Politécnico do Porto
Instituto Superior de Engenharia do Porto

Circuit Editor

Design, implementation, and integration into the *U=RIsolve* web application

Pedro Henrique Nunes Morim

LEEC
Degree in Electrical and Computer Engineering

Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto

April, 2023

This report partially satisfies the requirements defined for the Project/Internship course, in the 3rd year, of the Degree in Electrical and Computer Engineering.

Candidate: Pedro Henrique Nunes Morim, No. 1180798,
1180798@isep.ipp.pt

Scientific Guidance: Prof. Mário Jorge de Andrade Ferreira Alves,
mjf@isep.ipp.pt

Scientific Co-Guidance: Prof. André Teixeira da Rocha, anr@isep.ipp.pt

Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto
Rua Dr. António Bernardino de Almeida, 431, 4200–072 Porto

April, 2023

Abstract

Students tend not to use the U=RIsolve app due to its heavy reliance on the QUCS simulator. This app proposes solving that issue by integrating more profoundly with U=RIsolve. The main objective was to create a Web App capable of importing, editing, and exporting circuit schematics using an unprecedented JSON data model. This project should theoretically increase the students' adoption of U=RIsolve, although it can only be genuinely tested after deployment.

The app was built using the MERN stack, and its frontend was built almost entirely in-house. This project has also made some contributions to the community since it created the JSON model, a new way of saving circuit schematics' data, and also, the creation of a schematic editing interface in React since none were as complete as this one.

Keywords: Web App, Electrical Circuit Diagrams, JSON, Netlist, MERN Stack, MongoDB, ExpressJS, React, Node.js, PWA, online/self-learning tools, U=RIsolve web app

Resumo

Os alunos tendem a não usar a aplicação U=RIsolve devido à sua forte dependência do simulador QUCS. Esta aplicação propõe resolver esse problema ao ser integrando mais profundamente com U=RIsolve. O objetivo principal foi criar uma Web App capaz de importar, editar e exportar diagramas elétricos usando um modelo de dados JSON inédito. Este projeto deve teoricamente aumentar a adoção do U=RIsolve por parte dos alunos, embora só possa ser genuinamente testado após a implantação.

A aplicação foi construída usando a pilha MERN e o seu frontend foi construído quase completamente internamente. Este projeto também trouxe algumas contribuições para a comunidade: desde a criação do modelo JSON, uma nova forma de guardar os dados dos esquemáticos de circuitos, e também, a criação de uma interface de edição de esquemáticos em React já que nenhuma era tão completa quanto esta.

Palavras-Chave: Aplicação Web, esquemas de circuitos elétricos, JSON, Netlist, Pilha MERN, MongoDB, ExpressJS, React, Node.js, PWA, ferramentas de ensino/autoaprendizagem, aplicação web U=RIsolve

Contents

List of Figures

Glossary

Backend

Backend is the part of an application that is not directly accessed by the user. Used for storing and manipulating data, typically with a database and API.

Data Model

A Data Model is a structure that organizes elements of data and defined how they relate to each other and their properties.

Docker

Docker is a open platform that uses virtualized operating systems to deliver packages called containers. It allows developers to develop their applications without worrying about the infrastructure.

Draw2D

Draw2D is a library built with jQuerry that allows developers to create diagrams. It was planned to used this library to create the circuit schematics diagrams of this project, but those features were not fully developed and finished.

Frontend

The Frontend is the code that defined the interface that the user interacts with. Usually be defining the layout of text, buttons, images, and other graphical elements

jQuerry

jQuerry is a JavaScript library that was created in 2006 that aids with the creation of websites. Even nowadays it still is one of the most used libraries in the world.

Kanban Board

A Kanban Board is a physical or digital project management tool that helps visualize work, limit work-in-progress, and maximize efficiency by separating tasks into cards and columns

Linting

Linting is the act of using an external piece of software to analyse the source code and scan for potential problems.

Software Deployment

Software Deployment is the entire process of delivering the software to its intended users.

Version Control Software

Version Control Software is an external program that saves every modification made to the codebase so that, if a mistake is made, it is possible to compare it to previous versions and pinpoint the error while minimizing the amount of work to be done by the developers.

List of Acronyms

API	Application Programming Interface
CDN	Content Delivery Network
CSS	Cascading Style Sheets
DOM	Document Object Model
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
ISEP	Instituto Superior de Engenharia do Porto
JSON	JavaScript Object Notation
MERN	MongoDB, Express, React and Node.js
noSQL	Non Structured Query Language
ODM	Object Document Mapping
ORM	Object Relational Mapping
PWA	Progressive Web App
REST	Representational State Transfer
SEO	Search Engine Optimization
SQL	Structured Query Language
UI	User Interface
UX	User Experience
WIP	Work In Progress

Chapter 1

Introduction

1.1 Contextualization

The Inverted Classroom Model Model [1] indicates that students should try to find information and learn outside the classroom, hence leaving the class time for higher-order problems. In the Electrical Engineering field, this means that students should learn about circuit analysis methods by themselves, leaving the most complex circuits to the class. This feat is monumental since most of the friction in learning this branch of engineering comes exactly from this subject.

Therefore, some students pioneered a new web app called U=RIsolve at Instituto Superior de Engenharia do Porto. This app allowed users to learn, quickly and intuitively, the different circuit analysis methods by themselves by just uploading a Netlist of the desired circuit.

1.2 Problem Identification

However, one of the main downfalls of U=RIsolve [2] is that it heavily relies on third-party circuit simulators, specifically the one that the students already use a lot: QUCS [3]. This drawback added an extra layer of complexity, which negatively impacted the user experience and thus reduced the adoption of this app by the students. The user needed to create a circuit schematic in QUCS, generate its Netlist, download it, upload it to U=RIsolve, and only then could U=RIsolve do its job. This project comes to solve that exact issue by entirely removing the middleman.

1.3 State of the Art

The primary reference and target for this project were the "Multisim" simulator [4]. It is web-based and has an intuitive interface. It would be ideal to have an equivalent experience on this app, although that would be quite hard since there is a vast amount of concealed logic behind the scenes.

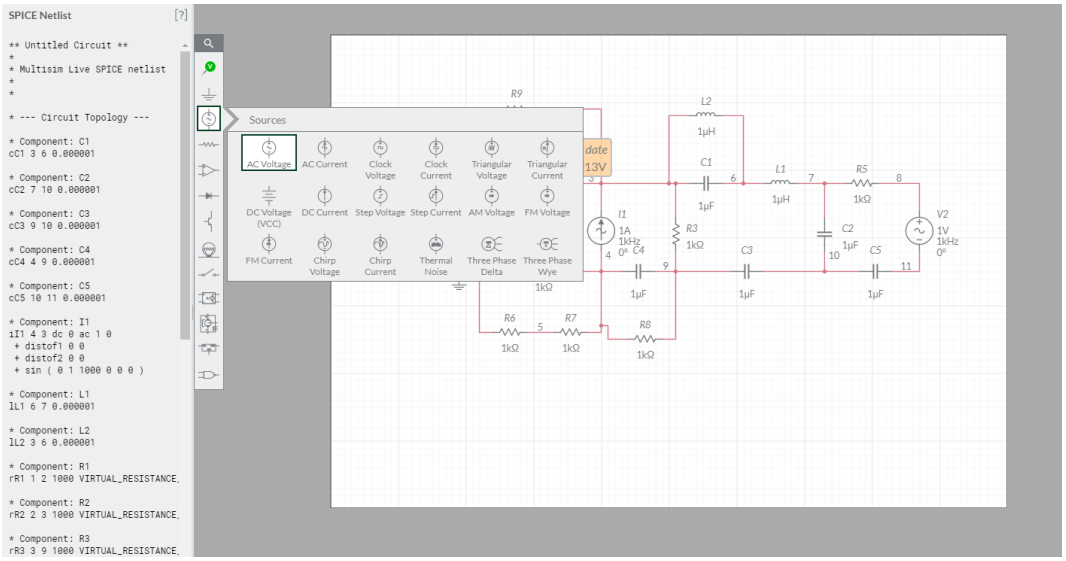


Figure 1.1: Multisim's interface

Another app that, although very similar to Multisim, looks much simpler is Spinning Numbers' "Circuit Sandbox" [5]. Even with its apparent simplicity, it is still more complete than this project can accomplish since it contains an entire custom-built simulation engine. It is also not visually appealing and does not make use of the newer Frontend development technologies, which is something we will hopefully improve in this project.

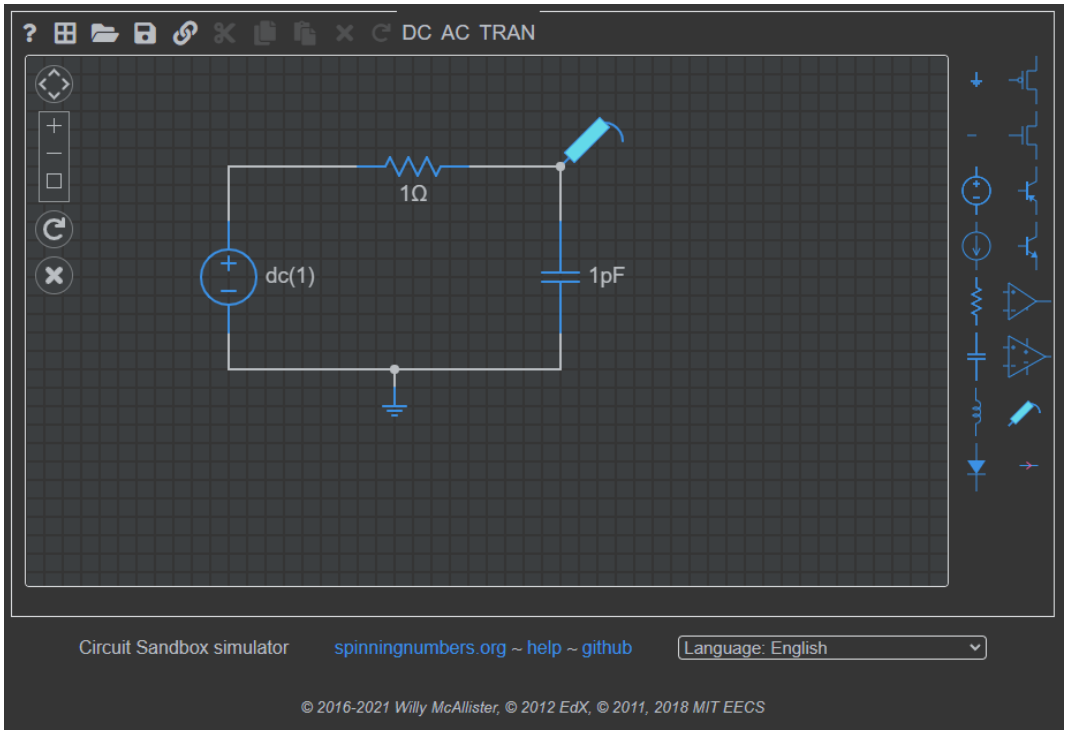


Figure 1.2: Circuit Sandbox' interface

Outside of this engineering field, some apps follow the same principle for the mathematics field. The most famous examples are "Symbolab" [6] and "Wolfram Alpha" [7]. They allow its user to learn how to solve complex equations by showing them the process, or as advertised by Symbolab: "Step by Step calculator." Symbolab also allows users to do group study sessions and quizzes, and Wolfram has since expanded their product to cover other fields like Physics and Chemistry.

