



Google Summer of Code 2024 **Project Proposal**



**StatWrap: Automated Reproducibility Checklists
Generation for Research Projects**

Adi Akhilesh Singh

Project Size: Large(350 hours)

Mentor: [Luke Rasmussen](#)

Table of Contents

Project Introduction	3
About StatWrap	3
Technologies used at present	3
Problem to be Addressed	3
Current State of the Problem	4
Proposed Solution	4
Project Goals	5
Project Objectives	5
Expected Deliverables	5
Future Work	11
Implementation Plan	12
Project Methodology	12
Technical Elements	12
Challenges	19
Project Timeline	20
Deliverables Schedule	20
Availability	21
Post-GSoC plans and Community Engagement	21
Biographical Information	22
Relevant Experience	22
Educational Background	25
Technical Interests	25
Contact Information	25

Project Introduction

About StatWrap

StatWrap is a free and open-source assistive, non-invasive discovery and inventory tool to document research projects. It inventories project assets (e.g., code files, data files, manuscripts, documentation) and organizes information without additional input from the user. It also provides structure for users to add searchable and filterable notes connected to files to help communicate metadata about intent and analysis steps.

Technologies used at present

- StatWrap is built using **Electron**, for cross-platform compiling and deployment.
- It uses basic **React** architecture for the user interface.
- **Yarn** is used for package management across different operating systems.
- For testing the platform, **Jest** Testing Framework is used.
- The Project is triggered via Native Git hooks, **Husky**, whenever a PR is created on StatWrap.

Problem to be Addressed

There are several challenges that researchers trying to reproduce the research project findings currently may face, which this project aims to solve, by introduction of customizable reproducibility checklists to document the methodology used and ensure transparency in code and report.

Computational Reproducibility

“An article about computational science in a scientific publication is not the scholarship itself, it is merely advertising of the scholarship. The actual scholarship is the complete software development environment and complete set of instructions which generated the figure.” - Jonathan B. Buckheit and David L. Donoho, “WaveLab and Reproducible Research” (1995).

Some of the possible problems that researchers currently face in its absence are listed below:

- **Difficulty in Replicating Results:** Incomplete or vague research documentation processes may lead to inconsistent outcomes, along with varying interpretation of results due to lack of clear guidance.
- **Ambiguity in Methodology:** Absence of standardized checklist of processes and procedural steps used, hampers reproducibility, potentially affecting results validity.
- **Delayed and Resource-Intensive Reproduction:** Lack of clear guidelines and procedural steps, reproducing research can be burdensome, requiring researchers to decipher ambiguous methods or contacting the original authors for clarification, delaying the reproduction process.

- **Lack of Code & Data Transparency and Selective Reporting:** In computational research, access to source code, relevant datasets, and data repositories used for data analysis and research simulations is essential. However, researchers often fail to make these resources and information sources such as websites, reference lists or registers accessible, hindering replication efforts. Additionally, selective reporting of results or omission of null or negative findings can contribute to discrepancies between original and replicated results.

Current State of the Problem

At present, there is no software to help automate the process of documenting reproducibility checklists along the progress of research projects as the researchers implement their models and computational findings in code. So, researchers and individuals attempting to reproduce research findings encounter significant challenges due to the absence of standardized reproducibility checklists.

While StatWrap primarily focuses on providing help with the management of project assets and people involved, and statistical analysis through its interactive data presentation along with documenting the process, it does not currently offer functionalities for generating reproducibility checklists. The existing support within StatWrap primarily revolves around project logging, which heavily relies on user interaction, manual notes, and monitoring of source-control progress (if enabled). As a result, there remains a crucial gap for a software solution that seamlessly integrates reproducibility checklist generation into the research project workflow.

Proposed Solution

In response to the prevalent challenges faced by researchers and individuals attempting to reproduce research findings, our proposed project idea aims to develop an **automated reproducibility checklist generation**. This comprehensive checklist will encompass vital details regarding the research project's methodology, the key entry points, relevant datasets and repositories, file paths, along with some procedural guidance and adherence to the coding practices used while implementing the research project, all this being completely **configurable** by the end-user. This will help the individuals streamline the setup process, with easy access to datasets or data repositories, and result in standardization of the implementation of research projects, along with guidance through user-added manual comments and notes, which will facilitate the comprehension of complex processes easily and understand the original author's intentions attached with these processes. Ultimately enhancing the reproducibility and transparency of research projects, fostering a culture of robust scientific research.

Project Goals

Project Objectives

The primary goal of this project is to develop an automated reproducibility checklist generation tool, which will involve introduction of a new interface page, that will significantly enhance the reproducibility of research projects across domains. Specifically, our objectives include:

- **New Checklists User Interface:** Implementing an intuitive and user-friendly interface for generating reproducibility checklists from the project data and user interactions, reducing the time to generate reproducibility documentation.
- **Flexibility:** Providing extensive configurability options to tailor the checklist according to specific needs of users, ensuring adaptability across diverse research projects. This involves the ability to **comment**, **update answer** states (be it textual or boolean), enable **image** attachments for display alongside each checklist point, etc with options to customize/edit each of these.
- **Community Collaboration:** Fostering knowledge sharing within the research community by enabling users to contribute to and share reproducibility checklists by **exporting** the checklist data as a file (**pdf**, docx, txt, csv or any other relevant format), promoting community engagement and collective refinement of checklists for a project, enabling multiple perspectives.
- **Commitment to Continual Improvement:** With receiving user and research community feedback, I'll be committed towards **iteratively improving** on our tool, beyond the initial development phase, ensuring that it remains aligned to the needs and standards of the research community.

All these will benefit the research community in multiple dimensions, by enhancing efficiency through time and effort reduction, providing customized solutions via configurable checklists along with knowledge sharing by enabling users to contribute to and share reproducibility checklists.

Expected Deliverables

In response to the pressing need for standardized reproducibility practices within the research community, our proposed initiative seeks to deliver a robust and adaptable solution. Our deliverables will focus on harnessing project data to initialize reproducibility checklists, offering customizable options for users to tailor checklist items, and providing an intuitive user interface. Additionally, we will prioritize the integration of advanced features which we'll discuss in detail in following points of our expected deliverables.

- **Standardization:** Our checklist will follow some of the internationally recognized reproducibility checklists as a base, namely [PRISMA checklists](#), [CONSORT checklists](#), [UVIC checklists](#) and others as we discuss further, to establish a set of standard checklist statements, and to initialize it with predicted answers.

