Manav Aggarwal
manavagr1108@gmail.com
Chhattisgarh, India
Timezone: Indian Standard Time (UTC+5:30)

GSoC Project Proposal for Studio

Facilitating H5P and SCORM import in Kolibri Studio

Personal Details and Contact Information:

Name: Manav Aggarwal.
Github: @manavagr1108
Preferred Name: Manav Aggarwal.
Email: manavagr1108@gmail.com.
University: National Institute of Technology Tiruchirappalli, Tamil Nadu, India.
Location: Raigarh, Chhattisgarh, India.
TimeZone: Indian Standard Time (UTC+5:30).
LinkedIn, Resume.

Project Title:

Facilitating H5P and SCORM import in Kolibri Studio

Synopsis:

Kolibri Studio is a content curation tool for Kolibri. Its primary purpose is to provide an easy and convenient way to organize learning resources and build channels for Kolibri. The Kolibri Studio user interface allows users to upload many different file formats for use in Kolibri. Currently, most of Kolibri's content resources are simple file formats such as audio and video files, PDFs, EPUBs, and zip files containing self-contained HTML5 apps.

However, many different container formats are used for educational resources that are not only self-contained but also bring additional metadata that Kolibri Studio should extract to add helpful information for learners and coaches in Kolibri. Two of the most prevalent are H5P and the IMS Content Package (commonly referred to as SCORM packages).

Lastly, while some container formats are self-contained and include everything needed to access the resource, many content creators assume that the end user will have access to a reliable Internet connection and include web URLs within the archive's contents to load additional resources. For users of Kolibri, this constitutes a huge barrier.

Mentors:

Richard Tibbles, José Redrejo

Duration:

250 hrs in 20 weeks

Difficulty Level:

Medium

Problems to Look for:

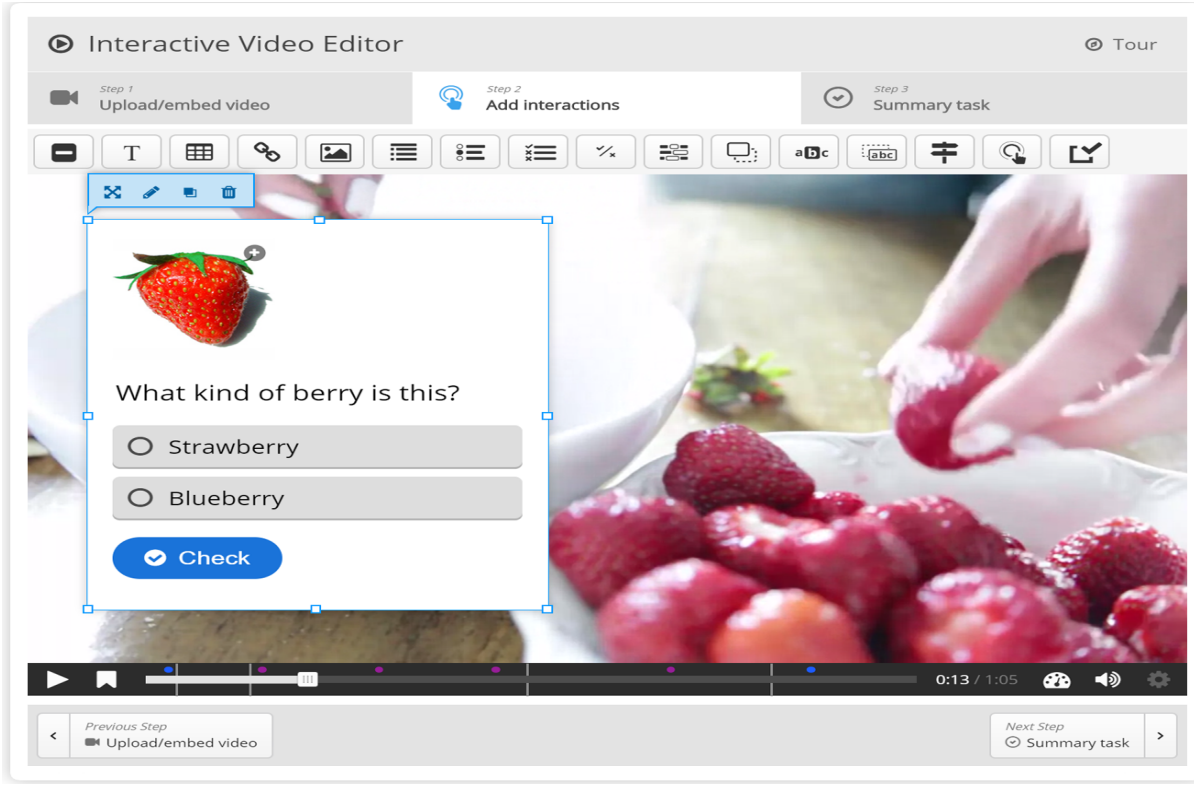
1. Currently, we cannot import H5P and SCORM files into a Kolibri Studio Channel, which only allows us to partially utilize the Kolibri Rendering Algorithm.
2. The Kolibri Studio Rendering Algorithm supports the rendering of these files, but due to the lack of content, it is not being utilized to its full potential.