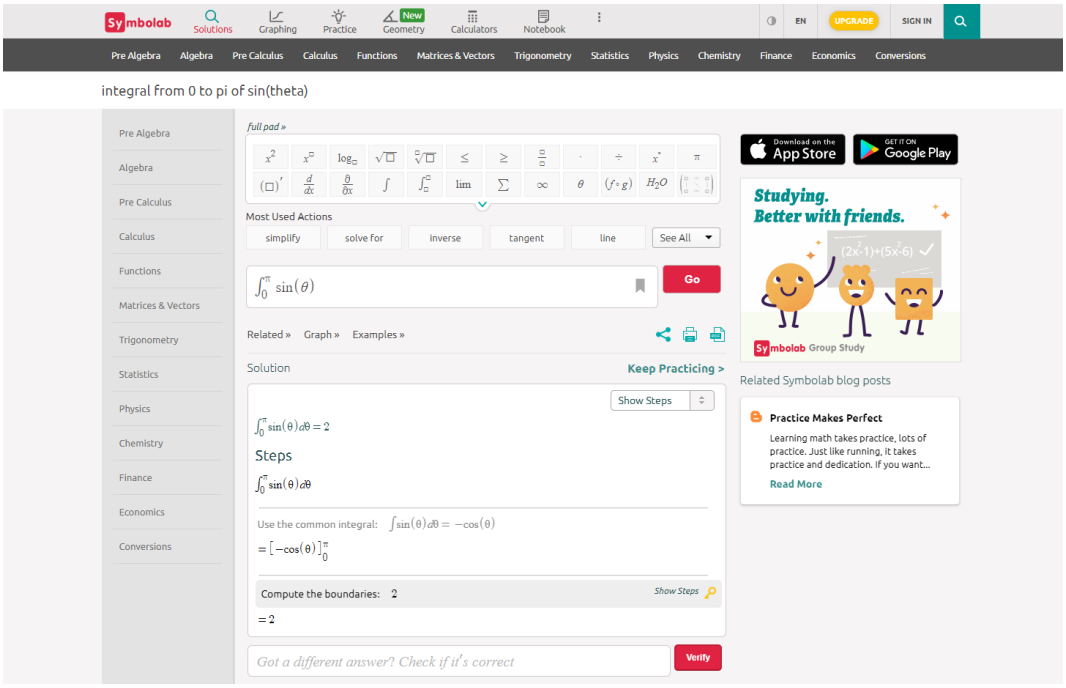


Figure 1.3: Solving a basic integral in Symbolab

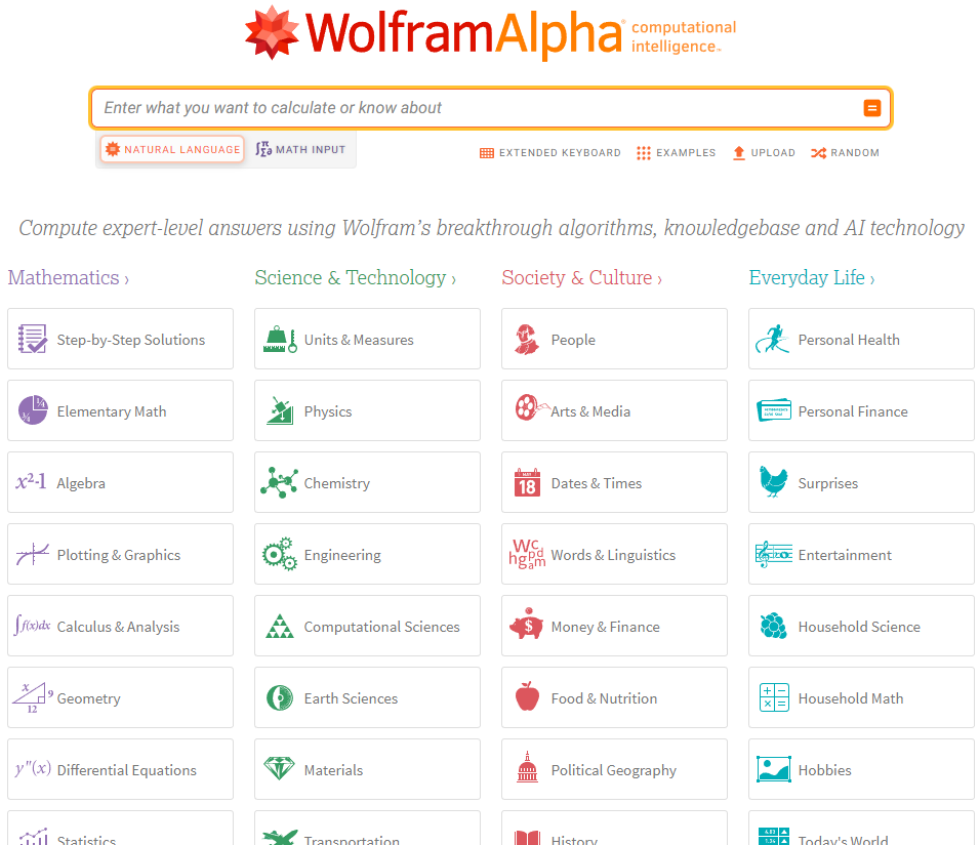


Figure 1.4: Wolfram Alpha's interface

A more traditional way of this learning/teaching method is to take online courses. One such implementation is "edX" [8]. They have partnered with big institutions like MIT and Harvard to provide their users with the best possible courses. Enrolling in a course is free, although a one-time fee is required to get a certification. This freemium model is helpful for students already enrolled in a university.

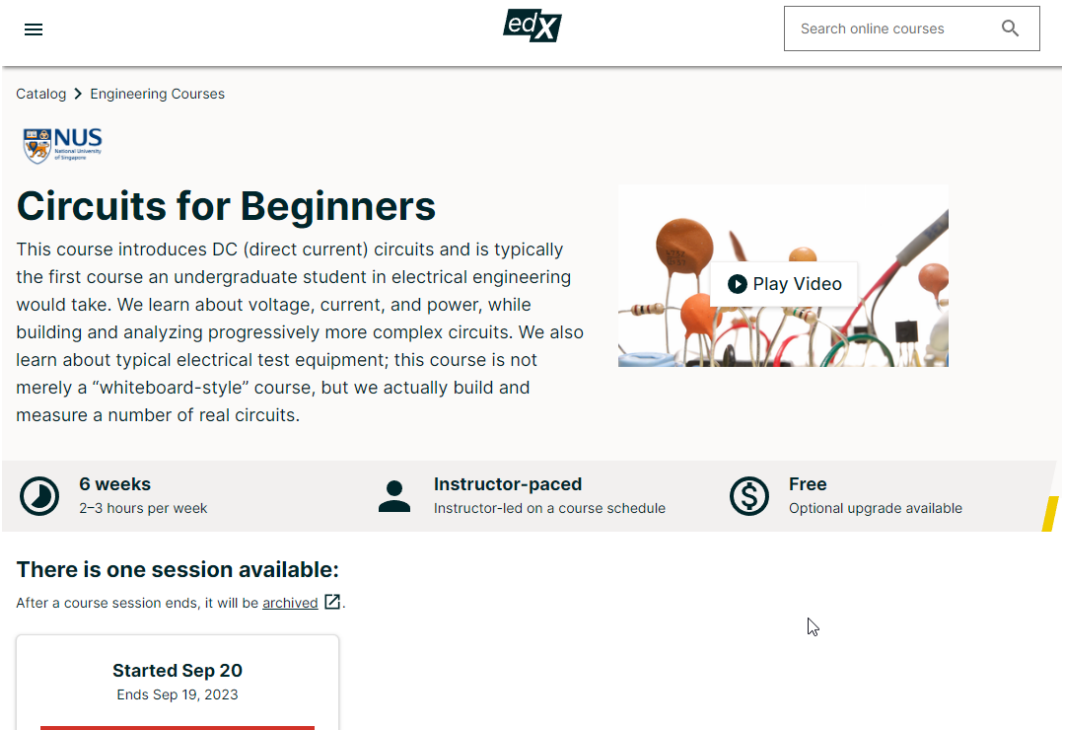


Figure 1.5: edX's "Circuits for Beginners" course

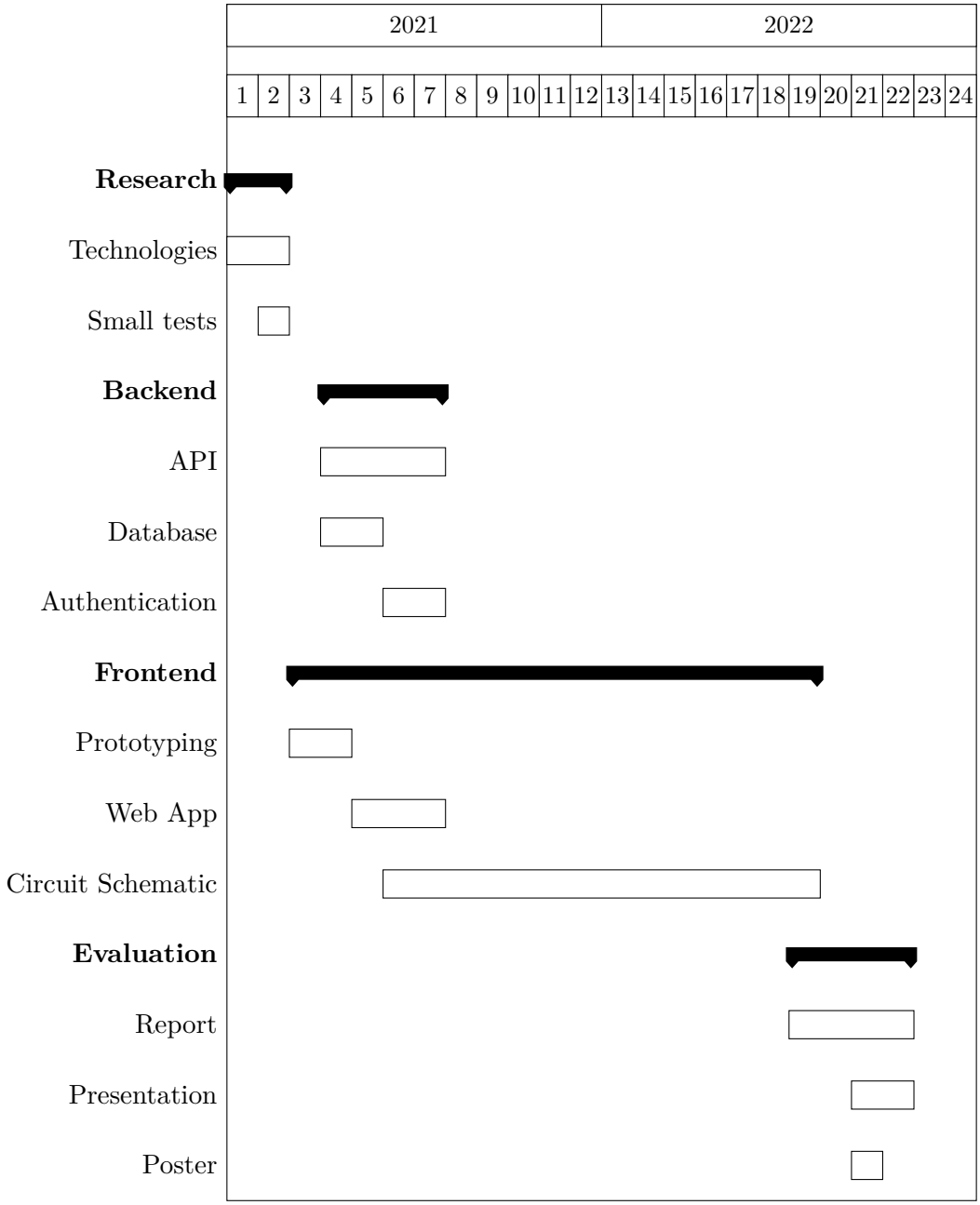
1.4 Objectives

The ideal outcome of this project is to have a Web App capable of creating circuit schematics and exporting their analytical model directly to U=RIsolve. The main objectives/tasks of this project are:

- Dynamic editing of the circuit parameters;
- Labeling of the Components, Nodes, Connections, and Meshes;
- Responsivity of the app for multiple screen sizes;
- Moving objects through a Drag&Drop action;
- Adding, deleting, and editing objects to the canvas;
- Exporting a cropped image of the canvas;

- Undo/Redo actions;
- Objects for real-time graphing of parameters;
- Importing and exporting the analytical model of the circuit in the JSON format;
- Generation and download of the circuit in the Netlist protocol for backward compatibility;
- Tutorial/Documentation for future users and developers;

1.5 Calendarization



1.6 Report Structure

There will not be a deep dive into the code in this report since this is entirely a software project, and the code speaks for itself; instead, we will try to explain how the different modules connect from a higher-level perspective.

