# Assurance Recipes Application

DESIGN DOCUMENT

**Group Name:** sdmay20-26
**Advisor and Client:** Myra Cohen
**Group Members:**
Matthew Smith: Meeting Facilitator
Garrett Harkness: Scribe
Kevan Patel: Report Manager
Qiwei Li: Chief Engineer
Shiwei Wang: Test Engineer
**Team Email:** sdmay20-26@iastate.edu
**Team Website:** https://sdmay20-26.sd.ece.iastate.edu/
**Revised:** 4/26/2020

# Executive Summary

## Engineering Standards and Design Practices

In our group and with our faculty advisor, we decided to work under an agile development process. We all meet as a group once a week to work on different things we need to do, as well as another meeting with our advisor once a week as well. One to two weekly sprints are created, with two major milestones in each semester. Using Trello, we plan on keeping track of stories, hours, and progress throughout the semester. We also use Gitlab for our version control so that we can update our latest code and keep track of commits and pushes. All communication is used via Slack and email for Team and Advisor communication.

We also have been running under a TDD (Test Driven Development) environment, we used CI/CD to implement this. CI/CD is a method to frequently deliver apps to customers by introducing automation into the stages of app development. The main concepts attributed to CI/CD are continuous integration, continuous delivery, and continuous deployment. CI/CD is a solution to the problems integrating new code can cause for development and operations teams. This improves our code quality and reduces bugs, along with reduction of time spent on rework of bugged code. We also want to have effective teamwork among our group. Because is only a whole year to implement a new architecture on an existing system, along with other course work, we want to allow optimal productivity in a limited amount of time. We also can get differing perspectives and feedback from each other. Good documentation is also extremely key when developing. Good open API documentation allows the project to be handed off to future developers, which this project will be, so they have an easy time understanding the code.

## Summary of Requirements

Functional Requirements:

- Creation and editing of safety case diagrams.
- Creating safety cases based off of a template.
- Free text editing on the diagrams.
- Description page of safety cases.
- Importing and exporting of safety cases on local machine.
- Opening an existing safety case.

Nonfunctional Requirements:

- Performance: Running the application in a short response time.
- Scalability: Users being able to add assurance cases.
- Security: Saving and creating in a secure environment. No sensitive information being leaked.
- Reliability: Safe stable software
- Usability: Easy to use and understandable interface for users not well versed in a technical background.
- Availability: An app that is available for those who need to use it.

## Applicable Courses from Iowa State University Curriculum

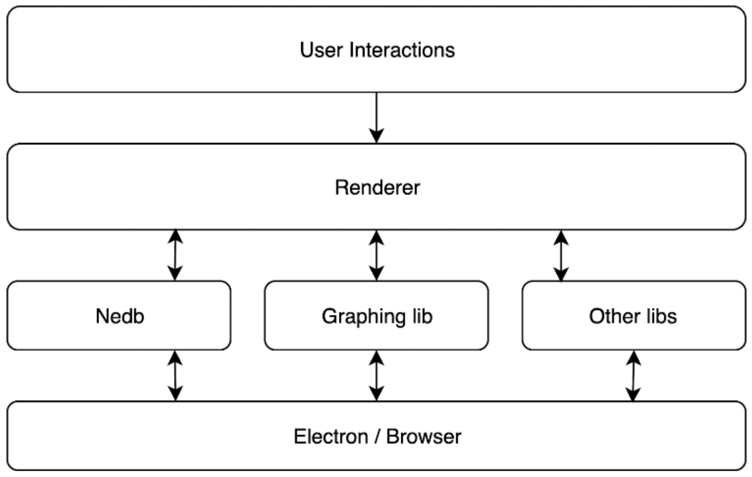
- Computer Science 309 – Software Development Practices: A practical intro to managing software development. Process models, requirements analysis, structured and object-oriented design, coding, testing, maintenance, cost and schedule estimation, metrics. Programming projects.
- Computer Science 319 – Construction of User Interfaces: Overview of user interface design. Evaluation and testing of user interfaces. Review of principles of object orientation, object-oriented design and analysis using UML in the context of user interface design. Developing Web and Windows-based user-interfaces.
- Computer Science 363 – Introduction to Database Management Systems: Relational, object-oriented, semi structured and query languages. SQL, XML, and NO-SQL. Database design using entity-relationship model, data dependencies, and relational database design. Application development in SQL-like languages and general-purpose host languages. Web application development. Programming projects.
- Software Engineering 329 - Software Project Management: Process-based software development. Capability Maturity Model (CMM). Project planning, cost estimation, and scheduling. Project management tools. Factors influencing productivity and success. Productivity metrics. Analysis of options and risks. Version control and configuration management. Inspections and reviews. Managing the testing process. Software quality metrics. Modern software engineering techniques and practices.
- Software Engineering 339 - Software Architecture and Design. Modeling and design of software at the architectural level. Architectural styles. Basics of model-driven architecture. Object-oriented design and analysis. Iterative development and unified process. Design patterns. Design by contract. Component based design. Product families. Measurement theory and appropriate use of metrics in design. Designing for qualities such as performance, safety, security, reliability, reusability, etc. Analysis and evaluation of software architectures. Introduction to architecture definition languages. Basics of software evolution, reengineering, and reverse engineering. Case studies. Introduction to distributed system software.

## New Skills/Knowledge acquired that was not taught in courses

Working with a faculty advisor and client face to face is one of the major differences not learned in other courses. Senior Design allows you to have a bit more freedom in how you want to design your project and how you want to communicate with your clients. This freedom gives you a free reign of any technology, software, development practices, etc. which allows us to really create the projects the way we want to within the client's limitations. That being said, you also have to address some client restraints as well to make sure you are achieving the goal of what the clients want as well. This is a very valid and applicable skill that will be used in real life when working with users and clients on large scale projects at companies. Many of the skills learned in this course may not be on a technical level, other than possibly learning some new languages, but more learning things on a one on one communication level with our client and advisor. We are able to apply skills learned in courses we have taken in the past, such as generating requirements by working closely with our client/advisor, as well as gaining practice with utilizing an Agile Development process. All but one of our developers have worked with Electron, so four of us will be new to working with this desktop application creation tool.

