Fall 2023

Graphical User Interface for Evidential Reasoning Models

Rohin Gopalakrishnan

Graphical User Interface (GUI) for Evidential Reasoning Models

A Project

Report

Presented to

Professor Leonard Wesley

Department of Computer Science

San Jose State University

In Partial Fulfillment

Of the Requirements for the Class

Fall-2023: CS 298

By Rohin Gopalakrishnan

May 2023

The Designated Project Committee Approves the Project Titled

Graphical User Interface for Evidential Reasoning Models

By Rohin Gopalakrishnan

Approved for the Department of Computer Science

San Jose State University

May 2023

Dr. Leonard Wesley	Department of Computer Science
Dr. William Andreopoulos	Department of Computer Science
Pranesh Ravi	UX III, Google

ABSTRACT

The Capri system is an evidential reasoning system based on the belief function calculus to support automated reasoning and decision making in uncertain environments. Example domains of application include, medical diagnosis, as well as identifying biological biomarkers. The purpose of this project is to build a Python web-based and app-based Graphical User Interface (GUI), called PyGrapher, that facilitates building graphical evidential reasoning models. The graphical models built using PyGrapher will then be converted to a form that is suitable for input to the Capri system. The PyGrapher system provides an intuitive means to build and manipulate evidential reasoning models as graphs that can be readily manipulated as the user desires. PyGrapher was built in a very user friendly manner and special care was taken to make sure that the interface highlights relevant errors to the user at the earliest occurrence.

Index Terms – **Belief Function, GUI, Electron, React, Graph, Nodes**

ACKNOWLEDGEMENTS

I would like to express my deep appreciation to Professor Leonard Wesley, for his guidance, support, and encouragement throughout my research. Professor Leonard Wesles's expertise and insights were invaluable in shaping the direction of my research and helping me to navigate the challenges and complexities of the research process. I am grateful for his patience and dedication, as well as for the countless hours he spent providing feedback on my work and helping me to improve my writing and research skills. Without his mentorship, this project would not have been possible, and I am truly grateful for his contribution to my academic and personal development. I would also like to extend my thanks to my committee members for their constant encouragement and support in this.

TABLE OF CONTENTS

TABLE OF FIGURES

1. INTRODUCTION

Pygrapher is an all-inclusive and efficient system designed specifically to manage graph data efficiently. In order to make the Pygrapher system accessible to users, the Pygrapher GUI was created as an internal tool providing an intuitive user experience when interacting with its system. The Pygrapher GUI was developed to give users an effortless and straightforward method for creating, loading, and manipulating graph data while still meeting specification guidelines accepted by the Pygrapher tool. Our aim is to make graph management as straightforward and accessible as possible while upholding high standards of efficiency and reliability.

1.1. Motivation

The motivation for embarking on this project stems from the shortcomings of existing tools to build graphical models, particularly those utilized in evidential reasoning models. Presently, there is a lack of user-friendly interfaces, as many of these tools heavily rely on the command line for operation. This poses a significant barrier for individuals who are not well-versed in programming or prefer more accessible interfaces. By addressing this limitation, the project aims to facilitate the use of evidential reasoning models and make them more widely accessible to researchers, practitioners, and users from various domains.

Moreover, the reliance on multiple flat files within existing tooling poses challenges in terms of data management and overall workflow robustness and efficiency. Working with numerous flat files for tasks such as data input, configuration, and output can be cumbersome and prone to errors. Recognizing this drawback, the project seeks to streamline the workflow by

providing a unified and intuitive platform where users can interact with their data and model seamlessly. This approach simplifies the process, reduces potential errors, and enhances the overall user experience.

Additionally, the steep learning curve associated with existing tools like Grasper-CL has been a limiting factor in their widespread adoption. Many of these tools require users to delve deeply into technical concepts and intricate workflows before they can effectively employ them. The aim of this project is to develop a tool that mitigates the steep learning curve, making evidential reasoning models more accessible to a broader audience. By providing a more user-friendly interface and simplifying the underlying processes, the project aims to empower users to leverage the potential of evidential reasoning models without requiring extensive technical expertise.

.

1.2 Belief Functions

Belief functions [1] (also referred to as Dempster-Shafer Theory or evidence theory) provide mathematical frameworks used to represent and reason with uncertainty and incomplete information. They were first proposed by Glenn Shafer during the 70s as an alternative approach to probability theory.

Belief functions can be invaluable tools in situations characterized by limited or conflicting evidence, providing a method to combine various sources of evidence or information and arrive at decisions or draw conclusions more efficiently.

Belief functions represent uncertainty by assigning degrees of belief or beliefs masses to sets of possible outcomes called propositions, representing any statement or hypothesis about any given scenario that has meaning in life and includes potential outcomes such as consequences or possible outcomes that might emerge; each proposition receives one belief mass which indicates the degree to which its assertion holds up against reality.

Belief functions revolve around their titular element - known as basic probability assignment or BPA). A belief function assigns specific belief masses for sets of propositions such as single propositions as well as combinations thereof.

1.4 Evidential Reasoning

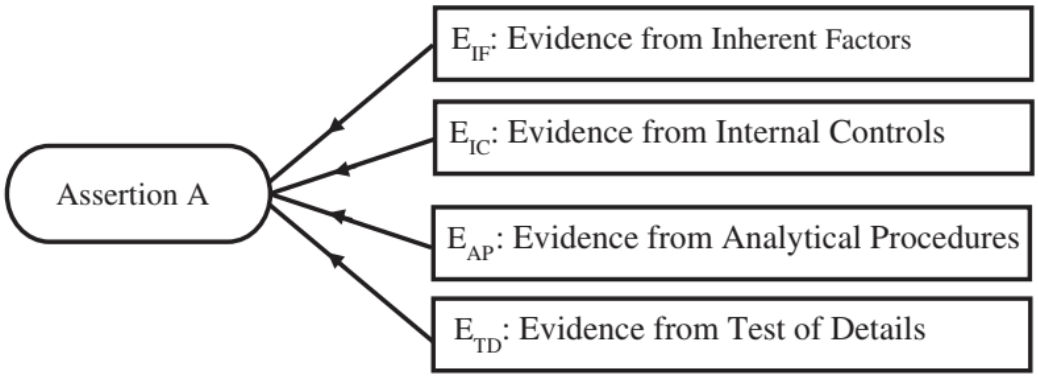

Figure 1. Evidential diagram for one variable with four independent items of evidence in an audit context

The Evidential Reasoning Approach (ERA) [2] is a decision-making methodology that synthesizes evidence from multiple sources by assigning belief functions or basic probability assignments that represent uncertainty and belief, then combining them using Dempster's rule of combination to form an overall belief representation. This approach facilitates integration of diverse evidence while managing conflicts and uncertainties more effectively while offering a systematic framework for reasoning under uncertainty. By considering the combined belief function, the evidential reasoning approach supports informed decision making and inference, offering a robust and reliable method for incorporating evidence from various sources.

1.4.1 Dempster's rule of combination

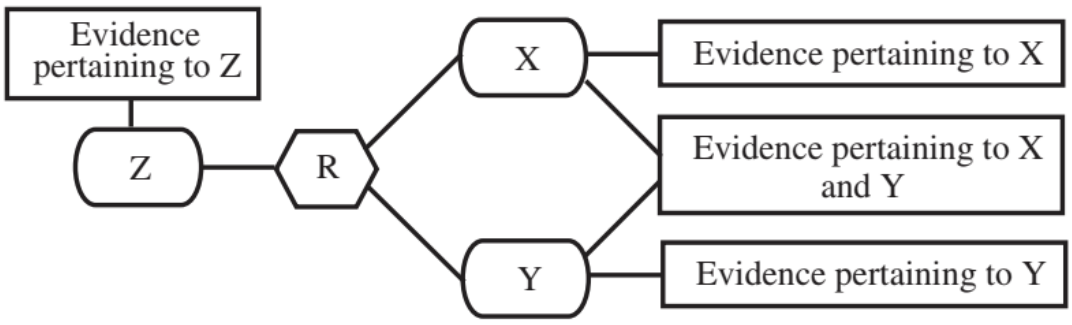

Figure 2. Evidential reasoning and evidential diagram.

Dempster's Rule of Combination [3] is a mathematical method utilized in belief theory for consolidating evidence from various sources. This process integrates belief functions or basic probability assignments associated with each source of evidence. This rule operates in four steps, combining belief masses assigned to individual propositions, calculating the commonality between the belief functions, normalizing the combined belief masses, and asasigning the remaining mass to the ignorance proposition. Dempster's rule takes into account individual belief masses, the overlap between belief functions, and ignorance proposition to allow for the fusion of evidence, handling conflicts efficiently, and creating a combined belief function that represents updated degrees of belief or support for various propositions.