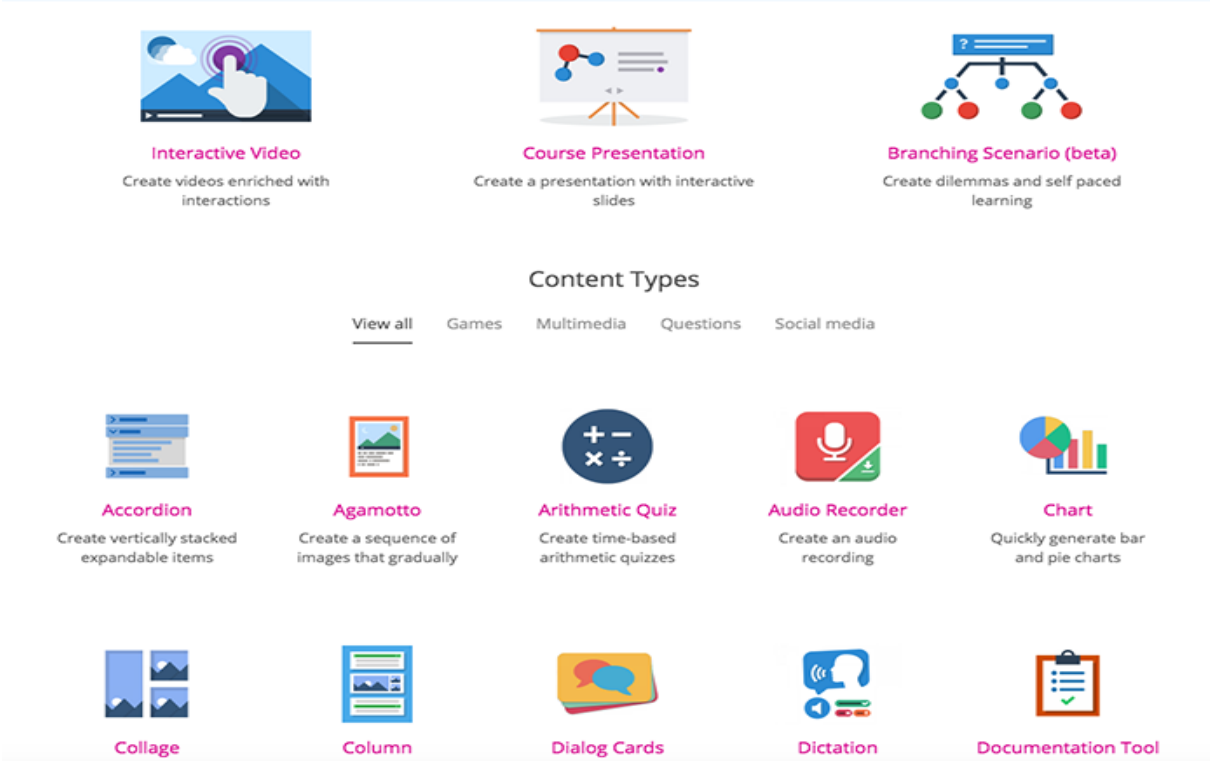
3. The current Kolibri Studio Needs a method to extract and store this data in a Data Structure that could be used to render it to the users.
4. Along with this, other than H5P and SCORM files, there can be many files like EPUB, ZIP, and other archive files that may contain a reference to external resources embedded in the HTML, CSS, Javascript, and other schematized JSON files. The user uploading these archive files should be informed about this beforehand.
5. The user should be informed about these external resources due to various reasons:
    a. It may contain Copyright issues.
    b. It may contain Personal resources.
    c. It may contain Restricted resources.
    d. It may be Unauthorized resources.
6. Suppose these external resources do not hinder any individual or organization. In that case, it should be Kolibri's job to download these external files and upload them to the database for the user's usage in offline mode.

Benefits to Community:

H5P (HTML5 Package) and SCORM (Sharable Content Object Reference Model) are e-learning content authoring tools offering different functionalities and use cases.

H5P is an open-source content authoring tool that allows educators and content creators to create interactive learning objects, such as quizzes, presentations, simulations, **and** interactive videos. H5P provides a wide range of content types that can be easily integrated into various e-learning platforms, including Moodle, WordPress, and Drupal. H5P's content types are based on HTML5 technology, which makes them responsive, mobile-friendly, and compatible with most devices and browsers. Some of the primary uses of H5P include creating engaging and interactive content, assessing learners' knowledge and skills, and enhancing learner retention.


Interactive Video Editor
Tour
Step 1
Upload/embed video
Step 2
Add interactions
Step 3
Summary task
What kind of berry is this?
Strawberry
Blueberry
Check
0:13 / 1:05
Previous Step
Upload/embed video
Next Step
Summary task


Interactive Video
Create videos enriched with interactions
Course Presentation
Create a presentation with interactive slides
Branching Scenario (beta)
Create dilemmas and self paced learning
Content Types
View all
Games
Multimedia
Questions
Social media
Accordion
Create vertically stacked expandable items
Agamotto
Create a sequence of images that gradually
Arithmetic Quiz
Create time-based arithmetic quizzes
Audio Recorder
Create an audio recording
Chart
Quickly generate bar and pie charts
Collage
Column
Dialog Cards
Dictation
Documentation Tool

On the other hand, SCORM is a set of technical standards that defines how e-learning content should be packaged and delivered across different learning management systems (LMS). SCORM content packages typically contain a collection of multimedia assets, such as images, videos, audio files, and instructional materials, such as quizzes, assessments, and interactive learning objects. SCORM content packages are designed to be interoperable and reusable to share and reuse across different LMS platforms. Some of the primary uses of SCORM include creating standardized e-learning content, tracking learners' progress and performance, and facilitating the exchange of e-learning content across different organizations.

Significant improvements that will be addressed after implementation will be:

- Users can utilize the H5P and ISM (SCORM) content packages by uploading them directly to the channels.
- Improvement in the file validation for external resources.
- Users can directly access the external resources once validated.
- This project can be the base for other e-learning models to be incorporated into Kolibri Studio

Current Status of the Project:

1. The Kolibri Studio does not allow users to upload H5P and SCORM files. Facilitating this is one of the main goals of this project.
2. Kolibri supports the rendering of H5P resources, but currently, these resources can only be imported into Kolibri Studio via the ricecooker library.
3. Kolibri Studio cannot detect and extract data metadata from Archive files like H5P, SCORM, EPUB, or ZIP.
4. Kolibri Studio should be able to download external resources attached with HTML, CSS, Javascript, and other schematized JSON appropriate to the file format.