## Table of Contents

## List of figures/tables/symbols/definitions (This should be the similar to the project plan)



| Node | Explanation |
|---|---|
| User Interactions | Handle user interactions with GUI |
| Renderer | Render the GUI whenever the underlying dependency changed |
| Nedb | Database we used for the project |
| Other libs | Other libraries we might use in future |
| Electron / Browser | The nodes that handle rendering and logic |

**Figure 1. System design diagram**

**Table 1. Nodes explanations**

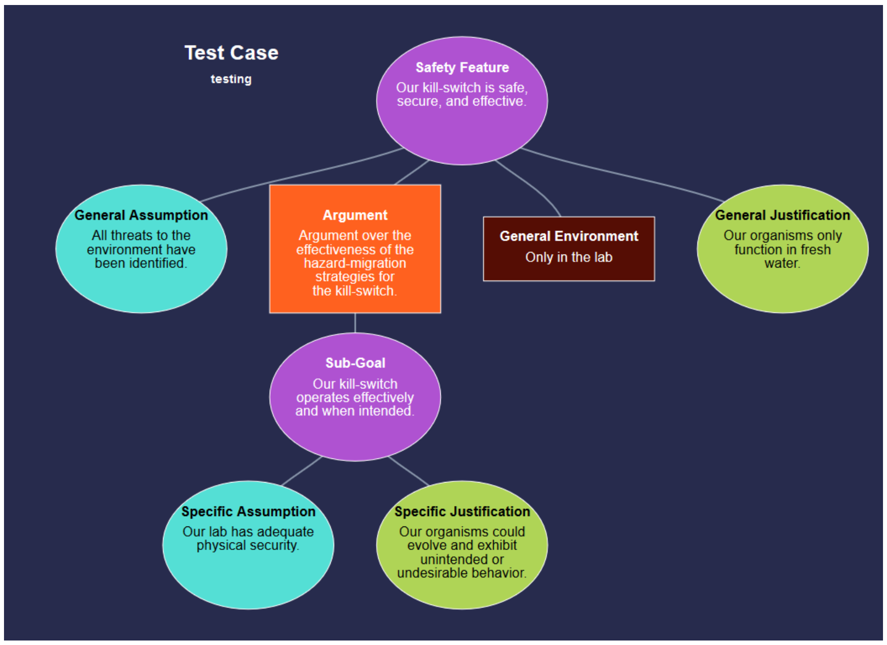

Test Case
testing
Safety Feature
Our kill-switch is safe, secure, and effective.
General Assumption
All threats to the environment have been identified.
Argument
Argument over the effectiveness of the hazard-migration strategies for the kill-switch.
General Environment
Only in the lab
General Justification
Our organisms only function in fresh water.
Sub-Goal
Our kill-switch operates effectively and when intended.
Specific Assumption
Our lab has adequate physical security.
Specific Justification
Our organisms could evolve and exhibit unintended or undesirable behavior.

**Figure 2 - A general assurance case from previous work**

**Table 2 - Table for use case diagram**

| Use Case | Users | Description |
|---|---|---|
| Create Assurance Recipe | General User | The user should be able to create an assurance recipe either from a template or from scratch. |
| Export Template | General User | The user should be able to export their template to their local computer so they can save it for a later time, transfer between machines, or share with another user. |
| Upload Template | General User, Admin | User and admin should be able to upload templates that are saved locally. The one the admin would upload will be available to all users through the open source application. |
| Delete Template | Admin | Admins should be able to delete templates that they upload, but not a user's template. |
| Add Node | General User | The user should be able to add any type of node they want to the assurance recipe. |
| Add Description | General User | The user should be able to add a description to the node. |
| Edit Description | General User | The user should be able to edit the description to a node after adding one, this would include deleting the description or the node. |

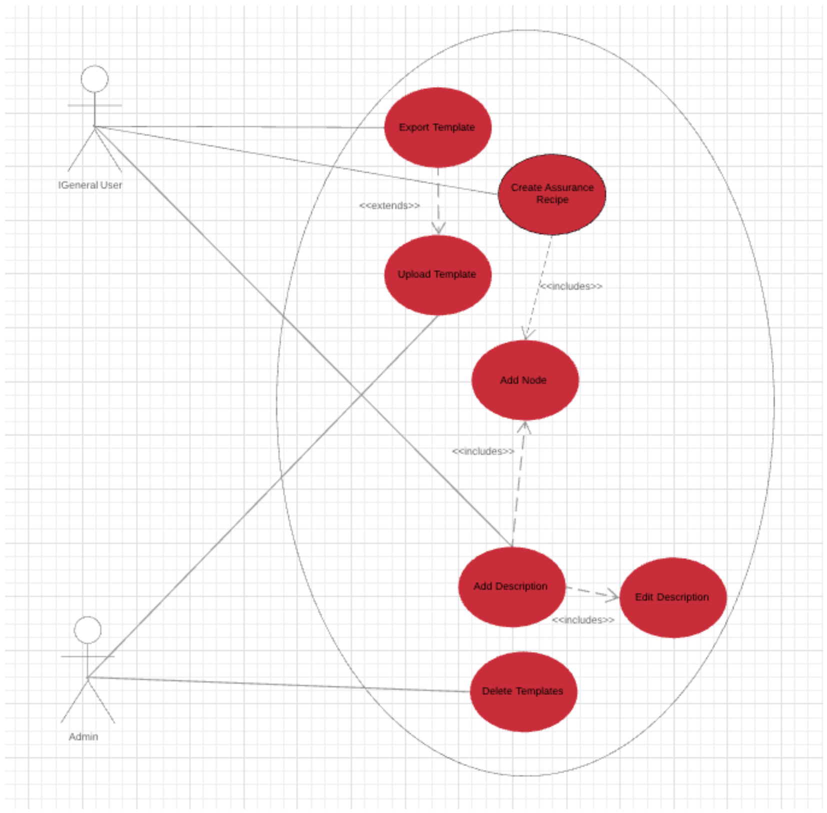


**Figure 3 - Use case diagram**

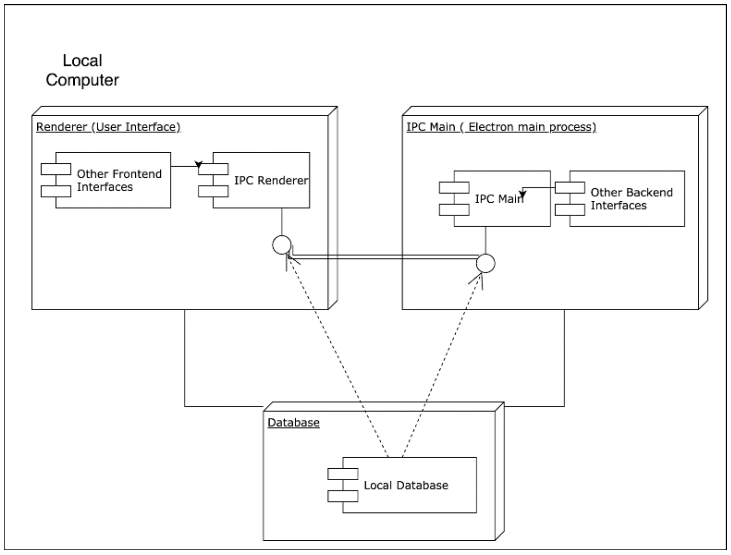

Local Computer
Renderer (User Interface)
Other Frontend Interfaces
IPC Renderer
IPC Main ( Electron main process)
IPC Main
Other Backend Interfaces
Database
Local Database