1.5 Previous Solutions

The existing tool used for creating and managing these nodes was known as Grasper-CL, developed by Intelligent Systems Lab at the Kyushu University of Japan as part of their graph mining efforts. Since being released for public use in 2002, it has gained significant attention among both practitioners of graph mining and researchers interested in its field.

Grasper CL is built on top of the Common Lisp language [4], which has been used for decades in artificial intelligence research. The software is highly customizable and can be extended using Lisp macros, allowing users to create their own algorithms and tools for graph analysis. One of its defining characteristics is that Grasper CL is one of the first graph mining tools to implement the closed frequent subgraph mining algorithm, which is used to

discover frequent subgraphs in a large dataset of graphs.

1.5.1 Grasper CL

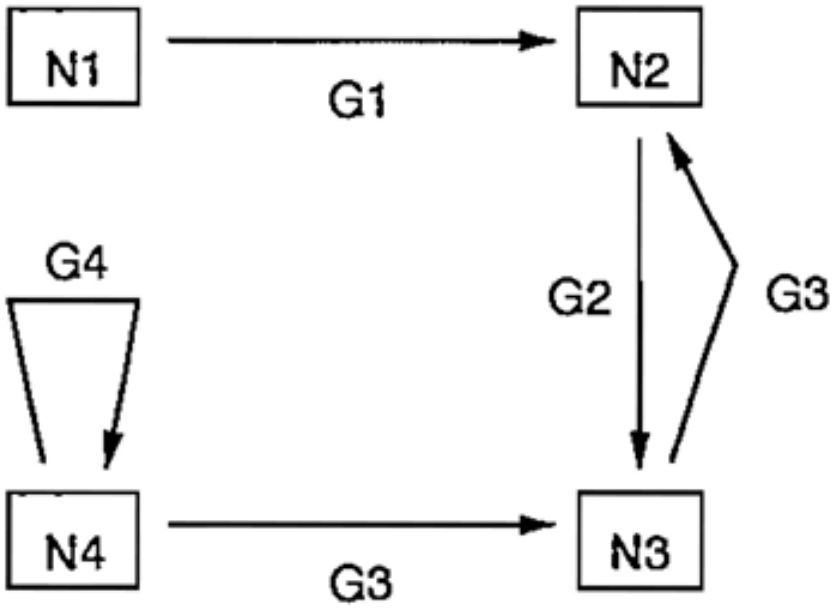

Figure 3. Sample Graph in Grasper-CL

Pros:

One of Grasper CL's major advantages lies in its capacity to manage large-scale graph data efficiently. The software was specifically developed to be highly scalable, making it suitable for handling massive datasets of graphs with ease. Furthermore, Grasper CL provides several algorithms for graph analysis including frequent subgraph mining, clustering, and classification; making it an invaluable tool for researchers and practitioners in graph mining alike. Moreover, this highly customizable platform also allows users to create their own algorithms and tools for graph analysis.

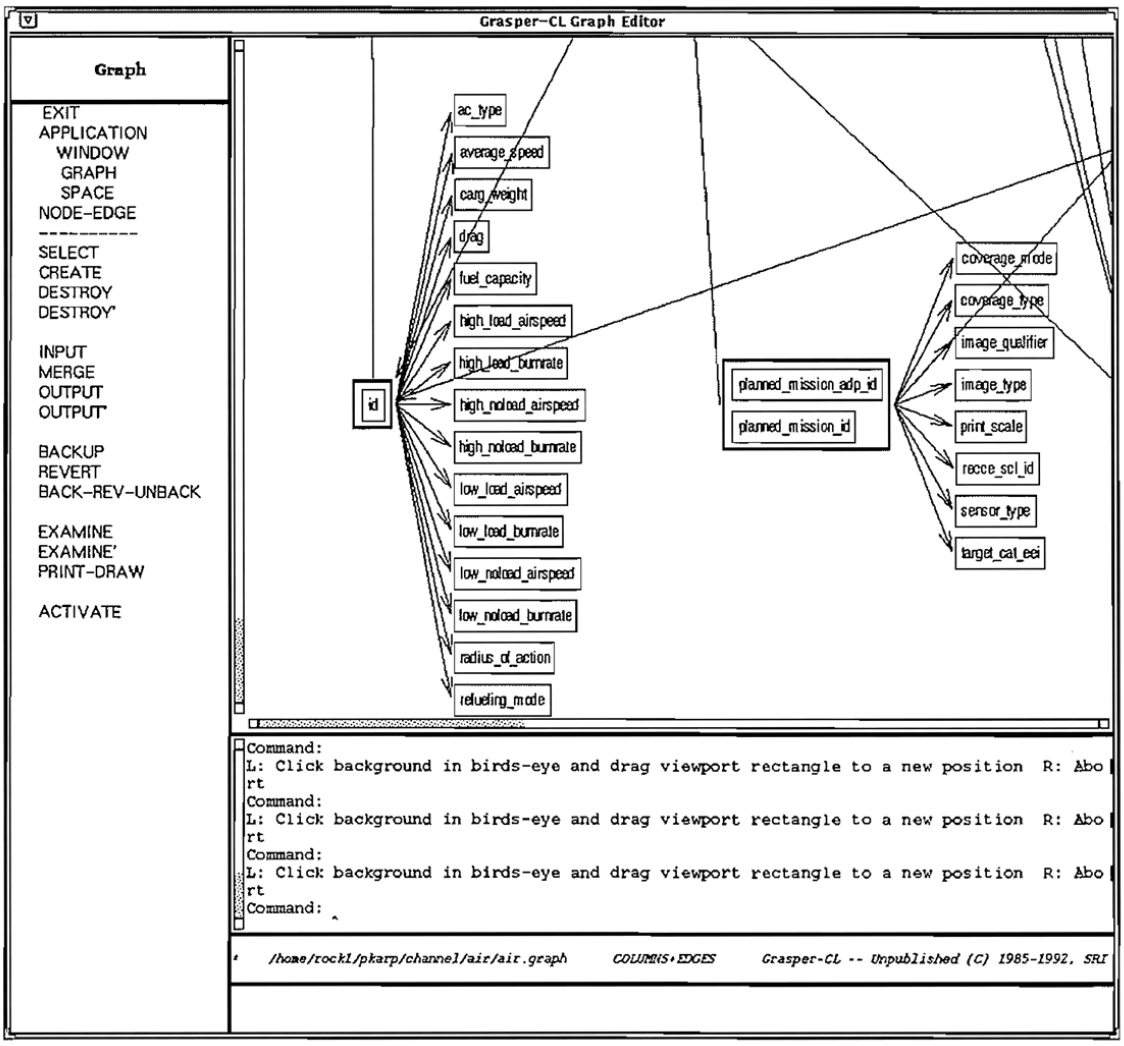

Figure 4. User Interface of Graph-CL

Cons:

One of the main disadvantages of Grasper CL is its steep learning curve for users unfamiliar with Common Lisp. Setting up and using this software may prove challenging for non-Lisp programmers. Additionally, this software was not specifically created to facilitate high levels of visualisation as its purpose lies mainly in computational analysis of graph data.

Due to an extremely steep learning curve and an interface that primarily prioritizes CLI usage, making even minor modifications to Grasper CL's user interface is challenging and time-consuming for beginner users - particularly for users not familiar with Lisp programming.

Overall, Grasper CL is an incredibly customizable and scalable graph mining tool, widely utilized both in research and practice. While its steep learning curve may make it less suitable than alternatives for users requiring high levels of graphical visualization, its versatility and efficiency make it a go-to option among researchers and practitioners in graph mining.

2. PROBLEM STATEMENT

Existing solutions to tackle the problem of large scale graph management with respect to evidential reasoning and belief functions are mostly CLI based and have a very steep learning curve. The problem statement is to build new graph management system that has the following features:

- **Multi-platform support:** Multi platform support refers to the capability of a software application to function on operating systems and hardware platforms. This implies that the software can be installed and utilized by users irrespective of the device or operating system they are using. Having platform support is crucial for software applications to reach a broader range of users and guarantee that individuals have access to the necessary tools regardless of their hardware or software preferences. Additionally it enables developers to expand their user base and enhance the value and significance of their software products.
- **Graphical User Interface:** A graphical user interface (GUI) plays a role in software applications that involve interfaces or data structures like graph management systems. It simplifies the visualization of data, facilitates data input and manipulation and ultimately enhances the user experience. With its user-friendly interface a GUI improves the functionality and usability of software applications making them more accessible and valuable to an audience.
- **Accessibility and User Interface:** Building an accessible application with a good user interface is crucial to ensure that all users, regardless of their abilities, can use the software effectively. It is important to design an interface that is easy to

navigate, intuitive, and has clear instructions.

