

Patrick da Silva Varela



2AI2C: Ambiente de Aprendizado Interativo para Construção de Circuitos Eletrônicos e Digitais

Rio Grande

Junho de 2023

Patrick da Silva Varela

**2AI2C: Ambiente de Aprendizado Interativo para Construção de Circuitos
Eletrônicos e Digitais**

Trabalho de Conclusão de Curso apresentado
como requisito parcial para a obtenção do grau
de Tecnólogo em Análise e Desenvolvimento de
Sistemas.

Orientador: Prof. Dr. Leonardo Bandeira Soares

Rio Grande, junho de 20123



Resumo

Este trabalho apresenta a plataforma 'Ambiente de Aprendizado Interativo para Construção de Circuitos Eletrônicos e Digitais' (2AI2C), uma ferramenta desenvolvida com o objetivo de aprimorar o ensino de circuitos eletrônicos e digitais, proporcionando maior eficiência e didática. A plataforma foi projetada como uma aplicação web, utilizando a biblioteca compartilhada do NgSpice. Para viabilizar sua implementação, foi necessário desenvolver um *wrapper*, distribuído na modalidade *open-source*, capaz de traduzir os símbolos e tipos de C para Go, a linguagem utilizada na criação da aplicação. Também foi desenvolvida uma interface customizada e interativa para a aplicação. Foram conduzidas pesquisas com usuários para validar a eficácia do 2AI2C no aprimoramento da compreensão de circuitos elétricos, demonstrando uma boa taxa de aprovação. Por fim, este estudo comprova que a ferramenta é um objeto de aprendizagem que, além de servir para a aprendizagem dos alunos, também contará com uma plataforma específica para o professor.

Palavras-chave: Edutech. Circuitos elétricos e digitais. NgSpice. Aplicação Web.

Abstract

This work presents the platform 'Ambiente de Aprendizado Interativo para Construção de Circuitos Eletrônicos e Digitais' (2AI2C), a tool developed with the aim of improving the teaching of electronic and digital circuits, providing greater efficiency and didactics. The platform was designed as a web application, using the shared library of NgSpice. To enable its implementation, it was necessary to develop an wrapper, distributed as open-source, capable of translating C symbols and types to Go, the language used in creating the application. A customized and interactive interface was also developed for the application. User research was conducted to validate the effectiveness of 2AI2C in improving understanding of electrical circuits, demonstrating a good approval rate. Finally, this study proves that the tool is capable of being used as a learning object, in addition to being tested with those who will use it, students, it will also have a specific platform for the teacher.

Keywords: Edutech. Electrical and digital circuits. NgSpice. Web application.

Lista de figuras

Figura 1 – Casos de uso funcionais, usuário genérico.	14
Figura 2 – Casos de uso de educador e aluno.	15
Figura 3 – Diagrama de funcionamento da plataforma.	21
Figura 4 – Cache com instâncias de bibliotecas sendo usadas no momento. . .	27
Figura 5 – Desenho de uma fonte.	36
Figura 6 – Menu clique esquerdo.	37
Figura 7 – Menu para criar novo fio.	38
Figura 8 – Nodos da fonte e terra conectados	38
Figura 9 – Quadro com inversor exportado.	39
Figura 10 – Resultado da simulação.	39
Figura 11 – Quadro com roteiro.	40
Figura 12 – Descrição dos componentes no roteiro.	40
Figura 13 – Definindo parâmetros dos elementos.	41
Figura 14 – Simulação em andamento.	41
Figura 15 – Inversor criado usando o passo a passo.	42
Figura 16 – Pergunta número 1 do formulário.	42
Figura 17 – Pergunta número 4 do formulário.	43
Figura 18 – Pergunta número 5 do formulário.	43
Figura 19 – Pergunta número 10 do formulário.	44
Figura 20 – Pergunta sobre curso ou formação do participante.	44
Figura 21 – Pergunta sobre a viabilidade do uso da plataforma para facilitar o entendimento de ferramentas como o NgSpice.	45

Lista de códigos

Código 3.1 – Exemplo de importação de bibliotecas em C em um pacote Go. . .	17
Código 3.2 – Exemplo de uso do Fiber.	17
Código 3.3 – Exemplo de componente desenvolvido usando o lit.	18
Código 4.1 – Função <code>init</code> com definição dos callbacks.	23
Código 4.2 – Função <i>gateway</i> para <code>sendChar</code> em C.	24
Código 4.3 – Função <code>sendStat</code> implementada em Go.	24
Código 4.4 – Struct <code>Callbacks</code> e <code>gongs.Init</code>	24
Código 4.5 – Função <code>newNgInstance</code> e métodos de <code>NgInstance</code>	25
Código 4.6 – Métodos <code>newI</code> e <code>Call</code>	27
Código 4.7 – Exemplo de tipos.	28
Código 4.8 – Transformação de tipos C para nativos em Go.	29
Código 4.9 – Interface para importação <code>gongs</code>	30
Código 4.10 – Criação de elementos.	31
Código 4.11 – Modelo e persistência.	32
Código 4.12 – Exemplo de <code>circuitusecase.go</code>	33
Código 4.13 – Declaração das rotas e uso dos <code>controllers</code>	34
Código 4.14 – Inicialização de elemento com desenho.	35
Código 4.15 – Declaração de <code>CircuitElement</code> com implementações.	36

Lista de tabelas

Tabela 1 – Observação das ferramentas e características das soluções atuais em comparação com a proposta.	12
---	----

Lista de abreviaturas e siglas



STEM	<i>Science, Technology, Engineering e Mathematics</i>
MOSFET	<i>Metal-oxide-semiconductor field-effect transistor</i>
2AI2C	Ambiente de Aprendizado Interativo para Construção de Circuitos Eletrônicos e Digitais
API	<i>Application Programming Interface</i>
MVP	<i>Minimum Viable Product</i>
HTML	<i>Hypertext Markup Language</i>
BSON	<i>Binary JSON</i>
JSON	<i>JavaScript Object Notation</i>
RESTF	<i>Representational State Transfer</i>
gongs	<i>go ngspice package</i>
ID	Identificador

Sumário

1	INTRODUÇÃO	10
1.1	Objetivos	11
1.1.1	Objetivos Específicos	11
2	ANALISE	12
2.1	Elicitação de requisitos	12
2.2	Diagrama de Casos de Uso	13
3	ARQUITETURA E PROJETO	16
3.1	Tecnologias	16
3.1.1	Go	16
3.1.1.0.1	Cgo	16
3.1.1.1	Fiber	17
3.1.2	Typescript	18
3.1.2.1	Paper.js	18
3.1.2.2	Floating-ui	18
3.1.2.3	Lit	18
3.1.3	MongoDB	19
3.1.4	Ngpsice	20
3.2	Arquitetura	20
3.2.1	Diagrama do Sistema	20
3.2.2	Wrapper	20
3.2.3	API	21
3.2.4	Interface	22
4	IMPLEMENTAÇÃO	23
4.1	Wrapper	23
4.2	API	30
4.3	Interface	35
5	TESTES	40
5.1	Roteiro de Testes	40
5.2	Formulário	42
5.3	Discussão	45
6	CONCLUSÃO	46

REFERÊNCIAS 47

1 Introdução



Cursos de *Science, Technology, Engineering e Mathematics* (STEM) frequentemente abordam conceitos que não são devidamente ensinados na educação básica, o que resulta em dificuldades de compreensão por parte dos alunos. Um exemplo disso são os circuitos elétricos/eletrônicos, que desempenham um papel fundamental nas engenharias e ciências da computação. Como mencionado por Holton, Verma e Biswas (2008), esses desafios podem desencorajar os estudantes a prosseguirem seus estudos em engenharias, o que pode interromper seu desenvolvimento curricular.



Na tentativa de tornar o aprendizado mais eficiente, os professores frequentemente recorrem a ferramentas digitais para demonstrar o funcionamento dos circuitos de maneira clara e didática. Essas ferramentas são objetos de aprendizagem (OAs), que de acordo com Dantas, Maia e Scaico (2021) possuem a finalidade de promover um modelo de aprendizagem mais adaptativo e personalizado. No entanto, nem todas essas ferramentas são intuitivas e capazes de realizar simulações elétricas precisas, tornando-as menos eficazes no processo de aprendizado, como explicado por Dantas, Maia e Scaico (2021), Já **que** nem sempre eles são validados com o público ao qual eles se destinam.” Portanto, há uma demanda por **softwares** que atendam a essa necessidade, auxiliando os educadores a apresentarem o conteúdo de forma clara e descomplicada.



Além disso, há uma crescente demanda por tecnologias educacionais inovadoras e interativas, especialmente nas áreas das ciências exatas. Vale ressaltar também a importância cada vez maior da educação a distância, especialmente devido aos eventos relacionados à pandemia da COVID-19.



Nesse contexto, o presente trabalho de conclusão de curso aborda o desenvolvimento da plataforma web intitulada Ambiente de Aprendizado Interativo para Construção de Circuitos Eletrônicos e Digitais (2AI2C). Essa plataforma tem como principal objetivo oferecer um ambiente de aprendizagem interativo que combina a construção e simulação de circuitos elétricos e digitais com atividades gamificadas. A intenção é proporcionar aos alunos uma compreensão aprofundada do assunto, permitindo a exploração de diferentes níveis de abstração, incluindo o uso de transistores *Metal-Oxide-Semiconductor Field-Effect Transistor*¹ (MOSFET) e portas lógicas. E para os professores, um OA eficiente e enxuto, que atenda as necessidades para uma didática aprimorada.

Para viabilizar essa plataforma, foi desenvolvido um **wrapper em Go** para a **bibli-**

¹ Considerados nesse trabalho MOSFET de tipo N, NMOS, e tipo P, PMOS.



oteka compartilhada do ngspice, originalmente escrita em C, que será disponibilizado como *Open-Source*. Utilizando o pacote "cgo", que permite a chamada de código em C, o *wrapper* atualmente é compatível apenas com o sistema operacional Windows devido a diferenças de comportamento dos sistemas operacionais no carregamento e implementações dos métodos de bibliotecas compartilhadas. No entanto, há a possibilidade de expandir a compatibilidade para outras plataformas no futuro.

O *wrapper* desenvolvido permitiu a criação de uma *Application Programming Interface* (API) capaz de fornecer múltiplas instâncias de simulação, atendendo às demandas de diferentes usuários. Além disso, a API oferece recursos avançados de gerenciamento de simulações e instâncias, bem como funcionalidades de análise de desempenho dos alunos. Os professores possuem contas especiais que lhes permitem criar turmas, atribuir tarefas e associar dispositivos aos alunos, facilitando a avaliação e o acompanhamento do progresso individual e coletivo.

1.1 Objetivos

Desenvolver um *software* que exerça o papel de OA e permita a simulação de circuitos elétricos e digitais com tarefas guiadas e gamificadas, além de permitir a criação de portas lógicas a partir do uso de transistores, em um ambiente interativo, com uma plataforma integrada para o acompanhamento dos alunos pelos professores.