**Figure 4 - Communication diagram**

| Component | Explanation |
|---|---|
| Renderer | Render the GUI based on the model. This is the place where user can do the interaction with the user interface. |
| IPC Main | This is the place where the Electron logic happens. This will handle the logic between user computer and renderer. It will send some signal when user click on the Electron widgets and the renderer will render the page based on the signal. |
| Database | This is the place where user's data will be stored. This will communicate to both IPC Main and renderer. Every user's data will be stored locally and only be communicated with IPC Main and renderer. |

**Table 3 - Communication diagram explanation**

# 1 Introduction

## 1.1 Acknowledgement

We would like to acknowledge Myra Cohen for being our faculty advisor and our client during this entire process. She has been an extremely huge help in guiding us in our project and helping us come up with how we want to implement this project. We also want to acknowledge Justin Firestone, Myra's grad student who helped in our development and helped write some of the project literatures on Assurance Cases. We also want to thank Mohammed Gesalla, our TA who went over our design document with us in the first semester and helped us in our editing process. Lastly, we would like to thank Mark Hernandez who wrote all of the original code in the original assurance case website. He provided a starting point for us to go off of and allowed us to see how he created diagrams and what technologies he used.

## 1.2 Problem and Project Statement

Our project is the Assurance Recipes application. These recipes involve the use of synthetic biology to genetically modify organisms like E. Coli to help daily routines. Many people want to be certain that this new and exciting opportunity will be safe for both them and the community. That is where assurance cases come in. They ensure the safety of various parts of their experiments and map out their design structure in an efficient way. Safety Cases take their design strategy from the aeronautics and software engineering communities where they can also be seen under the title Assurance Arguments using Goal Structuring Notation (GSN). There they are used to ensure the safety of various parts of the aircraft and target certain problem areas in the functions and dangers of the process of flight. Unlike aeronautics, synthetic biologists do not have to worry about engine and wing design or console displays, but they do have to worry about accidental release of bacteria and plasmid conjugation as well as other concerns. As synthetic biology grows to new heights and levels of complexity, the number of safety concerns a single project or application needs to address will also grow. Just as people trust the engineering of an airplane despite the many risks, Safety Cases can help people who use genetically modified organisms feel confident that what they are using is safe. Our project will allow students and experts to create assurance cases easily and efficiently on an all new app and will help students out with how to create these assurance recipes. We want the application to be easily editable where users can create assurance cases easily. With a clean and highly functional UI, being able to edit and create assurance cases is the biggest thing we want to accomplish in this project. There is already a website created that accomplishes this, but the client wants us to make a better and more secure application for Iowa State biology students to easily use. The problem with the already implemented design is the fact that there is a database implemented that the client does not own, and it also is storing user information in a not safe way. Not only this, but the project in general was incomplete. It was meant as a prototype for the iGEM competition and was hosted on website that needs to be moved. So, the goal is to eventually release a brand new and improved application to the community.

## 1.3 Operational Environment

After discussion with our client/advisor, we have decided that we will develop an Electron application, which utilizes HTML, CSS, and JavaScript to run as a desktop application on the users' computer. There are no hazards, temperatures, or weather elements that we need to be aware of when creating this project because it is purely created using software. This also means that we do not need to consider having web security because it will be a local application. The previous application had problems with security and storing user's information on the database. We would not have problems with this using Electron because security for data stored locally would have to be on the user's end.

## 1.4 Requirements

Functional Requirements:

- Creation and editing of safety cases: Users should be able to freely edit and layout their safety cases the way they want to.
- Templated safety cases: Users should be able to choose from a list of templates since safety cases can be quite complex.
- Free text: Users should be able to type free text in each diagram creation.
- Description page of safety cases: We want a description page to describe what assurance recipes and cases are. We also want to describe the importance of them and how they are used in synthetic biology.
- Data safe: User's data should be stored safely. No private information should be leaked or have any security threats.
- Saving of safety cases: Users should be able to import and export their cases on their local machines.

Nonfunctional Requirements:

- Performance:
    - Users can run on multiple machines.
    - Short response time.
- Scalability:
    - Users are able to add assurance cases.
- Security:
    - Users, program's data, and ports are protected.
    - Follows EU GDPR regulations (European Union General Data Protection Regulation).
- Reliability:
    - Provide a stable, safe software.
- Usability:
    - An easy interface for users to learn and understand: Effective, intuitiveness, low perceived workload.
- Availability:
    - Testability
    - Manpower
    - Detailed diagnostic procedures.

## 1.5 Intended Users and Uses

The intended users for this project are biology students who plan on making assurance recipes in their synthetic biology courses. The application will allow them to easily create these diagrams on their own and from templates so that they can make sure they are running safe and well-designed experiments. These synthetic biology students also participate in a competition called iGEM. iGEM is the International Genetically Engineered Machine competition where worldwide undergraduate synthetic biology students compete and work on projects together. Our assurance cases application will be used by these students so that they can lay out their safety cases and create diagrams for their experiments. We also plan on expanding it further towards real world synthetic biologists and researchers so that they have a well-designed tool to use to create Assurance Cases.

## 1.6 Assumptions and Limitations

Assumptions:

1. We have to assume that the users are non-technical people. This means that the app we are creating has to be easy to use and navigate for them to create the assurance cases. It also should be easy for them to run on their machines without too many requirements. It would be helpful to include a how to run and how to use guide or read me file when someone downloads the application.
2. Because our application is a desktop application, we have to assume that the user does not have a large amount of space on their hard drive. We have to assume that their specifications are not very good so this application can run on a variety of machines, no matter how bad they are.
3. Assume the user does not completely know what an assurance recipe is or how to make a complete one. To solve this problem, we will make a template and a description of how to make a thorough assurance recipe.

Limitations:

1. Our biggest limitation is a time constraint. Many of us are busy with work and other time-consuming course work. Many times, it is difficult to find a time where all five of us can meet together to work. Because of this, all of us can only meet together mostly on Sundays. In the following semester many of us are taking lighter courses so meeting for working on our sprints should not be as much of a problem.
2. We found that another limitation was due to COVID-19. Many of us had other classes to worry about and we were all suffering during a weird time like quarantine. This put a damper on how we would all work together.