- **Data Structure Creation:** After identifying the reproducibility checklists, we can move forward with creating the data structure to configure these checklists, depending upon all the data requirements of checklist statements, and the anticipated feature set we intend to integrate.
- **Initialization with collected StatWrap data:** Utilizing the existing project data using StatWrap, to **automatically predict** and seed the checklists with essential project details, such as **file paths**, statements related to **sensitive information**, **dependencies** used in the project, and the **project logs** data. This initialization lays down a standard checklist creation, saving users valuable time and effort, also guiding them with the intent of these checklists to add more later.
- **Customizability:** Offer users the ability to customize checklists based on project specifics, enabling **updates**, **deletions** and **additions** to checklist statements and corresponding answers, as needed.
- **Feature Set:** We will look forward to integrate various feasible useful features such as, enabling users to provide additional context through **manual-comments**, **image attachments**, **hyperlink attachments**, **adding notes** of how users can achieve some checklists tasks which are marked as negative, and feature to **export** the complete reproducibility **checklist report** as a file (in **pdf**, docx, csv or any other relevant format).



Taking into account these outlined considerations, I've **designed** a few pages for the Reproducibility Checklists user interface, including a dialog box for checklist editing.

These initial UI designs and the feature set present are just to visualize this project, the additional functionalities or alterations with these features needed can be discussed as we proceed. I've organized the Reproducibility checklists, as we can see, on the basis of their type. So, **the complete Checklists is a set of different checklists which contains checklist statements, which may further contain some sub-checklists**. This structure is very similar to what is found in the [UVIC checklists](#).

Reproducibility Checklist

DEPENDENCY CHECKLIST

Q1: Have you mentioned all dependencies of your research project for reproducibility? Add comments if necessary.

 Attached Images 



Q2: Have you documented the steps to install all the project prerequisites and dependencies? Add comments if necessary.

1

10

 Attached Images 

Q3: Are there any dependencies which are deprecated or not updated to their recent working versions? Share your thoughts.

DATA CHECKLIST

Q4: Is the raw data available? If only processed data is provided, is there enough information to understand how the raw data was modified or transformed?

Notes:

adi @ 2024-03-20 19:07:24

Raw data is available in the "raw_data.csv" file, and transformations applied are described in detail in the "data_preprocessing.ipynb" Jupyter Notebook.



Q5: Are all data files necessary to rerun analyses provided? If not, are links to containing repositories specified?

 Related URLs 

- <https://github.com/StatTag/StatWrap/data/statwrap-project.json>
- <https://github.com/AdiAkhileshSingh15/MOSSE-ObjectTracking/tree/master/datasets/img>

ORGANIZATION CHECKLIST

Q6: Are all files encapsulated within one directory?









 Attached Images 



▼ travel-planner

- README.md
- > app
- > backend

Q7: Is the sub-directory structure clear and easy to navigate? Attach images to support your answer.

Sub-checklist 

- ☒ Are the names of subdirectories self-explanatory?  
- ☒ Is the raw / input data separated from the derived data?  
- ☒ Is the data separated from the code?  
- ☒ Are any outputs (figures, tables) provided? Are they contained in their own subdirectory?  




 Attached Images 



Save checklist as a file


Export 

Fig 2.1: Demo Reproducibility Checklists Page Interface



DEPENDENCY CHECKLIST

Researcher

 John D.

Status

Update

Task


Dependency List

Details

Q1: Have you mentioned all dependencies of your research project for reproducibility?
Add comments if necessary.


☒ Yes ☐ No

Supporting files

 No files attached

Attach

Related URLs

 No hyperlinks


Include

Q2: Have you documented the steps to install all the project prerequisites and dependencies? Add comments if necessary.

1


10

Supporting files

 No files attached

Attach

Related URLs


 No hyperlinks

Include

Q3: Are there any dependencies which are deprecated or not updated to their recent working versions? Share your thoughts.


☒ Yes ☐ No

Supporting files

 No files attached

Attach

Related URLs

 No hyperlinks

Include


Sub-checkpoints

☒ Verify versions

☐ Confirmation

Notes

Add your thoughts here.



Save checklist

Fig 2.2: Edit Dialog box of a Checklist

The initialization of these checklists presents a minor challenge in determining the initial statements to introduce. This process will require some discussion to decide upon the appropriate checklist statements. Personally, I'd advocate selecting the list of statements from the provided resources of standard checklists, which we can predict the answers for quite accurately, from the available project data that StatWrap collects, ensuring the checklists are both comprehensive and relevant.

The following are my personal selections:

REPRODUCIBILITY CHECKLISTS

Organization:

- Are all files encapsulated within one directory? (Yes, because of the way StatWrap currently deals with projects)
- Is the sub-directory structure clear and easy to navigate?
 - Are the names of subdirectories self-explanatory? (Requires user input for Yes / No / Maybe.)
 - Is the raw / input data separated from the derived data? (Requires user input for Yes / No / Maybe.)
 - Is the data separated from the code? (It can be predicted from the collected project data from StatWrap, by checking if there are data files(Text, Excel, Parquet, JSON, XML, Stata Data, R Data, SAS Data) and code(R, Python, SAS, Stata, HTML) files present under single directory or not.)
- Are file names self-explanatory? If not, how could they be improved? (Requires user input for Yes / No / Maybe and/or manual notes.)
- Are all the file names free from usage of words like “final”? If not, consider enumerating multiple versions of such files, like data_v4.csv. (It can be predicted from the collected project data from StatWrap, by checking all the file names for such keywords.)
- Are there multiple versions of the file? If yes, are versions clearly enumerated? (Predicting for file versions can be inconsistent across different project naming standards, so we can take this as user input.)

Coding Practices:

- Is there any repetition of large code sections? If yes, consider turning them into more-general functions. (We can look for some software to detect large repetition in code, given that we can have read access to the project files. Or else, we can simply choose to depend on user input.)
- Does the code follow a consistent coding style and convention, along with thorough annotation with comments? (Preferable to rely upon user input for Yes / No, than implementing an automated project code analysis.)

- Is there a README file? (Yes / No depending upon the presence of a readme file)
 - Is yes, does it specify author contact information, file contents, directory overview, dependencies, etc? What other information could it provide to improve reproducibility? (Requires user input for Yes / No / Maybe and/or manual notes.)