Two chapters are the core of this report: Circuit Schematic and Software Architecture and Implementation. The former indicates the implementation of the library that is capable of graphing and editing a circuit schematic. The latter shows the

architecture and implementation of the Web App with which the user will interact. Besides those, two smaller chapters show how we managed the development and deployed the app.

Chapter 2

Core Functionality and Implementation

A complex application like this one contains numerous features, most of which will go unnoticed by the user due to their practical and intuitive design. Each feature is an independent module that allows the developer to replace or improve it. All modules are combined to create the app's complex circuit editing section. We will succinctly explain some of these components below; for a more in-depth analysis of their implementation, the code is public on GitHub.

All sections from this chapter besides 2.5 and 2.6 were not planned to be developed manually. This project was devised with the Draw2D [9] library in mind, which is a jQuerry [10] package that would take care of most of the circuit editing features. Although, it was later found that this library had bugs, and its creator had stopped its development. Other packages were searched for, but none were complete enough to be used in this application. Hence, another year was added to this project's deadline to build all the needed features.

2.1 Undo/Redo History

Firstly, the undo/redo feature was one of the first to be developed. It is essential to get a system like this working before adding all the other ones, or it risks having to do considerable amounts of refactoring later since many other features rely on it.

To store the states of the circuit, we used a stack. This data structure is ideal for this feature since it follows the "First In, Last Out" principle. We also implemented it to automatically decide if it should add a state to the stack or not based on the type of change that the user performed.

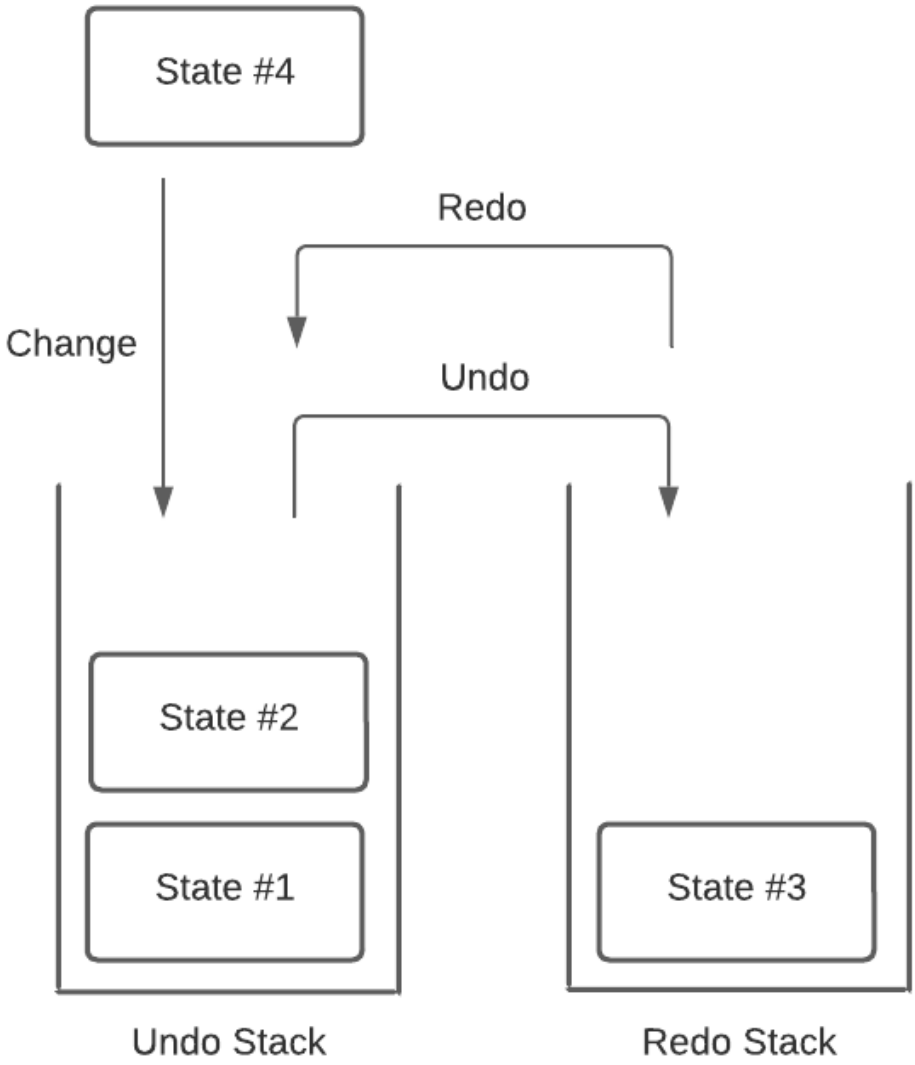


Figure 2.1: Diagram of the undo/redo algorithm

2.2 Edit Actions

No circuit editor is complete without some basic actions: adding, deleting, and editing components. Hence, before specifying any specific actions, we took time to build a robust API capable of executing the mentioned actions in a way that performs

competently with the previously mentioned undo/redo system. When adding a new component, this manager will automatically fill the default values for any missing parameters. When deleting, it will remove it from the model and any connections that depend on it. When editing, it provides an intuitive API inspired by React's useState Hook, which allows the developer to edit any property of the object easily - this is useful to, for instance, edit the coordinates of the component when dragging.

2.3 Drag and Drop

Since the browser has no native support for drag functionality, we used a library called react-draggable [11] to allow the user to drag components around the canvas. This library uses some native DOM events to detect the dragging of an element. This library is exceptionally robust; without it, we would need to write a lot more code. Thus, we created a generic draggable component. Then, all the movable canvas elements - such as components, nodes, and labels - were then merely extended from it. Hence, we reduced the logic behind editing the coordinates of the elements into a simple and reusable component.

2.4 Canvas Elements

Labels are simple Graphical User Interface (GUI) elements that smartly display a name, value, and unit. They can, then, be attached to any component, node, or connection to represent their name, value, or unit (if available). They can be customized in the internal properties of their parent element.

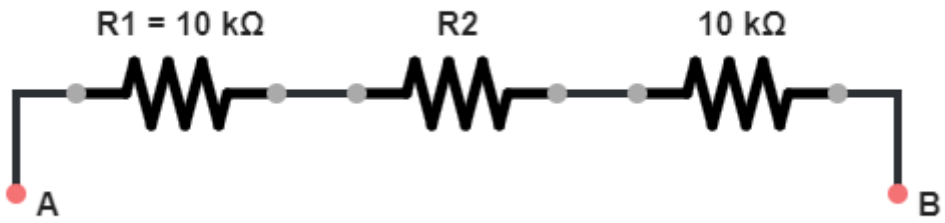


Figure 2.2: Example of the different label variations

Each component can have as many connection ports as needed by just providing an array of ports to be displayed, each with relative coordinates to their parent. We manually drew their electrical symbols in Figma [12] in a way that replicated the QUCS simulator's ones while maintaining a modern and minimalistic design. These

were thoroughly reviewed by Prof. Mário Alves and then adjusted as they need to be correct for the students to properly learn their official symbols.

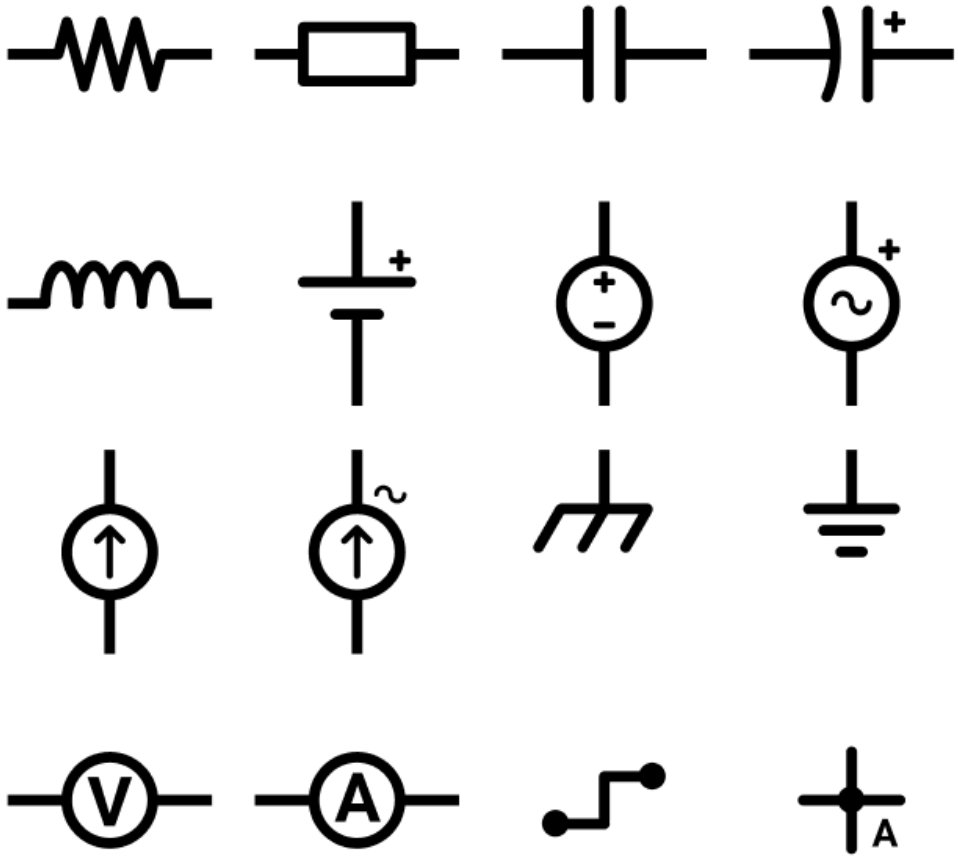


Figure 2.3: The symbols designed on Figma

Connections are the weak link of this entire app. To create them, we used a library called react-xarrows [13], which is quite lacking. It contains many features for creating diagrams, but unfortunately, not the ones that are useful for circuit schematics. For instance, there is no ability to drag the connections; this library only works by providing the start and end points of the connection, and it will then automatically draw it without giving the developer much control over it. Replacing this library with a custom one is a must for future development.

