



**KRISHNA MADHWANI**


**Project Title: Public Artifact Data and Visualisation**

**Subparts: Experiment Log and Record and Visualise experiment results**

**Project Size :- Large(350 hours)**

**Mentor: Anjo Vahldiek-Oberwagner**

# Krishna Madhwani

Email: [krishna.madhwani.cse21@itbhu.ac.in](mailto:krishna.madhwani.cse21@itbhu.ac.in)  
Github profile: <https://github.com/Roaster05>  
Slack: Krishna Madhwani  
Mobile: +917974032798  
TimeZone: +5:30 GMT  
LinkedIn: <https://www.linkedin.com/in/krishna-madhwani-978798223/>  
Resume/CV:  Krishna\_Resume.pdf

## About Me:

I am a sophomore pursuing B.Tech in **Computer Science and Engineering** from **Indian Institute of Technology Varanasi**. I love coding and creating projects and am curious to learn and contribute to the Open Source projects. Most of my projects in the past have been in the field of Web Development and UI designing. I have contributed to a few open-source organizations like **Replay.io**, previously selected as a GSOC organization.

## Proposal Abstract:

The Experiment Log project aims to provide tools to **log, capture, and visualize** experiments to improve the reproducibility and accessibility of research results. The project will develop a client and server-side agency to start, stop, and timestamp experiments and document **each experiment iteration**.

The project will also create a database to store and visualize the log of experiments, making it easier for researchers to access and review their experimental data. The tool will enable researchers to track the progress of their experiments and easily compare and contrast results across different iterations.

## Proposal:

### Client-Side Interface

Some of the client-side features I will focus on implementing for this Experiment Log project are:

### Start/Stop Experiment:

We will need to design a user interface allowing users to initiate and stop an experiment. This could be done using a **button or a command line interface (CLI)**, depending on the platform and the user's preference. When the user initiates an experiment, we must record the start time and associated metadata, such as the experiment name, description, and parameters. We can store this information in a database or a file related to the experiment.

```
axios.post('Url_here', {
  {data}
})
.then((response) => {

// On successful request

})
.catch((error) => {

// if Error is received

});
```

### Timestamping:

The tool should timestamp each experiment iteration, recording when each iteration **started and ended**, allowing the user to track the progress of the experiment over time and identify any issues or anomalies that arise.

```
timestamp : {

  type : Date,
  required : true,

},
```

### Iteration documentation:

During the experiment, for all the iterations related to it, **we'll need to monitor its progress and record any relevant information about each iteration**. That could include metrics, logs, and other data the user wants to capture, which will be saved and used for further analysis and visualization of the experiment.

```
const IterationSchema = new Schema({
  timestamp : {
    type : Date,
    required : true
  },
  data : {
    input : {
      parameter1 : {
        type : Number,
        required : true
      },
    },
    output : {
      metric1 : {
        type : Number,
        required : true
      },
    }
  }
});
```

## Visualization:

The tool should provide a user interface for visualizing the log of experiments, allowing the **user to filter and search the register by various criteria** (e.g., experiment name, date range, iteration number). The visualization should also summarize each experiment iteration's key results and observations.

```
Experiments.find()
  .then((experiment) => {

  })
  .catch((error) => {
```

```

    });
  })
  .catch((error) => {

  });
});

```

```

Iterations.find()
  .then((iteration) => {

  })
  .catch((error) => {

  });
})
.catch((error) => {

});

```

We will use the [Chart.js](#) library to improve our analysis **using various metric comparison tools** to achieve more visual analysis of an experiment across all the iterations.

For example, suppose you want to create a line chart to visualize the results of multiple iterations of an experiment. In that case, you could make an array of objects that contains the data for each iteration. Each object could have properties for the x and y values of the data point and any other relevant information.

```

const options = {
  title: {
    display: true,
    text: 'Comparison of Experiments'
  },
  scales: {
    yAxes: [{
      ticks: {

```

```

        beginAtZero: true
      }
    }]
  }
};

const myChart = new Chart(document.getElementById('myChart'), {
  type: 'line',
  data: data,
  options: options
});

