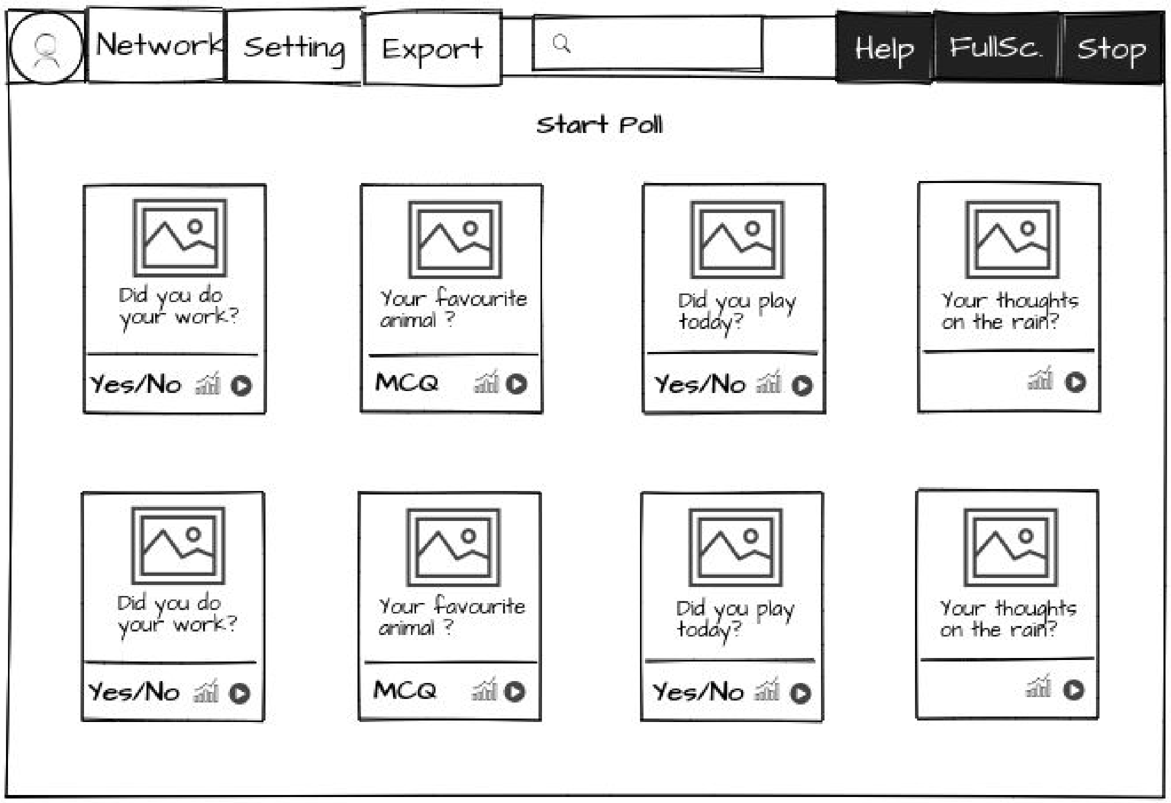
- Localisation
  - Localisation will be done with the webL10n Library.
- Tutorial
  - We will use Bootstrap 3 Tutorial, as in the majority of Sugarizer activities to make the tutorial interactive
- Responsiveness
  - We will use the Bootstrap 3 Library and the Sugar Web Library for building the UI and taking care of responsiveness on all the screens.
  - The application will be tested to function on all the major Web Browsers (Google Chrome, Mozilla Firefox, Safari, Microsoft Edge) and Electron.
- User Interface
  - This will be the basic layout of the UI. I propose to discuss with the mentors on where will be the best to remove text and use figures and geometrical shapes. The UI will be Sugar Friendly and will use Buddy Colors for the Background and for the User Icons. I emphasize on using less text and more figures, in respect of the Sugar Standards.
  - If we use Vue.js as an inline framework:
    - mounted() lifecycle hook can be accessed so that we can access the reactive components after the DOM loads.
    - created() lifecycle hook can be accessed in multiplayer mode to do the initial synchronisation of data before loading the DOM, like an API fetch
    - v-for directive can be used to render the arrays in the DOM
    - v-else and v-if statements can be used for conditional render in the DOM

        - watch() property can be used to update the DOM one the data in the game changes in the Logic
- Time to acquire (additional)
    - The time a skill is added can be taken into account when the user checks that the skill is acquired. The **difference can be taken as the time to acquire the skill**. Can be used to calculate the user rating and add a statistic to the side Panel.
- Import categories and skills from other users (additional)
    - We can add a feature to explore the categories and skills created by the other user to take inspiration and create our own set. The **Student Object {}** and **Skills [] Array** can be shared with the users over the network and the user can see the skill set in read only.

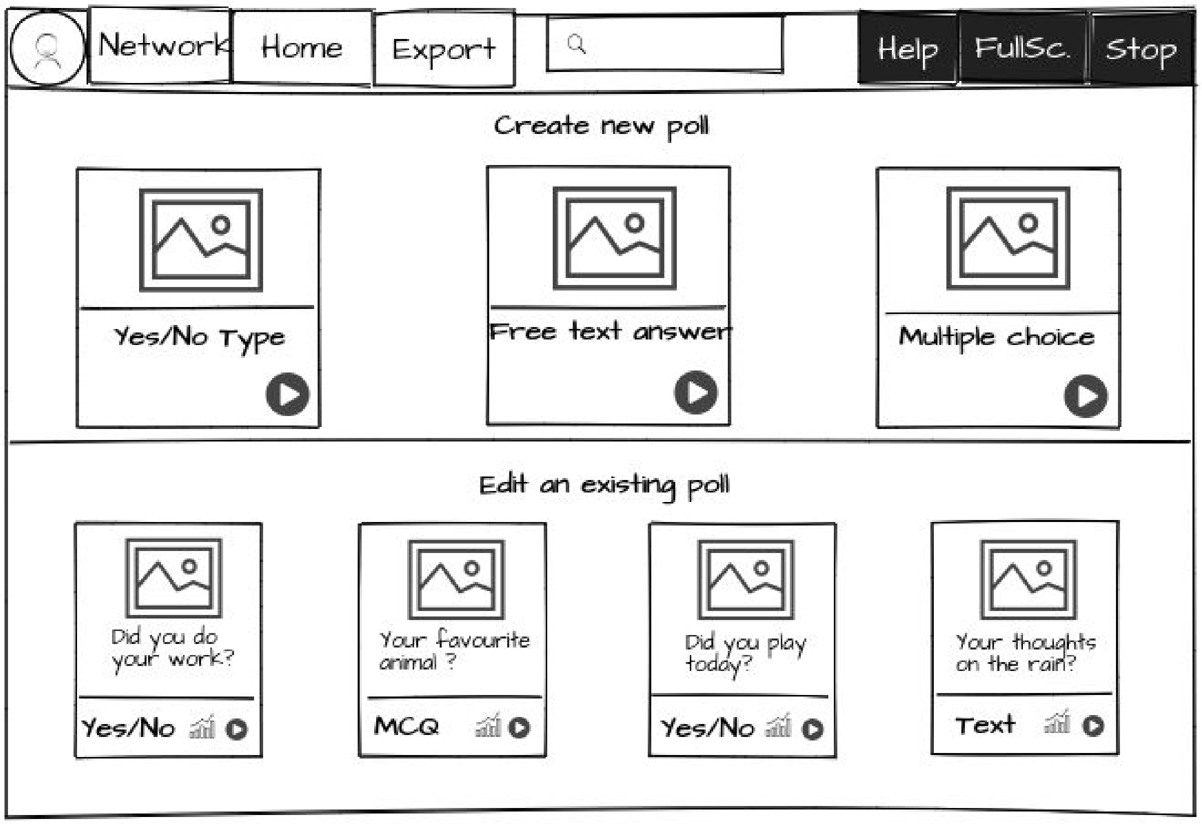
Vote Activity

The features proposed are discussed in the implementation level:

- Home Page
    - The home page will be similar to the **Exerciser Activity** - the user will be able to **start the poll from previously designed templates**.
    - There will be poll tiles for each of the designed polls.
    - The poll tiles will have -
        - The question
        - Type of the poll
        - Image
        - Play and Statistics Button
    - The poll can be started and stopped from the **play icon** in the poll tile.
    - The play button will start and open a **statistics page** for the host to see the live results of the poll in real time.
    - He can see **who first voted**, **who voted for what**, and other data. We will maintain the **timestamp** in the user response objects to implement this.
    - The stats button has two features -
        - Poll not started - The number of times this poll was taken, what were the results, will be visible.
        - Poll started - The live data related to the poll, the results and the users who have participated, will be shown.
    - For example - The poll with the question ‘Did you do your work’ is already designed by the user as a Yes/No type poll.
        - The multimedia (image, audio etc.) are already uploaded, and retrieved from the journal.
        - The title, description, options are also already defined.

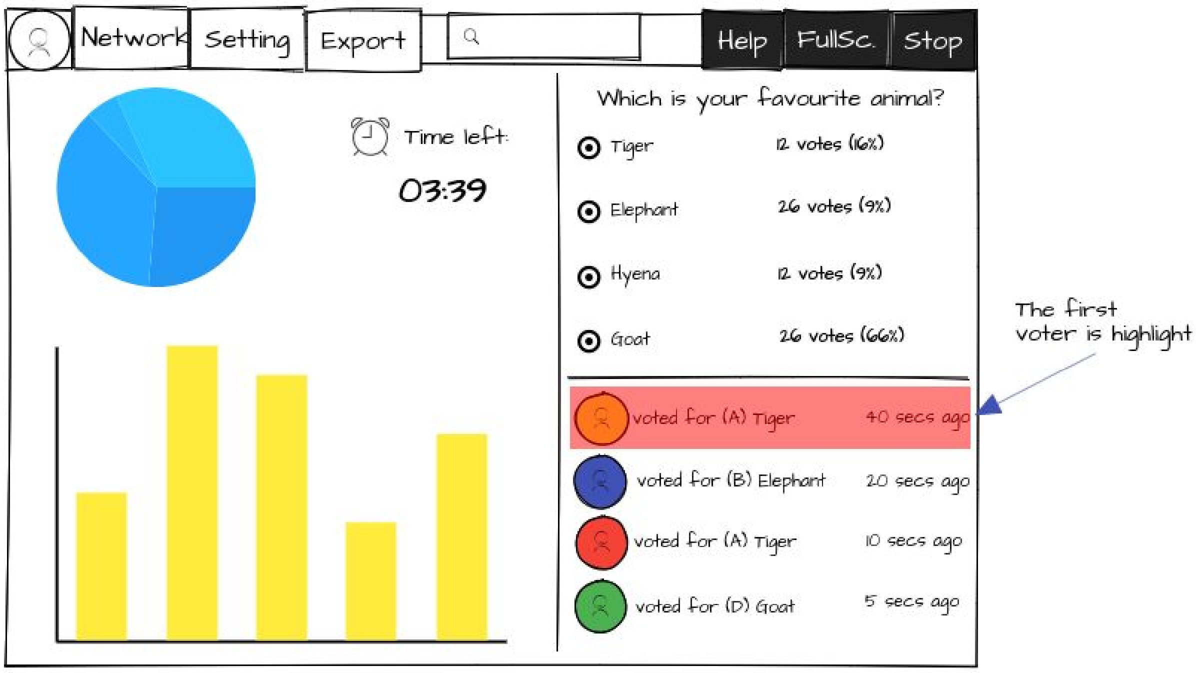


- Settings Page
    - The settings page will give three tiles in the top:
        - Yes/No poll template - To create a new Yes/No type vote
        - Free Input Text poll template
        - Multiple Choice Question poll template
    - The user can choose any one of the three and start building the poll. A screen will popup and will allow the user to fill in the requisite details and start the poll.
    - In the bottom, there will be an option to edit the existing polls.
        - We can change the question, type, time allocated, description, multimedia of the polls.
        - We can see the poll history and stats as discussed earlier



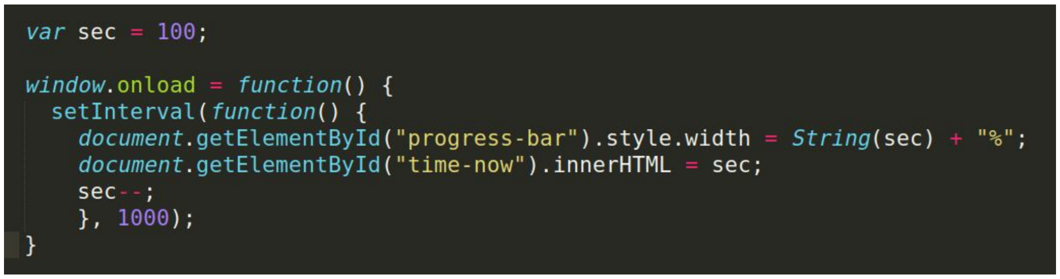
- Statistics Page
  - The statistics page will be shown to the host (and the users if the host allows) **when the poll is running** (or at the end, as the host decides). It will have charts and graphs and all the data required to analyse the results.
  - We will use Chart.js for the data analysis. It is a very versatile JavaScript library for HTML5 charts. It also redraws the charts on window resize for maintaining the response and scale.
  - There will be an element which will show the host the buddy icons, the option they voted for(in Yes/No, and MCQ) or the text they input(in Free Text) of the users as they vote.
  - It will take into account the timestamp of the user vote and show a scrollable list of users sorted according to earliest first.
  - The **first user to vote will be highlighted**.
  - The **current time of the system and the time the poll was started will be compared**, and the time left for the vote to finish will be displayed.
  - The type of graphs will change as the type of poll changes.

- More visual charts (pie charts, histogram) can be incorporated and more analysis metrics like most chosen responses can be displayed.
- The exact graphs and statistics to be displayed will be best discussed with the mentor.