Dependencies and Prerequisites:

- Are all the dependencies of your research project specified clearly, for reproducibility? (Yes / No depending upon whether or not the dependency listing file (package.json, requirements.txt etc) is updated as per project requirements.)
- Are all the prerequisite software to run/build the project locally, specified clearly? (Requires user input for Yes / No / Maybe)
- Are all the steps to install the project prerequisites and dependencies documented? (Requires user input for Yes / No / Maybe)
- Are there any dependencies which are deprecated or not updated to their recent supported versions? If yes, please specify them. (Requires user input for Yes / No / Maybe and manual notes.)

Document Data:

- Are all data files necessary to rerun analysis provided? If not, are links to containing repositories specified? (Requires user input for Yes / No / Maybe and/or attached URLs.)
- Is the raw data provided? If only process data are provided, is there sufficient description to understand transformations made to raw data? (Requires user input for Yes / No / Maybe and/or manual notes.)
- Is sufficient documentation provided to understand the data, its type and its interpretation, like a data dictionary? (Requires user input for Yes / No / Maybe.)

Document Software:

- Is the software environment specified? (Requires user input for Yes / No / Maybe.)
- Are all file conversion, data cleaning and analysis steps documented by scripts? (Requires user input for Yes / No / Maybe.)
- Is the execution of all the project code automated by a master script? In either case, is necessary documentation regarding the scripts provided? (Requires user input for Yes / No / Maybe and/or manual notes.)
- Are decisions behind data cleaning, analysis, and other scripts well documented within the code as annotations, or as a reproducible report (e.g. R markdown (*.Rmd))? (Requires user input for Yes / No / Maybe and/or manual notes.)

Results:

- Is the source-code made public sufficient, accurate and error-free to reproduce similar results as mentioned in the related research paper? (If applicable) (Requires user input for Yes / No / Maybe and/or manual notes.)
- Is relevant guidance, through research paper and documentation, provided to interpret the reproduced results correctly? (Requires user input for Yes / No / Maybe and/or manual notes and/or attached images.)
- Is there any modifications or processing of the results required before finally comparing it with the research findings? If yes, specify it clearly? (Requires user input for Yes / No / Maybe and/or manual notes.)

Licensing and Sharing:

- Is a license specified for the software? (for e.g. either in a README file or a separate license text file?) (Requires user input for Yes / No / Maybe.)
- Is a license specified for the data being used for research? (Requires user input for Yes / No / Maybe.)
- Is the repository(ies) containing the data and code registered with a unique DOI? (Requires user input for Yes / No.)

After completion of these tasks, the subsequent step involves generating the checklist report. This functionality will facilitate exporting the checklist in various file formats such as pdf, docx, and csv. This would probably not involve a lot of work, just iterating through the checklists data to write into the file, while maintaining the text format standards expected for a reproducibility checklist report, and enable export functionality for the file.

Future Work

Based on this Reproducibility Checklists Project, we can implement a lot more to it after its completion. Here are some suggestions that I've for future work and potential applications of this project:

- **Collaborative Checklist Editing:** This will enable multiple users to contribute to and edit the checklist simultaneously, related to the same project.
- **Import data to Configure Checklists:** This functionality will allow users to import checklists data (in JSON or CSV format possibly) to configure the checklists with this data. (This can be implemented during the project timeline itself if the project maintainers agree.)
- **Extension to Other Domains:** This will involve expanding the application of the checklist beyond research reproduction. For example, the checklists can be adapted for use in software development projects, quality assurance processes and assessments.
- **Integration with AI:** This specific project of checklists generation and answers prediction using available project and standard checklists data, to learn patterns and relationships between statements and data, can greatly benefit from the

integration of ML algorithms into it. (Although this integration requires additional time and expertise, and above all, the willingness of the project maintainers, I believe it holds great potential to enhance the checklist's relevance and functionality.)

Implementation Plan

Project Methodology

To achieve the project objectives, I will begin by identifying the initial checklists to serve as a starting point, as explained above. Following this, I will design a robust data structure capable of accommodating all necessary aspects of data manipulation based on the checklists and feature set we plan to implement. Once the data structure is in place, I will proceed to implement a checklist prototype incorporating all the features discussed and agreed upon to that point, drawing inspiration from existing checklists. Throughout this process, I will regularly engage with my mentor to review the functionality and user interactions, seeking feedback and guidance for further improvement.

Upon receiving approval from the mentor, I will focus on refining the overall UI of the checklist page inspired from the overall application user interface. Finally, I will work on integrating the functionality to export the checklist report as a file. Throughout these stages, I will maintain regular communication with the mentor, providing weekly updates on project progress and possibly scheduling biweekly meetings for in-depth discussions regarding project status and implementation.

In the event of progressing ahead of the established timeline, I will seize the opportunity to either work on additional features or speed up the completion of existing ones. By maintaining a collaborative approach with the mentor, I ensure the successful realization of project objectives within specified timeframe.

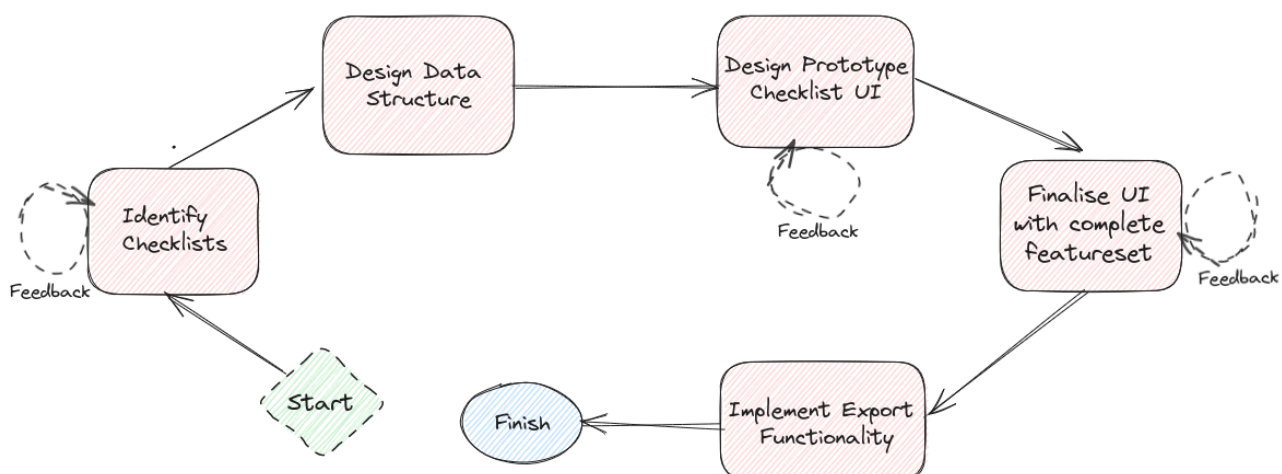


Fig 4.1 Methodology Flowchart

Technical Elements

The technical aspect of the project involves implementation of the project following the discussed methodology in code. The checklists identification part was discussed earlier, so

we will move forward with **designing the data structure** for our checklists. This will involve thorough understanding of the future requirements of the checklists, while also considering all the features that need to be implemented.