```

## Collaboration:

The tool must **allow multiple users to collaborate** by providing access to the experiment log. The user should be able to set permissions and control who can view and edit the record.

```

const memberSchema = new mongoose.Schema({
  name: { type: String, required: true },
  experiments: [{
    experimentId: { type: mongoose.Schema.Types.ObjectId, ref:
'Experiment', required: true },
    permissions: {
      read: { type: Boolean, required: true, default: false },
      write: { type: Boolean, required: true, default: false },
      delete: { type: Boolean, required: true, default: false }
    }
  }]
});

```

## Notifications:

The tool should **notify the user** when an experiment has been started or stopped. It should also inform the user when an experiment iteration has been added to the log. We will use the **Firestore Cloud messaging feature** for this feature, which allows us to notify

our users whenever an experiment is initialized or completed or an iteration related to the experiment is logged in. The feature can be implemented as follows:

```
var email = "user_email";
var fcmToken = "fcm_TOKEN";

var headers = new Headers({
  "Authorization": "API KEY"
  "Content-Type": "application/json"
});

var body = {
  "notification": {
    "title": "iteration logged/completed",
    "body": "Body of your notification after iteration/experiment"
  },
  "to": fcmToken
};

fetch("https://fcm.googleapis.com/fcm/send", {
  method: "POST",
  headers: headers,
  body: JSON.stringify(body)
}).then(function(response) {

  // On successful request
}).catch(function(error) {
  // if an error has occurred while logging
});
```

### Export/Import:

The tool should allow the user to **export the experiment log** in different formats such as CSV or JSON. This feature helps back up data or for sharing data with other researchers.

To achieve this functionality, we will:

1. Write a JavaScript function to gather the data and convert it into the desired format.  
Some examples for conversions are:
  - CSV: Use the `CSV` package to convert the data into a CSV string.
  - JSON: Use the `JSON.stringify()` method to convert the data into a JSON string.
  - PDF: Use a third-party library like `jsPDF` to generate a PDF file from the data.
  - Excel: Use a third-party library like `xlsx` to generate an Excel file from the data.
4. Once the data is converted into the desired format, we will use the Blob constructor to create a blob object for the file.
5. We will then create a download link for the file by creating an `<a>` tag with the href attribute set to a URL created from the blob object using the `URL.createObjectURL()` method.
6. Set the download attribute of the `<a>` tag to the desired filename for the downloaded file.
7. Trigger a click event on the download link to initiate the download.

## Server-Side features

### User authentication:

Users should be able to **create an account and log in to the system** to access their experiment logs. To achieve this, we will use `Auth0` authentication in our application as it can easily integrate authentication into your application, allowing your users to securely sign in with various identity providers such as **Google, Facebook, or Twitter** or with their email and password; also it provides Strong security measures such as encryption and multi-factor authentication, customizable login and registration pages and comprehensive analytics and reporting. An example of using Auth0 for login purposes is:

```
auth0Client.login({
  username: 'user_email',
  password: 'password',
  realm: ''
}, (err, authResult) => {
  if (err) {

  } else {
```



```
}  
});
```

For the auth0 authentication, we will need to initialize our application on the Auth0 portal which will provide us a **Client and a Client secret key** that will be used in headers while using different methods.

## Experiment log retrieval and sharing:

Users should be able to retrieve their experiment logs from the server-side tool. The logs should be accessible through a web-based interface.

**We can develop an API that will respond to queries related to data retrieval and sharing** and assign endpoints depending on the desired query request. We can use different API frameworks like the inbuilt Nextjs server-side scripting or frameworks like Express.js. An example of an endpoint would look like this.

```
axios.get('/searchExperiment', async (req, res) => {  
  const results = await Experiments.find({ id: req.query.experiment_id });  
  res.json(results);  
});
```

Similarly, we can find other endpoints which will respond to these requests related to data retrieval and sharing and will respond according to the permission the user holds, **Similarly, we will implement API endpoints to help upload the data into the database using the POST method and its permission.**

## Experiment log backup:

The server-side tool should **periodically back up the experiment logs** to prevent data loss.

To implement this, we will use the **mongodump** command line tool provided by the **MongoDB** database; it creates a binary export of the contents of a **MongoDB** instance. It can back up an entire database, a collection, or a document.