Goals:

1. Extract Metadata from H5P files to extract and populate the metadata for the Content Node.
    a. Allow Kolibri to upload files with extension **.h5p**.
    b. Then extract the required(maps along with the ContentNode) metadata from the files uploaded.
    c. Upload the metadata along with the files through the post request.
    d. The metadata required is:
        - Title
        - Language
        - License (optional **in** h5p.json file)
        - CopyWrite Holder (same **as** authors **in** h5p.json)
2. Extract Metadata from IMS Content Package (SCORM) and add required files and folders to the Content Package that is used to map the contents of the SCORM folder.
    a. Allow Kolibri to upload files with the extension **.zip or .imscc**.
    b. Then extract the required(maps along with the ContentNode) metadata from the files uploaded.
    c. The metadata required is:
        - Title
        - Language
        - License (optional **in** h5p.json file)
        - CopyWrite Holder (same **as** authors **in** h5p.json)
    d. Create a Course tree(JSON format) that creates the topic nodes and resource nodes in Studio that correspond
    e. Upload the metadata, and topic tree along with the files through the post request.
3. Determine if the uploaded Archived files (H5P, SCORM, EPUB, ZIP) contain any kind of reference to external links.
    a. Recursively read all the files that are being uploaded and if any kind of links or anchor tags or URLs are found, we can suggest to the user that there are references to the external links in the files, and if the user wishes to continue then upload the files to google cloud, else discard the process.
4. Stretch Goal: If any reference resources are attached, download them and add them to the Archived file to be used in offline mode.

Deliverables:

Deliverables 1

Expected by evaluation 1:

1. Implement the metadata extraction for both H5P **and** SCORM files.
2. Generate extra file resources for mapping Topics Nodes and Resource Nodes.

Deliverables 2

Expected by evaluation 2:

1. Implement the algorithm to check for any external resource links embedded in the files.
2. List the resource links and URLs and download the resources for offline usage.

Approach:

Implement the metadata extraction for **H5P** files:

1. **Allow Studio to import .h5p files:**

   Currently, only mp3, pdf, epub, mp4, webm, and zip these files can be imported to the studio. We will need to allow users to upload .h5p files. We can do this by changing the Display metadata for h5p files in `resources/presetlookup.json`

   File changes include:

```
"h5p": {
                "readable_name": "H5P",
                "multi_language": false,
                "supplementary": false,
                "thumbnail": false,
                "subtitle": false,
                "display": true,  // changed this to true.
                "order": 1,
                "kind": "h5p",
                "allowed_formats": ["h5p"],
                "convertible_formats": []
        },
"h5p_thumbnail": {
                "readable_name": "H5P Thumbnail",
                "multi_language": false,
                "supplementary": false, // changed this to false.
                "thumbnail": true,
                "subtitle": false,
                "display": true,
                "order": 2,
                "kind": "h5p",
```

```
            "allowed_formats": ["png", "jpg", "jpeg"],
            "convertible_formats": []
        },
```

2. **Decide Metadata That needs to be extracted:**

   The metadata that is required to be extracted needs to be figured out: let's take a look at the files that we will need to sync.

   a. The metadata needs to be aligned with the Content Node of Kolibri Studio. The data structure is as follows:

```
title = NOVALUE,
description = NOVALUE,
thumbnail_encoding = NOVALUE,
language = NOVALUE,
license = NOVALUE,
license_description = NOVALUE,
copyright_holder = NOVALUE,
author = NOVALUE,
role_visibility = NOVALUE,
aggregator = NOVALUE,
provider = NOVALUE,
extra_fields = NOVALUE,
prerequisite = NOVALUE,
complete = NOVALUE,
accessibility_labels = NOVALUE,
grade_levels = NOVALUE,
learner_needs = NOVALUE,
learning_activities = NOVALUE,
categories = NOVALUE,
suggested_duration = NOVALUE,
```

   b. Here's a sample file structure of the h5p.json file that would be used to extract the metadata from the .h5p files. This file is taken from the official .h5p documentation.

```
{
  "title": "Fill in the blanks",
  "language": "en",
  "mainLibrary": "H5P.Blanks",
  "embedTypes": [
    "div"
  ],
  "authors": [
    {
      "name": "fnoks",
```

```
      "role": "Author"
    }
  ],
  "source": "https://example.org",
  "license": "CC BY-SA",
  "licenseVersion": "4.0",
  "licenseExtras": "Extra license information is found here",
  "yearFrom": "1950",
  "yearTo": "2050",
  "changes": [
    {
      "date": "02-07-19 11:27:00",
      "author": "fnoks",
      "log": "Initial version"
    }
  ],
  "authorComments": "Author comments",
  "preloadedDependencies": [
    {
      "machineName": "H5P.Blanks",
      "majorVersion": "1",
      "minorVersion": "0"
    }
  ]
}
```


c. From the above contentNodeData, this data seems to be aligned with the H5P package definition. And these are the metadata that needs to be extracted from the .h5p files:

   i. Title
   ii. Description
   iii. Language
   iv. License (optional in h5p.json)
   v. CopyWrite Holder (authors in h5p.json it is optional as well)

Now we know what we need to extract, the only step that's left to do is to extract the metadata from the uploaded files.

3. Extracting metadata from .h5p files and sending the post request:

To extract the metadata we would use the package called JSZip. This package uses various compression and decompression methods to open various files, and it can be used to open .h5p files and help us extract the metadata that we need.