1.1.1 Objetivos Específicos

- a) Selecionar as tecnologias mais adequadas para o desenvolvimento;
- b) Desenvolver as funcionalidades para simulação de circuitos elétricos e digitais;
- c) Permitir a criação de portas lógicas a partir do uso de transistores.
- d) Criar a plataforma para alunos e professores;
- e) Desenvolver um ambiente interativo;
- f) Implementar a gamificação das tarefas;
- g) Validar a plataforma por meio de testes;

2 Analise

A **análise** é uma fase crucial no ciclo de vida do desenvolvimento de *software*, que ocorre antes do início efetivo da codificação. Ela envolve atividades de levantamento de requisitos, entendimento do domínio do problema e definição de soluções técnicas. Neste capítulo, são apresentados os principais requisitos funcionais e não funcionais, **juntamente** com diagramas de casos de uso, itens que definiram o processo utilizado no desenvolvimento da plataforma.

É importante ressaltar que a versão relatada nesse trabalho é o *Minimum Viable Product* (MVP) do ambiente. Portanto, pode não contemplar todos os casos de uso e/ou requisitos inicialmente planejados. Essa versão estabelece uma base sólida para as futuras melhorias e aperfeiçoamentos que já estão em desenvolvimento.

2.1 Elicitação de requisitos

Para o levantamento dos requisitos funcionais e não funcionais, foram identificadas as funções dos **softwares** já existentes no ecossistema educacional, bem como foi realizado o levantamento desses requisitos junto aos professores da área que utilizam ferramentas durante suas aulas.

A seguir, na Tabela 1¹, constam os *softwares* observados e suas ferramentas disponíveis para os usuários. Dentre elas foram consideradas, principalmente, a capacidade de ser usada por um educador para aprimorar a experiência do aluno.

Tabela 1 – Observação das ferramentas e características das soluções atuais em comparação com a proposta.

Características	2AI2C	NandGame	CircuitVerse	CircuitLab	PhET
Simulação elétrica	Sim	Não	Não	Sim	Sim
Plataforma web	Sim	Sim	Sim	Sim	Sim
Alterar abstração durante simulação	Sim	Não	Não	Não	Não
Rota de aprendizado guiada	Sim	Sim	Não	Não	Sim
Plataforma para professor	Sim	Não	Sim	Não	Não
Modo sandbox	Sim	Não	Sim	Sim	Não

Fonte: Elaboração Própria.

Através desse estudo, foi possível definir as funcionalidades principais do ambiente, assim como outras ferramentas para aprimorar a experiência já existente. Foram

¹ Mais sobre:
 NandGame em <<https://www.nandgame.com/>>
 CircuitVerse em <<https://circuitverse.org/>>
 CircuitLab em <<https://www.circuitlab.com/>>
 PhET <https://phet.colorado.edu/pt_BR/simulations/>

então elencados os seguintes requisitos:

a) Funcionais:

- Autenticação:
 - Registro de usuário;
 - *Login* de usuário;
- Características:
 - Criação de quadros, local onde se posicionam os elementos para construir o circuito;
 - Criação de diagramas de circuitos eletrônicos;
 - Simulação de circuitos eletrônicos;
 - Persistência de circuitos eletrônicos criados;
 - Utilização de circuitos eletrônicos como sub-circuitos;
 - Realização de tarefas guiadas;
 - Exportação como imagem do diagrama do circuito;
 - Ambiente gráfico *drag-and-drop*;

b) Não funcionais:

- Características:
 - Conta para educadores;
 - Capacidade de educadores monitorar o desempenho dos alunos;
 - Criação de tarefas guiadas;
 - Compartilhamento de quadros;
 - Colaboração em tempo real nos quadros;
 - Criação de turmas;
 - Adicionar alunos as turmas e compartilhar as tarefas;
 - Criação de rotas de tarefas para turmas;
 - Pontuação por aluno e turma;
 - Recompensas por desafios concluídos;
 - Exportação da instrução SPICE do circuito;
 - Simulações transientes interativas;
 - Uso de elementos lógicos, como portas lógicas;
 - Mudança de abstração durante simulação;

2.2 Diagrama de Casos de Uso

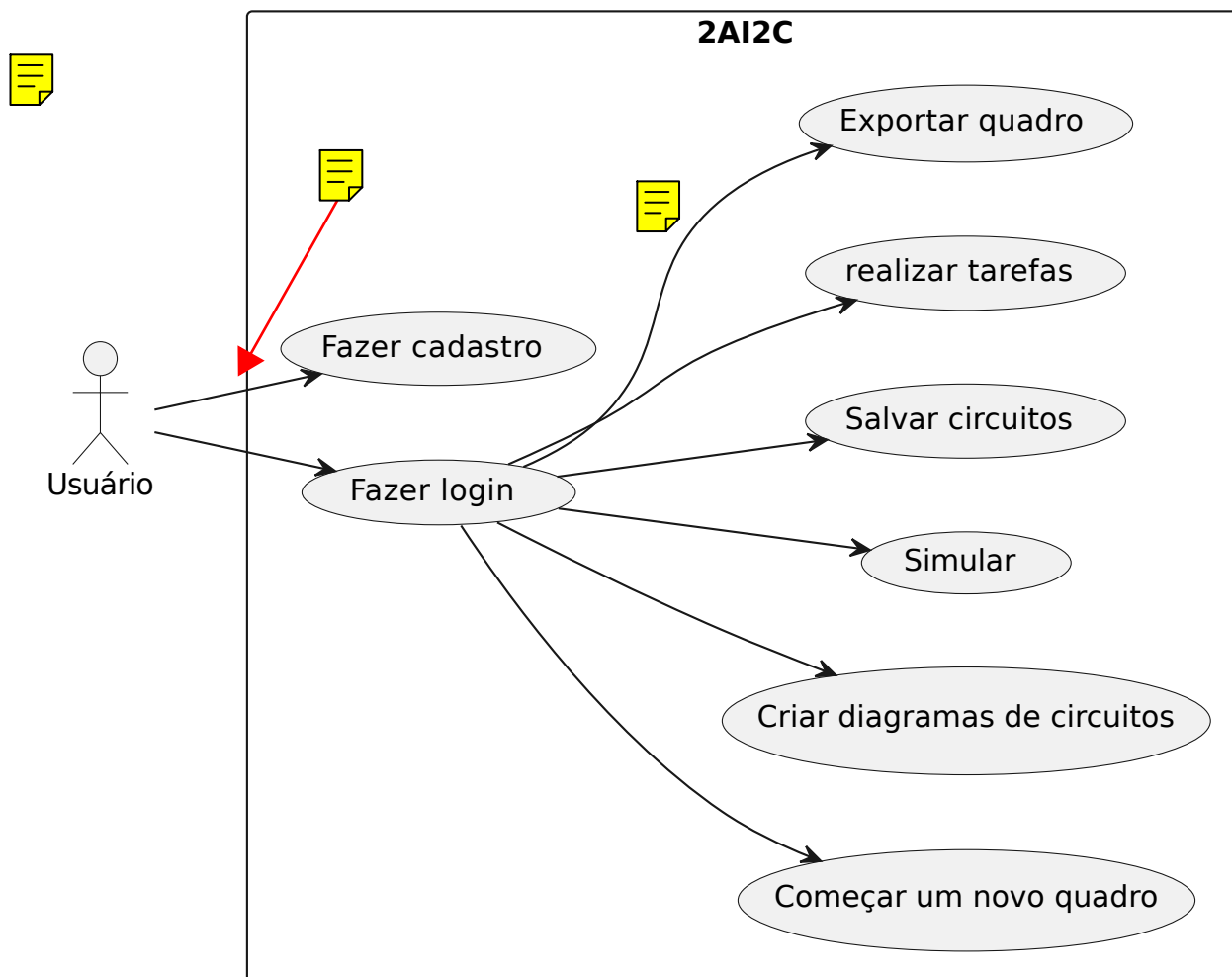
Os casos de uso são ferramentas essenciais para o desenvolvimento de *softwares*, pois permitem especificar os requisitos e as interações entre os atores e o sistema.



Por esse motivo, foram desenvolvidos 2 diagramas de casos de uso para demonstrar a atuação prevista pelos usuários. A Figura 1 demonstra o funcionamento de forma

mais genérica, com as funcionalidades mais importantes para a ferramenta. É possível observar que o usuário precisa estar conectado para realizar ações como simular e salvar circuitos, característica que não fica clara apenas com o levantamento dos requisitos.

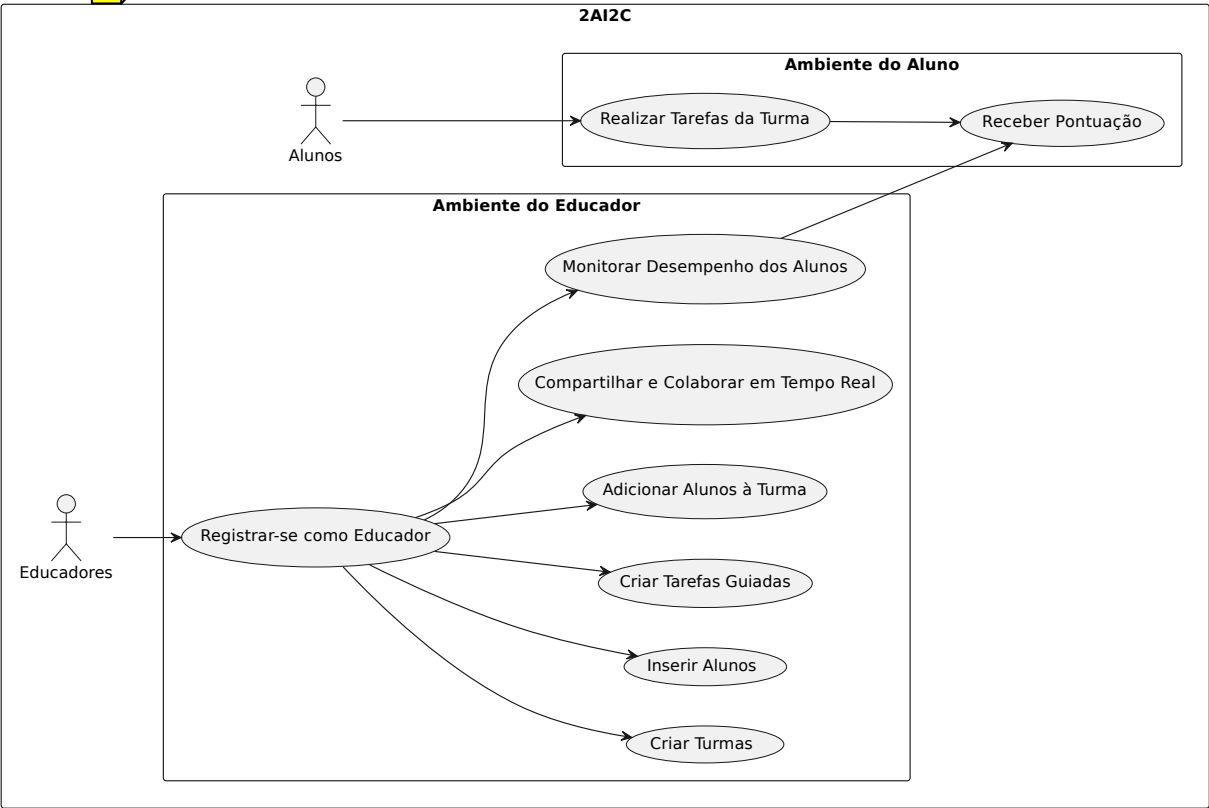
Figura 1 – Casos de uso funcionais, usuário genérico.