It provides several options that can be used to customize its behavior. Some standard options include:

- db: The name of the database to be backed up.
- out-: The path to the directory where the backup should be stored.
- collection: The name of a specific collection to be backed up.
- gzip: Compresses the backup using gzip.
- archive: Output the backup to stdout.

## Integration with other tools:

**Integrating the experiment logging tool with other tools commonly used in the scientific community**, such as Jupyter notebooks, makes it easy to log experiments and share data with other researchers.

We can use tools like [Jupyter](#), Docker, Git, and others that can be included in our application using their Public API for integration like

### **Jupyter**

- /api/kernels: GET request to list all available kernels.
- /api/kernels/{kernel\_id}/interrupt: POST request to interrupt a running kernel.
- /api/contents/{path}: GET request to get information about a file or directory.
- /api/contents/{path}: PUT request to save changes to a file or create a new file.
- /api/sessions: POST request to start a new session.

### **Docker**

- /containers/json: GET request to list all containers.
- /containers/create: POST request to create a new container.
- /containers/{id}/start: POST request to start a container.
- /containers/{id}/stop: POST request to stop a container.
- /images/create: POST request to create a new image.

## Accessing the application using a Command Line Interface:

We will also **create a CLI interface that will be used for Experiment data logging** and will allow our users or clients to access the application directly from a command line rather than going into the web application. We will use `Node.js` and the popular `node.js` library `Commander.js` to achieve this.

`Commander.js` is a command-line interface (CLI) tool allowing developers to create user-friendly interfaces using `Node.js`. With `Commander.js`, developers can quickly parse command-line arguments and options and **easily create powerful command-line tools**.

Now with the help of `Commander.js`, we will initialize a CLI that will be used to access our application through commands then,

Now using `Commander.js`, we will define the various commands and their features that will be associated with these commands, for e.g:

```
-a, -add <description>,"adding an experiment"  
-i, -iteration<iteration_id>,"adding an iteration of an experiment"
```

Similarly many other options can be created that will be used in experiment logging and retrieval.

```
const program = new Command();  
program  
  .option('-n, --name <name>', 'Name of the experiment')  
  .option('-d, --description <description>', 'Description of the  
experiment')  
  .option('-i, --iterations <iterations>', 'Number of iterations',  
parseInt)  
  .option('-s, --start <start>', 'Starting value', parseFloat)  
  .option('-e, --end <end>', 'Ending value', parseFloat)  
  .option('-t, --team <step>', 'Team size', parseFloat)  
  .parse(process.argv);
```

Now we have to **define actions related to these options**; for example, in order to add an experiment and its iteration data, we will have to **connect to our database and will then add these instances in our database to achieve this**, we have to create an action related to this commands that will help us in achieving our goal, the activity related to these options might look like:-

```
axios.post('Url_here', {
  {data}
})
.then((response) => {

// On successful request

})
.catch((error) => {

// if Error is received

});
```

Now afterward we can parse these command line arguments using `program.parse(process.argv);`

## **Proposal idea for integration of Project “Record and Visualize experiment results” in this project.**

Proposal idea for integrating the project “Record and Visualize experiment results” with this project, the previous project can be extended to include this tool as a part of the same application, which will give better control to the user in accessing and logging the experiment data through different methods.

**This project aims to develop a plug-in to enable users to convert various data formats into a standard JSON object for further visualization.** The plug-in will incorporate features such as **data format detection, transformation, validation, and loading** to ensure efficient and effective data processing.

The plug-in will be designed to **detect various data formats**, including CSV, Excel, XML, and JSON. Once the data format is identified, the plug-in will **transform the data** into a standard JSON format, which various visualization tools can easily consume.

The tool will also include **data validation features** to ensure the converted data is accurate and consistent. This will involve checking for missing values, data types, and other errors that may affect the output data quality.

Finally, the plug-in will be designed to load the **processed data into the visualization tool of choice**, making it easy for users to explore and analyze the data. By simplifying the data conversion process, this plug-in will help users save time and reduce errors in data visualization.

The project can be implemented by incorporating the following features in our Web Application.

### Data format detection

So to parse the data present in standard data formats into a unique Javascript Object, it is necessary to **detect the file format** to use respective tools for transforming them into the respective Javascript object.