After we can open the file by decompressing it using the package, we will need to locate the h5p.json file as this file contains all the metadata of the .h5p file. The general structure of the .h5p file is:

```
.
├── content
│   ├── images
│   │   └── explain.png
│   └── content.json
├── FontAwesome
│   ├── ...
│   └── library.json
├── H5P.Blanks
│   ├── ...
│   ├── library.json
│   └── semantics.json
└── h5p.json
```

We can see that the h5p.json file is located at the root level of the archive file. Now we know where the file is located. Here's the sample code to act.

```
var zip = new JSZip();
zip.loadAsync(h5pFile).then(function(zip) {
 // H5P file loaded
});
var metadataFile = zip.file("h5p.json");
if (metadataFile) {
        // Metadata file found
}
```

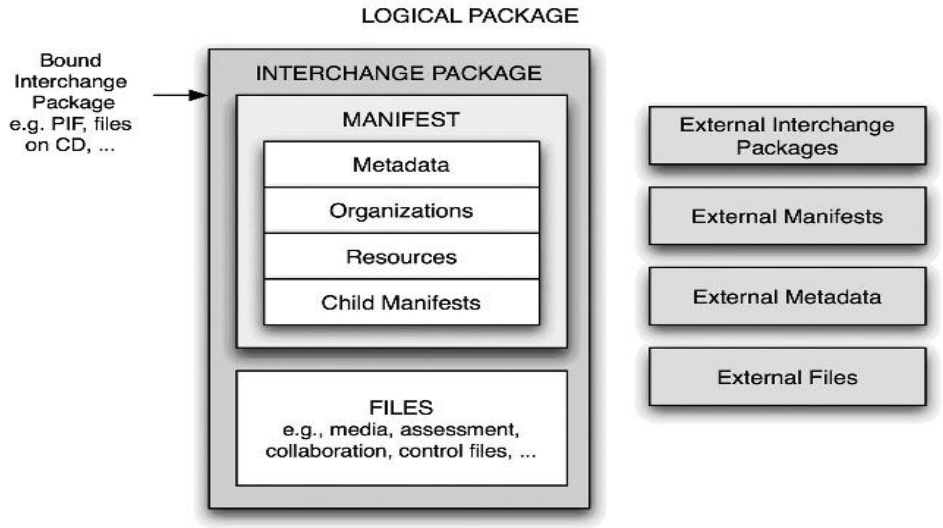
After locating the file we will need to parse the file so that we can read the content of the file and extract the data. JSZip has inbuilt functionalities that we can use to parse the file.

Now we can extract the data from the file, but before we send the extracted data to the backend we will need to look at the h5p encoded license just to map it with the way Studio/Kolibri encode their license.

Once this is done, we can add the metadata to the ContentNode and send it along the Post Request.

Implement the metadata extraction for IMS content package files:

The implementation for extraction of metadata from the IMS content package is a bit different from the extraction of H5P files. That's because the file structure is different and the file extension is .xml files. The file structure is as follows.




```
.
├── ims_xml
├── web_resources
├── assessment
├── resources
├── metadata
└── imsmanifest.xml
```

We would need to read both imsmanifest.xml and metadata files to get the required information. To read the Content Package, we will use JSZip to compress and decompress which is similar to .h5p Content Package. To extract the metadata from imsmanifest.xml and metadata files. We will use an XML parser called fast-xml-parser using this we can extract the metadata from the XML files.

Sample code from the docs:

```
const { XMLParser, XMLBuilder, XMLValidator} = require("fast-xml-parser");

const parser = new XMLParser();
let jObj = parser.parse(XMLdata);

const builder = new XMLBuilder();
const xmlContent = builder.build(jObj);
```

Generate extra file resources for mapping **Topics Nodes** and Resource Nodes:

For these SCORM content packages, we also need to generate a course tree that includes mapping the TopicNodes along with the ResourceNode. Then we will need to write it into their respective file format either JSON or XML and then embed them into the file.

To implement this first, let's understand what this course tree is and how these topic and resource nodes are structured. Here's an example of the Course Tree:

```
Course Tree
- Course Title
- Course Description
- Course Start Date
- Course End Date
- Topic Nodes
 - Topic Node 1
  - Topic Node 1 Title
  - Topic Node 1 Description
  - Resource Nodes
   - Resource Node 1
    - Resource Node 1 Title
    - Resource Node 1 Description
    - Resource Node 1 Type (e.g. video, document, quiz)
    - Resource Node 1 Content URL
   - Resource Node 2
    - Resource Node 2 Title
    - Resource Node 2 Description
    - Resource Node 2 Type (e.g. video, document, quiz)
    - Resource Node 2 Content URL
  - Child Topic Nodes (optional)
   - Subtopic Node 1
    - Subtopic Node 1 Title
    - Subtopic Node 1 Description
    - Resource Nodes (optional)
 - Topic Node 2
  - Topic Node 2 Title
  - Topic Node 2 Description
  - Resource Nodes
   - Resource Node 4
    - Resource Node 4 Title
```