## 1.7 Expected End Product and Deliverables

1. Easy to use interface. In saying this we want to deliver an application that can be used by anybody who is not specifically well versed in the technical field. We want users to be able to create assurance cases intuitively and efficiently. Since the application will be mostly used by people well versed in biology, it will be essential for us to create an organized and clean UI that allows easy diagram creation.
2. We want the user to be able to choose from a list of already templated out diagrams. Since Assurance Recipes can get quite complex and specific in their notation, we want users to be able to choose from a list of already templated and layered out assurance case diagrams. After they choose, they are freely allowed to edit the diagram as they wish so they can actually fill out the assurance diagrams specific to their experiments.
3. Free text editing of each part of the diagram.
4. Securely store the user's data. Although the users don't enter any personal information, we do want to allow their diagrams to be safely stored on their local drives.
5. Easy to install on every machine without too much pre-install requirements. We want the application to not require much installing for them to begin creating safety cases. The way we have it set up right now is that they will have a simple executable file that they can run on any machine and right then they can begin creating safety cases on their device.
6. Available on any platform (Linux, Windows, MacOS).
7. Easy to install on any platform without too much configuration.

# 2. Specifications and Analysis

## 2.1 Proposed Design

### *2.1.1 Electron*

Electron is a framework for creating native applications with web technologies like JavaScript, HTML, and CSS. And it works cross-platform which means you can deploy your application anywhere without contains.

### *2.1.2 Typescript*

TypeScript is an open-source programming language developed and maintained by Microsoft. It is a strict syntactical superset of JavaScript and allow you adding type for both variable and function. TypeScript is designed for development of large applications and trans compiles to JavaScript. Typescript will do the type checking during runtime so that it will add some type hints and checking during the development. And make it easier to cooperate with others.

### *2.1.3 ReactJS*

React is a JavaScript library for building user interfaces. React can be used as a base in the development of single-page or mobile applications, as it is optimal for fetching rapidly changing data that needs to be recorded. React can break one single application into different components so that we can reuse the components as we need. Also react is using declarative syntax of building user interfaces it will rebuild the components whenever the underlying data has changed. This will be useful when we are building a complex user interface where many components will interact with each other.

### *2.1.4 Nedb*

Nedb is an in browser JavaScript database which implements a subset of MongoDB operations. Nedb itself will store the data in browser by using the indexedDB API provided by modern browser.

### *2.1.5 KonvaJS*

KonvaJS is a 2d canvas library for desktop and mobile applications. We are using this library for graphing. By using this library, we can easily draw nodes and lines.

### *2.1.6 Jest and React-Testing-Library*

Jest is a delightful JavaScript Testing Framework with a focus on simplicity. It works with projects using: Babel, TypeScript, Node, and React. React Testing Library is a very lightweight solution for testing React components. It provides light utility functions on top of react-dom and react-dom/test-tills which can let developers to test GUI rendering without actually render the widget. This will accelerate the testing speed on a large-scale application.

### *2.1.7 Docker*

Docker is a set of platforms as a service (PaaS) products that use OS-level virtualization to deliver software in packages called containers. Containers are isolated from one another and bundle their own software, libraries and configuration files. Docker is used for deployment and testing. By using Docker, we can eliminate the differences between each computer so that it will provide stable test results. Because the implementation of Docker, it is faster than regular VM.

#### *2.1.7 Code-Gen*

We are using Code-Gen technology for our documentation page. This allows us to write documents in Markdown and automatically compile Markdown to typescript file.

### 2.2 Design Analysis

We were given an existing web app that implements assurance cases and our primary objective with the project is to redesign the architecture and overall application. We knew early on from experience that we wanted to utilize NodeJS and Typescript, due to safe typing and making JavaScript a more structured language.

Early discussions took place over whether we wanted to do a web app or potentially develop some sort of desktop application instead. We decided to utilize the Electron framework, due to its ability to develop cross platform desktop applications utilizing web technologies. A potential advantage we found when discussing this was that if Electron did not end up working out for what we wished to accomplish, the codebase we had will be easily migrated to a web application. We also noted that by developing a desktop application, opposed to a web application, meant that database and the additional security that comes with it will not be something we have to worry about. For example, certain specifications must be met for applications utilizing databases in the EU, but we are only utilizing local storage, therefore bypassing this issue.

The second major decision was what we wanted to use to develop the UI. Some of us had previous experience utilizing ReactJS. By analyzing Assurance Cases, we realized the ReactJS would work well with developing the UI portion of the assurance cases due to React emphasizing reusable components and assurance case diagrams are composed of many different components, many of which are reused across different types of assurance cases. After agreeing to utilize React, it added the issue of finding a library for drawing shapes that was compatible with React.

After looking at many different libraries to aid in the drawing of our diagrams, we currently have selected KonvaJs as our library of choice. We decided on Konva for a number of reasons, with the first being that Konva works very well with ReactJS. The second major criteria we had for selecting the library was to have it implemented in a way that had a lot of quality of life functionality, for us, built in, such as click and drag functionality and the ability to easily draw lines between nodes in the assurance case diagram.

### 2.3 Development Process

We are using the Agile development process. We plan to have 2 major milestones in the fall semester. We also plan on having weekly and bi-weekly sprints depending on how large the tasks are, as well as using Trello to keep track of these sprints or any issues we encounter. We wanted to use Agile because we all have experience working in an agile environment and because of how efficient it is to use, and it was suggested by our faculty advisor Myra. Agile is very organized and set process to follow which allows you to run a test-driven development throughout our project time. Using sprints, you can easily lay out the status of your projects and allows you to only take work that you can handle at a given time, instead of having constant ongoing story cards going on at once. We can tunnel in our focus together as a team on one to two tasks as a time to output production at the highest efficiency.