We can use the file-type library offered by the [Node.js](#) environment.

[file-type](#) detects the file type by reading the file data from the file path using the built-in fs module. [file-type](#) returns an **object with the detected file extension and MIME type**. If the file type is detected, the detected file extension is logged, and the file can be handled accordingly. If the file type is not seen, an appropriate action can be taken; [file-type](#) uses the file's header bytes to determine the file type, so it is generally more reliable than relying on file extensions alone.

```
const fs = require('fs');
const fileType = require('file-type');

const detectFileType = async (filePath) => {
  const fileData = await fs.promises.readFile(filePath);
```

```
const detectedType = fileType(fileData);
if (detectedType) {

    // handle the file based on the detected type
} else {

    // handle unsupported file types
}
}
```

## Data transformation

We have to use different **transformation functions for each file type** to transform this data into an everyday Javascript object. For this, various libraries must be imported to deal with specific data formats.

The transformation feature for some standard formats can be applied as follows:

### JSON:

**Node.js** has built-in support for parsing and serializing JSON data. The JSON object can be used to parse JSON data into a JavaScript object, or to stringify a JavaScript object into a JSON string.

```
const jsonString = '{"data1" : , "data2" : }';

const obj = JSON.parse(jsonString);
```

### CSV:

Several libraries are available in Node.js to parse CSV data. One of the most popular ones is csv-parser.

```
const csv = require('csv-parser');
```

```
const fs = require('fs');

const results = [];

fs.createReadStream('data.csv')
  .pipe(csv())
  .on('data', (data) => results.push(data))
  .on('end', () => {
  });
```

### XML:

There are several libraries available in Node.js to parse XML data. One of the most popular ones is `xml2js`.

```
const xml2js = require('xml2js');
const xml = `content`;

xml2js.parseString(xml, (err, result) => {

});
```

### YAML:

There are several libraries available in Node.js to parse YAML data. One of the most popular is `js-yaml`.

```
const yaml = require('js-yaml');
const fs = require('fs');

const doc = yaml.load(fs.readFileSync('data.yaml', 'utf8'));
```

These are just a few examples of the many data parsing libraries available in Node.js. The specific library we will choose will depend on our use case and the format of the data we need to parse.

## Data Validation:

To check whether the data is **accurate, consistent, and complete** can be performed by performing checks such as data type validation, data length validation, and data format validation,

and to perform data validation on the parsed data from various file formats, we will use a validation library like [Joi](#) or [Yup](#). These libraries allow you to **define a data schema**, specifying the expected data type, required fields, and any other validation rules.

Here's an example of how to use the Joi library to validate JSON data:

```
const Joi = require('joi');
const schema = Joi.object({

// mentioning the data conditions that have to be followed by
creating a schema for the same
});
const data = {

};
const { error, value } = schema.validate(data);
if (error) {

} else {

}
```

We can use similar validation techniques with other file formats like [CSV](#), [XML](#), or [YAML](#). The key is to define a schema that matches the structure of the data and any validation rules that apply to it.

## Data Loading:



To use this data for visualization, we first have to **log this experiment data in our MongoDB database** such that it can be used for visualization by the tools provided by the application or other external applications present; we will set up the connection with our database using the MongoDB **client**.

```
axios.post('Url_here', {
  {data}
})
.then((response) => {

// On successful request

})
.catch((error) => {

// if Error is received

});
```

### Development of an API for Plug-in usage :

**An API for a format parser plug-in would provide a standardized way for developers to interact with the plug-in, regardless of the tool in which it is being used.** This would enable developers to seamlessly integrate the plug-in into their existing software while reducing the need for custom code to handle different data formats.

### Using external Metric system for Visualisation

We can also **include external metric tools** for visualization which can be used for the analysis of experimental iterations and compare results from them

For e.g we can use **Prometheus** as our metric tool and then we can use its feature as

- Set up a visualization tool such as **Grafana** to visualize the metrics data collected by Prometheus. Configure a dashboard to display the key metrics in real-time, using visualizations such as line charts, bar charts, and tables.
- Use **Grafana** to gain insights into key performance metrics and to identify trends over time
- Use the data collected by **Prometheus** to create a database of historical performance metrics. This database can be used to analyze performance trends over time and deduce results from these experimental iterations


We can also incorporate better data visualization and form a dashboard for the same.