Notes.md

Checklists.md

AssetAttributes.js

AssetNode.js

AssetDetails.js

asset.js

Preview Checklists.md

docs > Checklists.md > ## Checklist Object > ### URL Object > ### Attributes

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

Checklist Object

About

The **Checklist Object** represents a single checklist item within the reproducibility checklist. It encompasses various attributes and sub-objects to capture different aspects such as statement, type, answer, user notes, attached images, attached URLs, and sub-checklists.

Attributes

Attribute	Type	Description
id	UUID	A generated unique identifier for the checklist item.
statement	String	The statement or question associated with the checklist item.
type	String	Indicates the type of checklist item, such as boolean or descriptive.
answer	String	Stores the user's response to the checklist item. ('yes'/'no' for boolean type, 'answer description' for descriptive type)
userNotes	Array ([Note])	An array containing user-added notes associated with the checklist item.
attachedImages	Array ([Image])	An array containing image data of images attached to the checklist item.
attachedURLs	Array ([URL])	An array containing URLs attached to the checklist item.
subChecklists	Array ([Sub_checklist])	An array containing sub-checklists associated with the checklist item.
updated	String	The timestamp indicating when the checklist item was last updated.

Checklist Object

About

The **Checklist Object** represents a single checklist item within the reproducibility checklist. It encompasses various attributes and sub-objects to capture different aspects such as statement, type, answer, user notes, attached images, attached URLs, and sub-checklists.

Attributes

Attribute	Type	Description
id	UUID	A generated unique identifier for the checklist item.
statement	String	The statement or question associated with the checklist item.
type	String	Indicates the type of checklist item, such as boolean or descriptive.
answer	String	Stores the user's response to the checklist item. ('yes'/'no' for boolean type, 'answer description' for descriptive type)
userNotes	Array ([Note])	An array containing user-added notes associated with the checklist item.
attachedImages	Array ([Image])	An array containing image data of images attached to the checklist item.
attachedURLs	Array ([URL])	An array containing URLs attached to the checklist item.
subChecklists	Array ([Sub_checklist])	An array containing sub-checklists associated with the checklist item.
updated	String	The timestamp indicating when the checklist item was last updated.

Fig 4.2 Checklist Object Data Structure documentation

Notes.md

Checklists.md

JS AssetAttributes.js

Preview Checklists.md

docs > Checklists.md > ## Checklist Object > ### Sub-Checklist Object > ### Attributes

1

21

27

30

31

32

33

34

35

36

Checklist Object

Sub-Checklist Object

Attributes

Attribute	Type	Description
id	UUID	A generated unique identifier for the sub-checklist item.
statement	String	The statement or question associated with the sub-checklist item.
type	String	Indicates the type of sub-checklist item, such as boolean or descriptive.
answer	String	Stores the user's response to the sub-checklist item. ('yes'/'no' for boolean type, 'answer description' for descriptive type)
updated	String	The timestamp indicating when the sub-checklist item was last updated.

Sub-Checklist Object

About

The **Sub-Checklist Object** represents a sub-item within a checklist item. It shares similar attributes with the Checklist Object, except for attachments, but is nested within the parent checklist item.

Attributes

Attribute	Type	Description
id	UUID	A generated unique identifier for the sub-checklist item.
statement	String	The statement or question associated with the sub-checklist item.
type	String	Indicates the type of sub-checklist item, such as boolean or descriptive.
answer	String	Stores the user's response to the sub-checklist item. ('yes'/'no' for boolean type, 'answer description' for descriptive type)
updated	String	The timestamp indicating when the sub-checklist item was last updated.

Fig 4.3 SubChecklist Object Data Structure documentation

Notes.md

Checklists.md U

JS AssetAttributes.js

...

Preview Checklists.md x

docs > Checklists.md > ## Checklist Object > ### URL Object > #####
1 ## Checklist Object
37 ### Note Object
43 ##### Attributes

47 | `id` | UUID | A generated
unique identifier for the
note.

48 | `author` | String | A display name
taken from StatWrap to indicate who created the
note.

49 | `updated` | String | The timestamp
indicating when the note was last
updated.

50 | `content` | String | The text content
of the
note.

Note Object

About

The **Note Object** stores user-added note associated with checklist items.

Attributes

Attribute	Type	Description
id	UUID	A generated unique identifier for the note.
author	String	A display name taken from StatWrap to indicate who created the note.
updated	String	The timestamp indicating when the note was last updated.
content	String	The text content of the note.

Fig 4.4 Note Object Data Structure (Inspired from current Notes.md)

Notes.md

Checklists.md U

JS AssetAttributes.js

...

Preview Checklists.md x

docs > Checklists.md > ## Checklist Object > ### URL Object > #####
1 ## Checklist Object
52 ### Image Object
58 ##### Attributes

61 | ----- | ----- |

62 | `id` | UUID | A generated
unique identifier for the
image.

63 | `data` | Blob | The image data
of the attached
image.

64 | `updated` | String | The timestamp
indicating when the image was last
updated.

Image Object

About

The **Image Object** stores image data of the image attached to checklist items.

Attributes

Attribute	Type	Description
id	UUID	A generated unique identifier for the image.
data	Blob	The image data of the attached image.
updated	String	The timestamp indicating when the image was last updated.