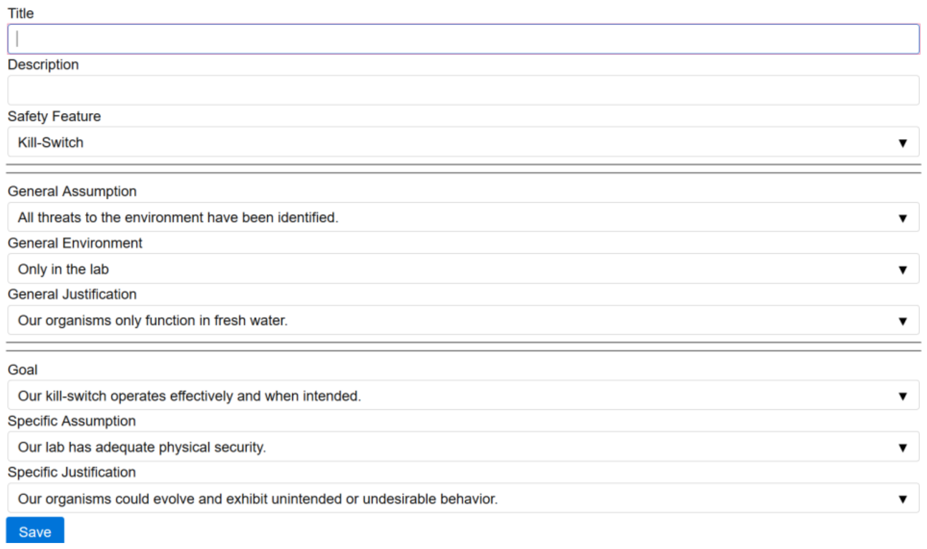
### 2.4 DESIGN PLAN

Our entire application will be built as an electron application and run as a desktop application. Our code will be written in Typescript and our User Interface will be built utilizing React components. Our development environments will be contained using Docker and all of our machines are currently able to participate in development. The way we will design our UI will adhere to the ReactJS philosophy of considering all aspects of the UI as components. For the diagrams themselves, the nodes in the diagrams will be React components that act as wrappers around shapes drawn using the KonvaJS library. By designing our application with the React mindset, we are able to maximize the amount of reuse among our different assurance cases. For example, in assurance recipes, justification nodes are used across almost all assurance recipes, and by designing our application using React, we are able to maximize code reuse and understandability.

# 3. Statement of Work

### 3.1 PREVIOUS WORK AND LITERATURE

We will be modeling the look of our charts based on some existing software that is available. We have been provided literature by our client/advisor regarding the relevant background knowledge needed for understanding the basics of Assurance Recipes. Assurance recipes are a lower level pattern used as an abstraction for Assurance Cases. According to paper, assurance cases can sometimes be too complex to individuals who are not as experienced in the fields of software engineering or assurance cases. In the case that a user lacks experience, these cases can sometimes provide too many degrees of freedom. The aim of assurance recipes is to provide the user a guided selection of what Myra Cohen and Justin Firestone called ingredients in the literature, and these ingredients can be filled in by the user as they traverse the assurance case.


Title

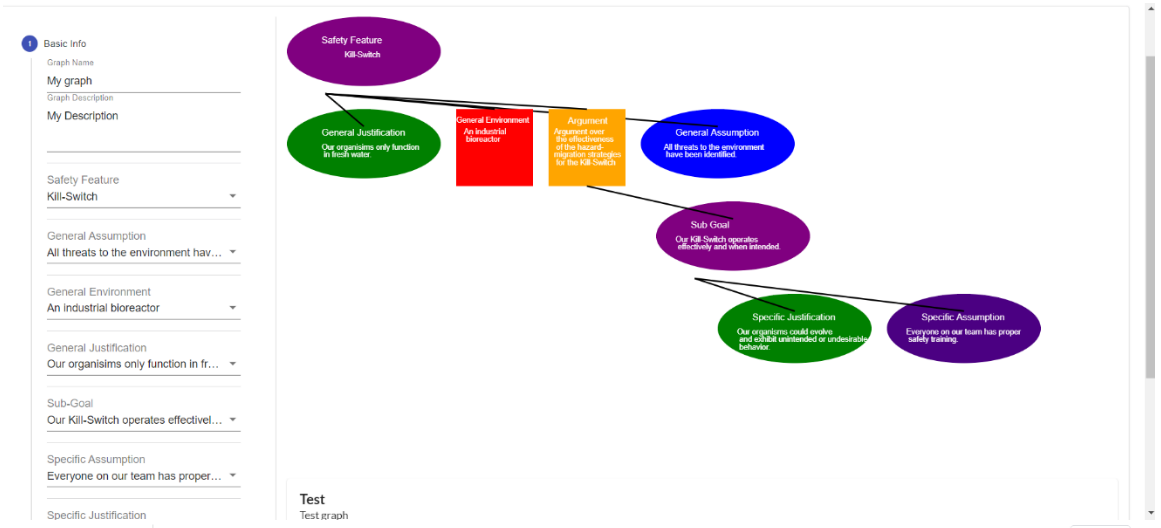
Description

Safety Feature
Kill-Switch

General Assumption
All threats to the environment have been identified.
General Environment
Only in the lab
General Justification
Our organisms only function in fresh water.

Goal
Our kill-switch operates effectively and when intended.
Specific Assumption
Our lab has adequate physical security.
Specific Justification
Our organisms could evolve and exhibit unintended or undesirable behavior.

*Figure 3.1.1 – The existing project's creation template for an assurance recipe*


Safety Feature
Kill-Switch
General Justification
Our organisims only function in fresh water.
General Environment
An industrial bioreactor
Argument
Argument over the effectiveness of the hazard-migration strategies for the Kill-Switch
General Assumption
All threats to the environment have been identified.
Sub Goal
Our Kill-Switch operates effectively and when intended.
Specific Justification
Our organisms could evolve and exhibit unintended or undesirable behavior.
Specific Assumption
Everyone on our team has proper safety training.

1 Basic Info

Graph Name
My graph

Graph Description
My Description

Safety Feature
Kill-Switch

General Assumption
All threats to the environment hav…

General Environment
An industrial bioreactor

General Justification
Our organisims only function in fr…

Sub-Goal
Our Kill-Switch operates effectivel…

Specific Assumption
Everyone on our team has proper…

Specific Justification

Test
Test graph

*Figure 3.1.2 – Our creation template for an assurance recipe*

## 3.2 Technology Considerations

We decided early on that we wished to develop a NodeJS based application. After some discussion, we decided that developing an application that runs on the Electron framework would be more advantageous when compared to a standard web app. Using Electron, we would still be able to develop using NodeJS, but we would also not have to worry about a database and the required security for user credentials and logins. This decision was made after considering the necessary security measures that must be in place to comply with the standards set by the European Union.

## 3.3 Task Decomposition

| **Name** | **Role** | **General Task Description** |
|---|---|---|
| Matthew Smith | Meeting Facilitator | Took on the role of assigning and making meetings. Stayed in contact with Myra who is our advisor and our client and was the one who arranged meetings and took into account who could show up for a given day. Also made sure our group met on our own at least once a week and worked on Lightning Talks, Design Document, code, etc. Also worked on organizing our schedule, setting up Trello board, and our website so that it is presentable. |
| Kevan Patel | Report Manager | Helping researching libraries and open source work we could use for project. CI/CD implementation for the project. Writing documentation and creating charts and graphs for documentation. Worked on Lightning Talks and setting up Trello issues as well. |
| Garrett Harkness | Test Engineer | Contributed to assignments, discussions on choices of libraries, and development done so far. Developed test cases to test the |

| | | |
|---|---|---|
| | | application. |
| Qiwei Li | Chief Engineer | Design the software and also setup the project structure. Write the documentation for the project. Choice libraries for project and implement the initial design. Also write the Docker file for testing and development. Also make a video of the project Q & A section. |
| Shiwei Wang | Scribe | Set up test case, look for hardware and software test. Decided how to test the project, make a plan on how to test module, how to test UI, how to test implemented library. Check out test result with members. |

*Table 3.3.1 – Task Description*

## 3.4 Possible Risks and Risk Management

Luckily, we haven't had to deal with too many risks in this project other than software architecture risk management. Our application does not require any hardware, materials, or equipment to run. The only thing we need to address is being careful in how we structure and run our architecture so that it can run quickly and efficiently. We also have the risk of any security threats or leaks and those need to be addressed as well. Since it is a desktop app, we aren't storing any user information and allowing the users to save their diagrams on their local drive so it becomes easy to address this risk as well.