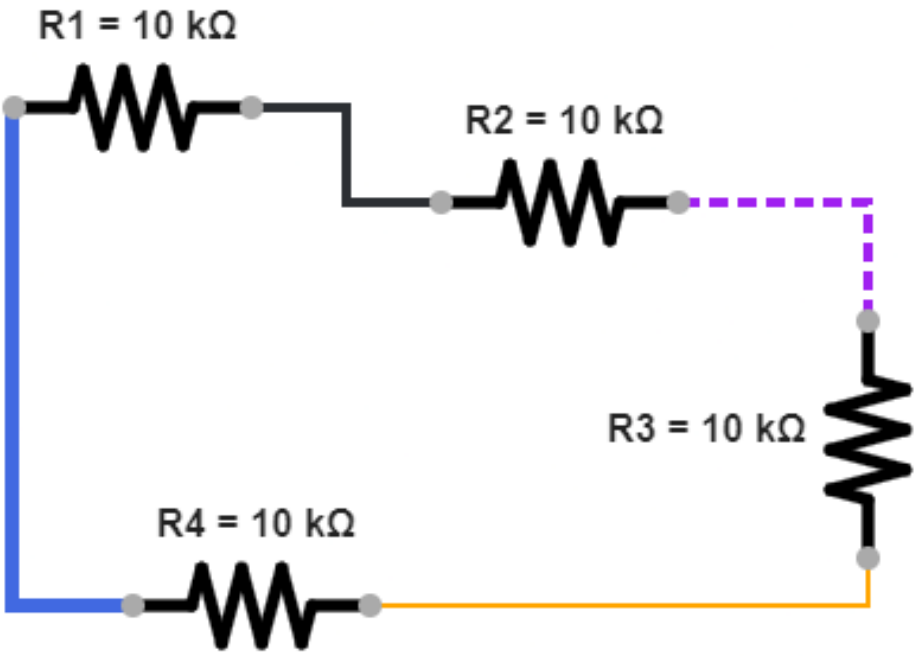


Figure 2.4: Example of different customizations of connections. The dashed connection can have an animation that obviously cannot be displayed in this image

Nodes are merely a hub for connections, just like components' ports, although nodes are not limited to any amount of connections, unlike their counterpart. Just like ports, they will be marked in red if there are not enough connections to them; this helps the user find possible problems with the schematic.

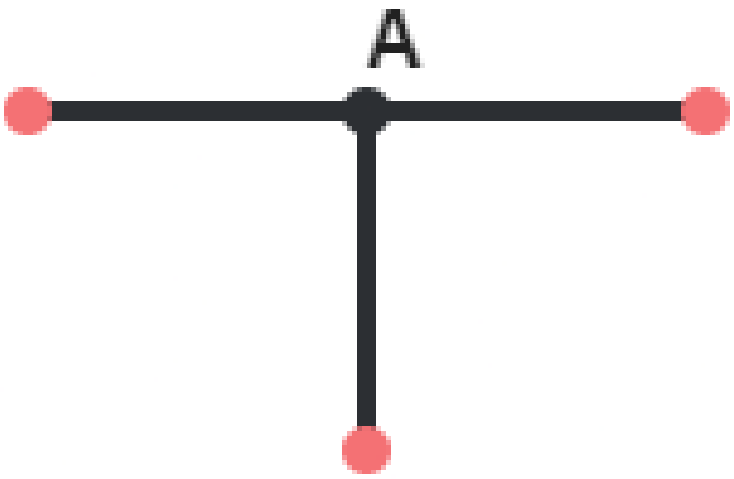


Figure 2.5: Example of a node with its label

These three elements, each with their internal label, can be combined to create basic circuit schematics. They cover most, if not all, circuits that students study in their first year of electrical engineering. Although this can be extended to cover more advanced circuits.

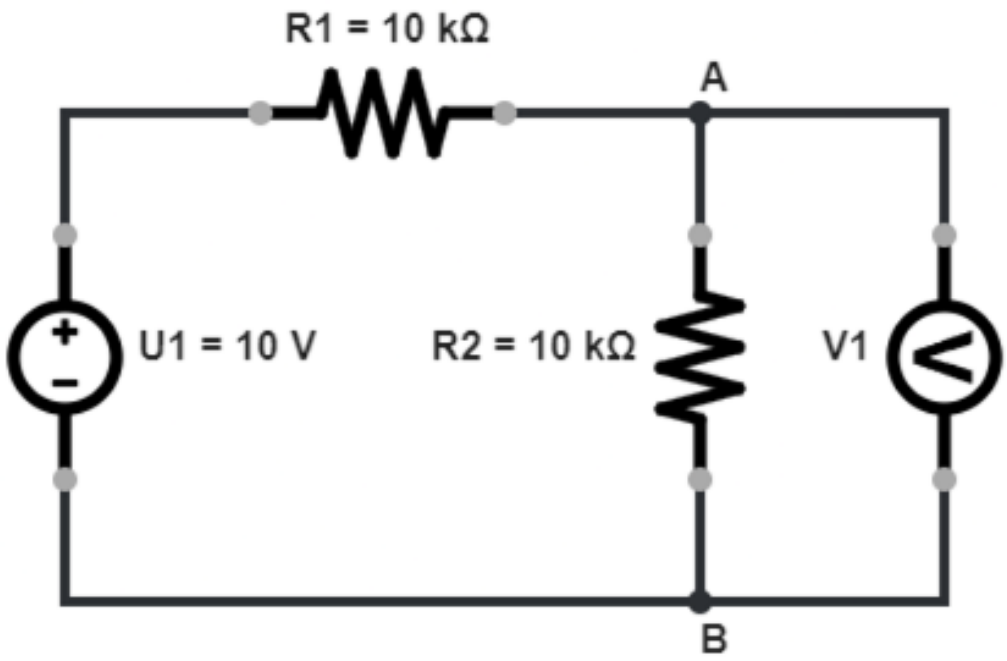


Figure 2.6: Example voltage divider circuit created with Circuit Editor

2.5 Data Models

A Data Model is a structure that organizes elements of data and defines how they relate to each other and their properties. One of the core objectives of this project was for the user to be able to export the circuit into U=RIsolve. Accordingly, we developed a module capable of converting between models and performing import and export actions.

It is possible to represent circuits in many distinct ways, each with its advantages and disadvantages. This project uses two data models that are complete opposites in terms of their composition. One is detailed and heavier, while the other is trivial and, thus, lighter.

2.5.1 JSON

Since no open text-based data models have complete data about the schematic, we idealized a JSON [14] schema capable of doing so. This model describes each component, node, connection, and their respective information individually. This data includes their coordinates, label, color, and much more - detailed documentation of this data model can be found in this report's appendix [15]. We then implemented this schema on the app and applied some minor refinements that aided its development. This sort of data could fit in a smaller amount of memory if encoded in a

binary format, although that would not be ideal since it would complicate its parsing, saving, and editing - both manual and automatic. The circuit presented in 2.6 is the visual representation of the enormous JSON file in the appendix A. A more in-depth explanation of how this model works is present in the appendix B.

To import and export, the JSON model is quite simple; the app's state revolves around that model. Thus, for importing: load the JSON into the app's state; and for exporting: download the current state as a JSON file. The user can also download a cropped image of the circuit schematic. We built this feature using the html-to-image library [16]. Its developments had some complications regarding the rendering of the connections; just another reason future developers should replace the current connections library.

2.5.2 Netlist

The Netlist [17] is one of the most superficial types of circuit-related data models. It merely indicates how components connect by using virtual nodes. This data is enough to simulate the circuit's behavior, though it does not hold any data that describes how to lay out the schematic. This trait makes it unattainable for the user to load a formerly saved schematic. This Netlist comes in a human-friendly text file rather than a binary file. The circuit presented in 2.6 would be equivalent to the following Netlist:

```
1 R:R1 _net0 A R="10 kOhm"
2 Vdc:U1 _net0 B U="10 V"
3 R:R2 A B R="10 kOhm"
4 VProbe:V1 A B
```

Each line of the Netlist represents a component, connections, and internal values. For instance, the first row indicates that there is a resistor with the name $R1$ connected between a virtual node named $\_net0$ and a real node named A. After that, it shows all of the component's properties: in this case, the resistor has a value of $10kOhm$. Other properties would show there as well, for instance, the resistor's internal temperature, or the internal resistance of a capacitor; although such properties are not yet implemented.

It is impossible to import this model because of its lack of data, so we only created a module for converting it from the JSON one. The algorithm is quite simple. Iterate through all components, display their internal data into a string, and append all those strings as a multiline text file for the user to download. It is to note that the Netlist is generated in real-time as the user performs edits to the schematic; this allows the students to learn the nuances of this model more efficiently by comparing how changes in the circuit affect the output.

2.5.3 Comparison

Each model has its advantages and disadvantages. For this app's purpose, the JSON model is an absolute need since the user needs to be able to save and load the circuit. The same cannot be said about the Netlist since it only contains information about how components are connected, not how the user placed them on the canvas, thus making the display of a loaded circuit impossible.

Why use the Netlist model, then? Previous students have built U=RIsolve with the said model in mind, so all its implementations heavily rely on it. Therefore, the mentioned model earlier requires support while it gets gradually deprecated and eventually phased out.

To make the comparison of both methods more evident, we only need to look at the representation of a simple component like a resistor. For example, in the Netlist a resistor can be represented by just one line: "R:R1 _net0 A R="10 kOhm"". In contrast, the JSON model needs a total of 25 lines for the same component as can be seen in appendix A.

2.6 Integration with U=RIsolve

Having to download the schematic to upload it to U=RIsolve manually is just as cumbersome as using the QUCS simulator. Consequently, it is vital to simplify that process for the user. Unfortunately, browsers do not support POST requests while redirecting to a new website. Therefore, we had to create an alternative way to send the schematic to U=RIsolve.

The solution was sending a standard POST request with the schematic to a backend that would then return the URL to open the new tab. This way, U=RIsolve could look at the unique URL and associate it with the schematic in the backend. At the time of writing, U=RIsolve is only a frontend app; this particularity made testing this feature impossible, although we already prepared all of the code on this project's end for the backend.

Chapter 3

Software Architecture and Implementation

3.1 Overview

This chapter explains the reasoning behind the choice of the technologies used. Together, they form the MERN Stack. This stack is one of the most used stacks in the industry. We chose it because of its simplicity and utility to the developer's future career.