Na Figura 2 é demonstrado o funcionamento de forma mais ampla, com funcionalidades mais específicas para educadores e alunos. Também é possível identificar que os alunos estão usando ambientes diferentes dos educadores dentro do 2AI2C. Os educadores devem se registrar e criar turmas para que, assim, possam adicionar os alunos. Com isso, os alunos podem realizar as tarefas da turma, recebendo uma pontuação por item concluído. O educador pode utilizar essa métrica para monitorar o desempenho dos alunos durante a realização das tarefas por ele criadas.



Figura 2 – Casos de uso de educador e aluno.



3 Arquitetura e Projeto



Neste capítulo, serão discutidas as decisões de arquitetura e projeto do sistema, assim como as tecnologias utilizadas. Essas tecnologias foram escolhidas **principalmente** para o desenvolvimento do *wrapper*.

3.1 Tecnologias

Para atender os objetivos deste trabalho, foi realizado um estudo para determinar as ferramentas mais adequadas. Portanto, este capítulo relata as tecnologias consideradas para o desenvolvimento do *wrapper*, já que esse é o componente central da plataforma. A seguir, serão apresentadas as tecnologias consideradas, juntamente com a conclusão sobre a sua viabilidade para a implementação.

3.1.1 Go

“Go é uma linguagem de programação criada pela Google em 2007”(GO, s.d.a, tradução nossa). Por ser uma linguagem compilada, o Go oferece um bom desempenho computacional. **Ele também oferece recursos nativos para interoperabilidade com programas em C, o que é essencial para o desenvolvimento do *wrapper*.** Além disso, compartilha algumas semelhanças com a linguagem C, mas com recursos como o *Garbage Collector*.

Levando em consideração esses aspectos, o Go se destacou como uma excelente opção para o desenvolvimento do *wrapper* e também para a API da plataforma. Ele possui uma documentação abrangente e uma comunidade considerável. Como um benefício adicional, o Go possui recursos nativos para o desenvolvimento de aplicações *web* e bibliotecas de terceiros, conhecidas por seu desempenho e qualidade. Assim foi definida a principal tecnologia do *wrapper* e da API do 2AI2C.

3.1.1.0.1 Cgo

O *cgo* é um pacote nativo do Go que permite a criação de pacotes Go que chamam código C. Com essa biblioteca, é possível importar bibliotecas compartilhadas de C para que sejam usadas como código nativo em Go. O *cgo* gera os cabeçalhos em Go da biblioteca C para a plataforma de forma automática. Um exemplo de como as bibliotecas podem ser importadas pode ser observado no Código 3.1¹. Neste exem-

¹ A inclusão de *headers* C em um pacote Go, é feita através de um comentário, chamado de preâmbulo, ele deve estar diretamente acima da declaração `import "C"` (C é um pseudo-pacote, essencialmente o *cgo*). Por esse motivo o preâmbulo é facilmente confundido com um comentário de documentação, quando na verdade ele possui a função de informar quais *headers* devem ser compilados.

plo, usando a *flag* CFLAGS é definido o caminho para os cabeçalhos necessários, ou seja, para este caso o `sharedspice.h`. Maiores detalhes serão discutidos na seção de desenvolvimento.

```
1 /*
2 #cgo CFLAGS: -I./inc
3
4 #include <stdbool.h>
5 #include <stdlib.h>
6
7 #include "sharedspice.h"
8 */
9 import "C"
```

Código 3.1 – Exemplo de importação de bibliotecas em C em um pacote Go.

3.1.1.1 Fiber

Fiber é um *framework web* baseado no Express², foi desenvolvido para ser rápido e fácil de ser usado. É um dos muitos *frameworks web* para Go, mas um dos mais rápidos (GOFIBER, 2023). Um exemplo de sua utilização pode ser observada no Código 3.2, que demonstra a sua simplicidade e facilidade para criar uma aplicação.

Outro ponto relevante a ser levantado é que o Fiber já possui *middlewares* comumente usados em seu pacote, o que torna ainda mais ágil o desenvolvimento da solução. Um desses *middlewares* é o `Session`, que gerencia e mantém as sessões de usuários.

```
1 app := fiber.New(fiber.Config{
2     Views: engine,
3 })
4
5 app.Use(logger.New())
6 app.Use(recover.New())
7 app.Use(requestid.New())
8 app.Use(compress.New())
9
10 app.Static("/svgs", "./public/svgs")
11
12 api.RouteApi(app)
13
14 app.Listen(":3000")
```

Código 3.2 – Exemplo de uso do Fiber.

² Também um *framework web*, mas para node.js

3.1.2 Typescript



Para o desenvolvimento da interface do 2AI2C foi utilizado Typescript, linguagem que quando compilada gera um código em Javascript. Typescript foi escolhido por ser uma linguagem fortemente tipada, fazendo com que, por exemplo, seja possível identificar erros durante a compilação, ao contrário do Javascript, que muitas das vezes erros de tipo só são identificados durante a execução, atrasando o desenvolvimento.

3.1.2.1 Paper.js

De acordo com o site oficial da ferramenta, (PAPER.JS, s.d.), o Paper.js é um *framework* de *script* de gráficos vetoriais de código aberto que é executado em cima do *Canvas* HTML5. Ele possui muitas funcionalidades poderosas para criar e trabalhar com gráficos vetoriais e curvas de Bézier. Essas características do Paper.js foram essenciais para desenvolver a interface gráfica da plataforma, permitindo modificar os gráficos vetoriais para responder às interações do usuário, como é o caso de conectar um fio a um nó de um elemento no circuito.

3.1.2.2 Floating-ui



É uma biblioteca que auxilia na criação de elementos flutuantes como *tooltips*, *popovers* e menus suspensos. A ancoragem de posicionamento que a biblioteca oferece é essencial, já que com ela foi possível exibir *popovers* com informações como os dispositivos do circuito, evitando que esses menus flutuantes acabassem saindo do campo de visão do usuário.

3.1.2.3 Lit




Lit é uma biblioteca simples para construir componentes da *web* rápidos e leves, renderizados diretamente no cliente, ao contrário de alguns motores de *templates*. O **lit** foi utilizado por ser fácil de implementar e principalmente pela boa integração ao código já existente, já que foi uma biblioteca adicionada quando alguns elementos já tinham sido criados, como os **popovers**, que fazem uso da biblioteca **floating-ui**. Um exemplo de componente pode ser visto no Código 3.3.


```
1 @customElement('context-menu')
2 export class ContextMenu extends LitElement {
3   static styles = css`
4     :host {
5       background-color: white;
6     }
7     button {
8       display: block;
```

```
9      width: 100%;
10     text-align: left;
11     padding: 0.5rem;
12     border: none;
13     background-color: #f0f0f066;
14     cursor: pointer;
15   }
16   button:hover {
17     background-color: #57575766;
18     transform: scale(1.05);
19   }
20 `;
21
22 @property({ type: Array })
23 buttons: { label: string; callback: () => void }[] = [];
24
25 handleClick(callback: () => void) {
26   callback();
27   hide(true);
28 }
29
30 render() {
31   return html`
32     ${this.buttons.map(
33       (button, index) =>
34         html`<button @click=${() => this.handleClick(button.
35           callback)}>
36           ${button.label}
37         </button>`
38     )}
39 `;
40 }
```

Código 3.3 – Exemplo de componente desenvolvido usando o **lit**.

3.1.3 MongoDB

 O MongoDB é um banco de dados de documentos escalável e flexível que armazena dados em documentos *Binary JSON* (BSON) que podem ser exportados em *JavaScript Object Notation* (JSON), o que o torna simples e poderoso (MONGODB, s.d.). Esse banco de dados foi escolhido **justamente** pelo uso do JSON, pois permite que os itens desenvolvidos sejam armazenados de maneira que possam ser facilmente transformados em objetos ou estruturas, tanto em Typescript quanto em Go.

 **Além disso**, MongoDB torna o desenvolvimento muito mais rápido, graças aos pontos já discutidos e também oferece recursos como indexação, agregação em tempo real, além de alto desempenho e disponibilidade. Por esse motivo, este banco de dados foi escolhido.

3.1.4 Ngspice

O ngspice é o simulador de circuitos elétricos e eletrônicos de código aberto do tipo spice, “de código aberto e baseado em três pacotes de **software**: Spice3f5, Cider1b1 e Xspice. É estável, confiável e amplamente utilizado em muitos projetos”(NGSPICE, s.d.)[tradução nossa], além de ser usado em cursos STEM para apresentar o conceito de simulação de circuitos. Por esse motivo, o ngspice foi a opção ideal, se não a única, para a aplicação proposta, pois além dos pontos já citados, ele oferece uma biblioteca compartilhada em **C**³ que pode ser utilizada para realizar simulações.

3.2 Arquitetura

Nessa seção do trabalho será discutida a arquitetura da plataforma 2AI2C, para que no **próximo** capítulo, os pontos abordados **aqui**, possam ser apresentados com mais clareza de sua função no fluxo do sistema.

3.2.1 Diagrama do Sistema

A Figura 3 demonstra como são separadas as partes da aplicação. Com esse diagrama é possível observar que as partes mais internas do sistema podem operar sem depender das mais externas, assim como demonstra a arquitetura limpa. Usando essa arquitetura é possível modificar os componentes que integram o 2AI2C com mais facilidade. Nas próximas subseções será demonstrado como cada elemento do diagrama é projetado, do mais interno ao mais externo.