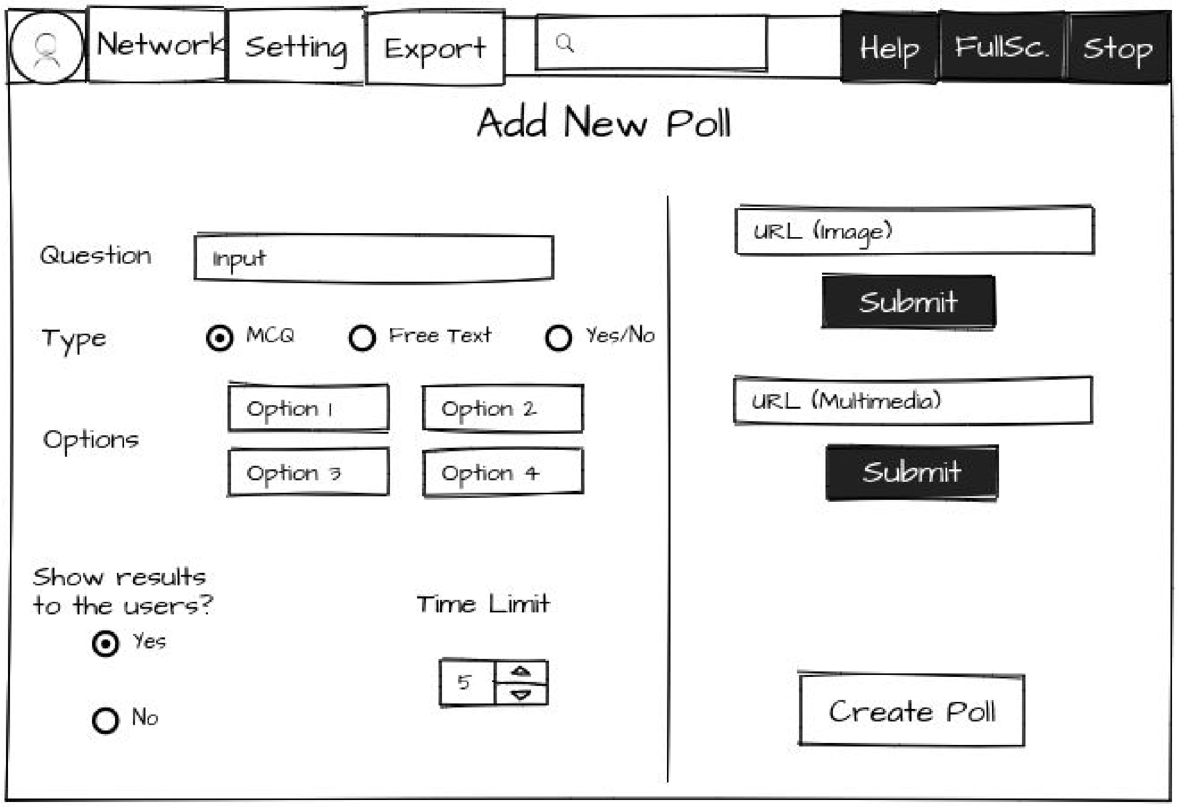
Network
Setting
Export
Help
FullSc.
Stop
Time left:
03:39
Which is your favourite animal?
Tiger 12 votes (16%)
Elephant 26 votes (9%)
Hyena 12 votes (9%)
Goat 26 votes (66%)
voted for (A) Tiger 40 secs ago
voted for (B) Elephant 20 secs ago
voted for (A) Tiger 10 secs ago
voted for (D) Goat 5 secs ago
The first voter is highlight

- Timer Integration
  - A countdown timer will be integrated to tell the time left for the poll.
  - From the start time of the poll and the user's current time, we will calculate the time left to finish the poll, in seconds.
  - We can make a variable called **sec** and set it to the number of seconds left. We can then count in the logic and update the DOM with the values. The DOM can be updated either by:
    - The **watch** property in Vue.js can listen to the updates in the value of the time count variable and update the DOM.
    - Updating the **<span>** with JS, we can use the **setInterval()** function and configure it to listen to the **window.onload** event and start counting



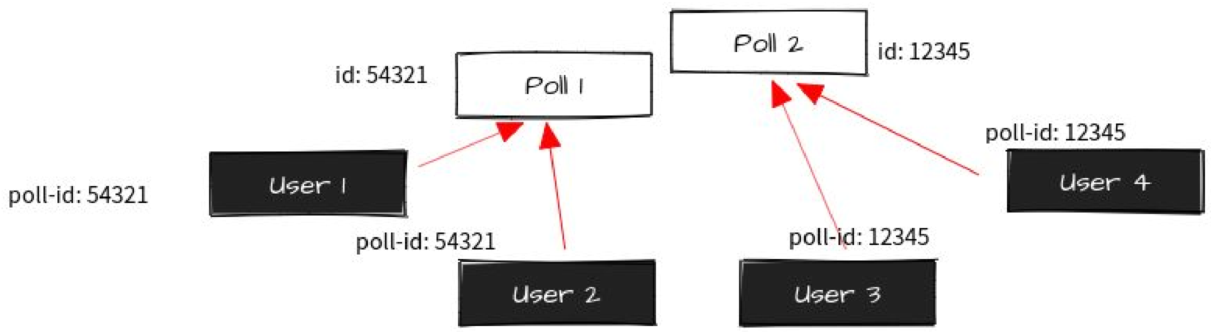
```
var sec = 100;

window.onload = function() {
  setInterval(function() {
    document.getElementById("progress-bar").style.width = String(sec) + "%";
    document.getElementById("time-now").innerHTML = sec;
    sec--;
    }, 1000);
}
```

- Create New Poll/Customise Existing Poll
    - The create and customise poll will have a similar UI and will give the user options to update the existing polls, or create a new poll out of the three types.



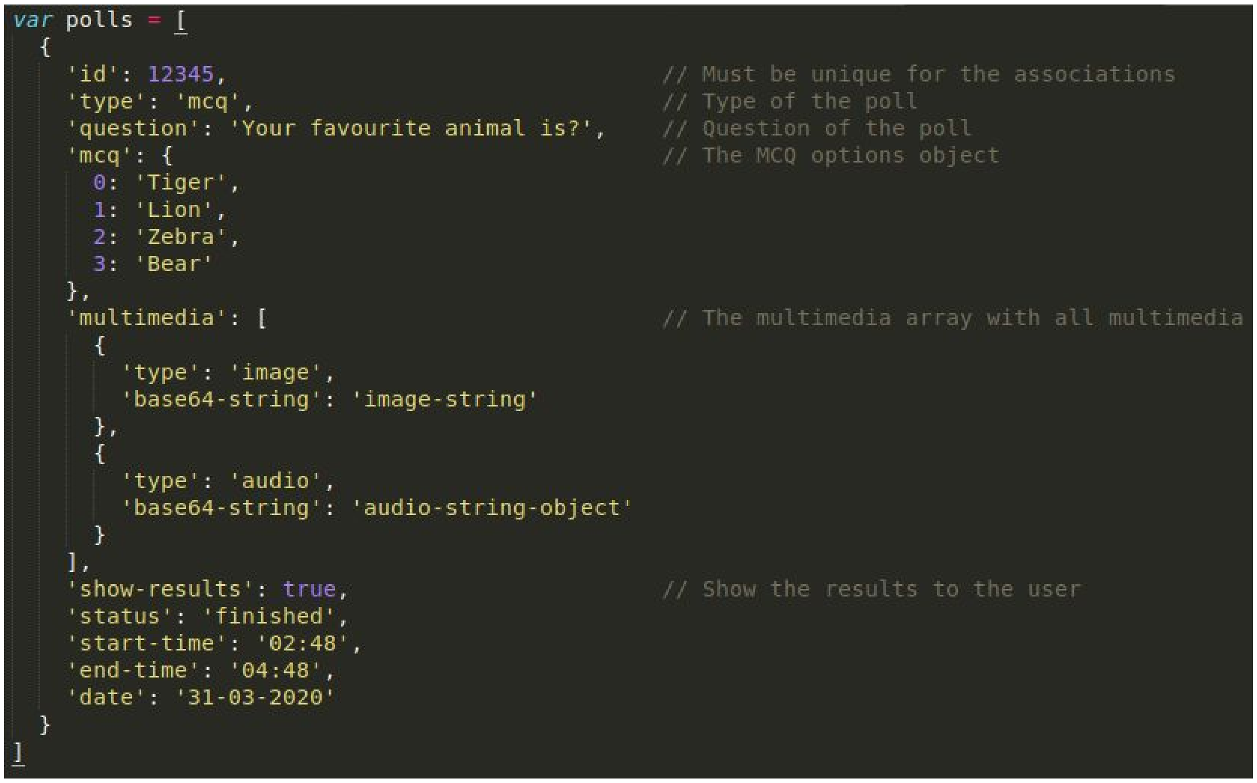
- The Journal Object
    - The data structures are:
        - Polls [] - The array of polls objects
        - Answers [] - The array of response objects obtained during a poll
        - currentlyRunning - Stores the information about the currently running poll in the runtime
        - voted - Maintains whether the user has voted for the currentlyRunning poll
        - isHost - Tells if the user is host or not and is utilised in checking privilege of the user.