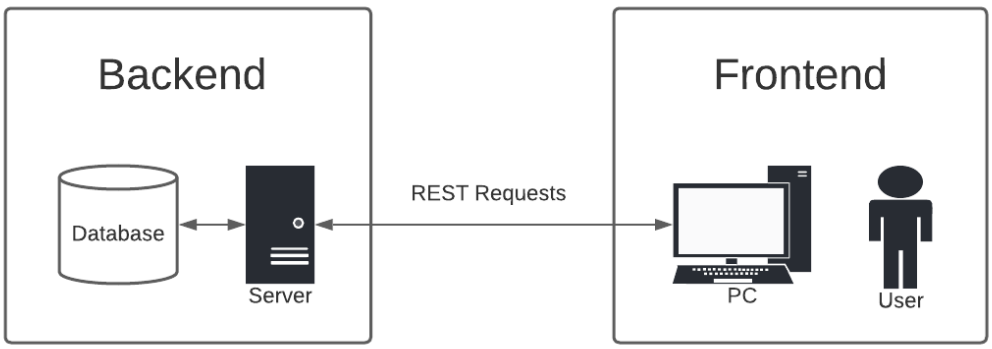


Figure 3.1: Diagram of the architecture of a web app

3.2 Software Stack

As said previously, the chosen stack is called "MERN". This name implies its four leading technologies: *MongoDB*, *Express*, *React*, and *Node.js*. We will be explaining this in more detail in the following chapters. [18]

3.2.1 MongoDB

MongoDB, as the "DB" in its name implies, is a database. It is the most used noSQL database in the current market. Opposing all of the other primary databases, *MongoDB* is not a relational database; in other words, it does not make use of SQL. Instead, it maps all of its data in a JSON-like format. This particularity gives developers exceptional ease of use since JSON is highly compatible with *JavaScript*. [19]

3.2.2 Express

The client needs to communicate with the server and the most common way of doing this is to create a REST API. This allows the client to send, request, or edit data by communicating via HTTP requests. To implement this architecture in *Node.js*, we used the *Express* package that provides the developer with an easy-to-use API capable of creating any REST server application. A different approach to this would be to, for example, use the *Apollo* package to build a *GraphQL* API instead; although, that would be too complex and unnecessary for such a frontend-heavy project. [20] [21]

3.2.3 React

To create our frontend, we used *React* due to its popularity and easy learning curve. Contrary to popular belief, *React* is a library, not a framework; meaning that it does not come "batteries-included", instead, a lot of other libraries need to be used to fully develop the frontend. Facebook created it in 2013, and it still is currently the most used library in this field. As the name implies, it is capable of creating reactive (and responsive) interfaces. Moreover, *React* opts to use a virtual DOM instead of accessing the DOM directly. This particularity makes it so that, after some event occurs, only some components need to be re-rendered instead of the whole page, which significantly increases the performance. It is even more potent if paired with *Next.js*, which allows the server to prerender the page on the server, making the app faster and increasing its SEO in the process. However, for simplicity's sake, this project does not use Next.js. [22] [23]

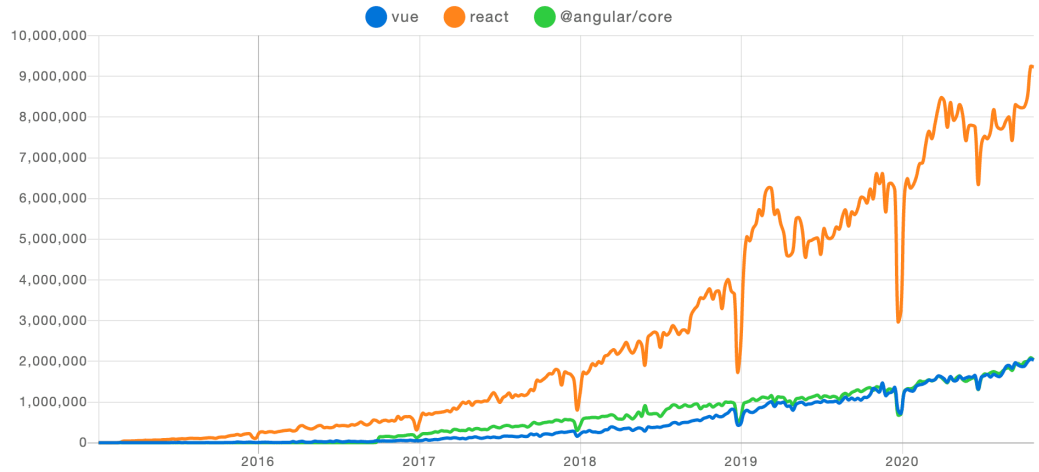


Figure 3.2: Comparison of the number of NPM downloads of the three most popular frontend frameworks/libraries (2021)

3.2.4 Node.js

Node.js has become a staple of the JavaScript ecosystem. It allows code to run on a server instead of a browser. It is mighty, although making a complex app requires some external packages like, for example, the previously mentioned *Express*. This process will enable developers to create the backend in the same language as the frontend, providing a better developer experience since only one language needs to be known or learned by the developer [24]. A more modern alternative to *Node.js* is the brand new *Deno*, both share the same creator. *Deno* supposedly fixes all of the issues that the creator regrets in *Node.js*, although it is still too new to be used in a serious project. Perhaps this project could be upgraded to use *Deno* in the future. [25]

3.3 Backend

3.3.1 Overview

The Backend is the middleman between the frontend and the database, where security is most important. Every time the user needs to interact with the database, that request goes through the backend to ensure the user is authenticated and has permissions for that specific request.

We decided to keep the backend in *JavaScript* instead of opting for other languages like *Python* or *PHP*. Since we also planned on building the frontend in this language, this decision massively accelerated the development process and lowered the learning curve. Therefore, we chose *Node.js* and *Express* to develop our API and *MongoDB* for the database.

3.3.2 Framework

There are many options for backend frameworks: *Express* for *JavaScript/Node.js*, *Django* for *Python*, *Laravel* for *PHP*, and many more. Since we were going with *Node.js*, there were only two main options: *Express* for REST APIs and *Apollo* for a *GraphQL* one. REST is the older and simpler one, while GraphQL is the newer and more complex one. Since this project is not backend-heavy, it was straightforward to choose *Express* as the framework. Even though having a *GraphQL* is extremely useful to big apps, it would be excessively complex for this one. [20] [21]

3.3.3 Database

To save all of the user data, we needed a database. There are many types of them, each one specialized for specific uses. For this project, we wanted a database that could quickly scale and use a document data model since storing the data in JSON format is more intuitive than saving them in SQL. Thus, an obvious choice was to use MongoDB as our database software. [19]

For organizing all of the data, every database needs an Object Relational Mapping (ORM) or Object Document Mapping (ODM), depending if its SQL or noSQL, respectively [26]. Otherwise, the database would accept any unstructured data. The ODM has the purpose of defining the structure and parsing the data to be saved. So, for our ODM, there were two choices: Prisma or Mongoose. While Prisma was more database agnostic and overall robust, Mongoose ended up being the chosen ODM due to its simplicity and larger community since it was the older of the two. [27] [28]

3.3.4 Authentication

Authentication can be precarious to implement due to the different types of protocols. Luckily, a library takes care of all that hassle: Passport.js. This library takes care of all cookies and sessions' logic, leaving only the form validation to the developer. With it, it is possible to implement all kinds of authentication strategies. Still, for this application, we only need the basic username and password combo, which Passport calls the "local strategy." [29] Before we save the user to the database, their password is encrypted using the *bcrypt* library for security reasons. [30]

3.4 Frontend

3.4.1 Overview

Having an intuitive and easy-to-use app is essential while keeping a beautiful interface. The Frontend is the code the user interacts with; it is the bridge between

humans and machines. It consists of an intuitive and reactive interface for the user to perform complex actions effortlessly. Its design is divided between UX and UI. The former describes the user's experience with the app, and the latter the interface. Most of the project's complexity and code are located in the frontend. [31]

3.4.2 Framework

Even though *React* is very powerful, it is still just a library, not a framework. Meaning that the developer must choose other libraries to fill in the gaps. For example: to create different pages we used react-router-dom [32], and to execute form validation we used react-hook-form [33].

3.4.3 Styling

We tried to achieve a modern and minimalistic design. To take care of all the styles of the page, we could use vanilla CSS, but that method, even though it provides as much customization as we could need, takes a long time to finish development. Thus, we opted to use a library that helps in that regard. There are many libraries, each with its approach to solving this problem. Each has its own set of advantages and disadvantages. Undecided on which to use, we built a small prototype with some of them.

Initially, we built the said prototype with Tailwind. This recently created library (2017) has much hype surrounding it due to its innovative approach. Instead of providing premade React components, it grants premade CSS classes that can add to any React component. It is compelling, especially if paired with styleless premade React components like those provided by Headless UI (which was created by the Tailwind team). With this library, the developer can have the customization level of vanilla CSS with a fraction of the development time. [34]

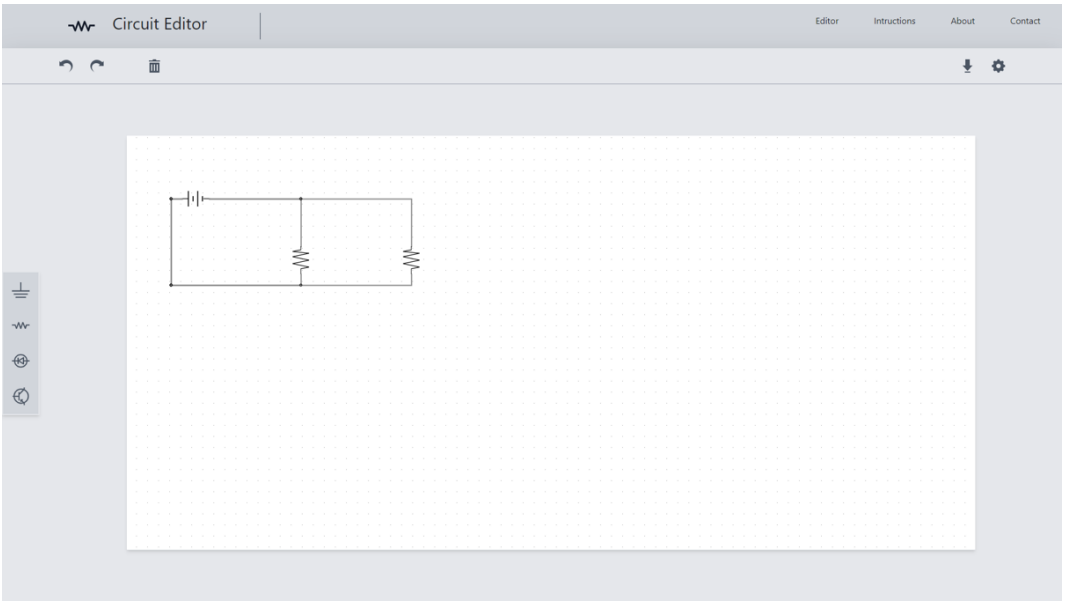


Figure 3.3: Prototype of the interface created with Tailwind

Even though Tailwind is a great library, we decided to use a simpler one that comes with premade components since the main objective of this project is the circuit editor and not the quality of the interface. Therefore, we tried out React-Bootstrap, the implementation for React of the widely famous Bootstrap library, which Twitter created in 2011. Still, this library only contains low-level components; this is more than enough for generic websites or blogs, but it is not quite enough for something as complex as this project. [35]

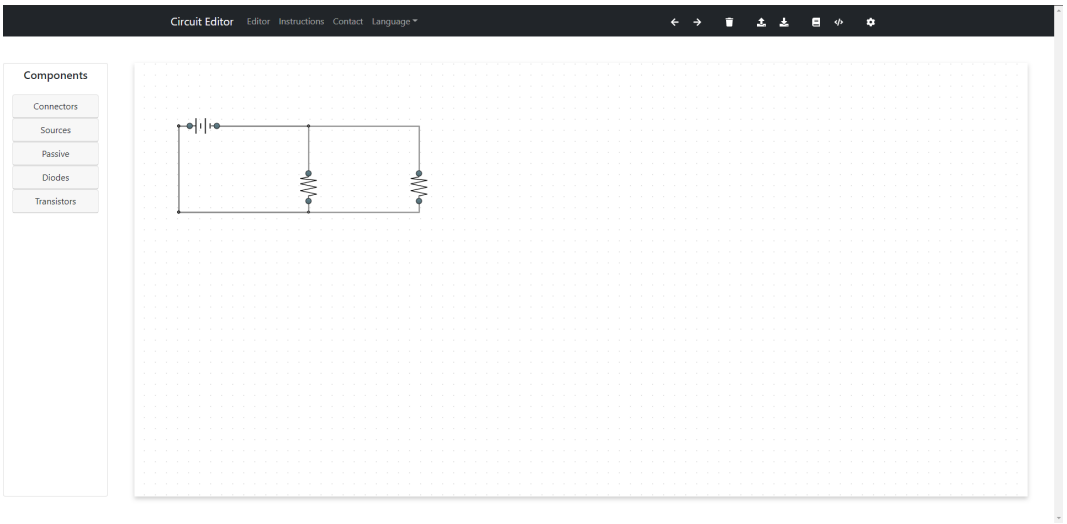


Figure 3.4: Prototype of the interface created with Bootstrap

With the previous tests, we knew we needed a component-driven library with some high-level components that would fit our needs. Hence, we searched for some

libraries that would suit those parameters. We found plenty of them and ended up using Material-UI. Out of all the ones we found, Material-UI has one of the best-looking designs because, as the name implies, it follows Material UI's design principles, which Google created for Android. As stated before, an obvious drawback of this kind of library is the lack of customization. Still, we were happy with the default look of Material-UI for this kind of application. [36]