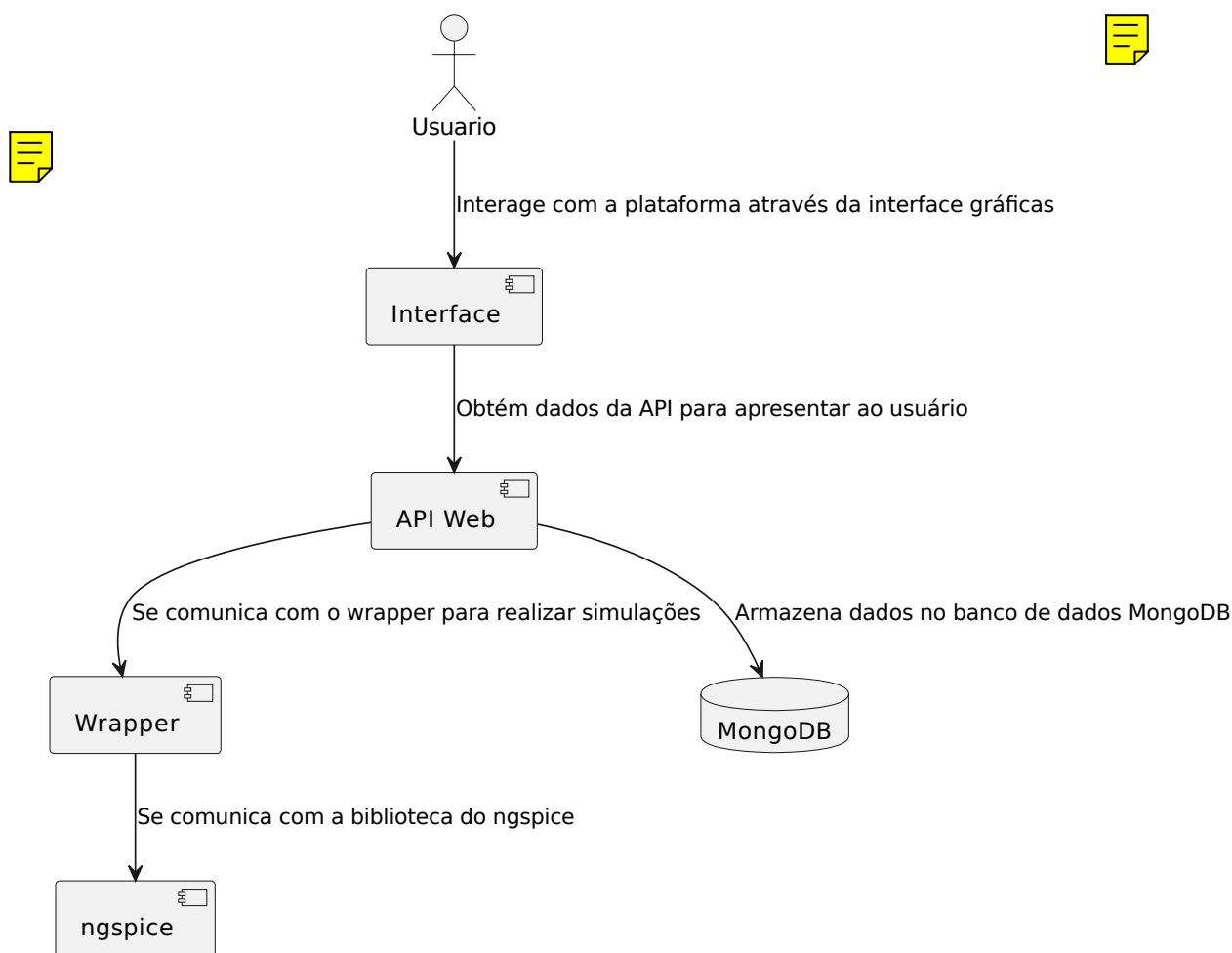
3.2.2 Wrapper

Para realizar simulações precisas, seria necessário desenvolver um simulador de propriedades físicas, o que levaria muito tempo. Para contornar essa questão, o uso do simulador ngspice é a melhor opção. Ele é amplamente utilizado, gratuito e possui bibliotecas compartilhadas escritas em C que podem ser usadas livremente. Para invocar as funções da biblioteca, foi projetada uma interface e posteriormente um *wrapper*. A arquitetura do *wrapper* é separada em 3 partes:

- Interface do ngspice *wrapper*: permite a importação do pacote em outros programas escritos em Go;
- Wrapper* da interface da biblioteca do ngspice: faz as transformações de tipos, trata os espaços de memória reservados e prepara argumentos para serem enviados à biblioteca;
- Interface da biblioteca do ngspice: permite que funções em Go sejam injetadas em funções em C, através de interfaces do cgo, permitindo a chamada

³ Mais informações podem ser acessadas em <<https://ngspice.sourceforge.io/extras.html>>

Figura 3 – Diagrama de funcionamento da plataforma.



das funções nativas em C. Tornando possível enviar *callbacks* através de um *gateway*;

3.2.3 API

A API desenvolvida atua como intermediária entre a interface do usuário e o ngspice *wrapper*. Ela mantém funções de autenticação, persistência de circuitos, lista de modelos de elementos⁴, tarefas guiadas, entre outros. A API atual não aplica o conceito *Representational State Transfer* (RESTF), ou seja, mantém informações de sessão. Foi desenvolvida de maneira totalmente agnóstica da interface gráfica, que pode ser alterada ou não ser utilizada, permitindo o desenvolvimento de outras ferramentas que usem a API.

Fazendo uso do banco de dados de documentos, é possível armazenar informações de elementos muito mais fácil, já que possuem mais flexibilidade do que os bancos relacionais. Com essa flexibilidade, diferentes elementos podem ter atributos

⁴ Elementos são itens que podem ser adicionados a um circuito, esses elementos podem ser sub-circuitos ou dispositivos como o transistor do tipo MOSFET

distintos aos demais.

A autenticação é mantida pela própria API, através de sessões, assim os usuários podem fazer requisições através da mesma origem sem precisar lidar com *tokens* de autorização. Essa decisão foi tomada para a versão v1 da API, que pode vir a ser diferente nas próximas atualizações.

3.2.4 Interface

A interface gráfica desempenha um papel fundamental ao tornar a experiência do usuário mais amigável e intuitiva. Ela permite a interação com a API de forma simplificada, sem a necessidade de lidar manualmente com requisições e respostas. Essa abstração proporciona ao usuário a facilidade de realizar operações, como a criação de circuitos e simulações, além de visualizar graficamente os resultados obtidos.

A interface foi projetada de forma simples, tendo como componente principal o ambiente de interação com o quadro dos circuitos e a caixa de elementos, componente com capacidade de *drag-and-drop*.

4 Implementação



Deste ponto em diante, será relatado como foram implementados os componentes apresentados com o uso das tecnologias expostas no capítulo anterior. De mesma maneira, será feito a partir do *wrapper*.



4.1 Wrapper

O *wrapper* é um pacote desenvolvido em Go com base na biblioteca *ngspice*. Dessa forma, será usado o nome *gongs* (Go *ngspice Package*) para se referir a este pacote deste ponto em diante. É importante destacar uma diferença entre as linguagens C e Go para entender o contexto da implementação. Go é uma linguagem com “*garbage collection*”, o que significa que a ferramenta precisa saber a posição de todos os ponteiros. Portanto, para transferir um ponteiro que aponta para um espaço alocado por Go para C, é necessário ter certeza de que nesse espaço não existem outros ponteiros com espaços alocados por Go. O tipo função sempre inclui um ponteiro para espaço alocado pelo Go, isso significa que existe uma restrição para transferi-los (GO, s.d.c).



Com base nisso, no Código 4.1 é possível observar uma função que aguarda seis ponteiros de diferentes funções, nesse caso *callbacks* para diferentes eventos do *ngspice*. Para fazer isso, é necessário criar funções *gateways* em C, já que não é possível transferir esses ponteiros de funções Go como argumentos para C (GO, s.d.b).

```

1 int ngSpice_Init(SendChar*, SendStat*, ControlledExit*,
   SendData*, SendInitData*, BGThreadRunning*, void)
2
3 typedef int (SendChar)(char*, int, void*)
4 typedef int (SendStat)(char*, int, void*)
5 typedef int (ControlledExit)(int, NG_BOOL, NG_BOOL, int, void*)
6 typedef int (SendData)(pvecvaluesall, int, int, void*)
7 typedef int (SendInitData)(pvecinfoall, int, void*)
8 typedef int (BGThreadRunning)(NG_BOOL, int, void*)

```



Código 4.1 – Função *init* com definição dos *callbacks*.

No Código 4.2 é possível observar um desses *gateways*, escritos em C. Na linha 5, é declarada uma função *sendChar*. Essa função é um *stub* gerado pelo *cgo* de uma função Go declarada e implementada em outro arquivo, como demonstrado no Código 4.3. Nesse mesmo trecho de código, o ponteiro da função em C, *sendChar_cgo*, é atribuída a variável *sendCharGtw*.


```

1 #include <stdbool.h>
2
3 int sendChar_cgo(char *c, int i, void *ptr)
4 {
5     int sendChar(char *, int, void *);
6     return sendChar(c, i, ptr);
7 }

```

Código 4.2 – Função *gateway* para `sendChar` em C.

```

1 /*
2 #include <stdbool.h>
3 #include <stdlib.h>
4
5 #include "sharedspice.h"
6 #include "gateway.h"
7 */
8 import "C"
9 import (
10     "unsafe"
11 )
12
13 var (
14     sendCharGtw      = unsafe.Pointer(C.sendChar_cgo)
15 )
16
17 //export sendChar
18 func sendChar(s *C.char, i int, ptr unsafe.Pointer) int {
19     ng := ngFromPointer(ptr)
20     if f := ng.cbs.SendChar; f != nil {
21         return f(C.GoString(s), i, ptr)
22     }
23     return -1
24 }

```

Código 4.3 – Função `sendStat` implementada em Go.

Para compreender melhor o Código 4.3, é importante entender que o ponteiro `sendCharGtw` é sempre enviado como argumento para a função `init` do `ngspice`. Ele é responsável por chamar a função `sendChar`, que invoca os *callbacks* em Go. Estes *callbacks* são inseridos através do *struct* `Callbacks` na função `wrapper.Init`, que é responsável por criar o *struct* do `NgInstance`. Isso pode ser observado no Código 4.4¹. Esse é o *struct* recebido como argumento do parâmetro `ptr` e transformado de volta para `NgInstance` na linha 21 do Código 4.3

```

1 ///Pacote ngcallback
2
3 type SendChar func(string, int, unsafe.Pointer) int
4
5 type Callbacks struct {

```

¹ As definições dos demais *callbacks* foram omitidas nesse exemplo.

```

6   SendChar      SendChar
7   SendStat      SendStat
8   ControlledExit ControlledExit
9   SendData      SendData
10  SendInitData  SendInitData
11  BGThreadRunning BGThreadRunning
12 }
13
14 ///Pacote wrapper
15
16 type NgInstance struct {
17     curVecs      *curVecsData
18     uid          string
19     libPath      string
20     cbs          *ngcallback.Callbacks
21     handler      libHdl
22     initP        procHdl
23     initSyncP    procHdl
24     commandP     procHdl
25     runningP     procHdl
26     GetVecInfoP  procHdl
27     circP        procHdl
28     curPlotP     procHdl
29     allPlotsP    procHdl
30     allVecsP     procHdl
31 }
32
33 func Init(c *ngcallback.Callbacks) *NgInstance {
34     i := newNgInstance(c)
35     i.init()
36     return i
37 }
38
39 func (ng *NgInstance) init() int {
40     ng.call(ng.initP, sendCharGtw, sendStatGtw, controlledExitGtw,
41             sendDataGtw, sendInitDataGtw, bgThreadRunningGtw, unsafe.
42             Pointer(ng))
41     return 0
42 }

```

Código 4.4 – Struct Callbacks e gongs.Init.

Nesse ponto do código, é persistente falar sobre o `NgInstance`. Esse *struct* é responsável por armazenar os *handlers* da biblioteca do ngspice e de seus processos. O processo de inicialização é feito através da função `newNgInstance`, utilizado na linha 34 do Código 4.4. Essa função pode ser visualizada no Código 4.5. Seguindo o seu fluxo de execução, dois passos chaves são executados: a criação de um Identificador (ID) na linha 18 e o método `newI` da linha 20.

```

1 const (
2     //counts as number of zeroes
3     uidHalfLen = 10000000
4
5     initName      = "ngSpice_Init"

```

```

6   initSyncName   = "ngSpice_Init_Sync"
7   commandName    = "ngSpice_Command"
8   runningName    = "ngSpice_running"
9   getVecInfoName = "ngGet_Vec_Info"
10  circName       = "ngSpice_Circ"
11  curPlotName    = "ngSpice_CurPlot"
12  allPlotsName   = "ngSpice_AllPlots"
13  allVecsName    = "ngSpice_AllVecs"
14 )
15 func newNgInstance(cbs *ngcallback.Callbacks) *NgInstance {
16     inst := new(NgInstance)
17     inst.curVecs = new(curVecsData)
18     inst.newUID()
19     inst.cbs = cbs.Copy()
20     inst.newI()
21     return inst
22 }
23
24 func (i *NgInstance) release() {
25     i.Release()
26 }
27
28 func (i *NgInstance) call(symbol procHdl, args ...unsafe.
    Pointer) uintptr {
29     return i.Call(symbol, args)
30 }
31
32 func ngFromPointer(p unsafe.Pointer) *NgInstance {
33     return (*NgInstance)(p)
34 }
35
36 func (i *NgInstance) newUID() {
37     p := int64(uintptr(unsafe.Pointer(i))) % uidHalfLen
38     t := (time.Now().UnixMicro() % uidHalfLen) * uidHalfLen
39     i.uid = strconv.FormatInt(p+t, 32)
40 }

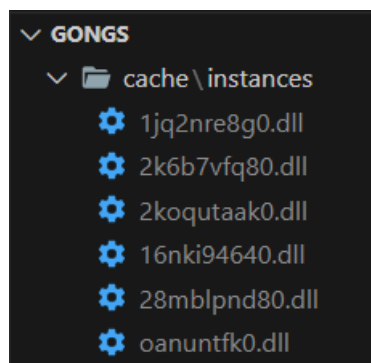
```

Código 4.5 – Função newNgInstance e métodos de NgInstance.