**NOTE:** The proposed application is built on Node.js frameworks if required I can incorporate the above-mentioned features using different frameworks as I have experience working in different frameworks as well.

**The Proposed UI of the application might look like (Not fixed will modify in order to make it more presentable )**

The UI/UX of the application is built using Framer and is in its initial status the UI can be viewed [here](#):

The image shows a web application interface for an experiment. The interface has an orange header bar with a logo, 'EventLogger', 'Pricing', and a 'Get started' button. The main content area is white and titled 'Experiment Title' in orange. Below the title is 'Experiment Details:' in orange. There are two input fields: 'Iteration ID:' and 'TimeStamp:'. Below these are six rows, each with a 'Parameter' input field and a 'Log' button. At the bottom, there are three orange buttons: 'Save Iteration', 'End Experiment', and 'Add Iteration'.

EventLoggerPricingGet started

Experiment -XYZ


Manage Collaborators

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi et convallis tortor. Pellentesque id risus eu tellus mollis ornare. Aenean sodales rhoncus lacus, sed bibendum nunc dictum et. Morbi viverra ipsum faucibus mi egestas, nec volutpat orci efficitur. In tincidunt bibendum nulla ac accumsan. Aliquam turpis dolor, ullamcorper id leo aliquet, interdum mattis ipsum. Nulla vel suscipit orci. Morbi euismod nulla ac lacus vestibulum condimentum.

Timestamp:

Iteration counts:

Collaborator Id	Name	Permissions	Day Joined	
E1x345fjh	Abcdef	abcdefghijklmnopqrstuvwxyz	13:08:34 22/03/23	Click
E1x345fjh	Abcdef	abcdefghijklmnopqrstuvwxyz	13:08:34 22/03/23	Click
E1x345fjh	Abcdef	abcdefghijklmnopqrstuvwxyz	13:08:34 22/03/23	Click
E1x345fjh	Abcdef	abcdefghijklmnopqrstuvwxyz	13:08:34 22/03/23	Click
E1x345fjh	Abcdef	abcdefghijklmnopqrstuvwxyz	13:08:34 22/03/23	Click
E1x345fjh	Abcdef	abcdefghijklmnopqrstuvwxyz	13:08:34 22/03/23	Click

EventLoggerPricingGet started

Experiment Title

Experiment details:

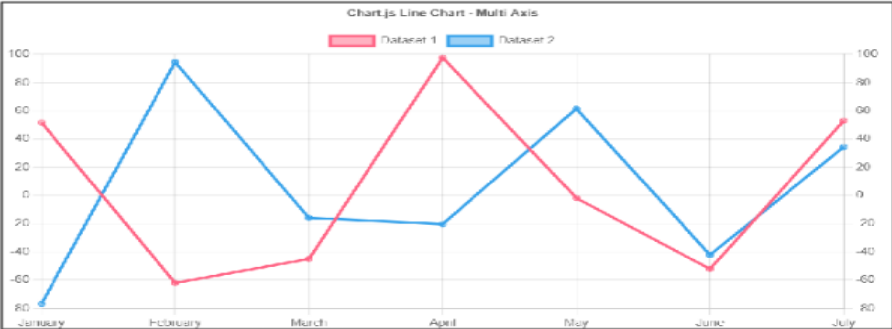
Data analysis

Feature 1

Chart.js Line Chart - Multi Axis

Dataset 1

Dataset 2





Feature 1

Chart.js Line Chart - Multi Axis

Dataset 1







Dataset 2





EventLoggerPricingGet started

My Experiments

	Collaborator Id	Name	Permissions	Day Joined	
	E1x345fjh	Abcdef	abcdefghijklmno	13:08:34 22/03/23	Click
	E1x345fjh	Abcdef	abcdefghijklmno	13:08:34 22/03/23	Click
	E1x345fjh	Abcdef	abcdefghijklmno	13:08:34 22/03/23	Click
	E1x345fjh	Abcdef	abcdefghijklmno	13:08:34 22/03/23	Click
	E1x345fjh	Abcdef	abcdefghijklmno	13:08:34 22/03/23	Click
	E1x345fjh	Abcdef	abcdefghijklmno	13:08:34 22/03/23	Click