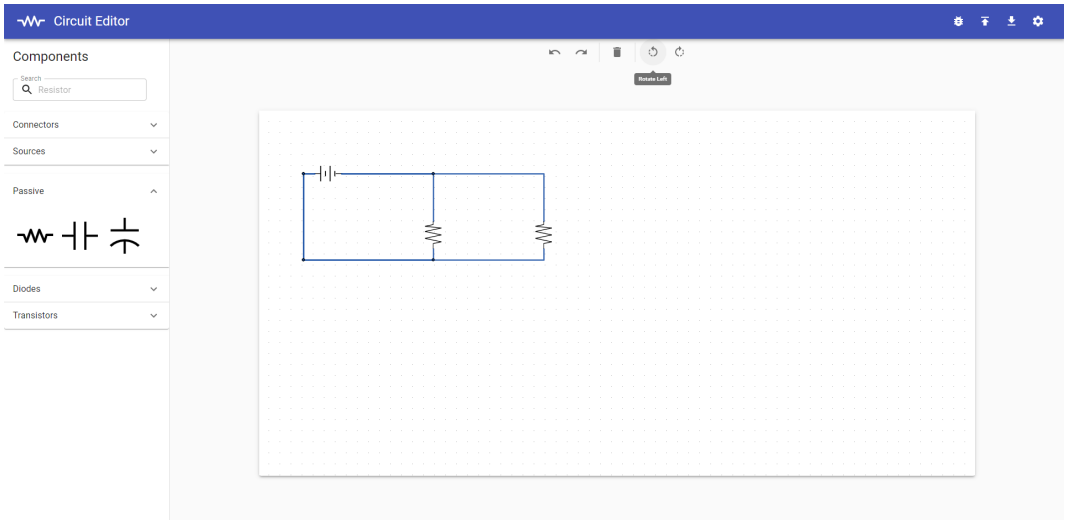


Figure 3.5: Prototype of the interface created with Material-UI

3.4.4 Progressive Web App

A Progressive Web App (PWA) is a web app that meets specific criteria, thus allowing it to be downloaded and function nearly as a native app. It can even work entirely offline in instances without a backend since the frontend code is used as the app's source. We took the time to do this conversion since it was a relatively simple setup with React.

To execute the said setup, we created a manifest file that describes the app's icons, name, description, theme color, and other properties. Then we created and registered a service worker, the code that will take care of routing, caching, accessing the file system, Bluetooth, and other complex tasks that only native apps contain - in our case, only routing and caching. Lastly, we executed some performance improvements, such as converting all images from PNG and JPEG to Google's new WEBP format.

This feature could be improved by increasing the app's offline functionality. For instance, circuits could temporarily be saved locally if offline and then automatically uploaded to the database when connections are returned.

Chapter 4

Project Management

Due to the high complexity of this project, we rigorously organized and structured it using multiple tools and approaches. The tools we used are far from perfect since they are free of charge, but we feel they are more than enough for a single developer project like this. This approach was employed to help this initial development phase and to aid future developers by intuitively displaying what needs to be improved.

4.1 Management Tool

We chose Trello [37] as our primary tool and the single source of truth. This application follows the basis of the Kanban Board, which is a board divided into multiple lists where each list contains cards that define a task. These cards can be moved from list to list and fully customized: such as adding due dates, checklists, descriptions, comments, and even integration with GitHub.

Of the multiple lists we used, the most important ones were: Todo, WIP, and Done. Every time a new task was defined, we would create a detailed card for it in the "Todo" list, when the task was being worked on it would be dragged to the "WIP" list, and when finished it was finally moved to the "Done" list. Another useful list was "Bugs" where we would post found/reported bugs in the application.

To improve communication between the team, another list was created to post Weekly Reports. These consisted of two simple checklists, one with the main objectives to accomplish in that week, and another with secondary objectives as a fallback

in case the other ones took less time to complete than expected. These cards were especially convenient for the writing of this report.

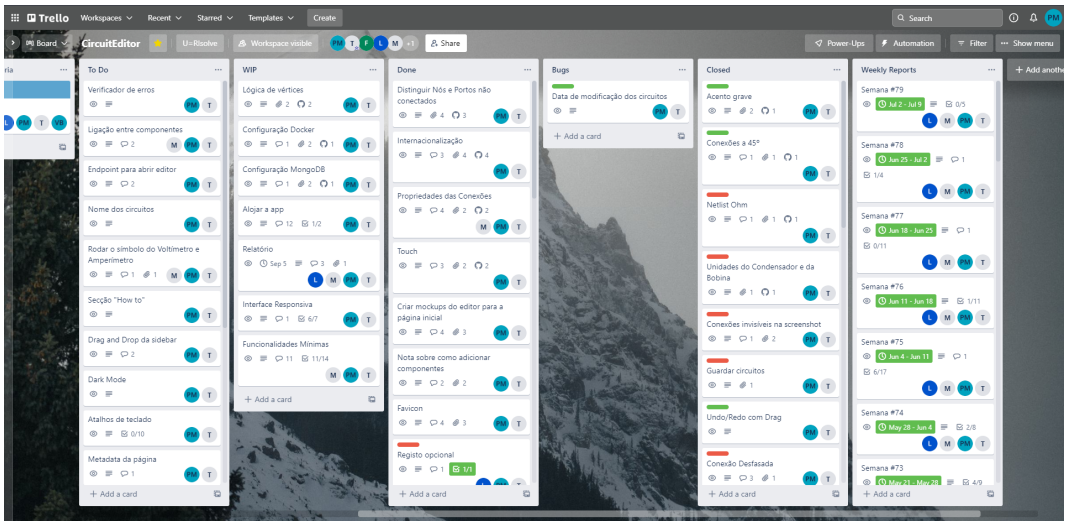


Figure 4.1: Screenshot of the Trello Board used for managing this project, in Portuguese

4.2 Version Control Software

Version Control Software is an external program that saves every modification made to the codebase so that, if a mistake is made, it is possible to compare it to previous versions and pinpoint the error while minimizing the amount of work to be done by the developers. This technology is of the utmost importance when developing a complex application, and it would be absurd to attempt the development of this project without it.

Hence, we used Git [38] as the main software and hosted the repository on GitHub [39]. The general rule for the development was to keep the *main* branch always fully functional and develop individual features on separate branches; once those features were fully developed and tested for bugs they could be merged onto the *main* branch. The code is currently in a private repository owned by the official U=RIsolve GitHub account [40].

4.3 Code Styling and Linting

Linting is the act of using an external piece of software to analyze the source code and scan for potential problems. This action becomes extremely useful when developing in a shared codebase since each developer has their writing style. Linting tools like Prettier [41] and ESLint [42] were the ones we used since they are the most widely used ones in the JavaScript ecosystem. This project had only one developer, but

other students will pick up where the previous ones left off; therefore, it is even more critical to have an automatic style enforcer since team member communication will be impaired.

4.4 Documentation

Due to the team communication issue described previously, it is also critical to have up-to-date code documentation. For that, we created multiple documentation pages using the Markdown language. The most important part of this mentioned documentation is present in this report's appendix.

Chapter 5

Software Deployment

Software Deployment is the entire process of delivering the software to its intended users. A typical computer program would be to create an installer and serve a download link for it, while for a web app, it is about hosting it online on a server, either automatically or manually.

While the frontend can just be compiled into a static page that can be served over a CDN, the backend is much more complex since it must constantly run and communicate with all active users. This inherent difference forces a different strategy to host both of them.

5.1 Development build

To launch an app into the world, we needed to ensure that many people thoroughly tested it. For that, it was evident that we needed to host a development build. This hosting does not need to be fully optimized or finished, as it only needs to allow the beta-testers to test the most recent features without building the app on their machines.

5.1.1 Frontend

To deploy our frontend, we used Netlify [43] since they offer a free option for smaller builds. It is effortless to set up: connect it to the GitHub repository and indicate the build command and folder. Netlify will automatically deploy the new build when the main branch receives new commits. It will also create previews for pull requests

which aids in testing if the new code broke any of the old features - it also provides a QR code to open the preview on the phone.

One downside of this free version of Netlify is that it only supports static pages. Thankfully, that is entirely acceptable for this project since it does not use server-side rendering like in Next.js, as we explained in an earlier chapter. The frontend is currently hosted at `https://circuit-editor.netlify.app/`.

5.1.2 Backend

For the backend, we used Heroku [44]. It is a service that can host a small server for free. However, it is considerably more complicated than Netlify. For it, we needed to generate a $Procfile$, a file exclusive to $Heroku$ that defines the environment that the server should run on. If no users send a request for a while, the server will enter a "sleep" mode to save costs; it will go back online once it receives a new request. The downside is that it can take up to ten minutes to go back online.

The backend hosting is not functional at the time of writing due to an unforeseen error on Heroku. Since it only takes care of the login and saves process - which is secondary to the editing of circuits - this is not a big issue since the beta-testers can still test the frontend.

5.2 Production build

The production build is the one that the users will see. It is fully optimized and capable of providing the best experience to the user. ISEP should later host it on a capable server that can support hundreds of connections simultaneously.

The cutting-edge way to host a server on any machine is to use Docker containers [45], which are small packages of virtualized operating systems created to run and distribute software. This technology allows developers to develop their applications without worrying about the infrastructure. It is only needed to define a docker-compose file to define and run our multi-container app: one container for the frontend, another for the backend, and a third for the database. Google then created a service named $Kubernetes$, which helps developers automate the deployment process and combines well with Docker. Although, it is too complex and excessive for this project. The actual hosting of the application on a server is kept out of this project and left in the hands of the professors behind the U=RIsolve team.

Chapter 6

Conclusion

6.1 Learned Lessons

We learned a lot during this app's development, ranging from managing and structuring a large project to minor implementation details. Regular classes cannot teach these kinds of lessons. Only actual project-building experience can convey this knowledge: hence the importance of projects like this.

Finding out about the entire web development environment was quite an experience itself. We had no idea it was such a progressive and thriving field. The more we researched, the deeper we found the hole to be. Incredible new advancements are being made in it every single day. This field has become one of our favorites. It is trivial to get started but incredibly hard to master. We are proud of all we have learned and would relish a career in this domain.

Also, it would be unfathomable to develop this entire project without using some managing tool like Trello. Anyone could easily get lost in between all of the features and bugs. Separating each segment into small, easily achievable steps is the best approach to a complex problem—just another application of the "Divide and Conquer" algorithm.

6.2 Shortcomings

This project is intricate, and its present implementation is far from perfect, especially since its developer was learning along the way. Therefore, some parts of it are not

as good as we hoped they would have been.

This fact is visible mainly on the editor page of the app. Especially the fact that the connections between nodes and components are not up to the current standard of professional circuit simulation software. They should be remade with custom logic since no JavaScript libraries offer those circuit-specific features.

Some less essential objectives were also not met due to unforeseen challenges in the development process—for example, the objects for real-time parameter graphing and labeling the circuit's meshes.