Primeiro, será abordada a necessidade de um ID para o *struct*. Quando uma biblioteca é conectada de forma estática ou dinâmica, todos os métodos ficam disponíveis para serem usados, mas essa instância não pode ser usada para operações em paralelo, pois seu estado é sempre compartilhado. Por exemplo, ao considerar uma aplicação *Web*, onde mais de um usuário possa fazer simulações ao mesmo tempo, se o usuário A solicitar uma simulação e o usuário B requisitar o valor de um nodo da última simulação que ele havia feito, o que será retornado é o resultado do nodo do Usuário A. Para lidar com esse problema, o **gongs** faz uma cópia da biblioteca com o ID como nome do arquivo, demonstrado na Figura 4, então, todos os usuários possuem ambientes isolados que não interagem entre si.

O método `new` é dependente da plataforma e pode ser observado no Código 4.6. No caso do presente trabalho, a plataforma de desenvolvimento foi o Sistema Opera-

Figura 4 – Cache com instâncias de bibliotecas sendo usadas no momento.



cional Windows. O `new` conecta todos os *handlers* com a biblioteca e seus respectivos processos, para que seja possível fazer as suas chamadas. Também faz a cópia da biblioteca para a cache e declara variáveis de ambiente para a instância em questão, como é o caso do `spinit`, arquivo de inicialização do `ngspice`. Feito esse processo, os *handlers* já estão prontos para receber e enviar chamadas para a biblioteca `spice`, através do método `call`, na linha 34. Esse método recebe como argumento o *handler* do processo e seus argumentos, e compara o nome recebido aos declarados no Código 4.5, a partir da linha 5. Concluída a chamada do processo escolhido, um `uintptr` com o endereço do resultado é retornado².

```

1 import (
2     "fmt"
3     "os"
4     "syscall"
5     "unsafe"
6 )
7
8 type procHdl = *syscall.Proc
9 type libHdl = *syscall.DLL
10
11 const (
12     libName      = "ngspice.dll"
13     spinitEnv    = "SPICE_SCRIPTS"
14     spinitPath   = "\\share\\ngspice\\scripts\\"
15     cachePath    = "\\cache\\instances\\"
16     fileExt      = ".dll"
17 )
18
19 func (i *NgInstance) newI() {
20     setSpinitPath(spinitPath)
21     i.cacheLib()
22     h := syscall.MustLoadDLL(i.libPath)
23     i.handler = h
24     i.initP = h.MustFindProc(initName)

```

² Existem algumas convenções para utilizar `unsafe.Pointer`'s, uma delas é que para chamadas como `syscall.Syscall` (similar ao `Call`), o ponteiro deve ser convertido para `uintptr` dentro da chamada da função, assim o compilador não deixa esse argumento ser destruído até o retorno ser recebido. Mais em <<https://pkg.go.dev/unsafe#Pointer>>

```

25 i.initSyncP = h.MustFindProc(initSyncName)
26 i.commandP = h.MustFindProc(commandName)
27 i.runningP = h.MustFindProc(runningName)
28 i.GetVecInfoP = h.MustFindProc(getVecInfoName)
29 i.circP = h.MustFindProc(circName)
30 i.curPlotP = h.MustFindProc(curPlotName)
31 i.allPlotsP = h.MustFindProc(allPlotsName)
32 i.allVecsP = h.MustFindProc(allVecsName)
33 }
34 func (i *NgInstance) Call(p procHdl, args []unsafe.Pointer)
    uintptr {
35     var ret uintptr
36     var err error
37     switch p.Name {
38     case initName:
39         ret, _, err = p.Call(uintptr(args[0]), uintptr(args[1]),
            uintptr(args[2]), uintptr(args[3]), uintptr(args[4]),
            uintptr(args[5]), uintptr(args[6]))
40     case commandName:
41         ret, _, err = p.Call(uintptr(args[0]))
42     case runningName:
43         ret, _, err = p.Call()
44     case getVecInfoName:
45         ret, _, err = p.Call(uintptr(args[0]))
46     case circName:
47         ret, _, err = p.Call(uintptr(args[0]))
48     case curPlotName:
49         ret, _, err = p.Call()
50     case allPlotsName:
51         ret, _, err = p.Call()
52     }
53     if err != nil {
54         // fmt.Printf("%s err.Error(): %v\n", p.Name, err)
55     }
56     return ret
57 }

```

Código 4.6 – Métodos newI e Call.

Além do *wrapper* interno do gongs, existe uma interface para os tipos e retornos da biblioteca, alguns exemplos estão no Código 4.7 e Código 4.8. Essas e outras ferramentas são utilizadas para tornar possível desenvolver outras aplicações sem a necessidade de lidar com os tipos em C.

```

1 type VectorInfo struct {
2     VName      string
3     VType      int
4     VFlags     int16
5     VRealData []float64
6     VCompData []complex128
7     VLength    int
8 }
9
10 type VecValues struct {

```

```

11  Name      string
12  CReal     float64
13  CImag     float64
14  IsScale   bool
15  IsComplex bool
16 }
17
18 type VecValuesAll struct {
19     VecCount int
20     VecIndex int
21     VecsA    []*VecValues
22 }

```

Código 4.7 – Exemplo de tipos.

```

1  /* Dvec flags. */
2  const (
3      vf_real      = (1 << 0) /* The data is real. */
4      vf_complex   = (1 << 1) /* The data is complex. */
5      vf_accum     = (1 << 2) /* writedata should save this vector.
6                               */
7      vf_plot      = (1 << 3) /* writedata should incrementally
8                               plot it. */
9      vf_print     = (1 << 4) /* writedata should print this vector
10                              . */
11     vf_mingiven   = (1 << 5) /* The v_minsignal value is valid. */
12     vf_maxgiven   = (1 << 6) /* The v_maxsignal value is valid. */
13     vf_permanent  = (1 << 7) /* Don't garbage collect this vector.
14                              */
15 )
16
17 /* Plot types. */
18 const (
19     plot_lin      = 1
20     plot_comb     = 2
21     plot_point    = 3
22 )
23
24 type curVecsData struct {
25     curVecValsAll *ngtype.VecValuesAll
26     curVecInfoAll *ngtype.VecInfoAll
27     curVecInfo     *ngtype.VecInfo
28     curVectorInfo  *ngtype.VectorInfo
29 }
30
31 func (vecData *curVecsData) storeCurVecInfoAll(pvia C.
32     pvecinfoall) {
33     veccount := int(pvia.veccount)
34
35     if vecData.curVecInfoAll == nil {
36         vecData.curVecInfoAll = new(ngtype.VecInfoAll)
37     }
38     if vecData.curVecInfoAll.VecCount != veccount {
39         vecData.curVecInfoAll.Vecs = make([]*ngtype.VecInfo,
40             veccount)
41     }
42 }

```

```

36
37   vecData.curVecInfoAll.Date = C.GoString(pvia.date)
38   vecData.curVecInfoAll.Name = C.GoString(pvia.name)
39   vecData.curVecInfoAll.Title = C.GoString(pvia.title)
40   vecData.curVecInfoAll.VecCount = int(pvia.veccount)
41   for i, v := range unsafe.Slice(pvia.vecs, vecData.
      curVecInfoAll.VecCount) {
42       storeVecInfo(v, &vecData.curVecInfoAll.Vecs[i])
43   }
44 }

```

Código 4.8 – Transformação de tipos C para nativos em Go.

Por fim, o **gongs** conta com a sua interface para importação como pacote em aplicativos Go. Todas as funções foram abstraídas para usar apenas tipos em Go. O exemplo de algumas chamadas pode ser visto no Código 4.9³.

```

1 type NgSpice struct{ inst *wrapper.NgInstance }
2 type NgCallbacks = ngcallback.Callbacks
3
4 func Init(callbacks *NgCallbacks) *NgSpice {
5     return &NgSpice{inst: wrapper.Init(callbacks)}
6 }
7
8 func RestoreNgSpice(ngp unsafe.Pointer) *NgSpice {
9     return &NgSpice{inst: (*wrapper.NgInstance)(ngp)}
10 }
11
12 func (ngspice *NgSpice) GetUID() string {
13     return ngspice.inst.GetUID()
14 }
15
16 func (ngspice *NgSpice) Command(command string) int {
17     return ngspice.inst.Command(command)
18 }
19
20 func (ngspice *NgSpice) Clear() int {
21     return ngspice.inst.Clear()
22 }
23
24 func (ngspice *NgSpice) Quit() {
25     ngspice.inst.Quit()
26 }

```

Código 4.9 – Interface para importação gongs.

4.2 API

A API, apesar de ser o conector da interface e do **gongs**, é talvez um dos componentes mais simples do ambiente atualmente, **já que devido a restrição de tempo,**

³ A documentação e comentários não estão presentes no exemplo, mas podem ser encontradas nos códigos fonte em <<https://github.com/Patrick-Varela/gongs/>>



não foram implementadas todas as funcionalidades previstas. No entanto, já estão em desenvolvimento novas atualizações. A API também é responsável por criar e persistir os elementos. Por exemplo, para que um usuário possa acessar um elemento na interface, tal elemento deve ser montado como no Código 4.10.

```
1 vdd := usecases.NewDevice(  
2     models.Device{  
3         Label:      "Fonte de tensão",  
4         Description: "Esse tipo de fonte é usada para simular uma  
                    fonte de tensão. Ela é ideal, ou seja, não possui resistê  
                    ncia interna. Além disso é possível fazer uso de funções  
                    lineares em trechos (PWL) para simular fontes de tensão  
                    mais complexas.",  
5         SVGPath:    "Voltage_Source.svg",  
6         Type:       models.VS,  
7         Nodes: []models.Node{  
8             {  
9                 Name:      "Positivo",  
10                Description: "Positivo da fonte",  
11                NodePosition: models.TopCenter,  
12            },  
13            {  
14                Name:      "Negativo",  
15                Description: "Negativo da fonte",  
16                NodePosition: models.BotCenter,  
17            },  
18        },  
19        Parameters: []models.Parameter{  
20            {  
21                Label:      "Tensão",  
22                Description: "Tensão da fonte",  
23                Unit:       "V",  
24                Min:        0,  
25                Max:        20,  
26            },  
27            {  
28                Label:      "PWL",  
29                Description: "Função linear em trechos. f: (tempo,  
                    tensão), ...",  
30                IsArray:    true,  
31                IsOptional: true,  
32                SubParamsStructure: []models.Parameter{  
33                    {  
34                        Label: "Tempo",  
35                        Unit:  "ns",  
36                        Min:   0,  
37                        Max:   100,  
38                    },  
39                    {  
40                        Label: "Tensão",  
41                        Unit:  "V",  
42                        Min:   0,  
43                        Max:   20,  
44                    },  
45                },  
33            },  
34        },  
35    },  
36    )
```



```

46         },
47     },
48 },
49 )

```

Código 4.10 – Criação de elementos.