    - The **Polls []** and the **Answers []** tables will be associated with the **poll-id** as the primary key. The tables association will be efficient because:



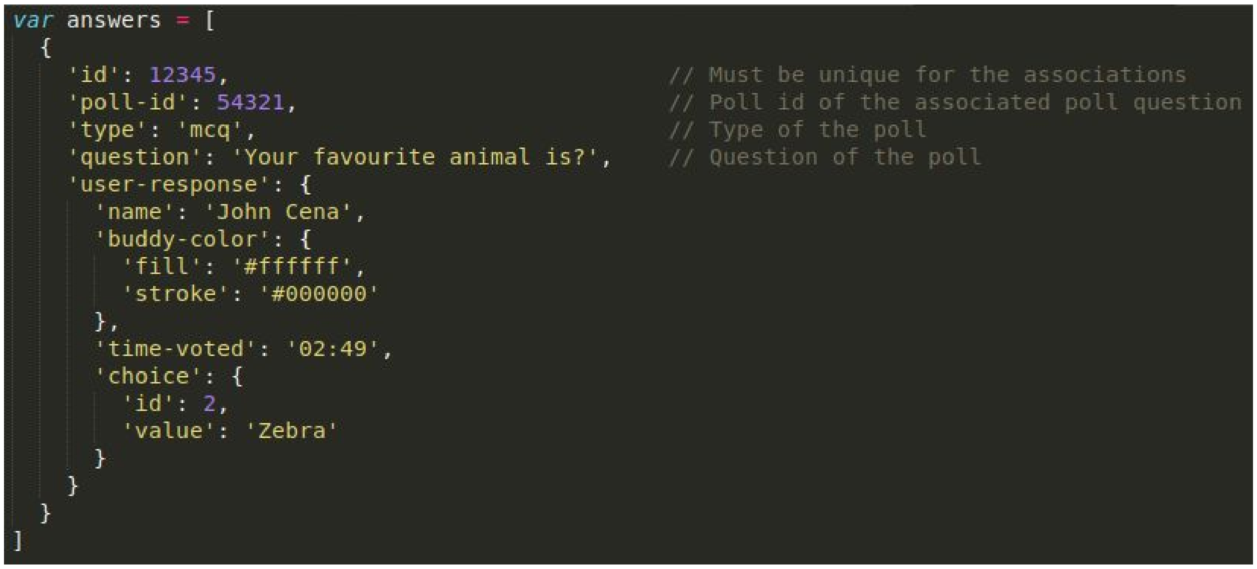
        - Data storage redundancy will be minimise
        - The **update when live voting will only have to be reflected in the Answers [] table** which contains the **user-response** objects.
        - A user answering multiple polls of the same or different users only needs to make an Answer object with the associated poll-id.

    - The sample polls object will be -



```
var polls = [
  {
    'id': 12345,                                    // Must be unique for the associations
    'type': 'mcq',                                  // Type of the poll
    'question': 'Your favourite animal is?',        // Question of the poll
    'mcq': {                                        // The MCQ options object
      0: 'Tiger',
      1: 'Lion',
      2: 'Zebra',
      3: 'Bear'
    },
    'multimedia': [                                 // The multimedia array with all multimedia
      {
        'type': 'image',
        'base64-string': 'image-string'
      },
      {
        'type': 'audio',
        'base64-string': 'audio-string-object'
      }
    ],
    'show-results': true,                           // Show the results to the user
    'status': 'finished',
    'start-time': '02:48',
    'end-time': '04:48',
    'date': '31-03-2020'
  }
]
```

    - The sample response object will be -



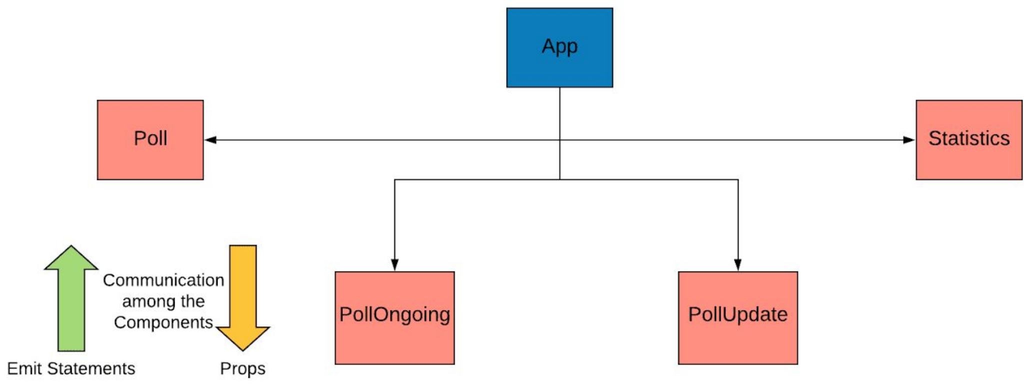
```
var answers = [
  {
    'id': 12345,                                 // Must be unique for the associations
    'poll-id': 54321,                            // Poll id of the associated poll question
    'type': 'mcq',                               // Type of the poll
    'question': 'Your favourite animal is?',     // Question of the poll
    'user-response': {
      'name': 'John Cena',
      'buddy-color': {
        'fill': '#ffffff',
        'stroke': '#000000'
      },
      'time-voted': '02:49',
      'choice': {
        'id': 2,
        'value': 'Zebra'
      }
    }
  }
]
```

The benefit of having such structure is **we only have to send the poll object once to all the connected users**, and it will not be disturbed. The **response objects coming to any user will be stored in the Answers [] array** and saved in the local database, so that we can use them for showing the results, and statistical purposes later. Only the answer object, which contains minimal information and is very lightweight, needs to be sent to other users.
Also, since people have different speeds on the internet, we can expect that there could be an anomaly in the order of response objects received by two users. For example, User 1 might get the response of User 3 before User 2, and User 4 might get the response of User 2 before User 3.
But, this will not have any adverse effects in this approach. **Since all the response objects are accumulated locally by each user, the results will be consistent across all users.** The only thing could be that the final updates might be slower in low network areas, but they will be correct nonetheless.

This method will ensure that even if the host leaves the network (due to probably a network error), the Answers [] array is distributed so the poll results can be retrieved upon connecting once again from the other users. This is a direct benefit of distributed data systems.