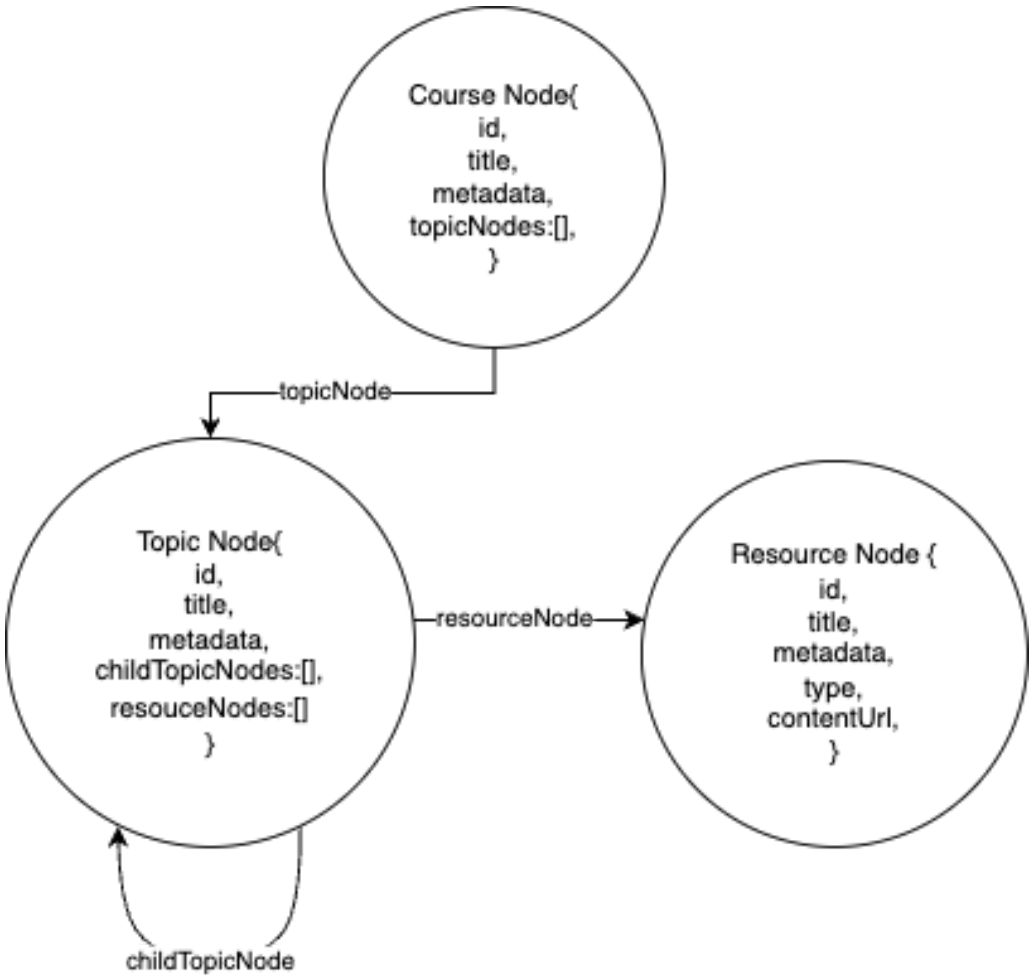
```
      - Resource Node 4 Description
      - Resource Node 4 Type (e.g. video, document, quiz)
      - Resource Node 4 Content URL
    - Resource Node 5
      - Resource Node 5 Title
      - Resource Node 5 Description
      - Resource Node 5 Type (e.g. video, document, quiz)
      - Resource Node 5 Content URL
  - Child Topic Nodes (optional)
    - Subtopic Node 2
      - Subtopic Node 2 Title
      - Subtopic Node 2 Description
      - Resource Nodes (optional)
        - Resource Node 6
          - Resource Node 6 Title
          - Resource Node 6 Description
          - Resource Node 6 Type (e.g. video, document, quiz)
          - Resource Node 6 Content URL
```

From the above tree structure, we can define a user-defined tree in this format that would allow us to store any kind of data in this format easily and also help us write it back to the JSON or XML files. The data structure that we can use is:

```
class ResourceNode {
 constructor({
  id,
  title,
  metadata,
  type,
  contentUrl,
 })
}

class TopicNode {
 constructor({
  id,
  title,
  metadata,
  resoureNode:[],
  childTopicNodes:[]
 })
}
class CourseNode {
 constructor({
  id,
  title,
  metadata,
  topicNodes:[]
 })
}
```

To help visualize the data structure here's a sample state map to understand the structure:



From the above flow diagram, we can generate the tree and write it in the requisite file. And embed it into the files using the JSZip, sample code:

```
//stringify the JSON_Object that we had created.
// save it in the required file name with the extension.
zip.file('h5p/fileName.json', JSON.stringify(JSON_Object));
```

In this way, we can embed the course tree in the SCORM file. Usage, this will be used to render out the folder tree and also help us navigate the files using the resource URL in ResourceNodes.

Implement the algorithm to check for any external resource links embedded in the files.

To implement this functionality first, we will need to read every file like HTML, CSS, JavaScript, and JSON files. To do that we will use the new FileReader().

```
var reader = new FileReader();
reader.onload = function(event) {
 var htmlContent = event.target.result;
 var urlRegex = /((http|https):\/\/[^\s]+)/g;
 var urlMatches = htmlContent.match(urlRegex);
 if (urlMatches) {
   console.log("URL links found:");
   console.log(urlMatches);
 } else {
   console.log("No URL links found.");
 }
};
```

Then once the file is open it will convert it into strings and use regex to check for patterns like

```
https?:\/\/[a-zA-Z0-9]+(?:-[a-zA-Z0-9]+)*\.[a-zA-Z]{2,}(?:\/\S*)?
ssh:\/\/[a-zA-Z0-9]+(?:-[a-zA-Z0-9]+)*\.[a-zA-Z]{2,}(?::\d+)?(?:\/\S*)?
curl\s+(-X\s+\S+\s+)?'?\S+'?(?:\s+-[a-zA-Z]\s+(?:\S+|"[^"]+"))*
```

And many more. Hence will have to identify various ways in which one can embed external resources in these files. Once all the various types of external links and resources are identified. We will have to categorize them according to their file type in a JSON object. For example all the external links for HTML a JSON objects. So that we can use these particular regular expressions to identify these resources.

Once we find that there is any kind of external resource link embedded in the file we can warn the uploader about the presence of external files and hence ask for his confirmation if we want to continue or not.

This is reading just a single file, but usually, the Archive file is a folder that contains multiple files and folders now we will need to write functionality to travel the folder structure.

To do this we can use a recursive function where we start with the root directory and create two arrays one for files and one for folders. Then call the same function again for each folder in the array. If we are left with 0 folders we can end the function. Here's a sample code to implement this.

```
function getFilesAndFolders(folderPath, callback) {
  window.resolveLocalFileSystemURL(folderPath, function(directoryEntry) {
    var reader = directoryEntry.createReader();
    reader.readEntries(function(entries) {
      var files = [];
      var folders = [];
      for (var i = 0; i < entries.length; i++) {
        var entry = entries[i];
        if (entry.isFile) {
          files.push(entry.name);
        } else if (entry.isDirectory) {
          getFilesAndFolders(entry.fullPath, function(result) {
            folders.push({ name: entry.name, files: result.files, folders:
result.folders });
          });
        }
      }
      callback({ files: files, folders: folders });
    });
  });
}
```

This function returns the list of all the files and folders present in a directory. Using the files array we use the files list to filter out the required files and then read the content to check for the external resource present in the files.

List the resource links and URLs and download the resources for offline usage.