- **Save/Load and Export:** Ability to Load/Save and export the graph to various formats
- It is essential to construct the system using up to date technologies that're user friendly and compatible. This will guarantee the systems manageability, scalability and ability to withstand advancements. By using programming languages, frameworks and tools developers can create software systems that're modular, adaptable and straightforward to maintain. Additionally these technologies offer a level of abstraction that enhances development efficiency.
- Additionally, the new system should be designed with the ability to export a web interface in the future. This requirement can significantly enhance the accessibility and versatility of the software, making it easier for users to access and use the system from anywhere, on any device. To achieve this, the new system could utilize modern web development tools and techniques. By leveraging these technologies, the new system could provide a frictionless experience for users, while also reducing its infrastructure costs and improving its scalability.

3. TECHNOLOGY FEASIBILITY SURVEY

3.1 Research

In order to choose a feasible technology for the GUI, multiple technologies were surveyed in order to find an optimal solutions each with its own advantages and disadvantages:

- **Electron:**

  Electron [5] is an increasingly popular cross-platform desktop application development framework using web technologies such as HTML, CSS, and JavaScript. Electron allows developers to build native-like applications using web technologies which can then easily be deployed across Windows, MacOS, and Linux platforms. However, its applications tend to be resource-intensive with significant performance overheads.

- **Qt:**

  Qt is an award-winning C++ framework used for developing cross-platform GUI applications. It provides libraries and tools that make creating native-like apps that run across Windows, MacOS and Linux easier than ever before. Qt boasts a strong community of developers as well as an abundance of features and functionalities. However its C++ nature can present more challenges when developing applications using it compared to other frameworks.

- **JavaFX:**

  JavaFX [6] is a Java-based framework for building cross-platform desktop applications. The tool offers libraries and tools for developing engaging, interactive applications that run across Windows, MacOS and Linux operating systems. JavaFX applications tend to be easy to learn with numerous features and functionalities available. However, due to overhead imposed by the Java Virtual Machine they may run more slowly compared to native apps.

- **GTK+:**

  GTK+ is an immensely popular C-based framework for building cross-platform GUI applications. This versatile framework offers libraries and tools to enable building native-like apps that run on Windows, MacOS and Linux, with minimal footprint requirements compared to other frameworks. Thus it is an ideal system for resource-constrained requirements but more challenging than other frameworks due to being C-based.

Overall, Electron has proven itself as an efficient cross-platform GUI building platform. It combines flexibility, customization, and multi-platform support in one package for developers seeking cross-platform GUIs that meet a range of specifications. Furthermore, Electron allows for extensive levels of customization and flexibility; developers can use CSS for user interface creation while supporting popular front-end frameworks like React and Angular as well as Node.js which allows server-side JavaScript functionality for powerful back-end functionalities.

3.2 React

React is a popular front-end JavaScript library that provides a declarative way of building user interfaces. Its virtual DOM model ensures efficient rendering and makes it easy to manage complex application state. When used with Electron, React can provide an easy and intuitive way to create interactive desktop applications that can run on multiple platforms. By using React, the ability to easily export the application into a website also satisfies the requirements.

3.2 Graphing Library

There are several frontend graphing libraries available for creating interactive graphs with nodes and edges. Each of these libraries has its own set of pros and cons. The below will focus on why vis.js was the best choice for this application.

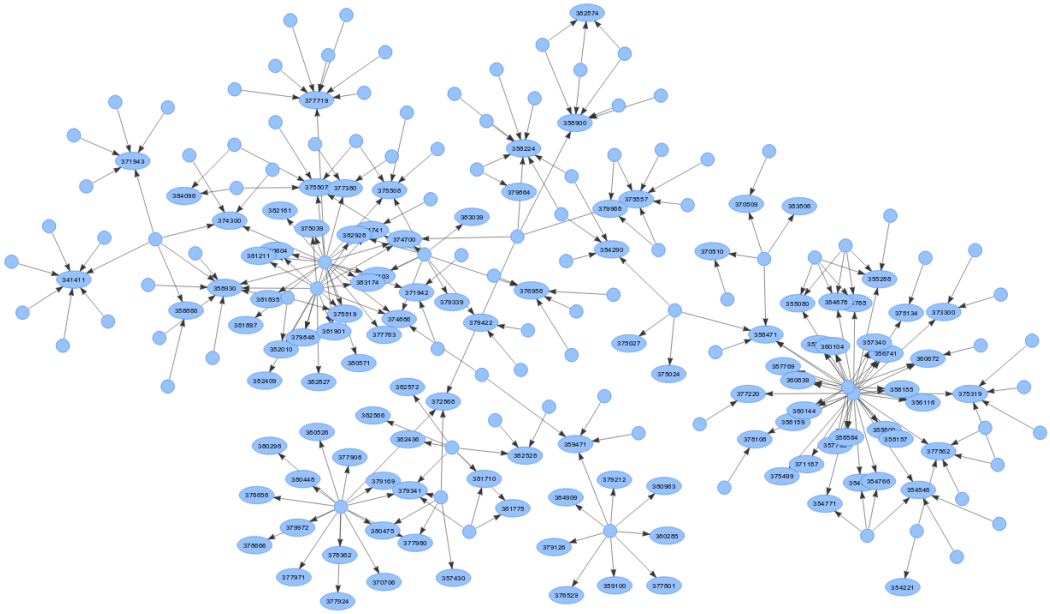

Figure 5. Sample Graph in Vis.Js

Pros:

- **Customizable:** Vis.js is a highly customizable library, which allows developers to modify every aspect of the graph, including nodes, edges, labels, colors, and more. This level of customization makes it easy to integrate the graph with existing applications and to create unique and visually appealing graphs.
- **Interactive:** Vis.js provides a highly interactive user experience. Users can zoom, pan, drag, and drop nodes and edges, making it easy to explore and manipulate the graph.
- **High-performance:** Vis.js is built for performance, which makes it suitable for handling large-scale graphs with hundreds or even thousands of nodes and edges.
- **Cross-browser support:** Vis.js works well with all modern browsers and has support for both desktop and mobile devices.

Cons:

- **Steep learning curve:** Vis.js may present a steep learning curve that may discourage developers from adopting it.
- **Limited documentation:** Vis.js boasts a robust user community, but its documentation may make it challenging for new developers to get up and started quickly.
- **Limited built-in functionality:** Vis.js is a low-level library, meaning developers will need to add custom functionality on top of it in order to fully leverage its potential.

In conclusion, Vis.js is the best solution for an interactive graph with nodes and edges due to its high level of customization, interactivity, and performance. While there may be a steep learning curve and limited documentation, the benefits of using Vis.js far outweigh the

drawbacks.

Vis.js also has a React implementation called react-vis-graph which is a React Component and extends into the ecosystem easily. Due to this, react-vis-graph was selected as the graphing library.

3.2 UI Framework and Component Library

There are many popular frontend frameworks such as Bootstrap, Materialize, Foundation, Bulma, and Tailwind. Each one of these frameworks comes with its own set of pros and cons.

Bootstrap: Bootstrap is a very popular frontend framework since its inception in 2011. It was really popular around 2020, but one of its major cons is in the customization. It boasts being very responsive and in being very consistent and having a strong user base.

Materialize: Materialize is a more recent frontend framework based on Google's Material UI framework. It has prebuilt UI components that are in the standards set by Google. It is another really popular framework, however, due to the overuse of this design system. Google is also stepping away from pure material designs.

Foundation: Foundation is a responsive frontend framework designed for easy customization and extension. It boasts a lot of very large enterprises and companies and also has features like pre-built components, templates, styles that would enable developers to quickly create websites.

Semantic UI: It's a front-end framework made to be simple to use and easy to understand. Furthermore, pre-built UI elements and styles based on linguistic principles are an advantage of Semantic UI, which enables the development of significantly more user-friendly designs.

Tailwind CSS: Tailwind is a more recent addition to the popular CSS frameworks, the benefits of tailwind being that it has prebuilt class names with predefined patterns. This means that developers will no longer have to create custom CSS styles and can follow the standards of prebuilt styles.

Of these five frameworks, **Tailwind CSS** stands out because it offers a really good approach to frontend development as mentioned in [7]. Instead of providing pre-built UI components and styles, Tailwind provides a set of predefined classes that can be used to style any element on a webpage. This allows developers to make changes without having to write custom CSS from scratch and also enables rapid prototyping and development.