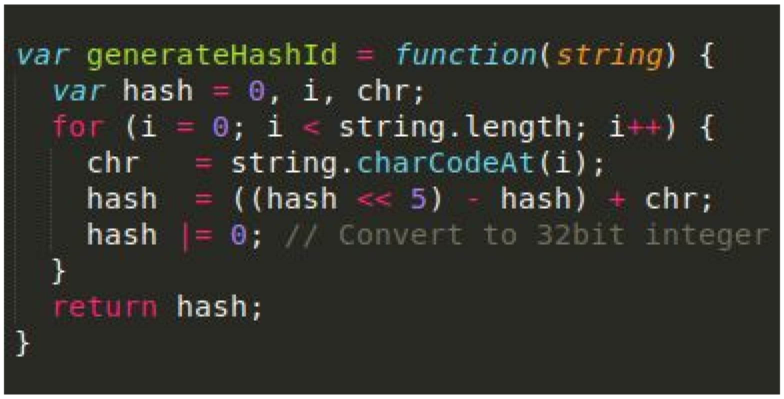
- The nature of the activity allows for us to have fragmentation into several Vue.js Components -
  - App - The root Component which handles the application and has all the data structures and other system components (toolbar, palette)

        - Statistics - The Component handles the data visualization methods and the Chart.js elements. This will be available to the Host all time but will be available to the user only if the host allows it.
        - PollUpdate - Handle the CRUD operation of the polls. Is visible from the settings mode.
        - PollOngoing - Handle the presence methods when the poll is started. Handles the DOM updates and the response objects from the users. Shows only the poll question and the option to users other than the Host.
        - Poll - Handle the Home screen. Has the poll templates both pre-defined and the user created ones. Can be used again in the settings page with addition to the edit mode.
    - Hierarchy among the Components can be shown -



The communication among the Components take place through **Props** and **Emit** Statements.

- We will have to make sure that all the Ids of the objects created are unique. For the same motive we will utilise the approach to make the strings into a Hash Value.

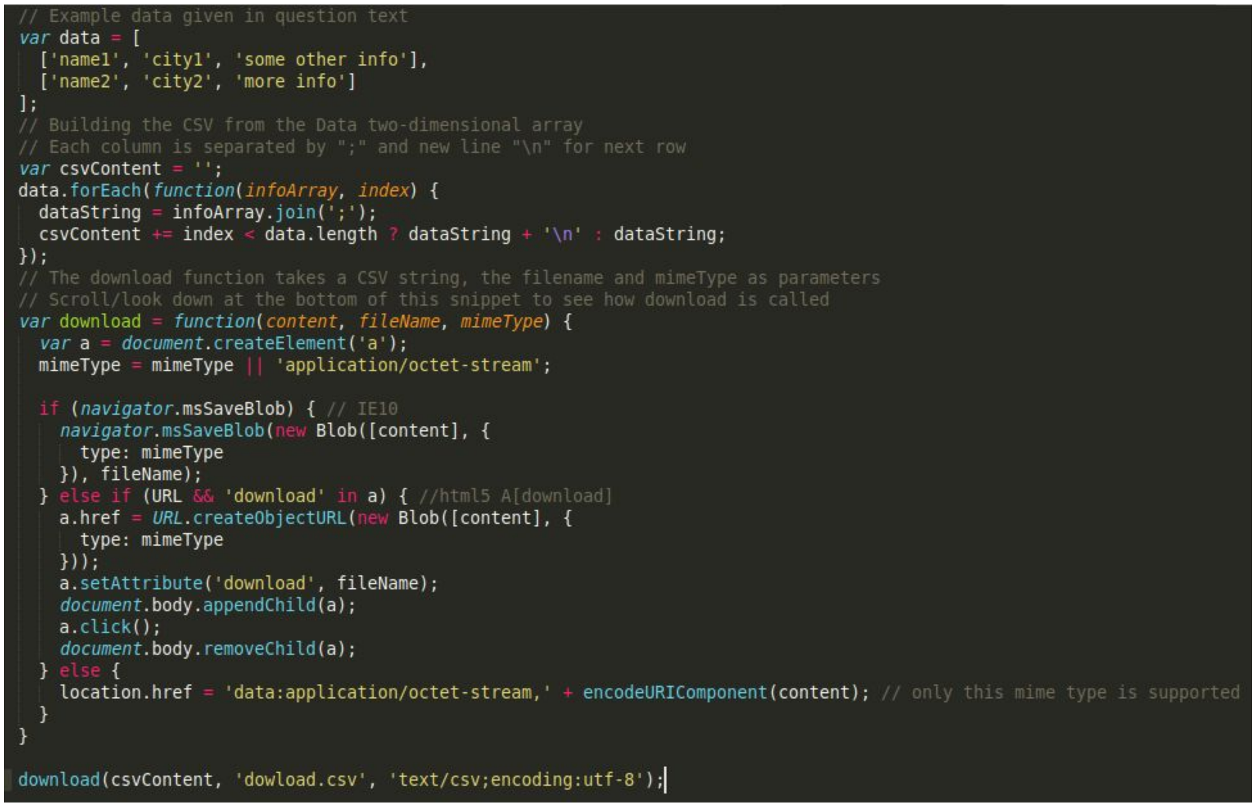


```
var generateHashId = function(string) {
  var hash = 0, i, chr;
  for (i = 0; i < string.length; i++) {
    chr   = string.charCodeAt(i);
    hash  = ((hash << 5) - hash) + chr;
    hash |= 0; // Convert to 32bit integer
  }
  return hash;
}
```

    - The hash value of any length string will be a fixed length string
    - The hash value of two different strings will never be equal
  - This will ensure that we have unique IDs for all the objects.

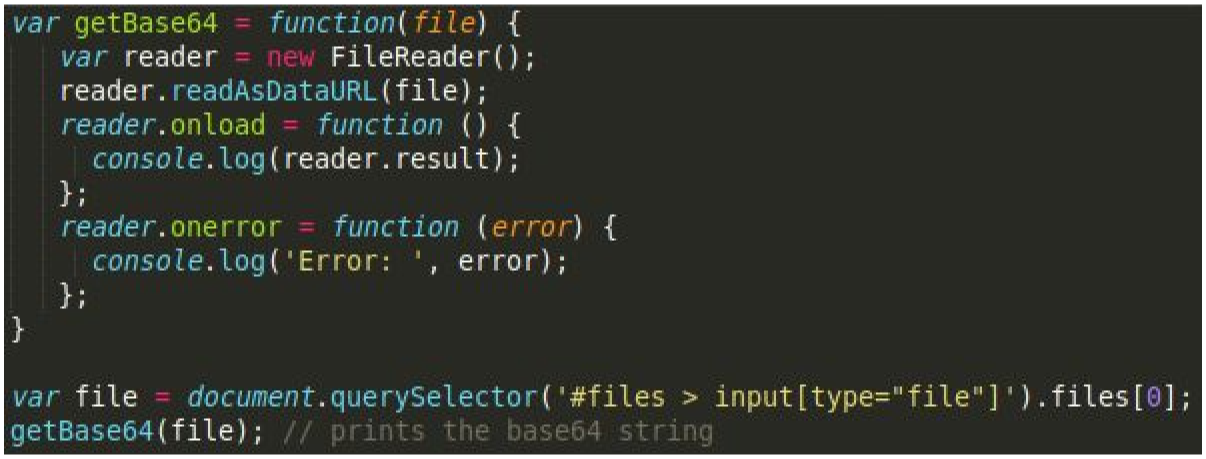
- Results Export to Files
  - PDF - We will export **the statistics page in external file format**. This option will be accessible to all who can see the result.
    We will follow the approach stated in the export file discussion of the previous activity and utilise the following libraries for the export -
    - **html2canvas** - The script allows us to take "screenshots" of webpages, directly on the user's browser. The screenshot is based on the DOM and will be used to generate the PDF.
    - **jsPDF** - Utilised with html2canvas to generate PDF documents.