Dessa forma, a interface gráfica tem todas as informações necessárias para desenhar o elemento para o usuário. Como exemplo de persistência, o tipo `Device` é armazenado diretamente com o MongoDB, já que seu modelo possui as descrições de persistência. O processo é realizado pela própria biblioteca do MongoDB, como demonstra o Código 4.11.

```

1
2 ///Pacote models
3
4 type Device struct {
5     ID primitive.ObjectID `json:"id" bson:"_id,omitempty"`
6
7     Name      string `json:"name" bson:"name,omitempty"`
8     IsFixed   bool   `json:"isFixed" bson:"isFixed,omitempty"`
9
10    Label      string `json:"label" bson:"label,omitempty"`
11    Description string `json:"description,omitempty"`
12    SVGPath     string `json:"svgPath" bson:"svgPath,omitempty"`
13
14    Type      DeviceType `json:"type" bson:"type,omitempty"`
15    Nodes     []Node      `json:"nodes" bson:"nodes,omitempty"`
16    Parameters []Parameter `json:"parameters" bson:"parameters,omitempty"`
17
18    Position Position `json:"position" bson:"position,omitempty"`
19    Rotation Rotation `json:"rotation" bson:"rotation,omitempty"`
20 }
21
22 ///Pacote deviceRepository
23
24 func (r *MongoDBDeviceRepo) Create(device *models.Device) error {
25     res, err := r.getCollection().InsertOne(context.Background(),
26         device)
27     if err != nil {
28         return err
29     }
30     device.ID = res.InsertedID.(primitive.ObjectID)
31     return nil
32 }

```

Código 4.11 – Modelo e persistência.

Definidas essas estruturas básicas, é possível criar controladores e casos de uso. Além dos `Device's` há também os `CircuitBoard's`, esses *structs* servem para determinar quais elementos estão no quadro, quais nodos estão conectados e também

verificam se seus nomes são únicos, bem como outros requisitos para serem simulados. O método do Código 4.12, implementado em `circuitusecase.go`, tem o resultado apresentado no final do código como comentário, depois de ser chamado com um circuito carregado com uma fonte `vdd`⁴ de 5 Volts, no modelo apresentado no código anterior. Depois desses exemplos, **mais abaixo** é demonstrado como esse resultado pode ser inserido ao gongs para a simulação.



```

1 func (c *CircuitUseCase) GetCircuitDescription() []string {
2     strArray := make([]string, 0)
3     strArray = append(strArray, c.Name)
4     strArray = append(strArray, ".include 32nm_HP.pm")
5     for _, element := range c.Elements {
6         if element.GetType() == string(models.GND) {
7             continue
8         }
9         strArray = append(strArray, element.GetCircuitDescription()
10        )
11    }
12    strArray = append(strArray, ".end")
13    return strArray
14 }
15
16 ///Exemplo
17 cir := usecases.NewCircuit("Circuito 1")
18 vdd.SetName("vdd")
19 vdd.SetParameterValue("Tensão", 5)
20 vdd.AddNode(0, "vdd")
21 vdd.AddNode(1, "gnd")
22 cir.AddElement(vdd)
23 dsc := cir.GetCircuitDescription()
24 for _, line := range dsc {
25     fmt.Println(line)
26 }
27
28 ///Saída
29
30 //Circuito 1
31 //.include 32nm_HP.pm
32 //Vvdd vdd gnd 5V
33 //.end
34
35 ///Simulando
36
37 cb := new(gongs.NgCallbacks)
38 cb.SendChar = func(str string, id int, ptr unsafe.Pointer) int
39 {
40     fmt.Println("NGSPICE: ", str)
41     return 0
42 }
43 ng := gongs.Init(cb)
44 ng.Circ(c.GetCircuitDescription())

```

⁴ Dreno de tensão.

```
44 ng.Command("tran 0.1ns 20ns")
```

Código 4.12 – Exemplo de `circuitusecase.go`.

Por fim, a API é inicializada como apresentado no Código 3.2 e já pode usar os *controllers*, responsáveis pelas validações e tratamento de requisições, como acontece no exemplo do Código 4.13, onde também é apresentado como as rotas são criadas.



```
1  ///api.go
2
3  package api
4
5  import (
6      "github.com/gofiber/fiber/v2"
7  )
8
9  func RouteApi(app fiber.Router) {
10     api := app.Group("/api")
11     getApiV1(api)
12 }
13
14 ///v1.go
15
16 package api
17
18 import (
19     "2AI2C.com/2ai2c/internal/controllers"
20     "2AI2C.com/2ai2c/internal/middlewares"
21     "github.com/gofiber/fiber/v2"
22 )
23
24 func getApiV1(app fiber.Router) {
25     api := app.Group("/v1")
26
27     deviceController := controllers.NewDeviceController()
28     userController := controllers.NewUserController()
29     circuitController := controllers.NewCircuitController(
30         deviceController)
31
32     api.Post("/user/register", userController.Register)
33     api.Post("/user/login", userController.Login)
34
35     api.Use(middlewares.AuthMiddleware())
36
37     api.Get("/devices", deviceController.FindAll)
38     api.Post("/devices", deviceController.Create)
39
40     api.Get("/circuit/simulate", circuitController.SimSSE)
41     api.Post("/circuit/simulate", circuitController.Simulate)
42     api.Post("/circuit/simulate/continue",
43         circuitController.SimulateContinue)
44 }
```

Código 4.13 – Declaração das rotas e uso dos *controllers*.

4.3 Interface

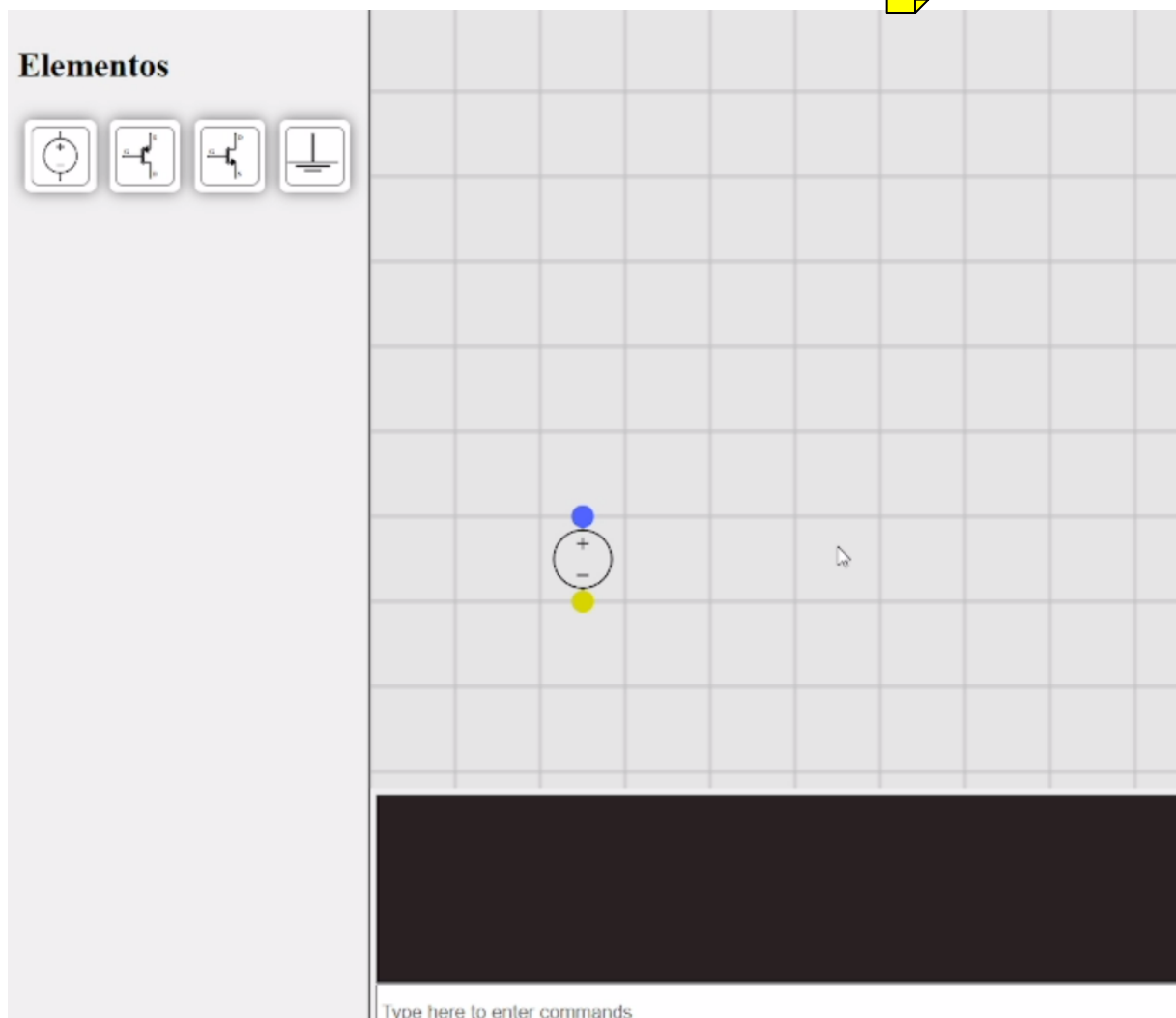
Para o desenvolvimento da interface, o Paper.js foi usado para implementação do quadro do circuito, enquanto que as funções de *popover* foram implementados com uso do floating-ui e Lit. O Paper.js funciona manipulando um elemento *canvas*, possibilitando desenhar os elementos gráficos necessários. Um exemplo de desenho pode ser observado no Código 4.14. O item gráfico é atribuído ao atributo *gObject* no construtor e, durante a inicialização, é desenhado no quadro na linha 2. O resultado do código para uma fonte é demonstrado na Figura 5

```
1 initialize(point: paper.PointLike) {  
2   this.gObject.draw(this.svgItem, {  
3     cellSize: cellSize,  
4     interactiveBounds: true,  
5   });  
6   const paperItem = this.gObject?.getPaperItem();  
7   if (!paperItem) {  
8     throw new Error('Paper item is null or undefined');  
9   }  
10  paperItem.fitBounds({  
11    x: 0,  
12    y: 0,  
13    width: cellSize,  
14    height: cellSize,  
15  });  
16  paperItem.strokeWidth = 1;  
17  this.gObject.setPosition(new Point(point));  
18  
19  this.device.nodes.forEach((DeviceNode: DeviceNode) => {  
20    this.nodes.push(new ElementNode(DeviceNode, this));  
21  });  
22  
23  this.interactiveShape = this.gObject.getInteractiveBounds()!;  
24  
25  if (!this.device.isFixed) this.generateUniqueName();  
26 }
```

Código 4.14 – Inicialização de elemento com desenho.



Figura 5 – Desenho de uma fonte.