## 3.5 Project Proposed Milestones and Evaluation Criteria

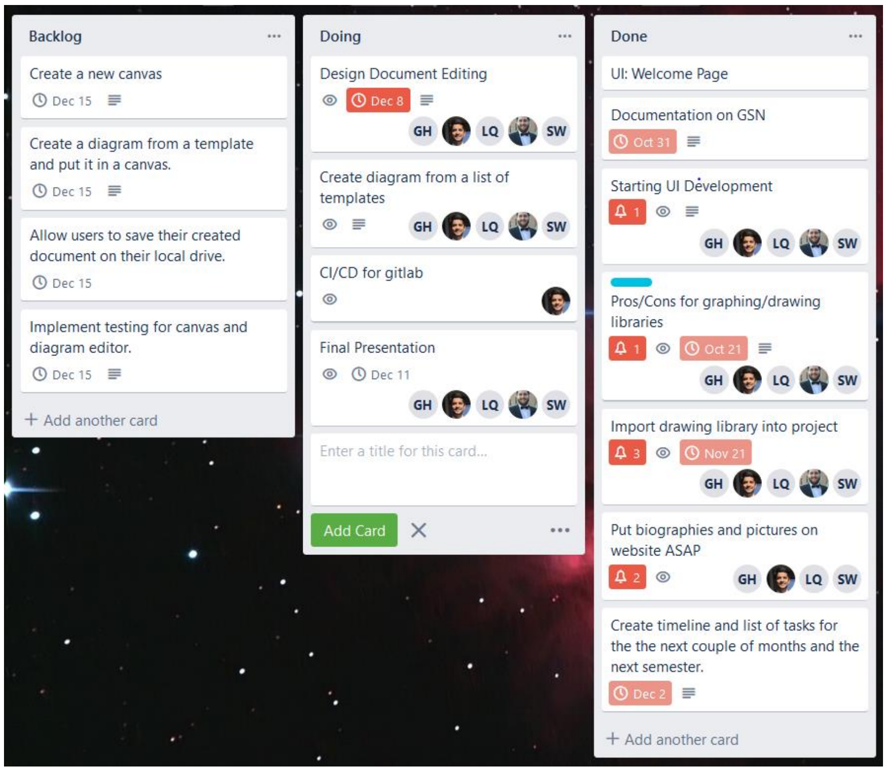
- Basic canvas that the user is able to create and use and assurance recipe.
- The user is able to edit the canvas further by adding and deleting nodes and descriptions.
- Implement Generalized GSN
- Generate Assurance Case Recipe from a provided template.
- Open existing assurance cases from the local drive and allow for saving to the local drive.

- Implement additional Assurance Recipe templates

## 3.6 Project Tracking Procedures

We have been using Trello to track project tasks and roles for any given week. We also meet weekly with our advisor and with ourselves to understand what we need to be getting done throughout the given week. Trello makes it easy because you can create story cards and put dates and statuses on them as you are working. You can also create story card descriptions which makes it easy to see who is working on what. It has an easy to use and organized interface which makes it easy to organize and plan our project timeline.

*Figure 3.7.1 – Trello Board*



**Backlog**

- Create a new canvas — Dec 15
- Create a diagram from a template and put it in a canvas. — Dec 15
- Allow users to save their created document on their local drive. — Dec 15
- Implement testing for canvas and diagram editor. — Dec 15
- + Add another card

**Doing**

- Design Document Editing — Dec 8 — GH, LQ, SW
- Create diagram from a list of templates — GH, LQ, SW
- CI/CD for gitlab
- Final Presentation — Dec 11 — GH, LQ, SW
- Enter a title for this card...
- Add Card

**Done**

- UI: Welcome Page
- Documentation on GSN — Oct 31
- Starting UI Development — 1 — GH, LQ, SW
- Pros/Cons for graphing/drawing libraries — 1 — Oct 21 — GH, LQ, SW
- Import drawing library into project — 3 — Nov 21 — GH, LQ, SW
- Put biographies and pictures on website ASAP — 2 — GH, LQ, SW
- Create timeline and list of tasks for the the next couple of months and the next semester. — Dec 2
- + Add another card

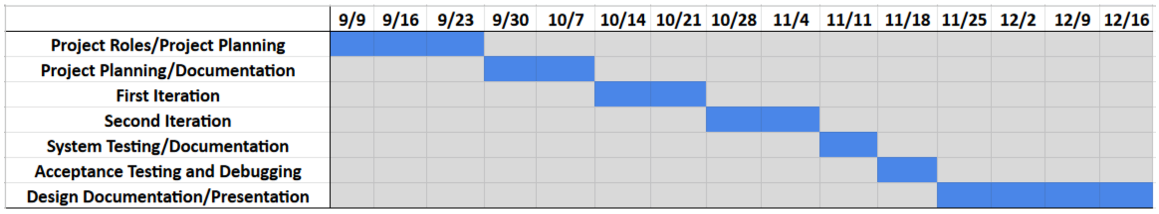
## 3.7 Expected Results and Validation

With our project, we plan to have an application that is widely available and easy to use for people working in the field of synthetic biology. Our project will be utilized by people who work with our advisor/client initially and then be utilized by more in the field of synthetic biology. Our advisor works in the field of synthetic biology and will be able to provide validation of progress over the course of the project.