  - CSV - We have the **Polls []** array and the **Answers []** array which have the data in JSON form, we can develop a method to take the data and make an array. We can dynamically make the array with the required data and call the given method to get the CSV.



```javascript
// Example data given in question text
var data = [
  ['name1', 'city1', 'some other info'],
  ['name2', 'city2', 'more info']
];
// Building the CSV from the Data two-dimensional array
// Each column is separated by ";" and new line "\n" for next row
var csvContent = '';
data.forEach(function(infoArray, index) {
  dataString = infoArray.join(';');
  csvContent += index < data.length ? dataString + '\n' : dataString;
});
// The download function takes a CSV string, the filename and mimeType as parameters
// Scroll/look down at the bottom of this snippet to see how download is called
var download = function(content, fileName, mimeType) {
  var a = document.createElement('a');
  mimeType = mimeType || 'application/octet-stream';

  if (navigator.msSaveBlob) { // IE10
    navigator.msSaveBlob(new Blob([content], {
      type: mimeType
    }), fileName);
  } else if (URL && 'download' in a) { //html5 A[download]
    a.href = URL.createObjectURL(new Blob([content], {
      type: mimeType
    }));
    a.setAttribute('download', fileName);
    document.body.appendChild(a);
    a.click();
    document.body.removeChild(a);
  } else {
    location.href = 'data:application/octet-stream,' + encodeURIComponent(content); // only this mime type is supported
  }
}

download(csvContent, 'dowload.csv', 'text/csv;encoding:utf-8');
```

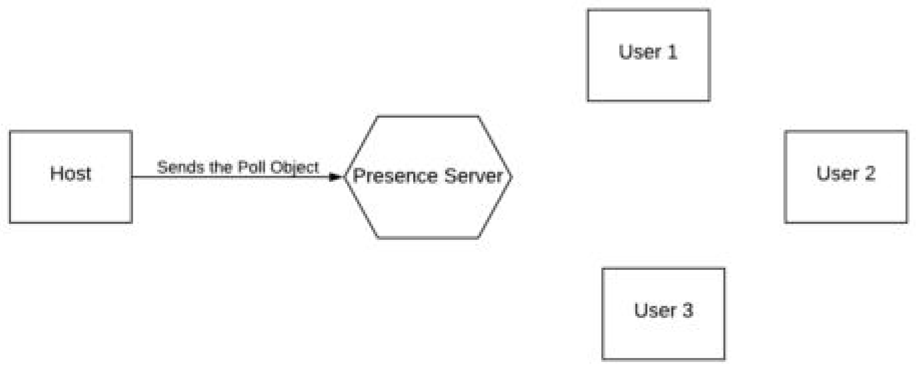
- Multimedia CRUD
  - We can utilise the **FileReader** class like discussed in the previous activity and shown here:



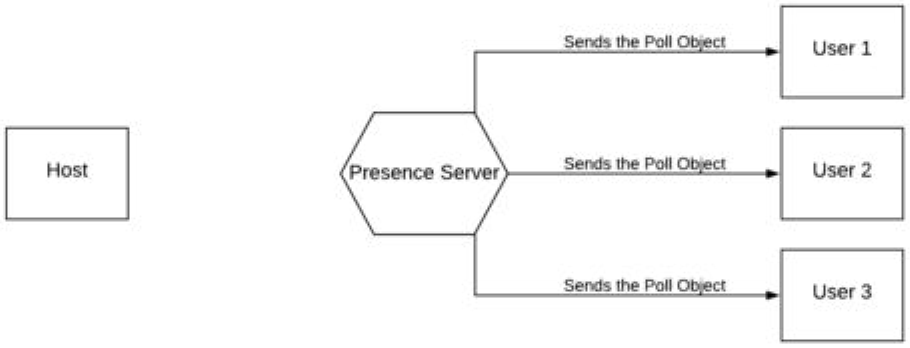
```javascript
var getBase64 = function(file) {
  var reader = new FileReader();
  reader.readAsDataURL(file);
  reader.onload = function () {
    console.log(reader.result);
  };
  reader.onerror = function (error) {
    console.log('Error: ', error);
  };
}

var file = document.querySelector('#files > input[type="file"]').files[0];
getBase64(file); // prints the base64 string
```

- Search feature
  - A search area will be present in the toolbar, which will allow the user to search for polls and poll templates.
  - All the matching poll templates will be displayed which are associated with the search query.

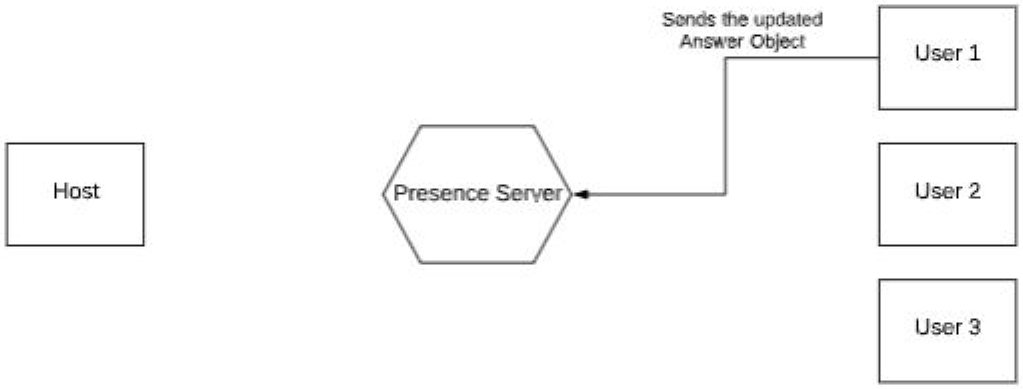
        - For implementing this feature, we will **maintain a dictionary of all the poll template names, with their object Id as the value**. An almost O(1) lookup can be designed
        - The results can be rendered with the **v-for** directive and **v-else** and **v-if** directives in Vue.js in the DOM
- Presence Integration
    - The following presence actions can be defined -
        - init - Connect the user and send any currentlyRunning polls, if possible.
        - startVote - The selected poll is shared to all the users by the host and the poll is started.
        - sendResponseObject - Send the response object to the presence server, which will send to all the users.
        - showResults - Always enabled for the host and if decided by him, also enabled for the users. Displays the results of the poll from the locally accumulated Answers [] array formed from all the response objects.
    - The activity is network intensive, so we define the workflow for the activity sharing:
        - The host shares the activity and sends the Poll object to the presence server.



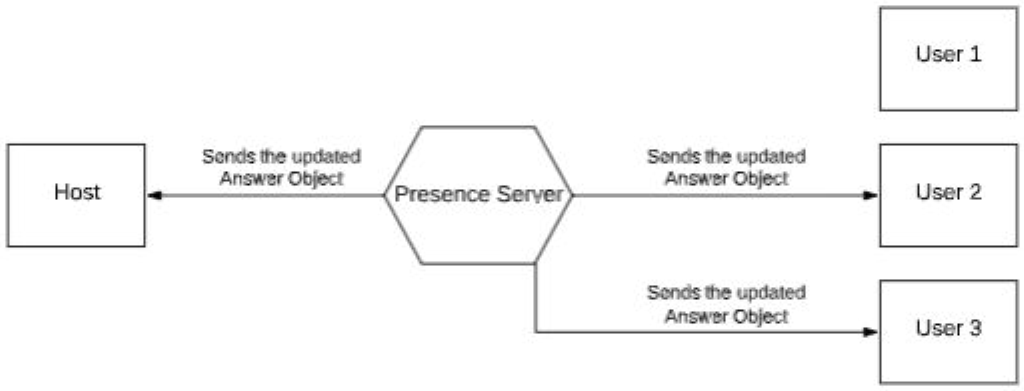
        - The presence server sends the poll object to all the connected users. They will be able to get all the data to **form the Poll in their local system**.