Fig 4.5 Image Object Data Structure documentation

65 ## URL Object
66
67 ### About
68
69 The **URL Object** stores hyperlink of the URL attached to checklist items.
70
71
72 ### Attributes
73
74 | Attribute | Type | Description |
75 | ----- | ----- | ----- |
76 | `id` | UUID | A generated unique identifier for the
URL.
77 | `hyperlink` | String | The hyperlink associated with the
URL.
78 | `updated` | String | The timestamp indicating when the URL was last
updated.

URL Object

About

The **URL Object** stores hyperlink of the URL attached to checklist items.

Attributes

Attribute	Type	Description
id	UUID	A generated unique identifier for the URL.
hyperlink	String	The hyperlink associated with the URL.
updated	String	The timestamp indicating when the URL was last updated.

This documentation outlines the structure and attributes of various objects involved in the reproducibility checklist project.

End of document

Fig 4.6 URL Object Data Structure documentation


```

1  import lockfile from 'proper-lockfile';
2  import { v4 as uuid } from 'uuid';
3
4  const os = require('os');
5  const path = require('path');
6
7  const DefaultChecklistFile = '.reproducibility-checklist.json';
8
9  export default class ReproducibilityChecklistService {
10    /**
11     * @param {string} id - The unique identifier for the checklist.
12     * @param {string} statement - The statement/question for the checklist item.
13     * @param {string} answer - The answer/expected response for the checklist item.
14     * @param {Array} [images] - The array of images associated with the checklist item.
15     * @param {Array} [urls] - The array of URLs associated with the checklist item.
16     * @param {Array} [notes] - An array of notes related to the checklist item.
17     * @returns - The initialized checklist object.
18     */
19    createChecklistConfig(id, statement, answer, images, urls, notes) {
20      const checklistConfig = {
21        id: id || uuid(),
22        statement,
23        answer,
24        images: images || [],
25        urls: urls || [],
26        notes: notes || [],
27      };
28
29      return checklistConfig;
30    }
31
32    /**
33     * Create a checklist configuration object.
34     * @param {string} id - The unique identifier for the checklist.
35     * @param {string} statement - The statement/question for the checklist item.
36     * @param {string} answer - The answer/expected response for the checklist item.
37     * @param {Array} [images] - The URL or path to the image associated with the checklist item.
38     * @param {Array} [urls] - The URL link associated with the checklist item.
39     * @param {Array} [notes] - An array of notes related to the checklist item.
40     * @returns - The checklist configuration object.
41     */
42
43    initializeChecklist(id, statement, answer, images, urls, notes) {
44      // Basic checks for required parameters
45      if (!id || !statement || !answer) {
46        throw new Error('Incomplete checklist information');
47      }
48
49      // Additional functionalities can be implemented here
50      const checklistConfig = this.createChecklistConfig(id, statement, answer, images, urls, notes);
51      // Other checks or operations can be added as needed
52
53      return checklistConfig;
54    }
55
56    lockChecklistFile(checklistPath) {
57    }
58
59    unlockChecklistFile(checklistPath) {
60    }
61
62    addNote(checklist, newNote) {
63    }
64
65    uploadImage(checklist, imageobj) {
66    }
67
68    attachUrl(checklist, url) {
69    }
70  }
71

```

Fig 4.7 ChecklistService

```

1  /**
2   * Lock the checklist file to prevent concurrent access.
3   * @param {string} checklistPath - The path to the checklist file.
4   */
5  lockChecklistFile(checklistPath) {
6      const filePath = path.join(checklistPath.replace('~', os.homedir), DefaultChecklistFile);
7
8      lockfile.lockSync(filePath);
9  }
10
11 /**
12  * Unlock the checklist file to allow concurrent access.
13  * @param {string} checklistPath - The path to the checklist file.
14  */
15  unlockChecklistFile(checklistPath) {
16      const filePath = path.join(checklistPath.replace('~', os.homedir), DefaultChecklistFile);
17
18      lockfile.unlockSync(filePath);
19  }
20
21 /**
22  * Add notes to a checklist.
23  * @param {object} checklist - The checklist object to which notes are to be added.
24  * @param {object} newNote - The array of new notes to be added.
25  */
26  addNote(checklist, newNote) {
27      if (!checklist || !newNote) {
28          throw new Error('Invalid checklist or note object');
29      }
30
31      checklist.notes.push(newNote);
32  }
33
34 /**
35  * Upload an image to a checklist.
36  * @param {object} checklist - The checklist object to which the image is to be uploaded.
37  * @param {object} imageobj - The image object to be uploaded.
38  */
39  uploadImage(checklist, imageobj) {
40      if (!checklist || !imageobj) {
41          throw new Error('Invalid checklist or image object');
42      }
43
44      checklist.images.push(imageobj);
45  }
46
47 /**
48  * Attach a URL to a checklist.
49  * @param {object} checklist - The checklist object to which the URL is to be attached.
50  * @param {object} url - The URL object to be attached.
51  */
52  attachUrl(checklist, url) {
53      if (!checklist || !url) {
54          throw new Error('Invalid checklist or url object');
55      }
56
57      checklist.urls.push(url);
58  }

```

Fig 4.8 Checklist Service Methods

(Divided code, as couldn't ensure clarity in one)

These **code snippets** and **data structures** for the checklists represent an initial **rough implementation**, aimed at understanding how we approach the foreseen implementation. They serve as a foundation to develop an idea of how we can manipulate the data effectively.

ReproducibilityChecklistService is designed following the pattern of our existing methods for manipulating data objects such as projects and assets. It also emulates the structure commonly seen in Java services, repositories and DAOs. All this data manipulation/checklists configuration will be similar to how we handle the project data in StatWrap using a .statwrap-project.json file. The anticipated implementation involves managing and accessing the data for all the checklists configurations within a JSON file. Eventually, exporting the standard formatted reproducibility checklist report as file.

While the provided methods offer basic functionality for returning checklist objects, the actual implementation will of course require more extensive data manipulation, which we will manage by breaking down the methods further into utility functions. The utility functions will include tasks such as dealing with file uploads, collecting the required data/metadata for checklist attributes, and updating the data objects, etc. My experience in object-oriented programming can be of decent help in the implementation, if we adhere to the current implementation style. The data structure I've designed anticipates the future data requirements and features, though we remain open to refining it through discussions and iterations.