## 4. Project Timeline, Estimated Resources, and Challenges

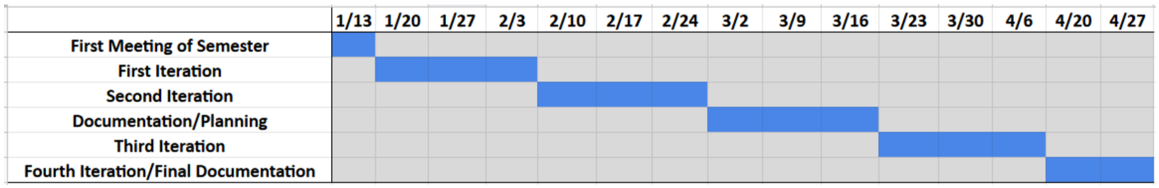
### 4.1 Project Timeline

1. Weekly meetings with our client/advisor every Monday, for some accidents, meeting time will be changed to other days.
2. We will have the basic skeleton of the project developed for our meeting on Nov. 4th.
3. We plan to meet as a group to work as we see necessary.
4. We implement library react at November.
5. Between November 11th and November 18th, we started to set every member's machine can run on early version of our software. Before thanksgiving break, November 25th, we make sure there were no issues on running software.
6. Between December 2th and December 8th, we start to edit our final version of design document, and make final presentation slides, and during that weekend, we will prepare the final presentation and distribute presentation part for each member.



| | 9/9 | 9/16 | 9/23 | 9/30 | 10/7 | 10/14 | 10/21 | 10/28 | 11/4 | 11/11 | 11/18 | 11/25 | 12/2 | 12/9 | 12/16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Project Roles/Project Planning | 🟦 | 🟦 | 🟦 | | | | | | | | | | | | |
| Project Planning/Documentation | | | | 🟦 | 🟦 | | | | | | | | | | |
| First Iteration | | | | | | 🟦 | 🟦 | | | | | | | | |
| Second Iteration | | | | | | | | 🟦 | 🟦 | | | | | | |
| System Testing/Documentation | | | | | | | | | | 🟦 | | | | | |
| Acceptance Testing and Debugging | | | | | | | | | | | 🟦 | | | | |
| Design Documentation/Presentation | | | | | | | | | | | | 🟦 | 🟦 | 🟦 | 🟦 |

*Figure 4.1.1 – First Semester Project Timeline Gantt*



| | 1/13 | 1/20 | 1/27 | 2/3 | 2/10 | 2/17 | 2/24 | 3/2 | 3/9 | 3/16 | 3/23 | 3/30 | 4/6 | 4/20 | 4/27 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| First Meeting of Semester | 🟦 | | | | | | | | | | | | | | |
| First Iteration | | 🟦 | 🟦 | 🟦 | | | | | | | | | | | |
| Second Iteration | | | | | 🟦 | 🟦 | 🟦 | | | | | | | | |
| Documentation/Planning | | | | | | | | 🟦 | 🟦 | 🟦 | | | | | |
| Third Iteration | | | | | | | | | | | 🟦 | 🟦 | 🟦 | | |
| Fourth Iteration/Final Documentation | | | | | | | | | | | | | | 🟦 | 🟦 |

*Figure 4.1.2 – Second Semester Project Timeline Gantt*

**Semester 1:**

Sprint 1- We plan to implement the basics of the Containment Recipe as well as basic implementations of the React components that can be used for other recipes.

Sprint 2 - Allow the ability to generate a Containment Recipe from a template. Allow for saving to a local drive and opening existing diagrams from a local drive.

Sprint 3 - Implement GSN: Safety Mechanism, Kill-Switch, and additional Recipes into the main graph page.

**Semester 2:**

Sprint 4 – Work on template addition into the project with proper GSN artifacts.

Sprint 5 – Recreating the introduction page.

Sprint 6 – Fix some of the bugs within template and graph page. Merge any code into final master branch.

## 4.2 Feasibility Assessment

The project will be a desktop application that allows the user to generate and edit Assurance Recipe diagrams. Since we used electron, so the software can be run on different operations, like Linux, Windows and MacOS with easy to use user interface and tutorials. The most difficult portion of the project that we are anticipating is the development related to generating the charts and also have them be aesthetically pleasing.

## 4.3 Personnel Effort Requirements

We would each spend around 8-14 hours per week working on development, assignments, documentation, and other aspects of the project. Some portions were done in pair-programming sessions, which we conducted via Discord once classes were moved to online instruction. Slack was also used for synchronous text communication and became increasingly important once courses were moved to online instruction.

There are two weekly assignments: weekly status reports and lightning talks. For weekly status reports, we had decided to make 6 reports, and we divide them equally and one member would two of them. The status reports usually record 10-14 days that include content about weekly meeting with advisor/client, weekly work process and weekly group meeting.

For lightning talks, we divided them into different parts to ask members to complete. Every time, we had different members to record voice for them and submit them to canvas.

For design document, we regularly updated the design document and periodically throughout the semester and spend time as a group looking at and updating it.

For programming, we have all contributed various portions of the codebase and to the project as a whole. What we each worked on varied by week, but we were all actively involved throughout both semesters.

## 4.4 Other Resource Requirements

The user will need to have around 300-400 megabytes of space available on their hard drive.

## 4.5 Financial Requirements

We have no financial requirements for this project. Everything we are using is open source and there has been nothing that we are creating that costs any money. We checked with our client Myra on this requirement and she has addressed that there are none.

# 5. Testing and Implementation

## 5.1 Interface Specifications

Currently, we have one model, which implemented the interface called Graph, need to be tested. This model includes everything we have so far for our business logic. There are two other data interfaces, which just defined the structure of the data, that doesn't need to be tested.

```
export interface NodeObj {
    x: number;
    y: number;
    width: number;
    height: number;
}

export interface GraphObj {
    _id?: string;
    name: string;
    description: string
    nodes: NodeObj[]
}


interface Graph{
    graphs: GraphObj[]
    selectedGraph?: GraphObj
    db: Nedb<GraphObj>
    addGraph(name: string, description: string):
Promise<GraphObj>
    addNode(node: NodeObj): Promise<void>
    deleteGraph(graph: GraphObj): Promise<void>
    selectGraph (graph: GraphObj): Promise<void>
    getAllGraph(): Promise<GraphObj[]>
 }
```

*Figure 5.1.1 – Code Interface for testing*

## 5.2 Hardware and Software

The most often used hardware is everyone's laptop or computer. Since our project is a desktop software, so a laptop or computer will be main hardware testing requirement. Therefore, our mainly developing working and testing are running on our laptop. Also, before weekly meeting, we will pull latest project version from Gitlab and run on our laptop or computer to make a brief testing.

The testing software is VS Code, since we are all doing developing at this software, so it's would be easy for us to test each part of feature and the whole project. It's important for us to have a same developing and testing tool, and VS Code fits our project pretty well and provide a convenient developing and testing environment.

## 5.3 Functional Testing

For this project, we divided the software into models and views by using React context api. This allows we separate business logic from the UI code. And we will use two testing techniques to test these separately. After we have testing files, we will use CI/CD tool like Travis CI or GitHub Actions to test our software automatically after each push request. We will also use docker for testing environment setup.