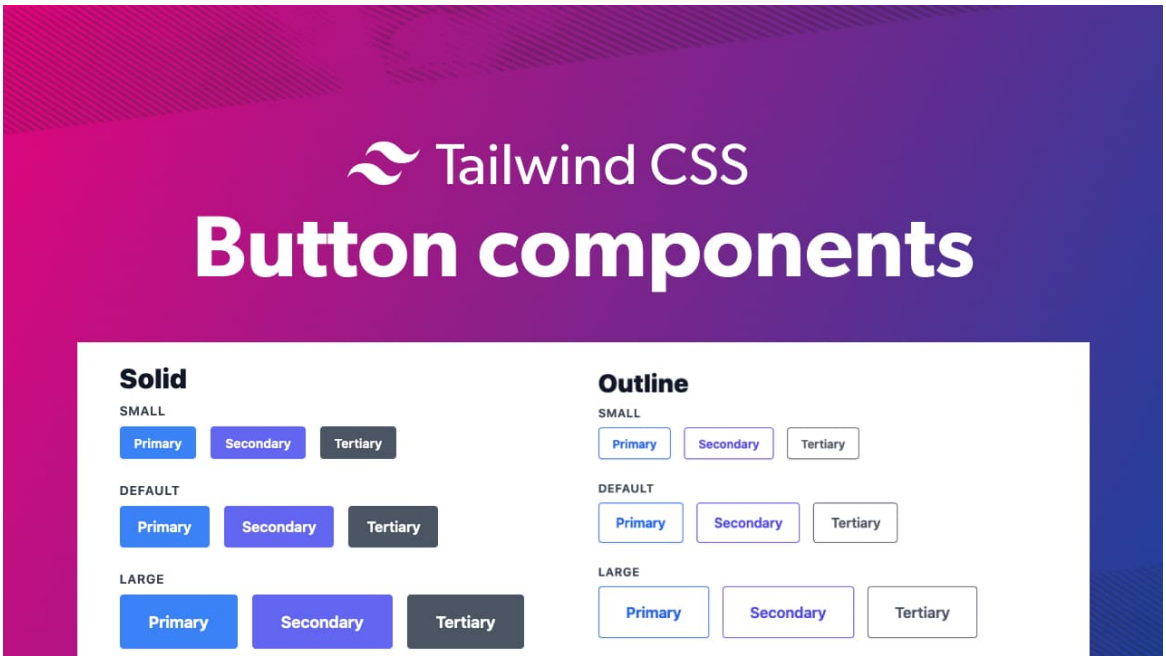

Figure 6. Example of Tailwind Button Components

Tailwind also includes a large number of pre-built utility classes that cover a wide range of styles and design patterns. This makes it easier for developers to create consistent designs across their website or application, while still allowing for a high degree of customization.

Overall, Tailwind is a good choice for frontend development because it offers a unique and powerful approach to styling web pages. Its utility-first approach can help developers create custom designs more quickly and efficiently, while still allowing for a high degree of flexibility and customization.

Additionally, Ant Design components are also being used as an additional component library. One of the strengths of Ant Design is its focus on accessibility and usability. The framework

provides a set of accessibility guidelines and best practices that developers can follow to ensure that their applications are accessible to all users, including those with disabilities.

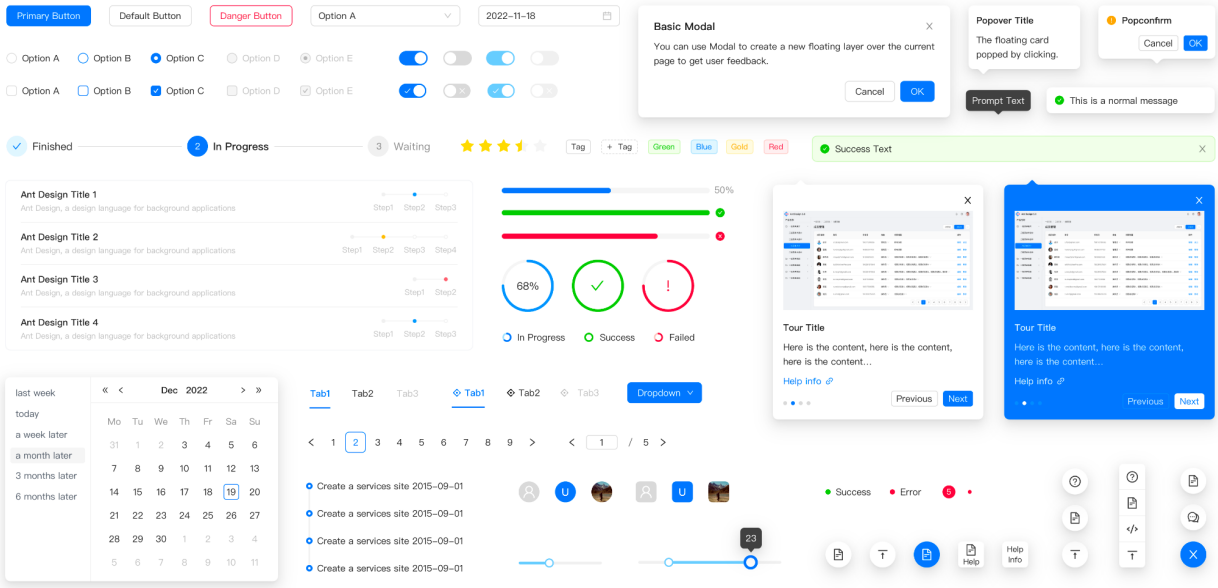

Figure 7. Example of Ant Design components

The combination of Tailwind and Ant Design components would help in the speed of development and since many of the generic components that are required can be extended from the already existing large set of components from here. This approach can save time and effort, allowing the ability to focus on more critical aspects of this project.

3.3 Electron Boilerplate

Boilerplate frameworks provide a foundation for building applications quickly and efficiently with a good starter template and comes bundled with lots of features [8].

Electron Forge:

Uses webpack as a Bundler focusing on simplicity and automation which makes it a good choice for someone getting started with electron development.

Electron Builder:

Electron Builder is another popular boilerplate framework that focuses on application packaging and distribution. It excels in simplifying the deployment process and offers several notable advantages.

Electron-React-Boilerplate:

Electron-React-Boilerplate combines Electron with React which makes it great for the use-case of this project. This also supports the ability to deploy and with hot reloading as well.

To conclude, the final technology stack based on the feasibility analysis was:

- Electron
- React
- Vis.js (react-vis-graph)
- TailwindCSS
- Boilerplate by Electron-React-Boilerplate

4. DEVELOPMENT ENVIRONMENT

4.1. Node Version Manager

Node Version Manager (NVM) provides users with a convenient means of switching among various versions of Node.js installed on their machines. As explained in [9] the steps to use the Node Version Manager are:

- Step 1: Open the terminal
- Step 2: Visit https://github.com/nvm-sh/nvm and copy the installation command for the Operating system that you are using
- Step 3: Paste the command from the website and run it in the terminal, this will install nvm on your machine
- Step 4: Once the installation is complete, re-open the terminal to finish installing
- Step 5: To verify the successful installation of NVM, run the command "nvm --version". You should see its version number displayed.
- Step 6: Once NVM has been successfully installed, it allows for quick and simple Node.js installations of various versions (nvm install 18 is the recommended choice in this project). To download one specific version: To do this use: "nvm install 18".

Figure 8. Installed nvm versions

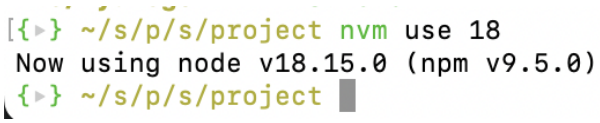

Figure 9. Set the nvm version for this project

4.2. Install the required packages

npm (Node Package Manager) is a powerful tool that allows developers to install and manage packages and dependencies for their projects. Using the Command Line Interface (CLI) provides a straightforward and efficient way to install npm packages.

- Step 1: Open your terminal or command prompt.
- Step 2: Change Directory (cd) into the project directory
- Step 3: Run the command npm install to install all the packages in package.json

Figure 10. Package.json scripts

The above scripts can be used to either expose the react application or use npm run dev to run the electron application.

4.3. Developer tooling and Debugging

React applications provide the developer by a set of developer and debugging tools which are explained in [10]. It is crucial for a smooth implementation of the project and hence the application.

4.3.1 Chrome Developer Tools

In Electron applications, Chrome DevTools are an integral part of the development process, aiding developers in debugging and optimizing their applications. However, it is important to note that by default, the DevTools are not open in Electron applications in order to ensure a

streamlined production environment.