- Now, suppose User 1 wants to Answer, **he will make the Answer Object with all the necessary information, which includes his**, and only his, response to the poll.



- The presence server now sends the **response object to all the other users**, including host. A listener will listen to the response object and do two things:
    - **Update the local Answers [] array, which accumulates all the response objects from all the users**. Note that there will be no race condition as the array will only increase in size and is cumulative. **It does not depend on the order in which the response objects arrive** to any user's computer.
    - Update the DOM with the new response object. According to the configuration settings, the host can decide if he wants all the users to see the changes live or only he can do.


User 1
Host
Sends the updated Answer Object
Presence Server
Sends the updated Answer Object
User 2
Sends the updated Answer Object
User 3

- Common use cases -
  - **Add a poll** - Go to settings and click on the template out of the three given. The **PollUpdate Component is displayed**. A form will be popped up. Fill with the details and upload the multimedia files from the Journal. The **data is emitted from the PollUpdate Component and sent to the App Component**, where the database manipulation happens, and will be **pushed to the Polls [] array**.

  - **Edit a poll** - Similar to adding a poll, go to settings and click on one of the existing polls in the bottom panel of the screen. A form will popup with the previously filled data. The user can update as desired and the **data will be sent to the App component from where the Polls [] array will be updated** (as we have the poll-id). However, **the older instance will be kept** in the **history** array with poll-id to show the statistical history.

  - **Delete a poll** - The **poll object is removed from the Polls [] array**, and since Vue components are reactive the change is reflected simultaneously.

  - **Start a poll** - **currentlyRunning will point to the selected poll object and pollOngoing Component is displayed and starts handling**. The activity will be shared with the network as soon as the play button is pressed. **isHost is set to true**. The **startVote presence action** is utilised to send the poll object to the connected users.

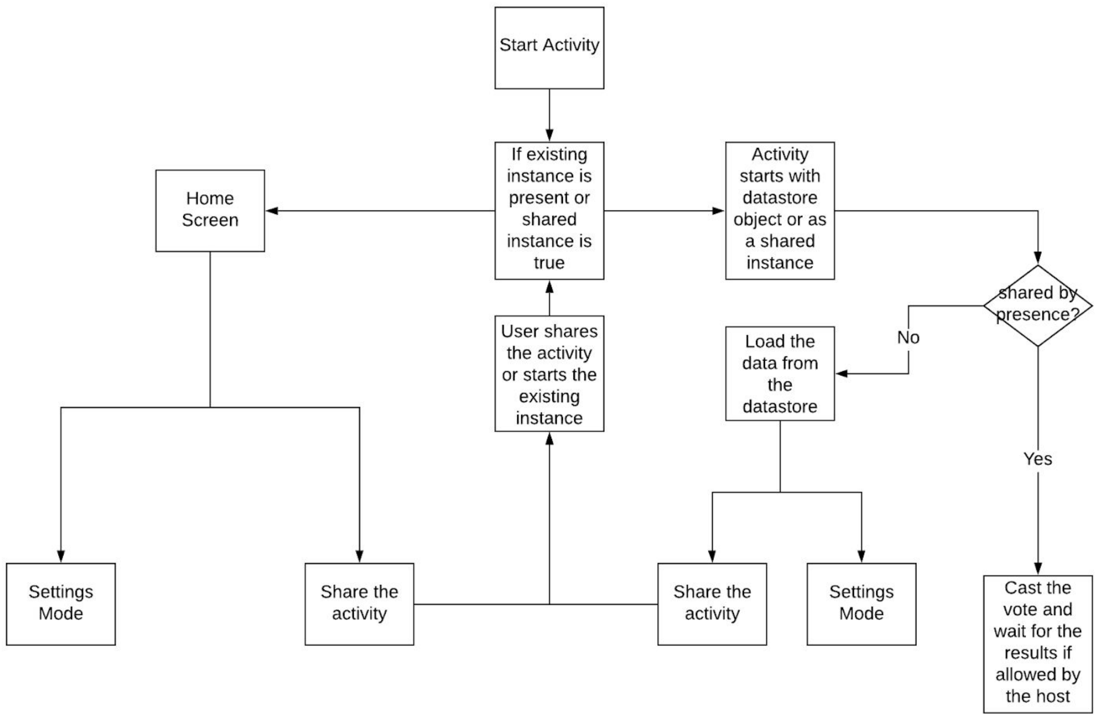
  - **Vote casting** - When a new user joins, **the pollOngoing Component** shows him the question and the input fields. Once the vote is casted, a thank you message is shown, along with a timer and an option to see the live results, if the Host has allowed this. **The sendResponseObject**

presence action is used to send the vote to the server. Each user maintains an Answers [] array and increments it with the response objects coming in by the network from the other users.

- **Displaying history** - **If a poll is not running, clicking on the statistics button will show the history of the poll**, exactly how many times the poll was run, with the dates and what were the exact results. The journal will store the old instances of the poll objects. **All the poll objects having the same poll-id are the time varied instances of the same poll**. ShowResults presence action is utilised to send the notification to all the users.

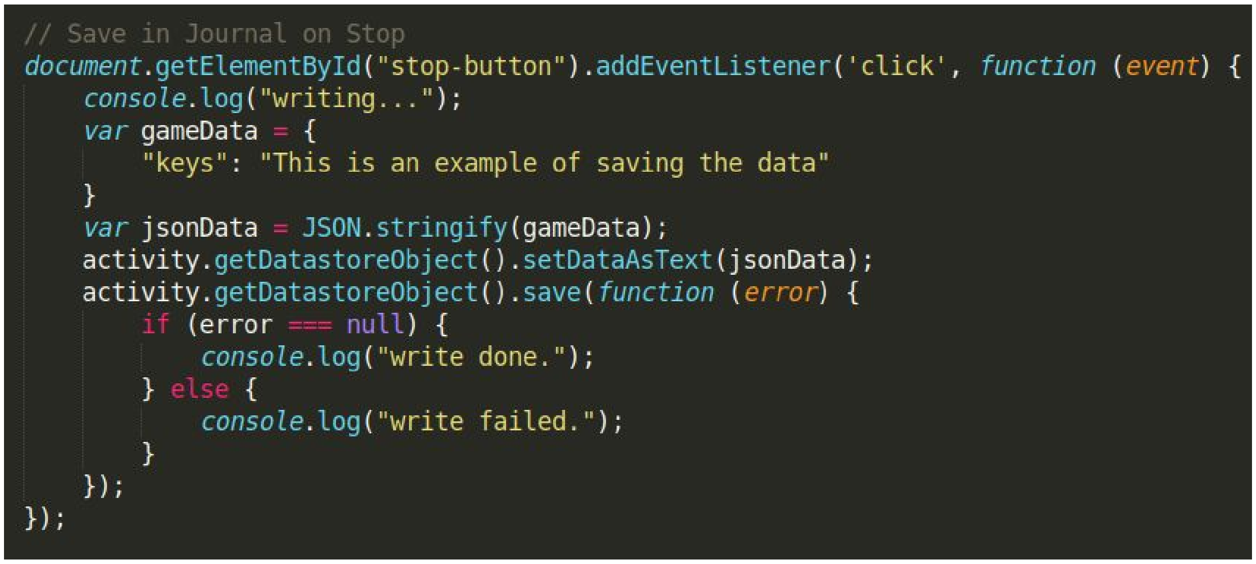
- **Results** - When the poll ends, by the host or by the timeout, the statistics are displayed. **The Statistics Component starts handling and the templates are displayed**. The **results are saved in the poll object** with the host. They are saved for future purposes.