### Testing models

To test the correctness of model, we will adopt unit testing along with Jest test framework.  Each model will have at least one testing file associated with it. The goal of model testing should achieve at least 90% code coverage.

```
describe("Test utils", () => {
    const rsp: Result<Post> = { count: 1, results: [{ title: "Some
title", content: "1234" }] };
    (axios.get as jest.Mock).mockResolvedValue({ data: rsp })
    test("Test search function", async () => {
        let result = await searchPost("Hello")
        expect(result.count).toBe(1)
        expect(result.results).toBe(rsp.results)
    })


})
```

*Figure 5.3.1 – Example of model testing*

### Testing UIs

To test UI, we will use React-Testing-Library, a library which handles rendering widgets for widget testing. The purpose of using this library is that by using this library, we don't need to actually render the whole website when testing the UIs. This will accelerate our testing speed and simplify the testing process.

```
test("home page testing", () => {
    const postResult: Result<Post> = {
      count: 3,
      next: "sss",
      results: [{ title: "p1", content: "p1" }]
    };
    (axios.post as jest.Mock).mockResolvedValue({ data: {} });
    const context = {
      value: 0,
      searchWord: "",
      progress: 0,
      onChange: (newValue: number) => {},
      onSearch: (e: React.ChangeEvent<{}>) => {},
      fetch: () => {},
      fetchMore: () => {}
    };

    const tree = (
      <DisplayContext.Provider value={context}>
        <HomePage></HomePage>
      </DisplayContext.Provider>
    );
    const { container } = render(tree);
    // should display a progressbar
    const progressbar = container.querySelector("#progress-bar");
    const err = container.querySelector("#err-msg");
    const list = container.querySelector("#post-list");
    const btn = container.querySelector("#load-btn");
    expect(progressbar).toBeDefined();
    expect(err).toBeNull();
    expect(list).toBeNull();
    expect(btn).toBeNull();
  });
```

*Figure 5.3.2 – Example of UI testing*

## 5.4 Non-Functional Testing

**Performance:**

So far, based on our diagram testing and UI testing, both parts work well.

**Security:**

For this project, we do not worry about security because this project will have all data stored locally on the user's computer so security will have to be on the user's end.

**Usability:**

We installed VS Code and electron on laptop, there are some issues on first time running, some of us did not install the module and some of us had problems with operation environment. Once we fixed those problems, all of us have no problem with running the software. After each pull from Gitlab, the project does not have any runtime issues.

**Compatibility:**

There were some compatibility problems with Windows operation while we installed the software for the first time. But there is no problem installing and running on Linux and MacOS. After we fixed problem with Windows operation, the software is working well and no issues anymore.

## 5.5 Process

As discussed in Section 2.2, we have decided on KonvaJS due to its inclusion of functionality that provides many qualities of life advantages for us as developers. The drawing library did make it easy to establish and create nodes of our graphs, and allow us to allow create text inside them, and create edges off each of the graphs. It was a process to learn how it all worked and how it would function well with react, but it turned out there was a lot of good tutorials out there that helped us with what we needed to do. A lot of us also had to learn react in general. Qiwei, our lead engineer was the most proficient with React and was a huge help for the setup of this project and the programming.

## 5.6 Results

We got the chance to really clean up a project and give it a new fresh coat of paint. One of our favorite things about it is that as the user is editing, all of the changes they are making to it is updating in real time. A user can take a templated assurance case that they have already created and add nodes to it as well which wasn't as easy to do previously in the old application. We also fixed one of the biggest issues with the old application and that was storing user information. That was easily fixed by creating using electron instead. We were able to easily create a desktop application that no longer needed user information to function.

# 6. Closing Material

## 6.1 Conclusion

It has been amazing to be able to start from scratch and recreate a high-level application. It was exciting to have the chance to work on something that will be used by synthetic biology students and biologists in the future. In result, we were able to create an application with a professional introduction page that clearly laid out and taught people what they will be working with. We have a graph page, where users can freely edit new nodes of their graphs and edit the text inside them. We also have a template page, where users can create graphs off a template, and select text to go inside them so they have a clear example of what their graphs should look like. It has been challenging to learn new technologies and make something good enough for our client where they can give it to future students to use. It was also rewarding and taught us a great deal about React, NodeJS, KonvaJS, and nedDB. It also allowed us to learn about a whole new field in synthetic biology and what kind of graphs and things they must create and learn. We look forward to giving this app to be worked on in the future and look forward to the app being used by many people.

## 6.2 References

Firestone J., Cohen M.B. (2018) The Assurance Recipe: Facilitating Assurance Patterns. In: Gallina B., Skavhaug A., Schoitsch E., Bitsch F. (eds) Computer Safety, Reliability, and Security. SAFECOMP 2018. Lecture Notes in Computer Science, vol 11094. Springer, Cham.

## 6.3 Appendices

Electron - https://github.com/electron

ReactJS - https://reactjs.org

KonvaJS - https://konvajs.org

Jest - https://jestjs.io

Typescript - https://www.typescriptlang.org

Original Web Application - https://unl-igem-test.netlify.com/

### 6.3.1 Appendix 1: Operations Manual

Help Docs of Running the project

**Getting start**

To install the project on your machine, you have three options.

1. Install natively using Yarn
2. Install using docker (This docker doesn't contain the automatically run, in fact it contains a code server which you can write code directly in your browser)
3. Using VM (This actually is running docker inside the VM for windows user. If you are using unix system like MacOS, you can ignore this)

**Install using Docker**

(1) First, get docker

(2) Get docker-compose. Note, if you install docker using above link on MacOS, you don't have to install docker compose.

(3) Run command

```
docker-compose up react
```

**Install using VM**

1. First, download the VM using this link. Note, the vm file is SeniorDesign.ova

2. Second, Install VitualBox
3. Finally, doing the following steps to import VM and run the VM.

**Config VM**

(1) First in your VitualBox click File > Import Appliance > clicked the little file icon on screen and select the VM file

(2) After VM has been imported, click VM and then click settings. Go to Network tab and click Adapter. Click port forwarding. Then have your settings match the screenshot.

(3) Start Vm then go to Senior-Design folder. In that folder, run

```
sudo docker-compose up code-server
```

Then, go to your home screen on your computer, open browser and enter

0.0.0.0:8080

or

localhost:8080

### 6.3.2 Appendix 2: Alternative version of the design

The original implementation of the application, which was done previously by Mark Hernandez, one of our advisor/client's students, runs as a web application in the browser. One of the major issues with this type of implementation was the lack of security features that would need to be enforced. We decided to implement the application as a desktop app, thus eliminating the need for web security features.

### 6.3.3 Appendix 3: Other Considerations

Due to studio's impact, we started accepting online classes after spring break, so all face-to-face meetings became online meetings. We use Discord and Slack for our daily discussions and group work. Compared with before, we have a more flexible date meeting schedule. Our project does not need additional hardware or lab, and all the work can be done on our personal computers or equipment, so online or offline meeting does not have a great impact on us.

### 6.3.4 Appendix 4: Code

Github Repository