The decision to enable or disable DevTools in an Electron application is typically based on the application's environment. During development, it is beneficial to have the DevTools readily available for debugging and testing purposes. Developers can open the DevTools by using specific keyboard shortcuts or through the application's menu options.

```
  // Create the browser window.
  const win = new BrowserWindow({
    width: 800,
    height: 600,
    webPreferences: {
      nodeIntegration: true,
    },
  });

  // and load the index.html of the app.
  // win.loadFile("index.html");
  win.loadURL(
    isDev
      ? 'http://localhost:3000'
      : `file://${path.join(__dirname, '../build/index.html')}`
  );
  // Open the DevTools.
  if (isDev) {
    win.webContents.openDevTools({ mode: 'detach' });
  }
}
```

Figure 11. Electron JS dev tools runner

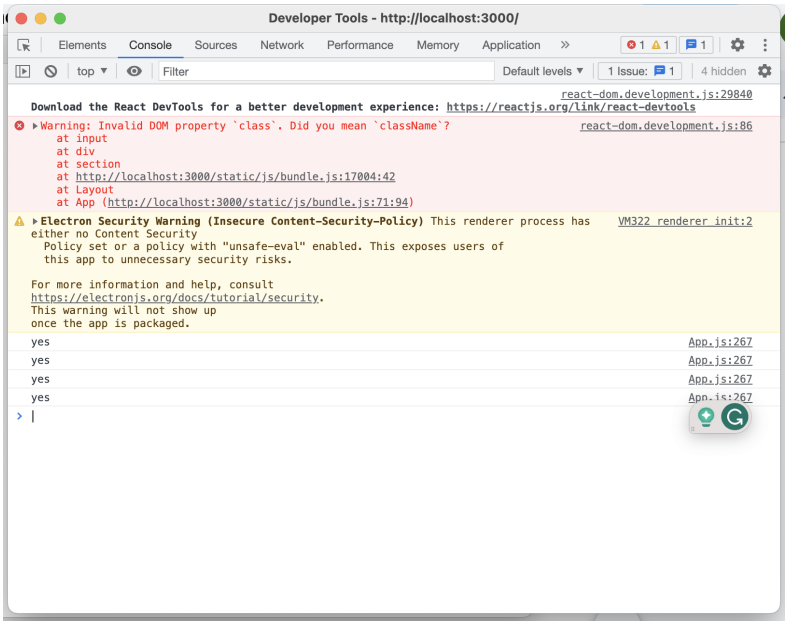

Figure 12. Chrome developer tools running on detached mode

4.3.1 React Developer Tools

React developer tools is a browser extension that helps in debugging and inspection for React Application. One of the key features is the ability to inspect the props and of React components. Due to this being a crucial component to debug and since it is time consuming to install this extension each time manually from the store. The use of an npm package called 'electron-devtools-installer' is being used.

```
const { default: installExtension, REACT_DEVELOPER_TOOLS } = require('electron-devtools-installer');

const { app, BrowserWindow } = require('electron');
const isDev = require('electron-is-dev');


function createWindow() {

  installExtension(REACT_DEVELOPER_TOOLS)
      .then((name) => console.log(`Added Extension:  ${name}`))
      .catch((err) => console.log('An error occurred: ', err));
```

Figure 13. Electron Devtools for React Developer Tools

This package automatically installs the React Developer tools at runtime and also caches the extension so it does not have to download the extension on subsequent loads.

5. IMPLEMENTATION

This section entails the various components of the User Interface and talks about the functionalities of each component.

5.1 Sidebar

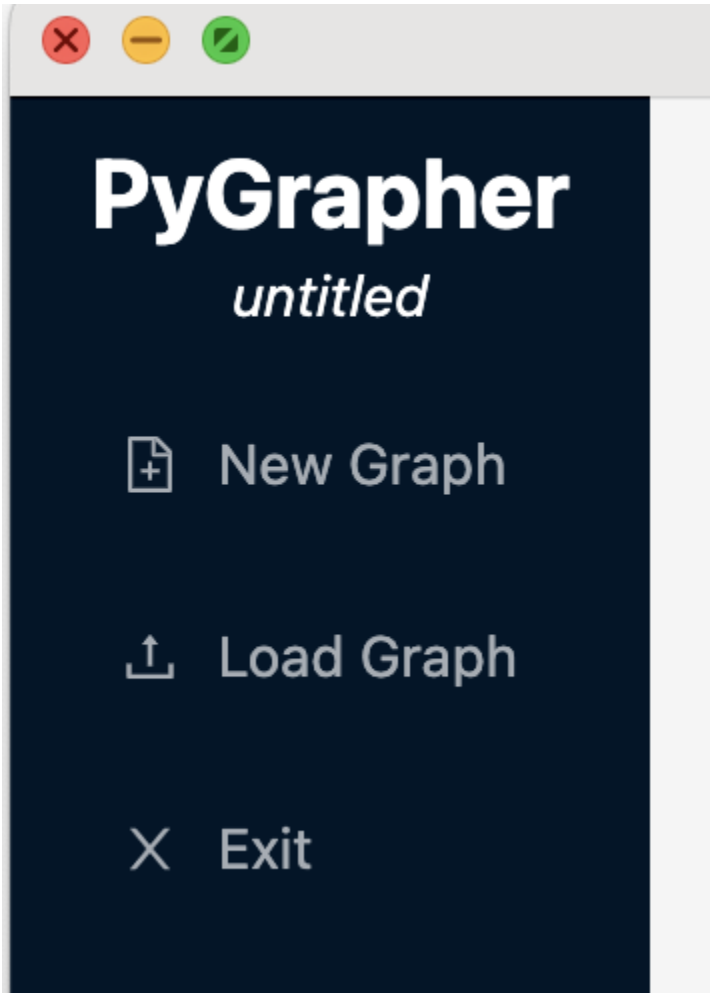

Figure 14. Sidebar of the Application (File Mode)

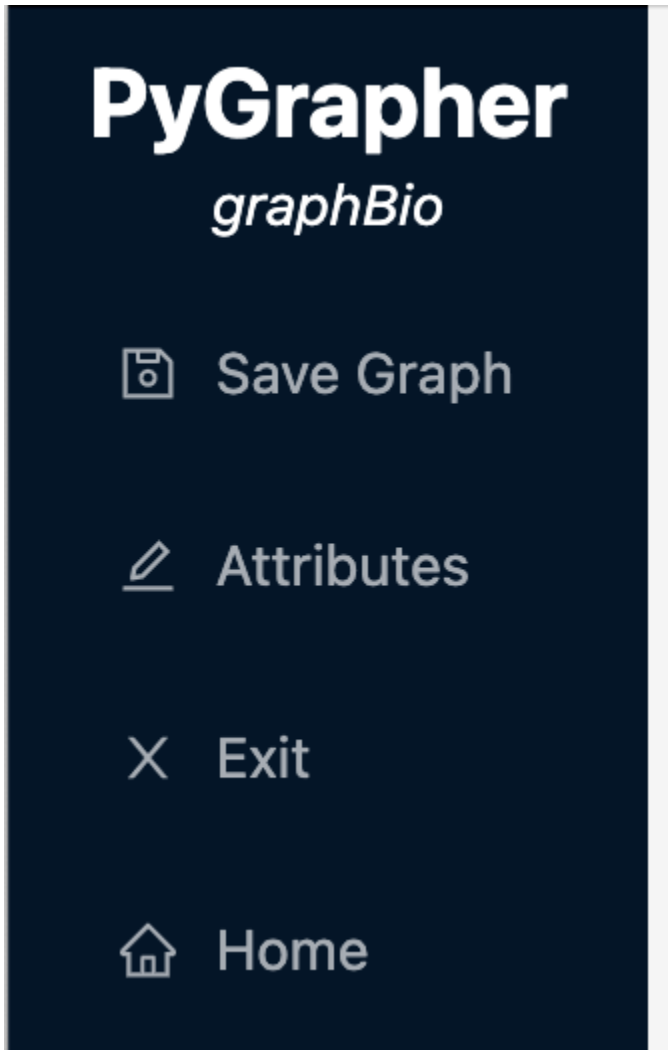

Figure 14. Sidebar of the Application (Graph Mode)

The sidebar supports the following functionalities to Add a New Graph, Load Graph, once the graph is loaded or selected, the UI moves into the Graph Mode, where the user has an option to Save Graph, Edit Attributes of the file, Exit and Go back to the File Mode (Home).