Now we have all the external links that we would want to download the resource from, so now we need to first download the resources and embed them into the folder so that they can be used in offline mode.

To download the data we can use different JS methods like

1. Fetch()
2. Anchor tag
3. XMLHttpRequest

Based on the types of files that Kolibri requires can we put them to use?

To Embed these files in the folders we can use the JSZip methods that we had used earlier for IMS Content Package can be similarly used.

Ricecooker Library has the implementation of downloading externally linked resources which could be used to implement the functionality. Let us take a closer look at the function that we can use for reference:

1. The main function is this https://github.com/learningequality/ricecooker/blob/6db33e577f25be8ae

a3f980080884860c20822e1/ricecooker/utils/downloader.py#L263. This function takes many arguments, to download the files like **.css, and .js.**

2. The major arguments include
    a. doc: The HTML page source as a string or BeautifulSoup instance.
    b. destination: The folder to download the static assets too!
    c. base_url: The base URL which assets will be downloaded from.
    d. request_fn: The function to be called to make requests, passed to
    e. ricecooker.utils.html.download_file(). Pass in a custom one for custom caching logic.
    f. url_blacklist: A list of keywords of files to not include in downloading. Will do substring matching, so e.g. 'acorn.js' will match '/some/path/to/acorn.js'.
    g. js_middleware: If specified, JS content will be passed into this callback which is expected to return JS content with any modifications.
    h. css_middleware: If specified, CSS content will be passed into this callback which is expected to return CSS content with any modifications.
3. This function also consists of many files specific functions which are used to download the files like:
    a. download_srcset: This function helps us to download the srcset (set of images we will allow the browser to choose between, and what size each image is).
    b. download_assets: Helper function to download all assets for a given CSS selector.
    c. js_content_middleware: JS content will be passed into this callback which is expected to return JS content with any modifications.
    d. css_node_filter: Filter btw rel and href CSS files.
    e. css_content_middleware: CSS content will be passed into this callback which is expected to return CSS content with any modifications.
    f. This function also consists of link scrapping that could be used as a reference.

Timeline:

Period	Task
After proposal submission [April 4 - May 4]	- Try to understand the Kolibri Studio Codebase. - Fix bugs and issues in the repository. - While learning about the code try to understand the coding style of Kolibri.
Week 1 and 2 [May 4 - May 18]	- Discuss with mentors and review the proposal. - Try to figure out various problems that we could face during the implementation of data extraction. - BrainStorm edge cases that can occur.
Week 3 [May 18 -May 28]	- Finalize the proposal requirements - Discuss the theoretical and technical aspects of every subproblem in the project. - Figure out different regexes and categorize them for various file types.
First Coding Period [May 29 - July 14]	
Week 4 [May 28 - June 2]	- Start implementing Allow Studio to import .h5p files. - Decide Metadata That needs to be extracted.

Week 5 and Week 6 [June 2 - June 16]	- Extracting metadata from .h5p files and sending the post request. - Extracting metadata from IMS content package files and sending the post request.
Week 7 [June 16 - June 23]	- Bug fixes and code refactoring. - Improvements that can be made. - Testing with various .h5p and IMS Content Files.
Week 8 [June 23 - June 30]	- Generate extra file resources for mapping Topics Nodes and Resource Nodes.
Week 9 [June 30 - July 6]	- Bug fixes and code refactoring. - Improvements that can be made. - Testing with various Archive Files.
First Evaluation date [July 10]	

Second Coding Period [July 14 - September 4]	
Week 10 and 11 [July 14 - July 28]	- Implement the algorithm to check for any external resource links embedded in the files
Week 12 [July 28 - August 4]	- Bug fixes and code refactoring. - Improvements that can be made. - Testing with all different types of resource files that Kolibri stores.
Week 13 [August 4 - August 11]	- List the resource links and URLs and download the resources for offline usage.
Week 14 [August 11 - August 18]	- Bug fixes and code refactoring. - Improvements that can be made. - Testing with various Archive Files.

Week 15 [August 18 - September 28]	- Proper Documentation and code submission.
Coding Period Ends	

Biographical Information:

About Me:

Hello, my name is Manav Aggarwal and I am currently a third-year student pursuing a B.Tech in Computer Science from the National Institute of Technology Tiruchirappalli in India. I have a keen interest in software development and enjoy traveling in my free time.

My passion for software development began in my first year at university when I joined the Delta Force coding club. I started with basic languages such as HTML, CSS, and **JavaScript** and built my first game called Match the Grid. I found the process of development fascinating and was eager to learn more.

Since then, I have expanded my knowledge by learning various languages and tools such as **TypeScript**, NodeJs, **ReactJs**, ElectronJs, AWS, Python, C++, Shell Scripting, AWS Lambda, AWS dynamoDB, MongoDB, and MySQL. I am excited to continue learning and growing in the field of software development.