- The flow diagram can be -


Start Activity
If existing instance is present or shared instance is true
Home Screen
Activity starts with datastore object or as a shared instance
shared by presence?
No
Load the data from the datastore
User shares the activity or starts the existing instance
Yes
Settings Mode
Share the activity
Share the activity
Settings Mode
Cast the vote and wait for the results if allowed by the host

- Journal Integration

    - We can use the **LZ based compression algorithm for JavaScript** to compress the string before storing them in the Journal, taking inspiration from the Exerciser Activity.
    - The current game variables of game state will be taken as the data.
    - The state variables will be converted to a JSON object.
    - The JSON object will be stringified and stored in the Journal in the standard manner
    - According to the features, we can discuss and decide as to exactly which variables and data structures should be stored in the object.



```
// Save in Journal on Stop
document.getElementById("stop-button").addEventListener('click', function (event) {
    console.log("writing...");
    var gameData = {
        "keys": "This is an example of saving the data"
    }
    var jsonData = JSON.stringify(gameData);
    activity.getDatastoreObject().setDataAsText(jsonData);
    activity.getDatastoreObject().save(function (error) {
        if (error === null) {
            console.log("write done.");
        } else {
            console.log("write failed.");
        }
    });
});
```

- Localisation
    - Localisation will be done with the webL10n Library.
- Tutorial
    - We will use Bootstrap 3 Tutorial, as in the majority of Sugarizer activities to make the tutorial interactive
- Responsiveness
    - We will use the Bootstrap 3 Library and the Sugar Web Library for building the UI and taking care of responsiveness on all the screens.
    - The application will be tested to function on all the major Web Browsers (Google Chrome, Mozilla Firefox, Safari, Microsoft Edge) and Electron.
- User Interface
    - This will be the basic layout of the UI. I propose to discuss with the mentors on where will be the best to remove text and use figures and geometrical shapes. The UI will be Sugar Friendly and will use Buddy Colors for the Background and for the User Icons. I emphasize on using less text and more figures, in respect of the Sugar Standards.
    - If we use Vue.js as an inline framework:

        - mounted() lifecycle hook can be accessed so that we can access the reactive components after the DOM loads.
        - created() lifecycle hook can be accessed in multiplayer mode to do the initial synchronisation of data before loading the DOM, like an API fetch
        - v-for directive can be used to render the arrays in the DOM
        - v-else and v-if statements can be used for conditional render in the DOM
        - watch() property can be used to update the DOM one the data in the game changes in the Logic

Project Timeline

The timeline incorporates all the considerations I can take into account at the time of developing, including the COVID-19 pandemic. My semester examination will be tentatively finished by the 10th of May, so I will put in extra hours to cover up for the deviation. I have no other projects in the Summer and can devote **40-50** hours per week (approximately 6-7 hours per day). The timeline is very flexible, and the exact dates can be decided with discussion with the mentors.

Keeping in mind the nature of the activities, I would like to work in the **Incremental** manner, developing the basic prototype first and then enhancing it with new features.

I have made the project into modules, and hereby explain what are the deliverables for each week -

Community Bonding Period (4 May to 1 June) -

- Discuss with the mentor and the members of the community about the use cases, technologies to be utilised and the best practices to incorporate.
- Discuss with the mentors about coding practices and how to exactly implement the features. Try to decide upon the activity state variables and other aspects according to the use cases and the requirements.
- Start Blogs about the project and update regularly.

Week 1 (June 1 - June 7) -

- Basic UI and Bootstrap 3 incorporation

- Skill tile Component and UI
- Categories Logic and UI
- Journal Integration

Week 2 (June 8 - June 14) -
- CRUD Skills
- CRUD Categories
- Bottom Panel Component and UI
- Side Panel Logic and UI
- Update the Blogs

Week 3 (June 15 - June 21) -
- Export files
- CRUD Multimedia
- Buffer time for errors and bugs

Week 4 (June 22 - June 28) -
- Localisation
- Hashing Function Integration
- Presence Integration
- Star rating system

Evaluation 1 (June 29 - July 3) -
- First prototype of the Curriculum Activity
- Everything except the tutorial and the documentation
- Update the Blogs

Week 5 (July 4 - July 10) -
- Testing on web browsers (Google Chrome, Mozilla Firefox, Safari, Microsoft Edge) and Electron
- Documentation and tutorial (Curriculum Activity)
- Update the Blogs
- Basic UI and Bootstrap 3 integration (Vote Activity)
- Polls Homepage (Vote Activity)

Week 6 (July 11 - July 17) -
- MCQ template and database object creation
- Yes/No template and database object creation
- Free Text Input template and database object creation
- Journal Integration

Week 7 (July 18 - July 24) -

- CRUD Polls
- Polls Settings Page
- Statistics Page
- Update the Blogs

2 Days before Evaluation 2 (July 25 - July 26) -

- Discussion and deliberation over the best methods for the presence integration
- Buffer time for errors and bugs

Evaluation 2 (July 27 - July 31) -

- Fully completed Curriculum Activity
- Partially completed Vote Activity
- Incorporates all the basic CRUD functionalities and the journal integration

Week 8 (August 1 - August 7) -

- Presence Integration (all the module)
- Buffer time for error and bugs

Week 9 (August 8 - August 14) -

- Export feature
- Timer Integration
- Search feature

Week 10 (August 15 - August 21) -

- Localisation
- Tutorial and Documentation
- Buffer time for errors and bugs
- Update the Blogs

2 Days before the Final Evaluation (August 22 - August 23) -

- Testing on different Web Browsers and on the Electron
- Extra time for unforeseen work and errors.

Final Evaluation (August 24 - August 31) -

- Completed both the activities
- The documentation and tutorial is finished
- The blogs are updated.

How will it impact Sugarlabs?

The curriculum activity will provide a reliable channel for self assessment. The importance of integrity and self discipline is known to us. I believe that the addition of the Curriculum Activity will help the students imbibe a healthy competitive temperament, which will help them keep a track of their skills in a plethora of curricular and co-curricular areas, and also be able to learn from each other. It will also provide a way for teachers to assess the students and see which kind of skills they want to learn.

The vote activity is very important, for it will provide a quick and trustworthy way of organising quick polls and help everyone voice their ideas. It will also be interesting for the students to make polls on new and trending problems and take the opinion of all their friends. This will be a great addition to the classroom, where the teachers will be easily able to see how the class is responding to the lessons, and whether they can answer satisfactorily. The activities will be great additions to the already highly valuable Sugarizer Activities.

My plans post GSoC 2020

I will be maintaining the activities and fixing the error and bugs regularly, and will keep discussing to further improve the activities. Since the activities are practical use cases based, we might require frequent feature changes and updates. I will be available all round the year for the issues related to the activities and all the Sugarizer activities.

I have been contributing and engaging in conversations and discussion in Sugarlabs and Sugarizer for more than a year. I will continue to do my best to be an active member of the community. I will love to welcome new members to the community and mentor when possible in the upcoming events. I would like to take this opportunity to thank Sugarlabs for being such a supportive place for my Open Source Journey, and all the admins, mentors and contributors, who have been wonderful in making the organisation a constructive and beautiful place to work and keep learning and developing for the students all over the world.