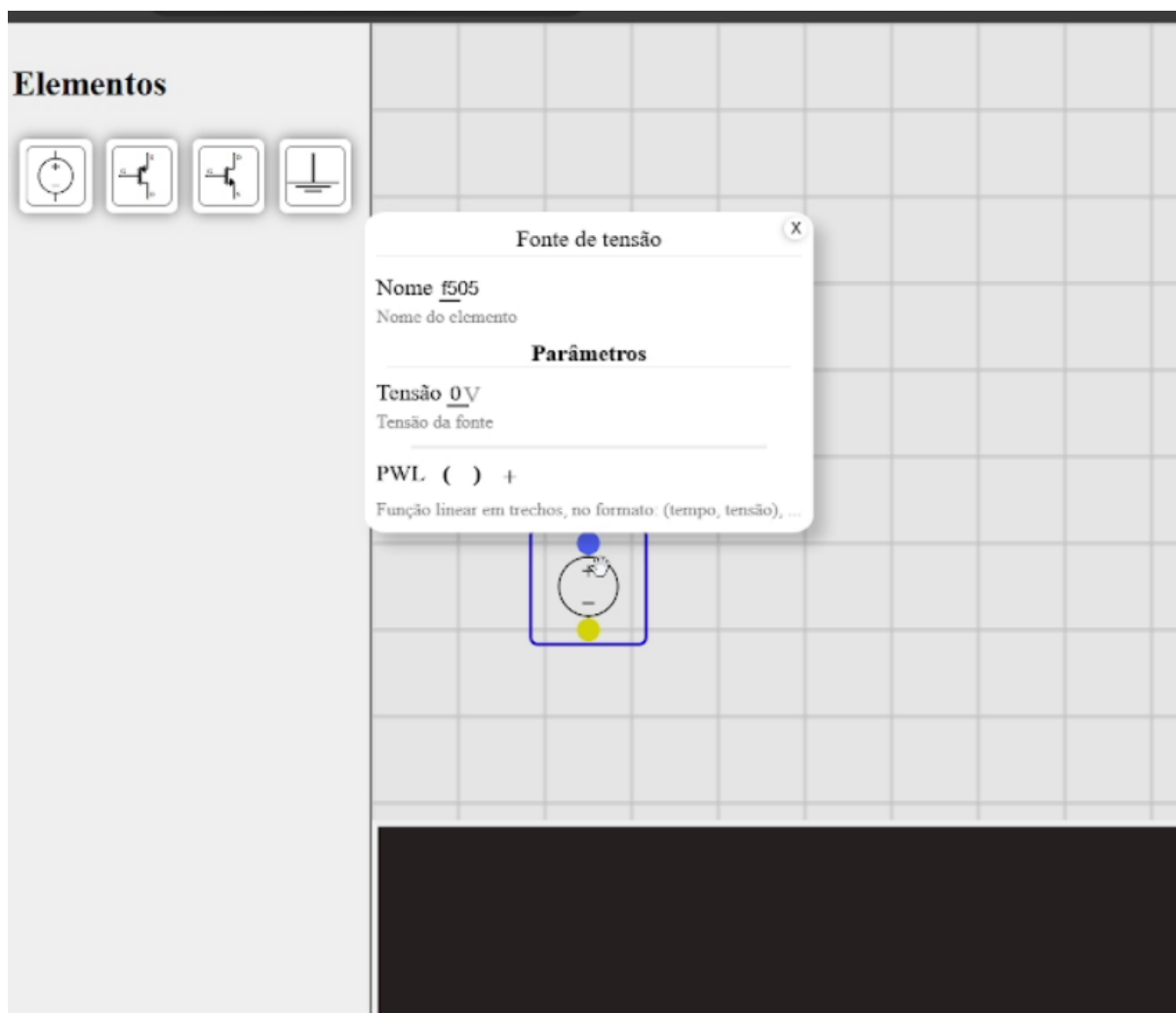
Depois de desenhado na tela, o usuário pode movimentar o elemento, deletar ou interagir. Quando o mouse passa por cima da área interativa de um elemento, como definido na linha 32 do Código 4.14, ocorre a exibição de três menus. Estes menus foram desenvolvidos com o uso do floating-ui, para posicionamento sobre o item, e o Lit, para a construção do conteúdo. No Código 4.15 é possível ver que o objeto implementa os três menus possíveis e mais outras três interações. O resultado de usar o botão esquerdo do mouse sobre o elemento pode ser visto na Figura 6. O conteúdo deste menu é feito através do Lit, como no Código 4.12. Esse menu é o responsável por disponibilizar as configurações dos parâmetros dos elementos.

```
1 export class CircuitElement
2   implements
3     ClickableContext<paper.Item>,
4     HoverAble<paper.Item>,
5     RightClickable<paper.Item>,
6     Selectable,
7     MoveAble,
```

```
8     RotateAble
9 {
10 ...
11 getClickMenu(): HTMLElement {
12     const clickMenu = new ElementClickContext();
13     clickMenu.element = this;
14     return clickMenu;
15 }
```

Código 4.15 – Declaração de CircuitElement com implementações.

Figura 6 – Menu clique esquerdo.

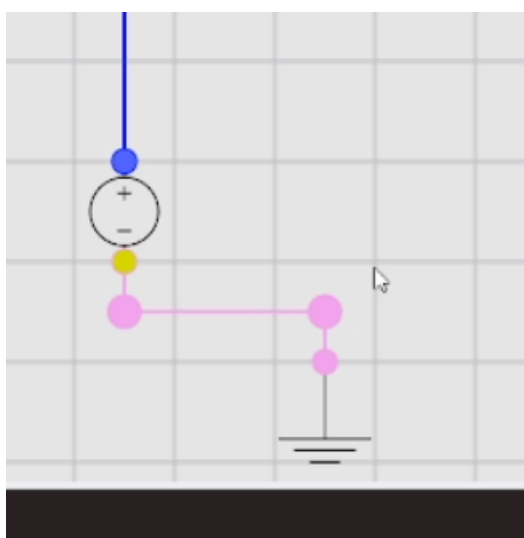


Quando o usuário já terminou de configurar o elemento, é possível clicar nos nós, representados por círculos, e criar novos fios sobre eles, como na Figura 7. Então é possível criar o caminho até o próximo elemento e conectá-los, como no exemplo da Figura 8.

Figura 7 – Menu para criar novo fio.



Figura 8 – Nodos da fonte e terra conectados



Com os elementos disponíveis no exemplo dessa imagem, é possível criar um inversor, como na Figura 9, que é um exemplo de como o quadro pode ser exportado. Para fazer a simulação, basta clicar no botão de “executar simulação”, que fica ao lado do terminal. Durante a simulação, o resultado, no MVP, é apresentado em um gráfico, quando clicado nos fios do circuito, como na Figura 10. Neste circuito, uma fonte de 1V está sendo simulada.

Figura 9 – Quadro com inversor exportado.

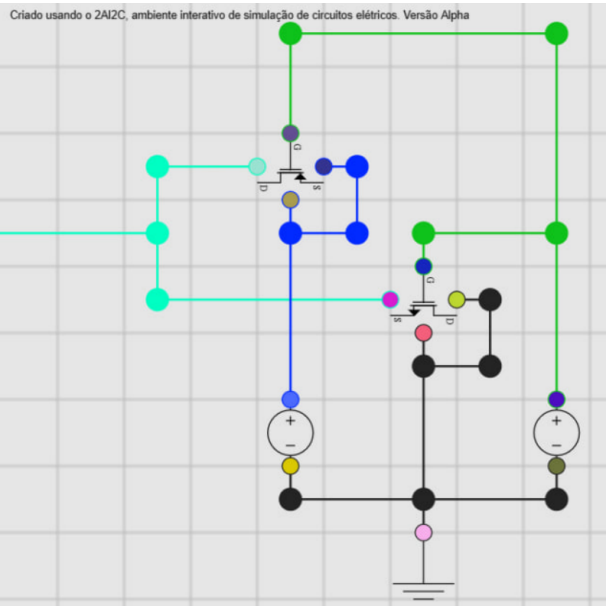
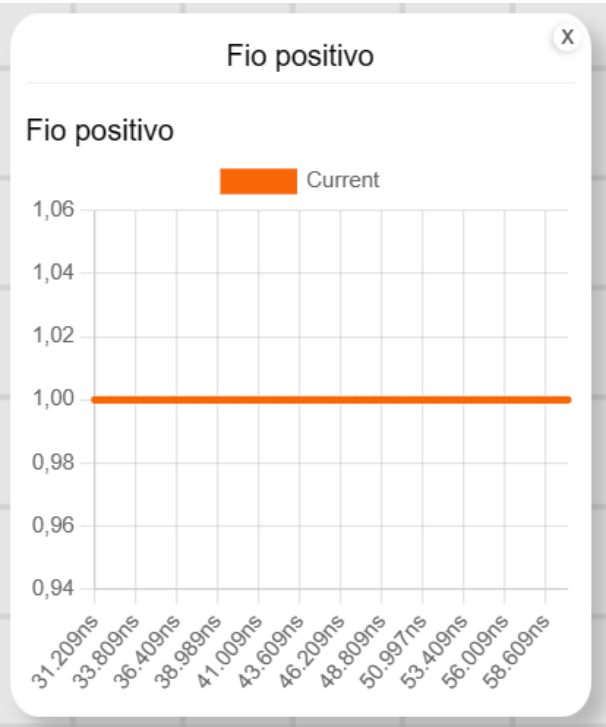


Figura 10 – Resultado da simulação.



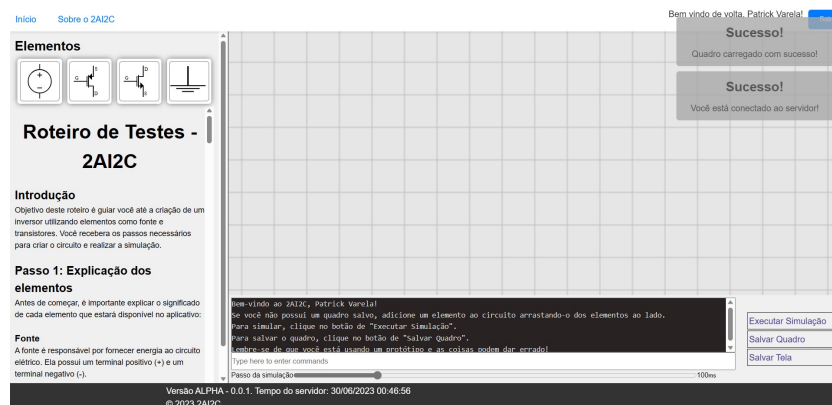
5 Testes



5.1 Roteiro de Testes

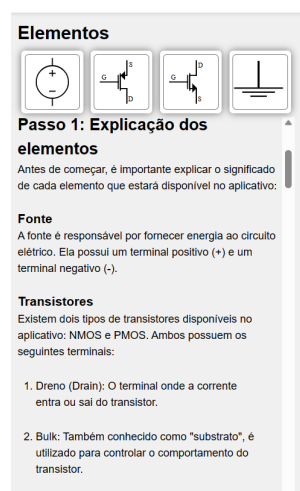
Para validar a plataforma desenvolvida, um roteiro de testes foi elaborado com o objetivo de verificar o funcionamento e a usabilidade da plataforma. O roteiro possui um guia que ensina o usuário a criar um inversor, utilizando elementos como fonte e transistores PMOS e NMOS. Na plataforma, o roteiro está localizado na parte esquerda, como na Figura 11.

Figura 11 – Quadro com roteiro.



Inicialmente, o roteiro explica o significado de cada elemento disponível na plataforma. Como os transistores NMOS e PMOS, que foram descritos abordando os terminais de cada um, como Dreno (*Drain*), Substrato (*Bulk*), Porta (*Gate*) e Fonte (*Source*). Isso pode ser observado na Figura 12.