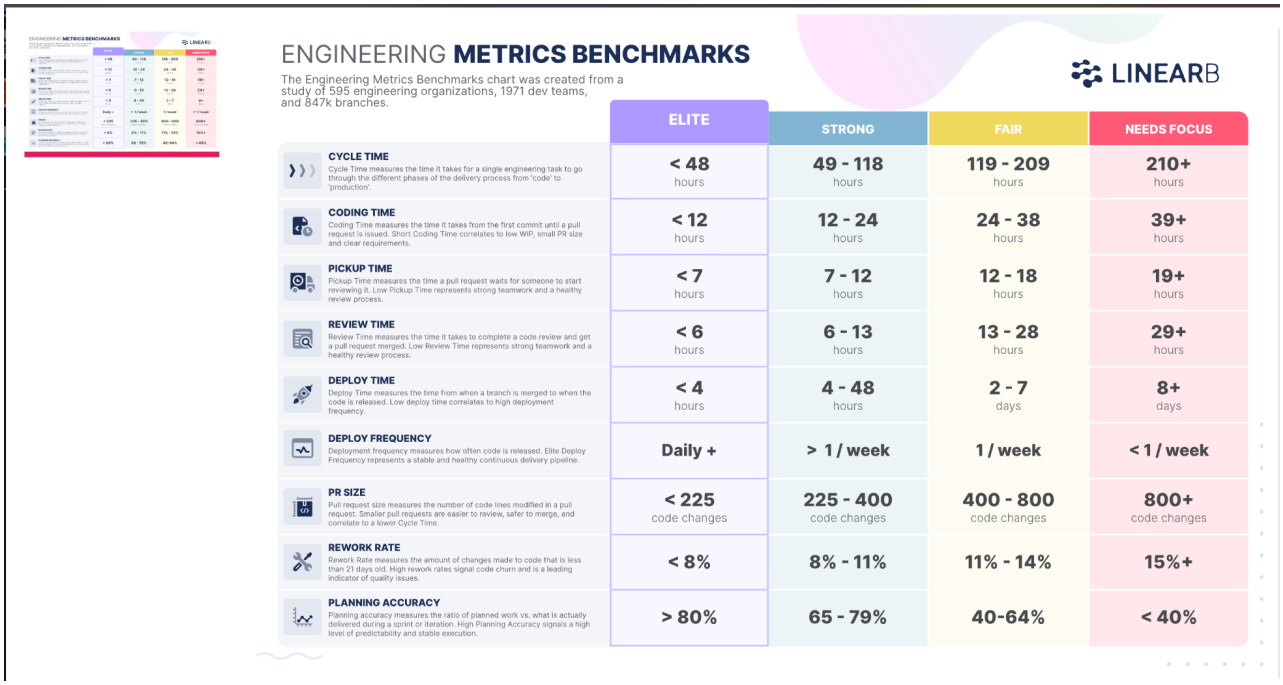
Workduck (8 months internship):

Since then, I have expanded my knowledge by learning about various new technologies. Recently, I had the opportunity to work with a product management startup called workduck.io. During my time there, I gained valuable experience in several areas, such as

reading and understanding large codebases and adding new functionalities to their web app and chrome extension.

One of my major accomplishments was building a chat application from scratch using PubNub. This involved several tasks, including figuring out the backend routes, implementing them, providing the ER model for the database, and connecting the back end with the front end through secure and authenticated methods.

In addition, I also worked on understanding and calculating engineering metrics for the organization using JavaScript scripts (similar to the current project).



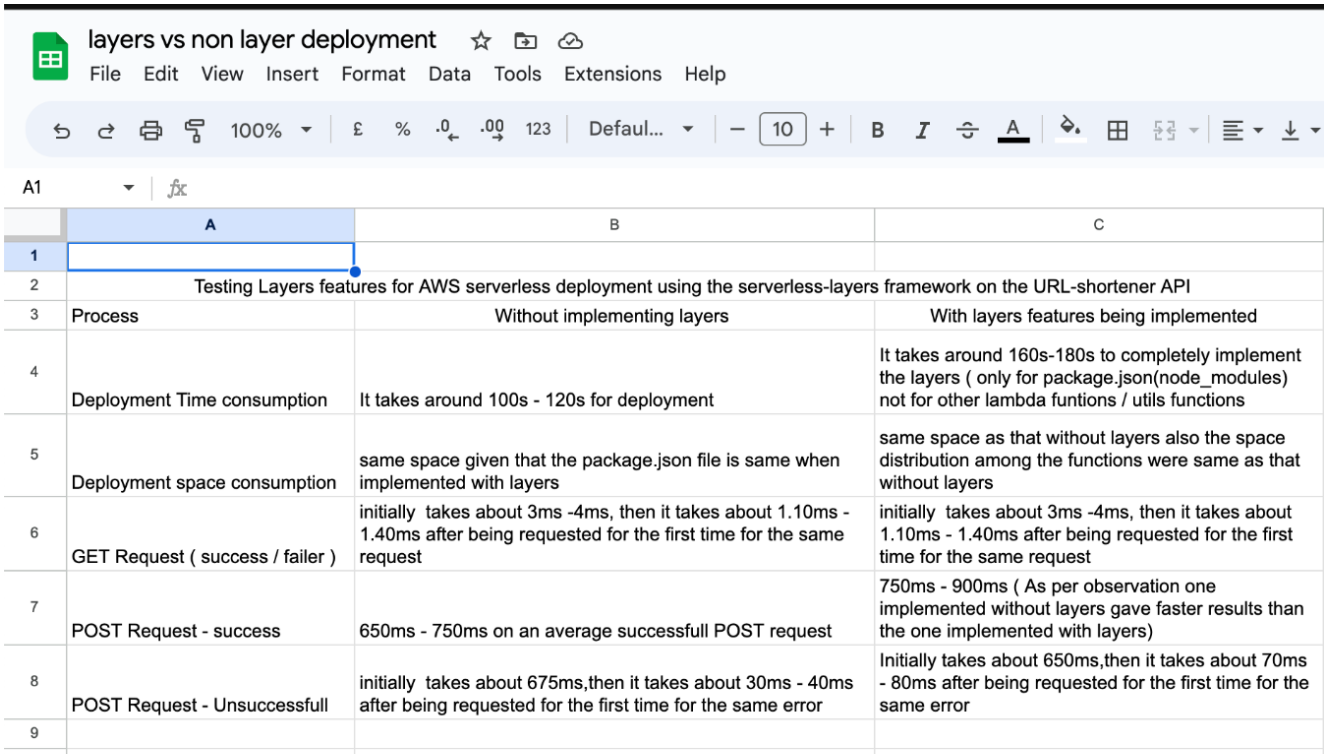
ENGINEERING **METRICS BENCHMARKS**

The Engineering Metrics Benchmarks chart was created from a study of 595 engineering organizations, 1971 dev teams, and 847k branches.