- **New Graph** creates new instances of the Graph and creates a sample untitled.json file.
    - When a new graph is created, the user is prompted to enter the graph attributes like below:

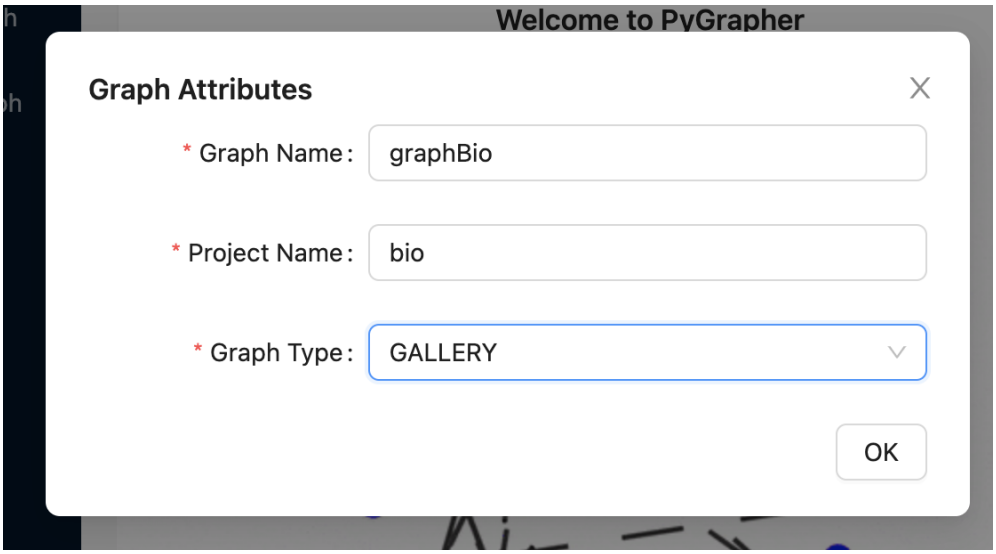

Figure 15. Graph Attributes Interface

- **Load Graph** opens the File Browser and accepts only json files. Will throw an Error if there are any parsing errors.

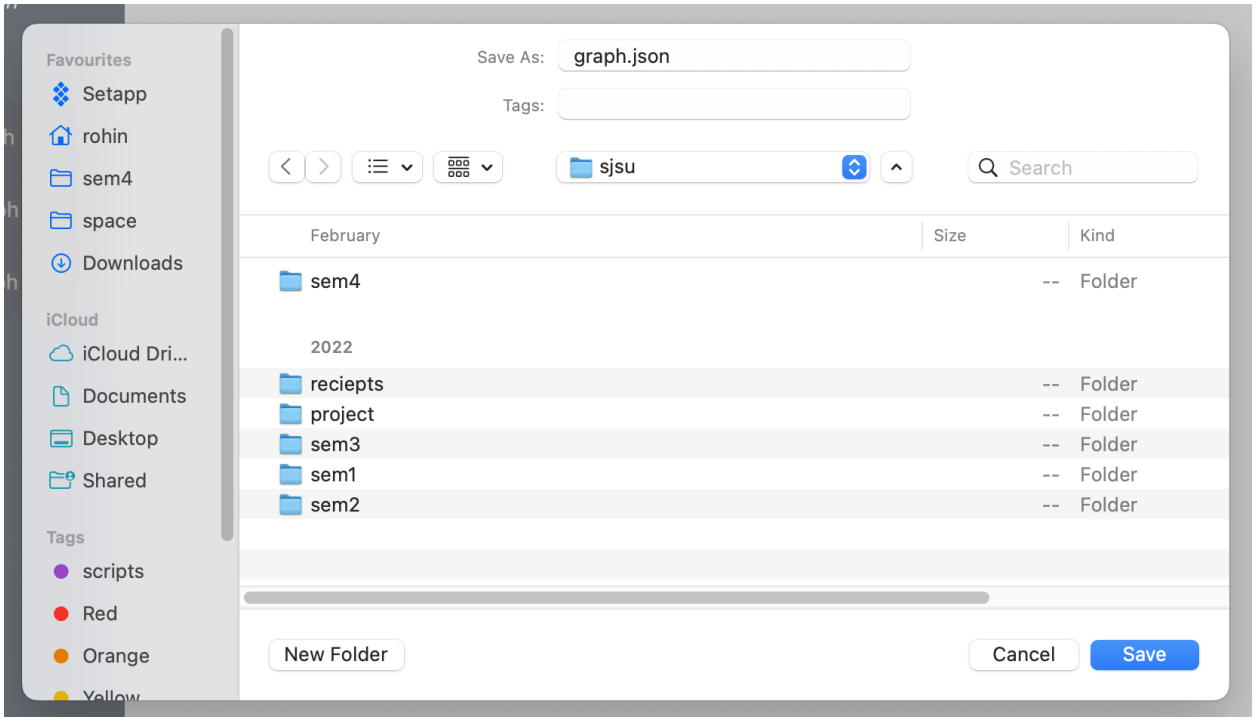

Figure 15. Save Graph File Interface

The save and load graph converts the nodes and edges in the graph into a JSON file. This function has been written in a generic way to be able to build format extenders in the future to convert the graph, nodes and its attributes into JSON.

```
{
  "nodes": [
    {
      "id": "1c5aa57e-23bb-45d7-ae16-b980041c3ca1",
      "size": 150,
      "label": "Sx021\nCAGE: 4\nSFE: 23\nAttrib3: Sample",
      "properties": [
        {
          "key": "CAGE",
          "value": "4"
        },
        {
          "key": "SFE",
          "value": "23"
        },
        {
          "key": "Attrib3",
          "value": "Sample"
        }
      ],
      "color": "#FFCFCF",
      "shape": "circle",
      "font": {
        "face": "monospace",
        "align": "left"
      }
    },
    {
      "id": "5f25625e-3766-4a12-bf92-8cbe64234a28",
      "size": 150,
      "label": "Sample Node 2",
      "properties": [],
      "color": "#f8e71c",
      "shape": "box",
      "font": {
```


Figure 16. Node attributes of the saved JSON file

```
"edges": [
  {
    "id": "b8563d33-4a98-49d5-a2be-31b8e8e3265c",
    "from": "1c5aa57e-23bb-45d7-ae16-b980041c3ca1",
    "to": "5f25625e-3766-4a12-bf92-8cbe64234a28"
  },
  {
    "id": "2f1b763f-5e90-40d3-80ca-b8bda41115ec",
    "from": "1c5aa57e-23bb-45d7-ae16-b980041c3ca1",
    "to": "4e6daa79-cd6d-4b18-8276-82b9a0e9f5d1"
  },
  {
    "id": "3171b762-e921-412a-90d9-cd7fb814ccca",
    "from": "5f25625e-3766-4a12-bf92-8cbe64234a28",
    "to": "1c5aa57e-23bb-45d7-ae16-b980041c3ca1"
  }
],
"options": {
```