Add Experiment

Title

Time

Details

Start Experiment









EventLoggerPricingGet started

Experiment -XYZView Iterations

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi et convallis tortor. Pellentesque id risus eu tellus mollis ornare. Aenean sodales rhoncus lacus, sed bibendum nunc dictum et. Morbi viverra ipsum faucibus mi egestas, nec volutpat orci efficitur. In tincidunt bibendum nulla ac accumsan. Aliquam turpis dolor, ullamcorper id leo aliquet, interdum mattis ipsum. Nulla vel suscipit orci. Morbi euismod nulla ac lacus vestibulum condimentum.

Timestamp:

Iteration counts:

Collaborator Id	Name	Permissions	Day Joined	Add permission Read,Write,Delete	
	E1x345fjh	Abcdef	abcdefghijklmno	13:08:34 22/03/23	<div><div></div><div></div><div></div></div>
	E1x345fjh	Abcdef	abcdefghijklmno	13:08:34 22/03/23	<div><div></div><div></div><div></div></div>
	E1x345fjh	Abcdef	abcdefghijklmno	13:08:34 22/03/23	<div><div></div><div></div><div></div></div>
	E1x345fjh	Abcdef	abcdefghijklmno	13:08:34 22/03/23	<div><div></div><div></div><div></div></div>
	E1x345fjh	Abcdef	abcdefghijklmno	13:08:34 22/03/23	<div><div></div><div></div><div></div></div>
	E1x345fjh	Abcdef	abcdefghijklmno	13:08:34 22/03/23	<div><div></div><div></div><div></div></div>

## Proposed Timeline

### 1. Community Bonding Period:

- Will research more about the Technology stack i will be using and will try to adapt to the best coding standards
- Will enhance my plan of action and will try to incorporate many more functionalities and modify the current ones(if required)
- Will enhance the UI/UX design and make it more presentable for the users and also suites the feature it provides

### 2. Week 1-2:

- Will develop the basic database schemas and models the application will require.
- Host these schemas and models on a MongoDB database and generate an API key for using the cluster.
- Will develop the API endpoints that will be used for making the requests in the database.
- Will test these API endpoints using Postman by making fake API requests.

### 3. Week 3-4:

- Will start the development of the Command Line Interface by initializing the options and their documentation of the purpose they have to serve.
- Will develop the actions these CLI options will provide and then integrate the MongoDB client into these actions, which will be used for data parsing.
- Will enhance the state of the CLI by using the libraries provided by Node.js for CLI development
- Will test the working of the CLI by testing it with the API endpoints and will do modifications if necessary.

### 4. Week 5-6:

- Will start working on the Web application by setting up a Nextjs

application and installing the required dependencies for the Project.

- Will define the Layout of the project by defining the Folder structure and the redirecting urls for each page of the Web App.
- Will form the basic structure of these pages by building the reusable components that will be used in the Web application.
- Will then integrate these reusable components to build the page and will include Hooks to make the app user interactive.

5. **Week 7-8:**

- Will integrate the basic API endpoints and the backend to the frontend Framework so that the application can now be used for data retrieval and logging.
- Will integrate User authentication to the whole application using Auth0 authentication and use its token feature through its Hooks.
- Will work on the data analytics portion of the Web application by using the several libraries that will be used in visualizing the data among different iterations, which then will be used for research purposes.

6. **Week 9-10:**

- Will develop the UI/UX of the application by using different libraries and using the CSS styling in the UI Developing software.
- Will make the application responsive across all the standard screen sizes so that it is accessible from each network device
- Will integrate the application with data-linked tools like Jupyter, docker, etc., making it easier for data to flow across different applications.

[If the project “\*\*Record and Visualize experiment results\*\*” is integrated with this project. In that case, the Timeline can be extended to](#)

7. **Week 11-12:**

- Will modify the UI design of the application by incorporating the plugin in our application
- Will develop a design structure on how to integrate the plug into our

application.

- Will include all the dependencies and the libraries required to develop the plug-in.