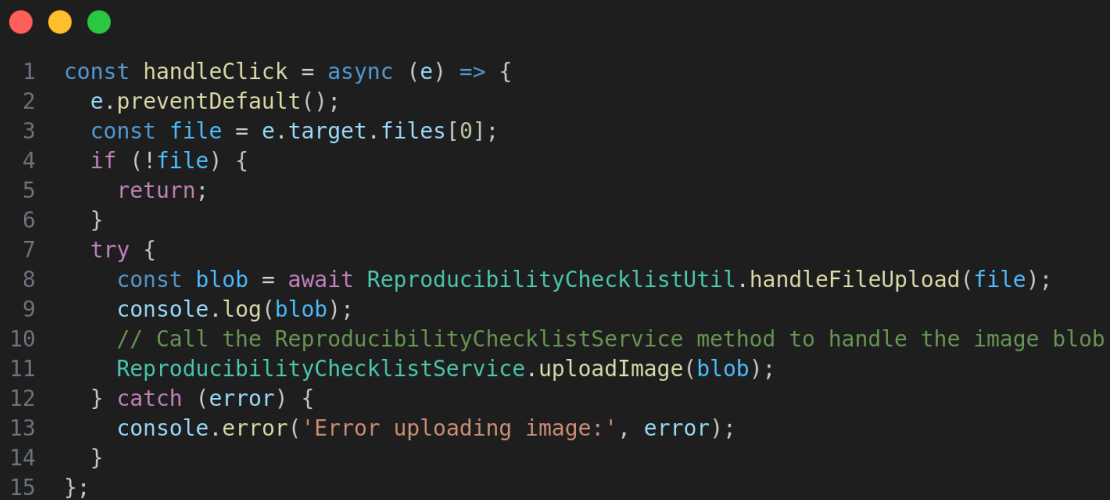
Managing the file uploads can be seamlessly done using the Javascript form event handling. By encapsulating the functionality within ChecklistUtil class, we can define methods that facilitate the transformation of uploaded files into Blobs(array buffers). These image blobs can then be stored into the image object of the checklist utilizing the methods provided by ChecklistService, as outlined earlier.

```

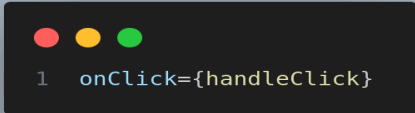
1  export default class ReproducibilityChecklistUtil {
2    /**
3     * Function to handle file upload and convert the file to a Blob.
4     * @param {File} file - The file to be uploaded.
5     * @returns {Promise<Blob>} - A promise that resolves with the Blob object representing the file data.
6     */
7    static async handleFileUpload(file) {
8      return new Promise((resolve, reject) => {
9        const reader = new FileReader();
10       reader.onload = () => {
11         const arrayBuffer = reader.result;
12         const blob = new Blob([arrayBuffer], { type: file.type });
13         resolve(blob);
14       };
15       reader.onerror = (error) => {
16         reject(error);
17       };
18       reader.readAsArrayBuffer(file);
19     });
20   }
21 }

```

Fig 4.9 File Upload Util



```
1  const handleClick = async (e) => {
2    e.preventDefault();
3    const file = e.target.files[0];
4    if (!file) {
5      return;
6    }
7    try {
8      const blob = await ReproducibilityChecklistUtil.handleFileUpload(file);
9      console.log(blob);
10     // Call the ReproducibilityChecklistService method to handle the image blob
11     ReproducibilityChecklistService.uploadImage(blob);
12   } catch (error) {
13     console.error('Error uploading image:', error);
14   }
15 };
```

Fig 4.10 File Upload Event handler

```
1  onClick={handleClick}
```

Fig 4.11 Click event

Implementing other features and functionalities will likely pose fewer challenges compared to handling file uploads, as they won't involve data transformation tasks. Therefore, I haven't included the code implementations for those features here. Instead, we can focus on discussing potential challenges we may encounter while progressing with the project.

Another important feature of our project involves exporting checklists as report files, in pdf or other formats. Below is the pdf version of the code implementation. This process entails writing checklists data to a file as a stream through piping, while also ensuring that the file maintains its styling using this `pdfkit` library.

```
1 export default class ReproducibilityChecklistUtil {
2   /**
3    * Export the checklist data as a PDF file.
4    * @param {Array} checklistData - The checklist data to be utilised for report generation.
5    * @param {string} outputFilename - The name of the output PDF file.
6    */
7
8   static exportChecklistAsPDF(checklistData, outputFilename) {
9     const doc = new PDFDocument();
10    doc.pipe(fs.createWriteStream(outputFilename));
11
12    doc.fontSize(24).text('Reproducibility Checklist Report', { align: 'center' });
13    doc.moveDown();
14    doc.fontSize(18).text('Checklist Items:', { align: 'left' });
15    doc.moveDown();
16
17    // Loop through each checklist item and add statements, answers to the PDF
18    checklistData.forEach((item, index) => {
19      doc.fontSize(12).text(`${index + 1}. ${item.statement}`, { continued: true });
20      doc.fontSize(12).text(`Answer: ${item.answer}`);
21      doc.moveDown();
22    });
23
24    doc.end();
25
26    console.log(`Reproducibility Checklist Report generated as: ${outputFilename}`);
27  }
28 }
```

Fig 4.12 Report Export Util

Challenges

The project we have undertaken to develop an end-to-end solution for generating automated reproducibility checklists, while also ensuring adherence to UI standards, presents several challenges that we need to address effectively. Some of these challenges include:

- **Deciding on Reproducibility Checklists:** One significant challenge is determining the initial set of reproducibility checklists. Since there is no universally accepted standard checklist to follow for research reproducibility. We will need to engage in constructive discussions to establish a comprehensive list of standard checklists that can serve as an initial point.
- **User Input Handling:** Handling user input, particularly when it comes to uploading files, requires careful consideration. We need to implement robust mechanisms to handle file uploads securely, validate user inputs effectively, and prevent potential security vulnerabilities or data corruption issues.
- **Well-formatted/Visually appealing Reports:** We need to ensure that the exported reports are not only accurate and comprehensive but also visually appealing. This will involve integrating npm libraries to do the heavy-lifting for us by customizing the report to align with specific styling aesthetics.

- **Documentation:** We need to create clear and concise documentation that covers all the aspects of the introduced feature, including usage instructions, troubleshooting guides, and most importantly developer documentation.