Figure 17. Edge attributes of the Saved JSON File

5.1 Graph View

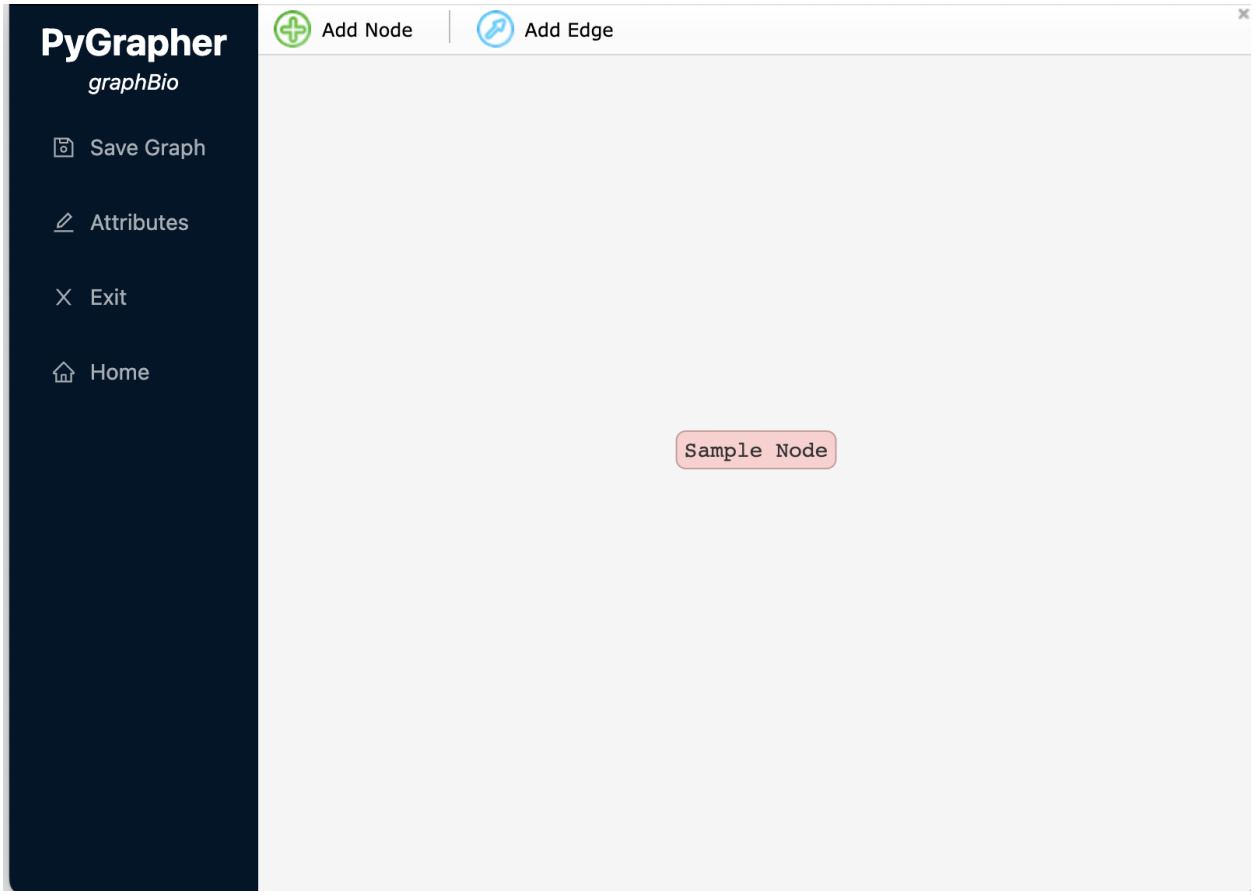

Figure 18. Sample Graph UI

The Graph View is a visual representation of a graph data structure and offers various functionalities to manipulate the nodes and edges within the graph. Let's further extrapolate the abilities mentioned:

Add Node: The "Add Node" button allows users to add nodes to the graph. When the button is clicked, the UI prompts the user to click on any area within the Graph View. Upon clicking, a new node, often represented as a shape or symbol, is created at the clicked position. Initially, the newly created node may not have any attributes or data associated with it.

Add Edge: The "Add Edge" button enables users to establish connections between nodes in the graph. When this button is clicked, the user is prompted to click on one node to initiate the edge creation process. After selecting the starting node, the user can drag the edge to the desired target node. This action creates a directed or undirected edge between the selected nodes, representing a relationship or connection between them.

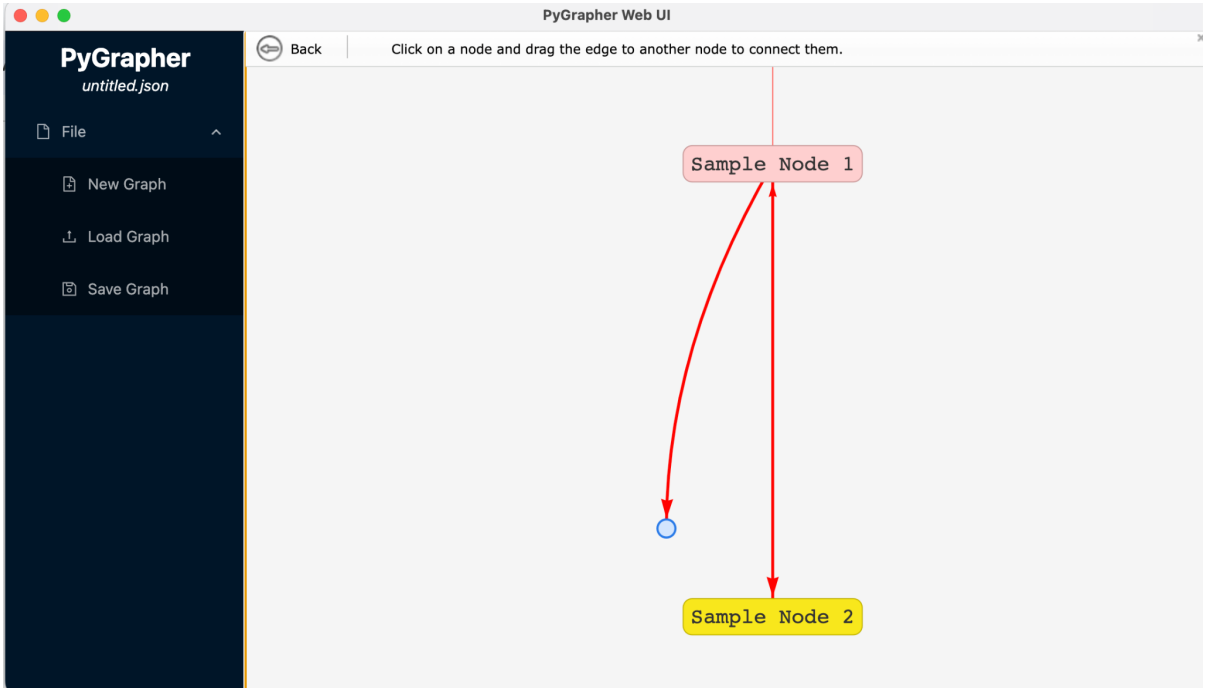

Figure 19. Add Edge UI

Delete Edge: The Graph View allows users to delete edges from the graph. When an edge is selected or highlighted, the user is provided with an option to delete it. By choosing to delete the edge, the connection between the associated nodes is severed, and the edge is removed from the graph.

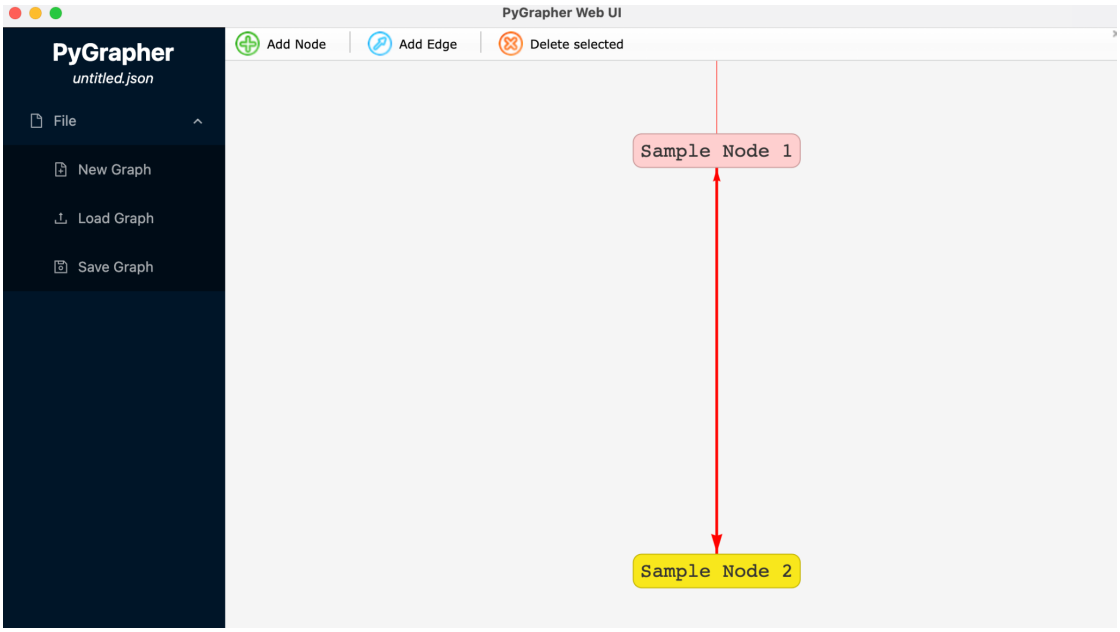

Figure 20. Delete edge UI

Delete Node: The ability to delete nodes is another functionality provided by the Graph View. When a node is selected, the user is presented with an option to delete the node. If the user chooses to delete the node, the system also removes all incoming and outgoing edges connected to that node. This ensures that the graph remains consistent and maintains data integrity.