**8. Week 13-14:**

- Will build the components and modify the page structure required to integrate the plug-in into our web application.
- Will develop the pages by combining various components with building up the plug-in required
- Will develop the functionality of this plug-in by incorporating the features mentioned earlier

**9. Week 15-16:**

- Will work on the styling of this plug-in created so that the viewer experience is constant throughout the web application
- Will also work on including the data uploading feature through the command line interface (CLI)
- Will develop an API that can be used to incorporate this plugin irrespective of the tool we are using to scale up our tools across different sites.

**10. Week 17-18:( If required)**

- Will work on including external metric tools and libraries that can help in a visualization like Prometheus and others

**NOTE:**

**The Timeline is drafted by considering the delays in work and any changes that must be made during the Coding period so the Project can be completed earlier than the Timeline states.**

## Future Scope

One potential future scope idea is to develop **machine learning algorithms** that can analyze experimental data and use it to predict the outcome of future experiments. By feeding the algorithm data from previous experiments, it can learn **patterns and relationships between variables, allowing it to make increasingly accurate predictions.**

In addition, **AI could be used to identify which experimental variables are most important in determining the outcome.** By analyzing large datasets, AI could help identify key factors that drive results, which could in turn help researchers **refine their experiments and improve their understanding of the underlying processes.**

## More about me and Why me:

I am an undergraduate student at **IIT Varanasi**, which is one of the most prominent institutes in India and secured a Rank of **803 in JEE Advance**, which more than a million people give,

My technology stack has a variety of technologies ranging from Front end frameworks like **Nextjs, Reactjs, Vuejs, Angularjs, and Django** and various backend technologies like **Express.js, Django, and Flask**. Also, I have experience performing e2e tests using technologies like **Selenium and Cypress**. I have worked on various full-stack real-world projects on an individual and team basis. I also practice DSA and competitive programming, and I am rated **1732 (Expert) on [Codeforces](#) and rated 2004( 5 stars ) on [CodeChef](#)** platforms, which can be used to justify my coding experience.

## My Projects:

### A Movie Database and Wishlist App

The Web Application is a combination of a Movie database which also provides the feature of creating your wishlist along with viewing the wishlist of your friends. Also, it allows us to add users as your friend and talk about movies.



### Tech Stuff:

The Application is built on top of the **Nextjs** Framework. It uses **Auth0** authentication uses, external APIs for fetching the Movie Data, and built-in API, which have been constructed in the Nextjs App itself. Also, I have used **MongoDB** as the database for storing the data. Similarly, for styling, I have used **Tailwind and Bootstrap** and would integrate **Threejs** to improve the overall look of the Website.

## [A Study Portal for my Institute](#)

This application was a community project where a bunch of developers from our university came together to build a Study Portal for the institute, which provides a single stop for all the Academics related issues.

### Tech Stuff:

**Could be read from the Readme.MD of the GitHub Repository**

## [Open Source Contributions at Replay.io](#)

During the **Hacktoberfest** and Post-Hacktoberfest, I started working on open-source contributions and worked on raising several issues to Solve some of them to recommend and implement some additional functionalities; therefore, I have some experience in Open Source.

**Also, I have worked on contributions and issues for a previously selected organization like Replay.io**

## [An Activity Manager App for my institute](#)

I also worked on a project which is an Activity Manager and will help individuals to

deal with their hectic schedule by reminding them of the Events and work they have to do by adding some community events for a wide range of people..

Tech stuff:

**The App is built on top of Django and uses MySQL as a database also for the styling the app uses Tailwind CSS and Bootstrap classes.**

## My availability:

I have no other commitments this summer, so GSoC will be my only priority, Also I am working on this project only for GSOC-2023, so I can provide my total commitments to this project

I will be able to devote ample time to work on the project. I am comfortable with working hours but prefer to work during office hours, i.e. **(9:00 -14:00) and (16:00-19:00) +5:30 GMT.**

Also, I have a sheer interest in Open Source and am sure that I will land upon the expectations of my mentors and my organization.

## Post GSOC:

Post Gsoc, I would love to **contribute to other Open source projects** conducted in various Open source programs organized by OSPO also, in the future **would love to work as a mentor** and be on the other side of this journey and help budding open source contributors like me