Project Timeline

Deliverables Schedule

Timeline	Milestone
Community Bonding Period (May 1 - May 26)	<ul style="list-style-type: none"> • Onboarding and getting to know the team members and their roles. • Coordinate to get my pending PRs merged and follow up needed for them. • Finalize the project requirements, and further improvements in the plan of action, through discussions with the mentor.
Week 1 - 2 (May 27 - June 9)	<ul style="list-style-type: none"> • Identify and finalize the reproducibility checklists to use as guides. • Discussing and finalizing the way to configure checklists, specifically, to use JSON or not. • Discussion on whether to pre-populate the initial checklists with predicted answers using project data or leave it to the users for accuracy.
Week 3 - 4 (June 10 - June 23)	<ul style="list-style-type: none"> • Design the checklists data structure to configure and manipulate checklists, along with required features. • Discussion with the mentor and team, on the feature set to be included, and finalizing the data structure.
Week 5 - 6 (June 24 - July 7)	<ul style="list-style-type: none"> • Design a prototype checklists UI in the newly integrated reproducibility tab, to showcase key features, possibly 'adding notes', 'updating checklists', 'adding new checklists', 'adding images', 'adding hyperlinks', 'adding sub-checklists' . • Working on feedback to the features integrated in the prototype checklists component. • Finalize the checklists component UI with complete feature set implementation.
Week 7 - 8 (July 8 - July 21) (July 8 – July 12 {Midterm Evaluation})	<ul style="list-style-type: none"> • <u>Midterm Evaluation.</u> • Perform local testing of the end-to-end functionality and address any issues. • Implement unit testing using Jest for the checklists component added.
Week 9 - 10 (July 22 - August 4)	<ul style="list-style-type: none"> • Discuss reproducibility checklist report format and styling standards. • Implement functionality to generate reproducibility checklist reports to export as files, primarily in PDF format. • Extend export functionality to other file formats, such as

	DOCX, CSV and JSON (to share the configuration to others). <ul style="list-style-type: none"> Implement tests for export functionality, thereby ensuring end-to-end unit testing coverage for the introduced checklists component.
Week 11 - 12 (August 5 - August 18)	<ul style="list-style-type: none"> Conduct final project review with mentor and team, incorporating feedback and suggestions. Provide comprehensive documentation (developed progressively throughout the timeline) for the checklists project, for the end-users and developers. Utilize remaining time before final evaluation for additional features, likely the import of JSON checklist-config & collaborative editing. Improve UI/UX for the checklists component, and the overall StatWrap Project. Improve overall StatWrap Project documentation, and add create a detailed contributing guide for developers.
Week 13 (August 19 – August 26) (Final GSoC contributor evaluations)	<ul style="list-style-type: none"> Project Presentation Submit <u>Final Evaluation</u> for the standard coding period.

Availability

During this summer, I am fully committed to participating in GSoC, and it will be my sole focus. I do not have any travel plans or other commitments to distract me from my dedication to the project. While I may have institute exams for my upcoming 7th semester, the same are over in under a week and would pose no barrier to my work schedule. I can easily devote around **30-40 hours per week**. Therefore, I am applying for a project duration of 350 hours, as I am confident to meet the project's requirements within this timeframe. I'd also inform you about any unexpected engagements (if any) and will ensure to compensate for the missed time by working extra hours in the preceding and succeeding weeks, maintaining progress and meeting project deadlines effectively.

I am flexible with working hours but prefer to work during office hours, i.e. **(9:00 - 14:00)** and **(16:30 - 19:30) UTC +5:30**.

Post-GSoC plans and Community Engagement

It is highly unlikely that any aspects of the project will remain unimplemented within the 12-week timeframe allotted for GSoC. However, in the rare-event that this does occur, I am fully committed to completing any remaining tasks during the extended timeline period.

Throughout my participation in the GSoC program, I will actively contribute to StatWrap and UC OSPO, engaging in community discussions. Having been an active member of the project for over 2 months now, I've developed an understanding of the philosophies and community values found to be of principal importance here. Furthermore, I am eager to extend my contributions to other open-source projects within various Open source

programs organized by OSPO. In the future, I also aspire to serve as a mentor and be on the other side of this journey and support aspiring open source contributors. Looking forward to building lasting connections with its members.

Biographical Information

Relevant Experience

I have 2+ years of hands-on experience developing software using **Javascript** and **React**. Additionally, I've learned quite a bit about ElectronJS recently, after I started to work with StatWrap.

I am a responsible individual and a great team player and believe in the utmost importance of active communication. As for my experiences, I am serving as the coordinator in the tech team for [Codefest](#), the Annual Coding Fest organized by Department of Computer Science and Engineering, IIT BHU, and thus know about working in large groups to achieve collaborative success. I am gold medalist in the recent **Inter IIT Tech Meet 12.0**, in the software development event, **drone software challenge**, by the Ministry of **Panchayati Raj**.

Projects:

Here are the projects we undertook during the Inter IIT selection phase and the main event:

- **GramSim** (Inter IIT 12.0 Gold Winning Project)

Deployed Link : <https://gramsim.vercel.app/>

Github Links: [Frontend](#) [Backend](#)

Project Presentation Link : [Solution Presentation](#)

TechStack: **Typescript, Python, React.js, Three.js**

Description:

- **Approach:** Utilizes a 3D simulator developed with [three.js](#) for immersive rural development planning.
- **Zero-cost:** Uses [publicly available or open-source resources](#), providing access to such planning tools for resource-constrained Gram Panchayats (Village Councils).
- **Cross-platform:** [Web-based](#) platform accessible from any browser, promoting inclusivity and flexibility in diverse rural settings.
- **Dynamic Data Integration:** Dynamic data, for [real-time feedback](#) on financial implications and optimizing resource allocation.
- **Happiness Index:** [Quantifies](#) overall village [well-being](#) through a collective Happiness Index based on [facility proximity and type](#).

- **TravelPlanner** (Collaborative Trip Planning Project)

Github Link: <https://github.com/AdiAkhileshSingh15/travel-planner>

Video Demonstration Link: [Project Demo](#)

TechStack: **Javascript, ReactNative, Express.js, Node.js**

Description:

- **Communities and Trips:** Users can create, join, leave, and delete communities, as well as organize and join trips within these communities.
- **Zero-cost:** Utilizes no-cost solutions for database, deployment, map-services, map data, flight data, and train data APIs.
- **Emergency SOS:** Enables users to send emergency alerts to nearby security offices, trips and community members, based on real-time location.
- **Price-prediction:** Predicts trip prices using historical and real-time data from flight and train data APIs.