6.3 Future Work

This project is far from finished, as it is an app that should keep being developed and improved over the years. Besides the many minor changes and bug fixes, the lack of two main features keeps this app from its true potential.

Improving upon the schematic's connections mentioned in the previous chapter is the most superficial and of the greater importance of the two. They severely lack the features that modern circuit simulators have. These deserve to be created on a separate library since that would be not only important for this app but also incredibly useful for the open-source community since, at the moment of writing, there is no JavaScript/React library that can give the user complete control of the connections like they are used to in most native software. The used library, react-xarrow [13], was severely lacking in this regard, and the next developers to touch this codebase should consider creating a general-purpose connection library to replace the currently used one.

Secondly, the integration with U=RIsolve [2] is quite simplistic at the time of writing. It would be ideal if both apps were just one: this app's frontend, with the U=RIsolve's functions on the backend. This change would give the U=RIsolve team the most control over how the apps interact. Unfortunately, remaking both apps into just one is quite a task. Developing this kind of change will only be realistically done in a slightly far future, but it is a must if this app's plan is to be opened to the international community eventually.

6.4 Possible Changes

If the project were to start from scratch again, we would make many changes to improve it. These changes were not made initially because of a lack of knowledge in this Full-Stack Web Development field.

Firstly, replacing JavaScript with TypeScript [46] is the most significant change. JavaScript is an incomplete scripting language which is what to expect from a language that took five days to create. This fact is even more noticeable on large and

complex projects. Microsoft made TypeScript specifically for situations just like this one. As the name implies, it adds types to the standard JavaScript syntax. This small change is the difference between having a lot of tiny but hard-to-fix bugs or no bugs. We highly recommend that one of the future developers make the upgrade; it also doubles as a great exercise to get used to the codebase.

Secondly, Material-UI is not the best component library for creating an app. Its default Android look makes the app appear as if we grabbed it from a template, even though that could not be further from the truth. A much better and more modern alternative would be to use Mantine [47] since it's more complete than Material-UI and provides a more remarkable ability to customize the look of the components. This library was not considered at the start because it did not exist then. It is a young library, but it is the one everyone uses and loves.

Another necessary change, although much more complex than the previously mentioned ones, would be to use a state-handling library like XState [48] instead of doing it manually with React. Complex apps need to manage complex states, and this app just so happens to have a complicated JSON schema for its circuits. Separating all global app states into separate state machines would be the best way to simplify and modularize much of the current code. This change takes a lot of time and know-how, but it would significantly improve the developer experience since most future bugs would never even exist. However, it would force the new developers to learn how to use that new library.

Other possible but less significant changes would be to: replace CRA with Vite [49] since it is a better and more straightforward framework for React; update the repository to use TurboRepo [50] instead of Yarn Workspaces; add more linting rules.

References

[1] H. Jürgen, N. Kiesler, and L. Wiemeyer, *The inverted classroom model the 2nd German ICM Conference - Proceedings*. Oldenbourg, 2013. [Cited on page 1]

[2] "U=risolve simulator." [Cited on pages 1 and 34]

[3] "Qucs project: Quite universal circuit simulator." [Cited on page 1]

[4] M. L. O. C. Simulator, "Multisim live online circuit simulator," 2019. [Cited on page 2]

[5] W. McAllister, "Circuit sandbox," Jul 2022. [Cited on page 2]

[6] Symbolab, "Symbolab math solver - step by step calculator." [Cited on page 3]

[7] Wolfram, "Wolfram: Alpha: Making the world's knowledge computable." [Cited on page 3]

[8] edX, "Free online courses by harvard, mit, & more." [Cited on page 5]

[9] A. Herz, "Draw2d." [Cited on page 9]

[10] J. Foundation, "Jquery." [Cited on page 9]

[11] "React-draggable," 08 2022. [Cited on page 11]

[12] Figma, "Figma: the collaborative interface design tool.," 2019. [Cited on page 11]

[13] "react-xarrows." [Cited on pages 12 and 34]

[14] "Json." [Cited on page 15]

[15] A. Rocha, "Documento de trabalho," Feb 2021. [Cited on page 15]

[16] "html-to-image," 08 2022. [Cited on page 16]

[17] "Netlist," 05 2022. [Cited on page 16]

[18] MongoDB, "What is the mern stack? introduction & examples," 2022. [Cited on page 20]

[19] MongoDB, "Mongodb atlas database | multi-cloud database service," 2022. [Cited on pages 20 and 22]

[20] O. Foundation, "Express - node.js web application framework," 2017. [Cited on pages 20 and 22]

[21] Facebook, "Graphql: a query language for apis.," 2012. [Cited on pages 20 and 22]

[22] Facebook, "React – a javascript library for building user interfaces," 2022. [Cited on page 20]

[23] Vercel, "Next.js by vercel - the react framework." [Cited on page 20]

[24] N. Foundation, "Node.js," 2019. [Cited on page 21]

[25] R. Dahl, "A modern runtime for javascript and typescript." [Cited on page 21]

[26] "What is the difference between an orm and an odm?," Jan 1960. [Cited on page 22]

[27] "Mongoose odm v5.8.2," 2011. [Cited on page 22]

[28] P. D. tools for modern application development, "Prisma," 2019. [Cited on page 22]

[29] J. Hanson, "Passport.js." [Cited on page 22]

[30] Okta, "Password encryption: How do password encryption methods work? | okta." [Cited on page 22]

[31] "Ui vs. ux design: What's the difference?." [Cited on page 23]

[32] "React router: Declarative routing for react." [Cited on page 23]

[33] "Home." [Cited on page 23]

[34] T. Labs, "Tailwind css - rapidly build modern websites without ever leaving your html.." [Cited on page 23]

[35] M. Otto, "Bootstrap," 2000. [Cited on page 24]

[36] "Mui: The react component library you always wanted." [Cited on page 25]

[37] Trello, "Trello," 2014. [Cited on page 27]

[38] Git, "Git," 2019. [Cited on page 28]

[39] GitHub, "Github," 2018. [Cited on page 28]

[40] urisolve, "Circuit editor," 05 2022. [Cited on page 28]

[41] "Prettier · opinionated code formatter." [Cited on page 28]

[42] "Eslint - pluggable javascript linter," 2013. [Cited on page 28]

[43] N. A. in-one platform for automating modern web projects, "Netlify: All-in-one platform for automating modern web projects," 2019. [Cited on page 31]

[44] Heroku, "Cloud application platform | heroku," 01 2020. [Cited on page 32]

[45] Docker, "Enterprise application container platform | docker," 2018. [Cited on page 32]

[46] Microsoft, "Typescript - javascript that scales.," 2015. [Cited on page 34]

[47] "Mantine." [Cited on page 35]

[48] "Xstate - javascript state machines and statecharts." [Cited on page 35]

[49] "Vite." [Cited on page 35]

[50] "Turborepo." [Cited on page 35]