LINEARB

	ELITE	STRONG	FAIR	NEEDS FOCUS
CYCLE TIME Cycle Time measures the time it takes for a single engineering task to go through the different phases of the delivery process from 'code' to 'production'.	< 48 hours	49 - 118 hours	119 - 209 hours	210+ hours
CODING TIME Coding Time measures the time it takes from the first commit until a pull request is issued. Short Coding Time correlates to low WIP, small PR size and clear requirements.	< 12 hours	12 - 24 hours	24 - 38 hours	39+ hours
PICKUP TIME Pickup Time measures the time a pull request waits for someone to start reviewing it. Low Pickup Time represents strong teamwork and a healthy review process.	< 7 hours	7 - 12 hours	12 - 18 hours	19+ hours
REVIEW TIME Review Time measures the time it takes to complete a code review and get a pull request merged. Low Review Time represents strong teamwork and a healthy review process.	< 6 hours	6 - 13 hours	13 - 28 hours	29+ hours
DEPLOY TIME Deploy Time measures the time from when a branch is merged to when the code is released. Low deploy time correlates to high deployment frequency.	< 4 hours	4 - 48 hours	2 - 7 days	8+ days
DEPLOY FREQUENCY Deployment frequency measures how often code is released. Elite Deploy Frequency represents a stable and healthy continuous delivery pipeline.	Daily +	> 1 / week	1 / week	< 1 / week
PR SIZE Pull request size measures the number of code lines modified in a pull request. Smaller pull requests are easier to review, safer to merge, and correlate to a lower Cycle Time.	< 225 code changes	225 - 400 code changes	400 - 800 code changes	800+ code changes
REWORK RATE Rework Rate measures the amount of changes made to code that is less than 21 days old. High rework rates signal code churn and is a leading indicator of quality issues.	< 8%	8% - 11%	11% - 14%	15%+
PLANNING ACCURACY Planning accuracy measures the ratio of planned work vs. what is actually delivered during a sprint or iteration. High Planning Accuracy signals a high level of predictability and stable execution.	> 80%	65 - 79%	40-64%	< 40%

I was also fortunate to gain exposure to various AWS cloud services, such as AWS Lambdas, DynamoDB, AWS S3, and AWS Layers, during my time at workduck.io. This experience helped me further expand my knowledge and skill set in the field of cloud computing.



	A	B	C
1			
2	Testing Layers features for AWS serverless deployment using the serverless-layers framework on the URL-shortener API		
3	Process	Without implementing layers	With layers features being implemented
4	Deployment Time consumption	It takes around 100s - 120s for deployment	It takes around 160s-180s to completely implement the layers (only for package.json(node_modules) not for other lambda funtions / utils functions
5	Deployment space consumption	same space given that the package.json file is same when implemented with layers	same space as that without layers also the space distribution among the functions were same as that without layers
6	GET Request (success / failer)	initially takes about 3ms -4ms, then it takes about 1.10ms - 1.40ms after being requested for the first time for the same request	initially takes about 3ms -4ms, then it takes about 1.10ms - 1.40ms after being requested for the first time for the same request
7	POST Request - success	650ms - 750ms on an average successfull POST request	750ms - 900ms (As per observation one implemented without layers gave faster results than the one implemented with layers)
8	POST Request - Unsuccessfull	initially takes about 675ms,then it takes about 30ms - 40ms after being requested for the first time for the same error	Initially takes about 650ms,then it takes about 70ms - 80ms after being requested for the first time for the same error
9			

I had the opportunity to compare the performance of AWS Lambdas with and without AWS Layers. To conduct this analysis, I went through various documentation and tutorials to gain a better understanding of these technologies. This experience not only allowed me to improve my analytical and problem-solving skills but also helped me gain a deeper understanding of the nuances of cloud computing.

Delta Force (Coding Club):

In addition to my experience at workduck.io, I was also a member of Delta Force, the official coding club at my college. As a member of the club, I participated in various activities, such as building and maintaining the college's main website and updating its content while improving its user interface.

I also worked on the college's official scholarship portal, where I contributed by designing and developing new features to improve its functionality. Moreover, I was involved in organizing our college's Tech Fest (Pragyan) and was responsible for designing and developing one of the main gaming events called "Attack on Robots." This event had around 600-700 active participants and was only playable during the pre-fest events.

I also had the opportunity to work on several other Delta projects, such as Festember's main website and other projects. These experiences helped me enhance my technical and collaborative skills while working on different projects in a team environment.

Vortex (Computer Science Dept. Symposium):

In addition to my other experiences, I was also a member of the Computer Support Group for our department's symposium. In this role, I was responsible for managing the front end of the website. This experience allowed me to hone my web development skills while working collaboratively with others toward a common goal. I am grateful for this opportunity and look forward to applying these skills in future projects.

My Contribution to Open Source:

Replay.io:

1. Proposed a feature implementation that is accepted and has the potential of solving 2-year long-standing issues.
2. First Pr for the above-mentioned solution.

Nertralino.js:

1. Working on adding net API to the framework.

Why did you choose to work with Learning Equality?

1. Collaboration: Learning Equality values collaboration and works with a wide range of partners, including educators, technologists, and other organizations. Contributing to the organization can provide opportunities to work with a diverse group of individuals and organizations to advance education.
2. Equity: Learning Equality's mission is to provide access to education for all, regardless of geographic location, socio-economic status, or other barriers. Contributing to the organization can help promote equity in education.
3. Personal growth: Contributing to Learning Equality can also provide opportunities for personal growth and development, particularly in the areas of education, technology, and social impact.

What are my plans after GSOC?

I'm always curious about backend development and adding functionalities, and I would love to further help the organization in any way possible. I want to get involved in the plans of the organization and help implement new features. When I studied the work of my mentors I was deeply inspired by their work and would want to further my knowledge and experience under your able guidance and expertise.

How much time each day would I devote to GSoC?

The Project demands 12.5 hrs per week i.e., 2 hrs(approx) per day. But I will be comfortably devoting 4(minimum) hrs per day. In case of extraordinary circumstances, if the situation demands I can even put in 10 to 12 hours a day to complete the task.

What other commitments do you have for this summer?

I have no other commitments other than GSoC 2023 on my list.