There are several other projects that I have been involved within my institute, which are listed below:

Project	Contributions
Codefest	PRs
Training & Placement Portal	PRs
LatexGenerator	PRs
Arithmetica	PRs





I have over an year experience in open-source, some of the projects I've worked on, are listed below:


Project	Contributions
Konveyor	PRs
Kyverno	PRs

Over the past two years, I've developed and worked on numerous personal and team projects, many of which are also present on my [github profile](#).

Involvement with UC OSPO and StatWrap:

I've been actively involved with UC OSPO and StatWrap for over the past 2 months, and have made many contributions to the project. Below is a summary of my involvement, including engaged issues and submitted pull requests:

- **Pull Request #181:** [Updates few project dependencies](#) : Issue Linked: [#171](#):
 - This PR updates the existing project dependencies to their latest LTS version. It involves upgradation of 17 devDependencies (major and minor) and 3 major dependencies. With end-to-end testing and checks conducted ensuring the overall functionality.
- **Pull Request #184:** [Updates local development documentation for errors](#) :
 - This PR updates the development documentation, to provide solutions to potential errors faced while developing locally.
- **Pull Request #186:**  [Fixes dependency tree-view](#) : Issue Linked: [#182](#) :
 - This PR enhances the Dependency Tree View appearance, and renders it free from unwanted UI glitches, ensuring that the tree skeleton retracts and opens up smoothly. Additionally, it provides other features such as zooming and moving the dependency tree structure.
- **Pull Request #189:** [\[PACKAGE\]](#) : [Updates !\[\]\(d7a34a706cfa4ef37c62a369101e1b36_img.jpg\) project dependencies !\[\]\(7325769475e8f4bf67f57a0cbebc8ab9_img.jpg\)](#) : Issue Linked: [#171](#):
 - This PR updates the existing project dependencies to their latest LTS version. It involves upgradation of 20+ devDependencies and 10 dependencies (major and minor). With end-to-end testing and checks conducted ensuring the overall functionality.
- **Pull Request #193:** [\[FEATURE\]](#) : [Introduces a component to display the endpoints of the project.](#)  : Issue Linked: [#192](#) :
 - This PR introduces a feature to display the list of project's entry points. Additionally, it includes functionality to navigate directly to a selected endpoint node in the asset tree. And, I've also implemented **unit testing** for an asset util method that was created.
- **Pull Request #197:** [\[FIX\]](#) : [Resolves ESLint Failures](#)  : Issue Linked: [#169](#) :
 - This PR focuses on resolving ESLint issues by adjusting the configuration, downgrading the `eslint-formatter-pretty` package, and aligning files with ESLint rules. Additionally, it refactors unit tests to adhere to the updated code standards. With end-to-end testing and checks conducted ensuring the overall functionality.
- **Pull Request #198:** [\[FEATURE\]](#) : [Introduces feature to add notes from the notes page](#)  : Issue Linked [#140](#) :
 - This PR introduces a feature enabling users to add project notes directly from the notes page interface, enhancing user convenience. It leverages the existing notes editor for input and manages update and delete functionalities using project notes handler functions.
- **Issue #171:** [Refactor : Update Project Dependencies](#) :

- I've actively contributed to resolving this issue of updating the project dependencies, successfully completing two PRs as mentioned earlier, upgrading a total of around 40 devDependencies and 13 dependencies, out of approximately 150 dependencies listed in the issue.
- **Issue #182:** [Dependency tree branch once opened does not retract back on closing](#) :
 - This issue that I opened, and myself undertook the task of addressing it, as mentioned in the PR related to tree-view. Additionally, it involved raising an [issue](#) in the source repository of a package utilized by StatWrap.
- **Issue #192:** [\[FEATURE\] : Component to display the list of endpoints of the project.](#)  :
 - This issue deals with a feature enhancement of listing the project's entry points, and is addressed by the PR mentioned earlier.
- **Issue #169:** [Eslint failing for pre commit hooks](#) :
 - I've contributed to resolving this issue of failure of ESLint, as mentioned in the PR related to ESLint above.
- **Issue #140:** [Add notes from Notes page](#) :
 - I've contributed to introducing this feature of adding project type notes from the notes page interface itself.

While working on this project, I've gained valuable insights and actively participated in discussions, offering solutions to challenges. Despite joining the project relatively late, I've committed myself to making substantial contributions to StatWrap, enhancing it as a project overall. I've tried my level best to operate efficiently within an open source community project, drawing from my experience in team environments, prioritizing communication.

Educational Background

I am an undergraduate student at IIT Varanasi, which is one of the most prominent institutes in India, pursuing an Integrated Dual Degree course (Bachelor of Technology + Master of Technology) in Computer Science and Engineering.

Major Course Work:

Introduction to Programming	Algorithms
Data Structures	Operating Systems
Bash Scripting & Python	Artificial Intelligence
MySQL & Django	Intelligent Computing
Computer Graphics	Computer System Organization
Database Management System	Software Engineering

Theory of Computation	Computer Architecture
Discrete Maths	Operations Research

Technical Interests

I was first introduced to the world of programming in my school years. Since then, I have explored various fields such as Web Development, Data Structures and Algorithms, Operating Systems. Along the way, I've gained hands-on experience with a diverse range of technologies and languages, which are listed below based on my expertise.

Beginner	Intermediate	Advanced
Docker, AWS, Firebase	Django, GoLang, Next.js	HTML5, CSS, Javascript , Typescript, Python, C++, React.js , Node.js , MongoDB, MySQL, PostgreSQL

- Technologies relevant to this project.

Contact Information

Name <i>Adi Akhilesh Singh</i>
Email adiakhilesh.singh.cse21@itbhu.ac.in
Github AdiAkhileshSingh15
LinkedIn adi05022003
Slack <i>Adi Akhilesh Singh</i>
Phone (+91) 9305717073
Resume Link
University <i>Indian Institute of Technology (BHU), Varanasi</i>
Major <i>Computer Science and Engineering</i>
Degree <i>IDD (B.Tech.+ M.Tech.)</i>
Date of Enrollment <i>November 2021</i>
Expected Graduation Date <i>July 2026</i>
Postal Address <i>33/11A/2, Dariyabad, Prayagraj, UP (211003), India</i>
Nationality <i>Indian</i>
Timezone <i>Indian Standard time (UTC +5:30)</i>