Appendix A

Example JSON

Below is the enormous JSON model that equates into the example circuit shown in 2.6:


```
{
  "components": [
    {
      "id": "99ee28d6-4cb3-4b43-a8e1-2215b6c6b8e4",
      "type": "R",
      "fullName": "Resistor",
      "label": {
        "name": "R1",
        "value": "10k",
        "unit": "Ω",
        "position": {
          "x": 360,
          "y": 150
        },
        "id": "d5778fcd-e076-471c-98d6-42589a530770",
        "isNameHidden": false
      },
      "ports": [
        {
```

```
          "id": "d2ef584c-b8e2-4811-ab3d-fbbbfa1fa01a",
          "owner": "99ee28d6-4cb3-4b43-a8e1-2215b6c6b8e4",
          "position": {
            "x": 0,
            "y": 0.5
          },
          "connection": "4e1b1256-a878-4c2a-b374-0e6ffd2cc5b3"
        },
        {
          "id": "ef2534d5-4f50-4b37-a0eb-31a5f7eb2d22",
          "owner": "99ee28d6-4cb3-4b43-a8e1-2215b6c6b8e4",
          "position": {
            "x": 1,
            "y": 0.5
          },
          "connection": "003e6aa5-6b5b-4559-953b-299d6968241c"
        }
      ],
      "position": {
        "x": 350,
        "y": 140
      },
      "properties": {}
    },
    {
      "id": "6d8b1c38-b469-4a8b-ab78-fb92fdcc2f4f",
      "type": "Vdc",
      "fullName": "DC Voltage Source",
      "label": {
        "name": "U1",
        "value": "10",
        "unit": "V",
        "position": {
          "x": 300,
          "y": 280
        },
        "id": "a7102e72-6af0-47b8-9524-89edc7e22a86",
        "isNameHidden": false
      },
      "ports": [
```

```
        {
          "id": "d7b95294-0f6f-442e-8f4b-ceddf8f7b601",
          "owner": "6d8b1c38-b469-4a8b-ab78-fb92fdcc2f4f",
          "position": {
            "x": 0.5,
            "y": 0
          },
          "connection": "4e1b1256-a878-4c2a-b374-0e6ffd2cc5b3"
        },
        {
          "id": "5b31e136-017d-4ba3-8efe-87c3c5345ca2",
          "owner": "6d8b1c38-b469-4a8b-ab78-fb92fdcc2f4f",
          "position": {
            "x": 0.5,
            "y": 1
          },
          "connection": "de6d0cf6-1c68-4ce5-989b-1663c55cbd20"
        }
      ],
      "position": {
        "x": 220,
        "y": 240
      },
      "properties": {}
    },
    {
      "id": "a5b04582-225d-4d9c-a919-047b19767ac0",
      "type": "R",
      "fullName": "Resistor",
      "label": {
        "name": "R2",
        "value": "10k",
        "unit": "Ω",
        "position": {
          "x": 420,
          "y": 280
        },
        "id": "bb3f934d-5da9-4eb7-bb02-482efc6a0730",
        "isNameHidden": false
      },
```

```
      "ports": [
        {
          "id": "feadc6ea-be78-4c34-8013-710e78b24745",
          "owner": "a5b04582-225d-4d9c-a919-047b19767ac0",
          "position": {
            "x": 0,
            "y": 0.5
          },
          "connection": "77e262f8-ddb1-481a-8b09-209e089cbe57"
        },
        {
          "id": "bc09e7aa-1df9-4d96-8e8d-fcabdf08f830",
          "owner": "a5b04582-225d-4d9c-a919-047b19767ac0",
          "position": {
            "x": 1,
            "y": 0.5
          },
          "connection": "45de0b45-be37-43df-bf17-10a7d65e2f0a"
        }
      ],
      "position": {
        "x": 480,
        "y": 240,
        "angle": 90
      },
      "properties": {}
    },
    {
      "id": "b80002a5-213f-42d2-9629-cea4906deb98",
      "type": "VProbe",
      "fullName": "Voltmeter",
      "label": {
        "name": "V1",
        "position": {
          "x": 580,
          "y": 280
        },
        "id": "2f0c94e5-63e6-4185-9b20-d6a64f1a6907",
        "isNameHidden": false
      },
```

```
      "ports": [
        {
          "id": "ef913272-f156-484a-a4a2-737731cc93d2",
          "owner": "b80002a5-213f-42d2-9629-cea4906deb98",
          "position": {
            "x": 0,
            "y": 0.5
          },
          "connection": "a8b986aa-2ead-4ce3-95bb-851a6e2f311a"
        },
        {
          "id": "c5463957-2532-4a69-841c-b8310ce4c9d0",
          "owner": "b80002a5-213f-42d2-9629-cea4906deb98",
          "position": {
            "x": 1,
            "y": 0.5
          },
          "connection": "a740f069-70ca-461f-bda6-106dc62c9389"
        }
      ],
      "position": {
        "x": 580,
        "y": 240,
        "angle": 90
      },
      "properties": {}
    }
  ],
  "nodes": [
    {
      "id": "6ad88b18-0c8c-464a-845b-489c8313f7e4",
      "label": {
        "unit": "V",
        "isValueHidden": true,
        "id": "a0dd31dd-b7e6-4c99-a1aa-06986f30aed3",
        "isNameHidden": false,
        "name": "A",
        "position": {
          "x": 530,
          "y": 170
```

```
        },
        "value": ""
      },
      "position": {
        "x": 530,
        "y": 190
      },
      "properties": {
        "color": "#2C2F33",
        "radius": 5
      },
      "connections": [
        "003e6aa5-6b5b-4559-953b-299d6968241c",
        "77e262f8-ddb1-481a-8b09-209e089cbe57",
        "a8b986aa-2ead-4ce3-95bb-851a6e2f311a"
      ],
      "type": "real"
    },
    {
      "id": "f6520abb-7fe3-4875-a4ff-683685642512",
      "label": {
        "unit": "V",
        "isValueHidden": true,
        "id": "39e1ddf7-cf00-4f60-8289-b3ee06cbcd6f",
        "isNameHidden": false,
        "name": "B",
        "position": {
          "x": 530,
          "y": 400
        },
        "value": ""
      },
      "position": {
        "x": 530,
        "y": 390
      },
      "properties": {
        "color": "#2C2F33",
        "radius": 5
      },
```

```
      "connections": [
        "de6d0cf6-1c68-4ce5-989b-1663c55cbd20",
        "45de0b45-be37-43df-bf17-10a7d65e2f0a",
        "a740f069-70ca-461f-bda6-106dc62c9389"
      ],
      "type": "real"
    }
  ],
  "connections": [
    {
      "id": "4e1b1256-a878-4c2a-b374-0e6ffd2cc5b3",
      "start": "d7b95294-0f6f-442e-8f4b-ceddf8f7b601",
      "end": "d2ef584c-b8e2-4811-ab3d-fbbbfa1fa01a",
      "label": {
        "isValueHidden": true,
        "id": "3a3185e5-c658-4fed-b4bb-985192f9b7b9",
        "isNameHidden": true,
        "name": "A",
        "position": {
          "x": 10,
          "y": 0
        }
      },
      "properties": {
        "dashedAnimationSpeed": 0,
        "color": "#2C2F33",
        "dashed": false,
        "gridBreak": 50,
        "strokeWidth": 4
      }
    },
    {
      "id": "003e6aa5-6b5b-4559-953b-299d6968241c",
      "start": "ef2534d5-4f50-4b37-a0eb-31a5f7eb2d22",
      "end": "6ad88b18-0c8c-464a-845b-489c8313f7e4",
      "label": {
        "isValueHidden": true,
        "id": "a3611a1a-0d18-4767-af94-40854fc42394",
        "isNameHidden": true,
        "name": "B",
```

```
        "position": {
          "x": 10,
          "y": 0
        }
      },
      "properties": {
        "dashedAnimationSpeed": 0,
        "color": "#2C2F33",
        "dashed": false,
        "gridBreak": 50,
        "strokeWidth": 4
      }
    },
    {
      "id": "77e262f8-ddb1-481a-8b09-209e089cbe57",
      "start": "6ad88b18-0c8c-464a-845b-489c8313f7e4",
      "end": "feadc6ea-be78-4c34-8013-710e78b24745",
      "label": {
        "isValueHidden": true,
        "id": "0e31d357-a2f3-450d-a1b0-27ff1196f9c4",
        "isNameHidden": true,
        "name": "C",
        "position": {
          "x": 10,
          "y": 0
        }
      },
      "properties": {
        "dashedAnimationSpeed": 0,
        "color": "#2C2F33",
        "dashed": false,
        "gridBreak": 50,
        "strokeWidth": 4
      }
    },
    {
      "id": "de6d0cf6-1c68-4ce5-989b-1663c55cbd20",
      "start": "5b31e136-017d-4ba3-8efe-87c3c5345ca2",
      "end": "f6520abb-7fe3-4875-a4ff-683685642512",
      "label": {
```

```
        "isValueHidden": true,
        "id": "912f5251-9d91-4621-825b-2f5d11983452",
        "isNameHidden": true,
        "name": "D",
        "position": {
          "x": 10,
          "y": 0
        }
      },
      "properties": {
        "dashedAnimationSpeed": 0,
        "color": "#2C2F33",
        "dashed": false,
        "gridBreak": 50,
        "strokeWidth": 4
      }
    },
    {
      "id": "45de0b45-be37-43df-bf17-10a7d65e2f0a",
      "start": "f6520abb-7fe3-4875-a4ff-683685642512",
      "end": "bc09e7aa-1df9-4d96-8e8d-fcabdf08f830",
      "label": {
        "isValueHidden": true,
        "id": "c0f9d040-2a97-492b-b09a-da6fd1874b82",
        "isNameHidden": true,
        "name": "E",
        "position": {
          "x": 10,
          "y": 0
        }
      },
      "properties": {
        "dashedAnimationSpeed": 0,
        "color": "#2C2F33",
        "dashed": false,
        "gridBreak": 50,
        "strokeWidth": 4
      }
    },
    {
```

```
      "id": "a8b986aa-2ead-4ce3-95bb-851a6e2f311a",
      "start": "6ad88b18-0c8c-464a-845b-489c8313f7e4",
      "end": "ef913272-f156-484a-a4a2-737731cc93d2",
      "label": {
        "isValueHidden": true,
        "id": "944dfc2b-09d0-46ec-ae84-8d0d48bd7505",
        "isNameHidden": true,
        "name": "F",
        "position": {
          "x": 10,
          "y": 0
        }
      },
      "properties": {
        "dashedAnimationSpeed": 0,
        "color": "#2C2F33",
        "dashed": false,
        "gridBreak": 50,
        "strokeWidth": 4
      }
    },
    {
      "id": "a740f069-70ca-461f-bda6-106dc62c9389",
      "start": "f6520abb-7fe3-4875-a4ff-683685642512",
      "end": "c5463957-2532-4a69-841c-b8310ce4c9d0",
      "label": {
        "isValueHidden": true,
        "id": "b34ba8ea-c1e7-41ad-956d-a7eef39a32a3",
        "isNameHidden": true,
        "name": "G",
        "position": {
          "x": 10,
          "y": 0
        }
      },
      "properties": {
        "dashedAnimationSpeed": 0,
        "color": "#2C2F33",
        "dashed": false,
        "gridBreak": 50,
```

```
380           "strokeWidth": 4
381         }
382       }
383     ]
384 }
```

Appendix B

Documentation

B.1 Schematic documentation

This app was built on top of a schematic JSON model that has the capability to completely describe a circuit in an intuitive way. It was initially idealized by Prof. André Rocha, and then modified and implemented by Pedro Morim.

This is the documentation for **v1.0.0** of the JSON schema and uses of the TypeScript syntax to describe the types of the schematic elements.

This model is the core of the circuit-editor app and it is used by others. Therefore, it is very important that it is followed strictly.

B.1.1 Basic types

These are some simple helper types created to remove the redundancy of the more complex types.

ID

Each element needs to have it's own ID so that it can easily identified in the schematic. The only requirement is that the ID needs to be a **unique** string.

```
type ID = string;
```

In this app, the IDs are generated by the v4 method of the UUID library. For example, an ID can be something like this: "8c0168ec-07fe-4b2c-96ee-4292ab34cfb2".

Properties

Each element can have its own specific set of properties. What those properties are depends on the type of element.

```
type Properties = Object;
```

For example:

- A **DC Voltage Source** can have a property to describe its internal resistance.
- A **Resistor** can have its internal temperature has a property.
- ...

Position

Each element must have a position so that it can be placed in the schematic. These come in two types:

- Global: The position relative to the origin of the schematic/canvas.
- Relative: The position relative to the origin of its parent element. Ranges from 0 to 1.

```
interface Position {
    x: number;
    y: number;
    angle?: number;
}

type GlobalPosition = Position;
type RelativePosition = Position;
```

Note: x and y refer to the distance, in pixels, to the origin. While, angle refers to the rotation, in degrees, of the element.

B.1.2 Secondary types

These types make use of the basic ones (shown above), but are, nonetheless, not as complex, nor as important as the main ones. They simply describe small parts/-modules of the aforementioned.

Port

The Port is the connection point of a Component. It allows its owner to be connected to the circuit. It can only allow an input, an output, be an hybrid (allow both) - the default.

```
interface Port {
    id: ID;
    type?: 'hybrid' | 'input' | 'output';
    position: RelativePosition;
    owner?: ID;
    connection?: ID;
}
```

It can also contain a reference to its owner and to the connection that is attached to it. It should also be noted that it can only hold one connection at a time - unlike a Node.

The Ports position has a unique twist to it. The values of x and y should range from 0 to 1, where 0 is the origin and 1 is the extreme of the component. It was created this way to simplify the creation of components, since, this way, it does not depend on its size to fit, it only depends on the its ratio.

Label

The Label is responsible for holding the text that visually differentiates each element.

```
interface Label {
    id: ID;
    name: string;
    value: string;
    unit: string;
    position: Position;
    isNameHidden?: boolean;
    isValueHidden?: boolean;
    owner?: ID;
}
```

It can hold a name, value and unit. For example, a **resistor** with the label "R1 = 10 kΩ" should have a label as follows:

```
{
    // ...
    "name": "R1",
```

```
    "value": "10k", // Or "10 k"
    "unit": "Ω"
    // ...
}
```

The label can have some variations:

- No value: **"R1 (Ω)"**
- No unit: **"R1 = 10 k"**
- No value and no unit: **"R1"**

It can also contain a reference to its owner, and flags that controls if some parts of the label should be hidden, isNameHidden and isValueHidden respectively.

B.1.3 Main types

These are the most complex types and hold the most data. They are the heart of this schema.

Component

The Component can represent any kind of electrical component: a resistor, a capacitor, a voltage source, a voltmeter, etc...

```
interface Component {
    id: ID;
    type: string;
    fullName: string;
    position: GlobalPosition;
    label: Label;
    ports: Port[];
    properties?: Properties;
}
```

Its important to distinguish the type from the fullName:

- The type, as the name implies, it describes the type of the component. For example, a resistor should have a type of **"R"**.
- The fullName describes a more human-friendly version of the type. For example, a resistor should have a fullName of **"Resistor"**

Connection

The Connection represents the physical wire connection between Components' Ports and Nodes. It holds two main values: start and end which are the IDs of the two Ports or Nodes that are connected.

```
interface Connection {
    id: ID;
    start: ID;
    end: ID;
    // vertexes: Vertex[]
    properties?: Properties;
}
```

A much needed upgrade is the ability to also store the vertexes of the connection. This addition would then allow the user to freely adjust the connection to their liking. Unfortunately, this feature is not yet implemented. Maybe in v1.1.0...

Node

Nodes are just connection "hubs": points that can hold multiple connections - unlike Ports.

```
interface Node {
    id: ID;
    type?: 'real' | 'virtual';
    position: GlobalPosition;
    label: Label;
    connections?: ID[];
    properties?: Properties;
}
```

A Node is considered of "real" if it holds more than 2 connections simultaneously. Otherwise it is just considered a "virtual" node.

B.1.4 Schematic

Joining all of the Main Types, each into their specific array, it is then possible to describe the entire circuit schematic in just a single JSON file.

```
interface Schematic {
    components: Component[];
    connections: Connection[];
    nodes: Node[];
}
```