5.2. Attributes View

The purpose of this interface is to add attributes to the nodes in the graph. The attributes that can be added are:

- Shape
- Label
- Key/Value pairs of open attributes

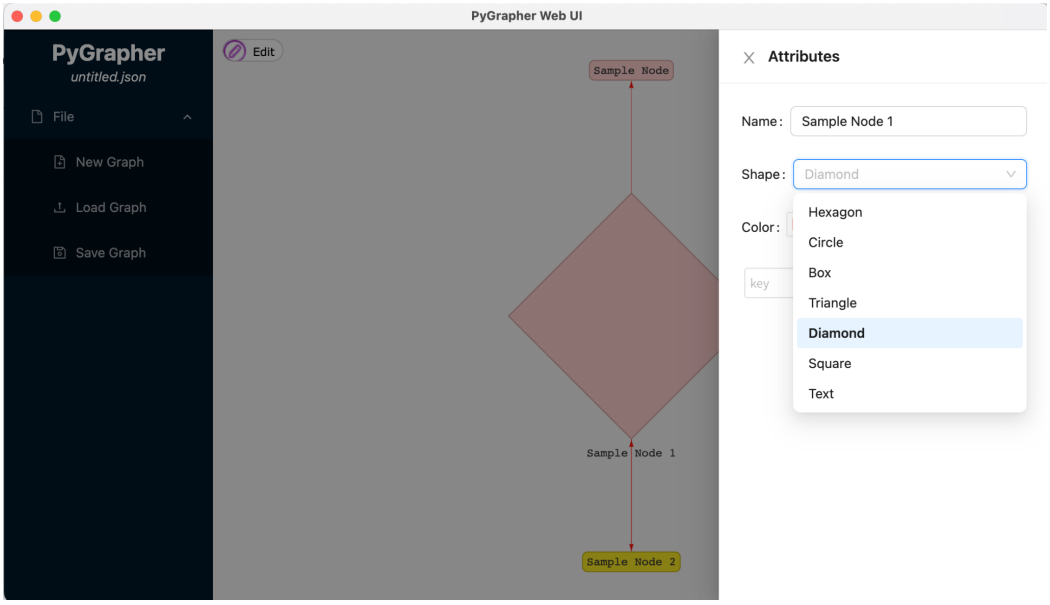

Figure 21. Change the shape and/or Label of the Nodes

The interface updates immediately on change of the Shape attribute or the Label attribute, more shapes can be added later based on functionality provided by vis.js. This interface is also programmed in such a way that it is very trivial to add more elements in the future.

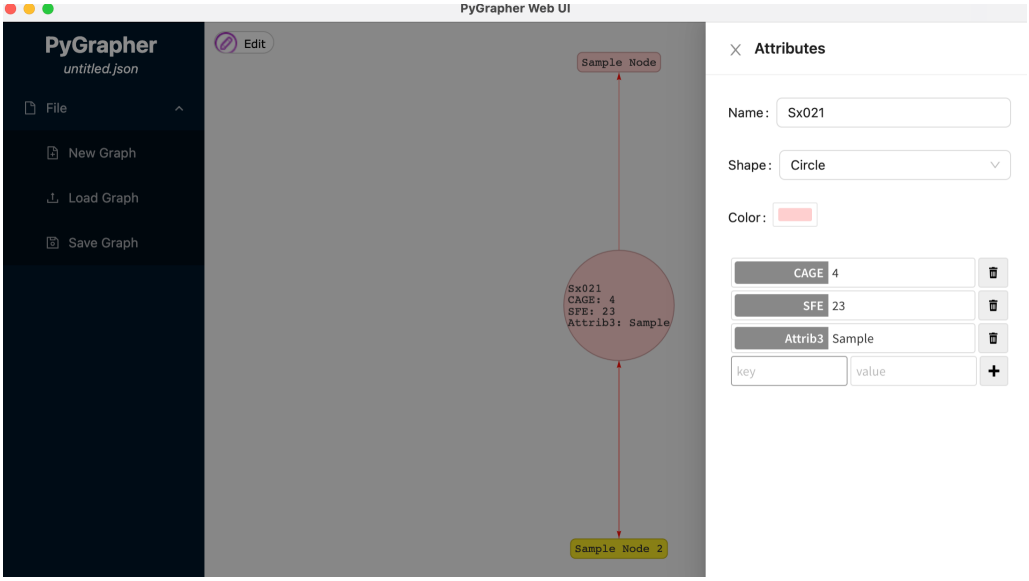

Figure 22. Key/Value pair attributes

6. TESTING

The testing strategy opted for this project was **Manual.** In this strategy, we manually go through known scenarios of the application by simulating real user interactions with the application. The behavior was validated with the given requirements to identify potential defects or issues.

Here is the list of some of the top level tests that were done on the system:

Platform Compatibility Testing: Since the application can be used across any platform. The strategy here included to test the final .dmg (MacOS), .exe/ (Windows), .bin (Linux) manually and ensure that the application works as expected.

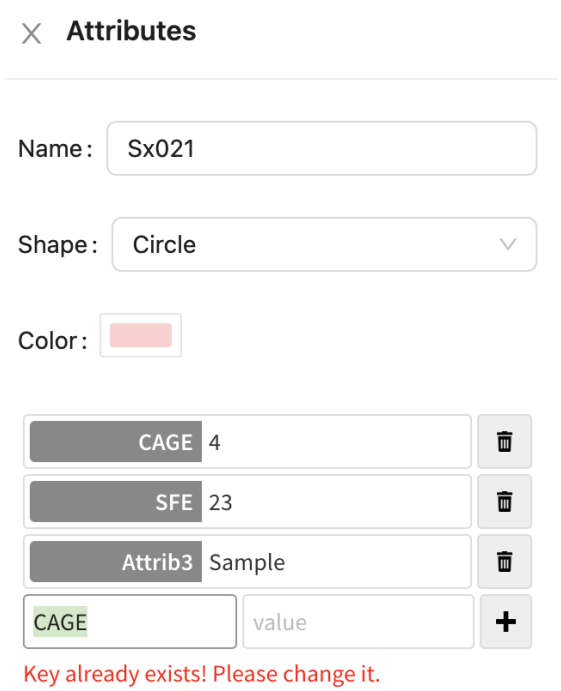

Figure 23. Example of an issue found in functionality test

UI and Functionality Testing: This test was done to verify that all the UI functionalities work as expected and error scenarios are taken into account and the issue is highlighted to the user in a timely manner.

Performance and Load Testing: This test involved largely if this system was capable of handling a large number of nodes and the system requirements (RAM/CPU) were within reasonable limits.

Minimal Automated Test: The system also involves some base tests that were created with Jest. Jest is a framework that is used to test react components. This was out of the scope of this application but it was still added to have the ability to extend it into more test cases in the future as the need calls for it.

7. RESULTS AND CONCLUSIONS

In conclusion, PyGrapher is a powerful GUI based graph management application that efficiently handles large scale graph data. It is extensible enough to be used across many use cases (including evidential reasoning with the internal Capri model). The development of this tool based on multiple technologies like Electron, React and the react-graph-vis library had provided a familiar and easy user interface compared to the existing tools available. PyGrapher is/was developed as a web-based an an app-based application, this project involved building the app-based application

The multi-platform support of the GUI ensures that this application can run across multiple operating systems and is easy to install and distribute. The innovative design and error highlighting capabilities of this system will contribute to its usability across a wide range of users.

8. FUTURE WORK

- Add support for multiple encoders and decoders for graph export. The ability to export this graph into popular format like gephi/d3 will further enable extensibility of this model
- Add more automated extensive unit and functional tests
- Additional graph manipulation features such as graph clustering, subgraph extension and community detection
- Integration with external data sources such as APIs or connectors to popular graph databases of data formats
- Collaborate and sharing features such as real-time collaborative editing and version control

REFERENCES

[1] Lowrance, John D. Thomas D. Garvey, and Thomas M. Strat (1989). A Framework for Evidential- Reasoning Systems. AAAI-86, pp. 896-903.

[2] Shafer, Glenn (1976). A Mathematical Theory of Evidence. Princeton University Press.

[3] Shafer. (2016). Dempster's rule of combination. International Journal of Approximate Reasoning, 79, 26–40. https://doi.org/10.1016/j.ijar.2015.12.009

[4] KARP, LOWRANCE, J. D., STRAT, T. M., & WILKINS, D. E. (1994). The grasper-CL graph management system. LISP and Symbolic Computation, 7(4), 251–290. https://doi.org/10.1007/BF01018612

[5] Rapley, Bellekens, X., Shepherd, L. A., & McLean, C. (2018). Mayall: A Framework for Desktop JavaScript Auditing and Post-Exploitation Analysis. Informatics (Basel), 5(4), 46–. https://doi.org/10.3390/informatics5040046

[6] Flatscher, & Müller, G. (2021). Employing Portable JavaFX GUIs with Scripting Languages. Central European Conference on Information and Intelligent Systems, 333–341.

[7] Tyson. (2021). Tailwind CSS: Learn the joys of functional, responsive CSS. InfoWorld.com.

[8] Sheiko. (2017). Cross-platform desktop application development, Electron, Node, NW.js, and React : build desktop applications with web technologies (1st edition). Packt.

[9] Ali. (2013). Instant node package manager: create your own node modules and publish them on npm registry, automating repetitive tasks in between (1st ed.). PACKT

Publishing.

[10] Adam Boduch. (2018). React 16 Tooling. Packt Publishing.