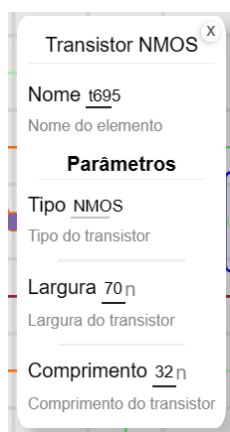
Figura 12 – Descrição dos componentes no roteiro.



Em seguida, é apresentado o passo a passo para a criação de um inversor. Após a montagem do circuito, é feito o ajuste de parâmetros dos elementos do circuito,

como a tensão da fonte e as dimensões dos transistores. Nesse roteiro, foi usada uma fonte com a função *pulse*, que, em pulsos, alterna a tensão. Um exemplo da alteração de parâmetros pode ser observado na Figura 13.

Figura 13 – Definindo parâmetros dos elementos.



Montado o circuito, como na Figura 15, o usuário deve realizar a simulação do mesmo. Com isso, é possível entender o comportamento do inversor e verificar se o funcionamento estava de acordo com o esperado. O resultado pode ser visto na Figura 14, mas também estará disponível para assistir em vídeo¹.

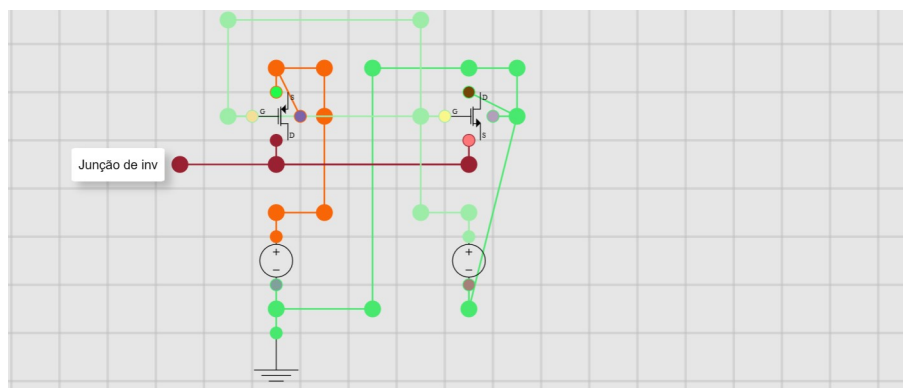
Figura 14 – Simulação em andamento.



Por fim, o roteiro aborda a funcionalidade de salvar a tela do circuito criado, como na Figura 9. O que permite exportar uma imagem do circuito projetado, possibilitando compartilhar e revisitar o circuito posteriormente. O roteiro abrangeu todos os aspectos implementados da plataforma, portanto validando o ambiente.

¹ Para assistir acesse: <<https://youtu.be/oojzuAOpTws>>

Figura 15 – Inversor criado usando o passo a passo.



5.2 Formulário



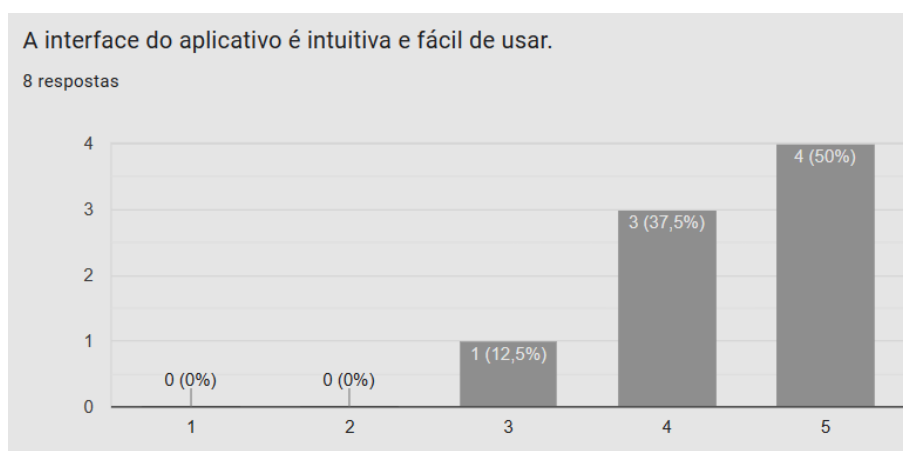
Usuários foram convidados para usar a plataforma a fim de testar as suas ferramentas e a capacidade de realizar simulações. Os participantes deveriam seguir o roteiro, simulando e salvando o circuito desenvolvido. Ao fim, um formulário era apresentado para que fossem respondidas perguntas sobre a sua experiência. Foram feitas 10 afirmativas que poderiam ser avaliadas entre 0 e 5, sendo 0 “Discordo totalmente” e 5 “Concordo totalmente”. No total **houveram** 8 participantes, sendo 4 que estão cursando ou já concluíram cursos STEM.



Serão apresentadas as afirmativas **mais** relevantes do formulário, assim como suas respostas. A primeira afirmativa é a que segue: “A interface do aplicativo é intuitiva e fácil de usar.”, os resultados podem ser vistos na Figura 16. Observando as respostas, pode-se concluir que os usuários concordam em grande maioria que o ambiente é intuitivo e fácil de usar.



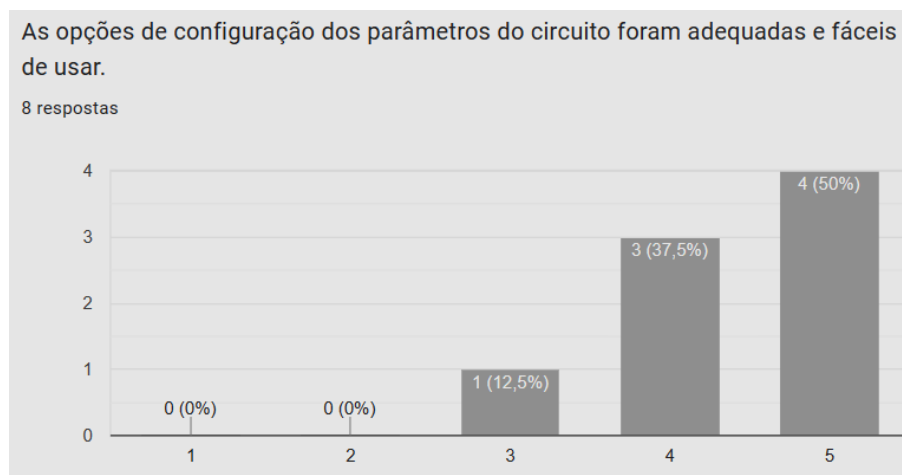
Figura 16 – Pergunta número 1 do formulário.



Depois foi solicitado que o participante avaliasse a afirmação: “As opções de configuração dos parâmetros do circuito foram adequadas e fáceis de usar.”. Essa afirmativa está relacionada com a função observada na Figura 13, e é pertinente pois pode

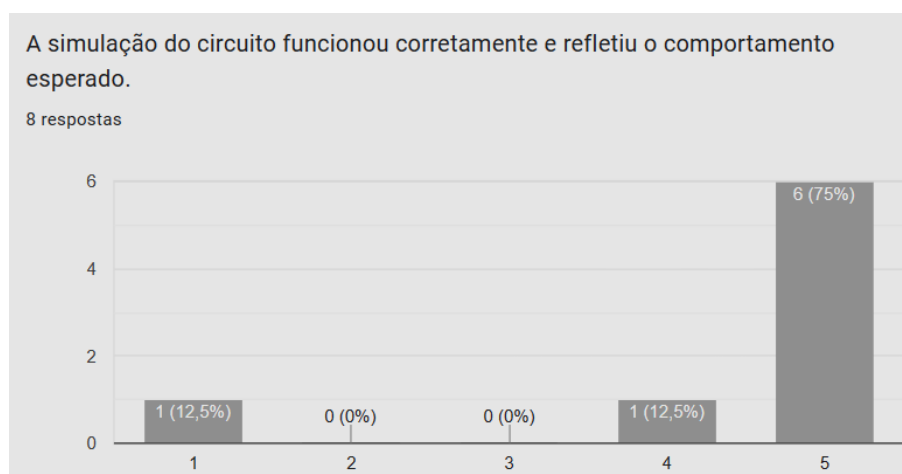
conter muitas informações e se tornar confusa. No entanto, a tendência de resposta parece condizer com a pergunta anterior, o resultado pode ser visto na Figura 17.

Figura 17 – Pergunta número 4 do formulário.



A próxima afirmativa estava interessada em compreender sobre a simulação do sistema. Ou seja, se ela se portou como esperado e se atendeu as expectativas do participante. Era pedido que avaliasse a seguinte frase: “A simulação do circuito funcionou corretamente e refletiu o comportamento esperado.”. Com essa afirmação, é possível identificar se a simulação está de acordo com o esperado. Como visto na Figura 18, diferente das demais perguntas até então, nesta houve uma avaliação como “Descordo totalmente”. Como comentário, o participante adicionou a seguinte informação: “A simulação não funcionou”. Por se tratar de uma versão inicial e pela grande carga observada no servidor da plataforma, a simulação nem sempre era executada, fazendo com que fosse necessário recarregar a página para que voltasse a funcionar.

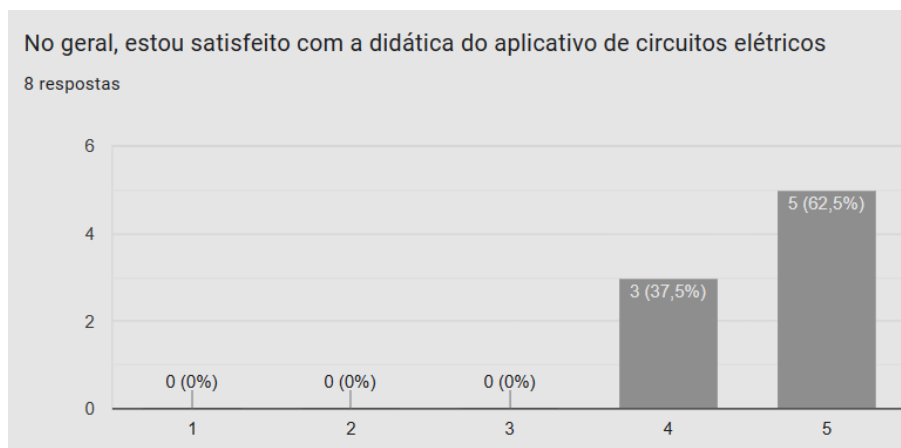
Figura 18 – Pergunta número 5 do formulário.



Por fim, os respondentes foram solicitados a avaliar a seguinte afirmação: “No geral, estou satisfeito com a didática do aplicativo de circuitos elétricos”. Com esse item,

o interesse é compreender a satisfação e percepção geral do usuário. O resultado pode ser observado a seguir na Figura 19.

Figura 19 – Pergunta número 10 do formulário.



Com essas avaliações concluídas, foi solicitado para que o participante preenchesse informações sobre a sua formação. Uma das perguntas questionava sobre o curso atual ou sua formação. A Figura 20 indica que metade dos participantes cursa ou possui formação na área STEM. Logo, foi questionado a esses usuários, se já haviam usado o NgSpice, e adicionalmente formulada a questão que segue: “Se sim, você acredita que essa aplicação facilita o entendimento dessas ferramentas?”. O resultado dessa pergunta pode ser visto na Figura 21

Figura 20 – Pergunta sobre curso ou formação do participante.

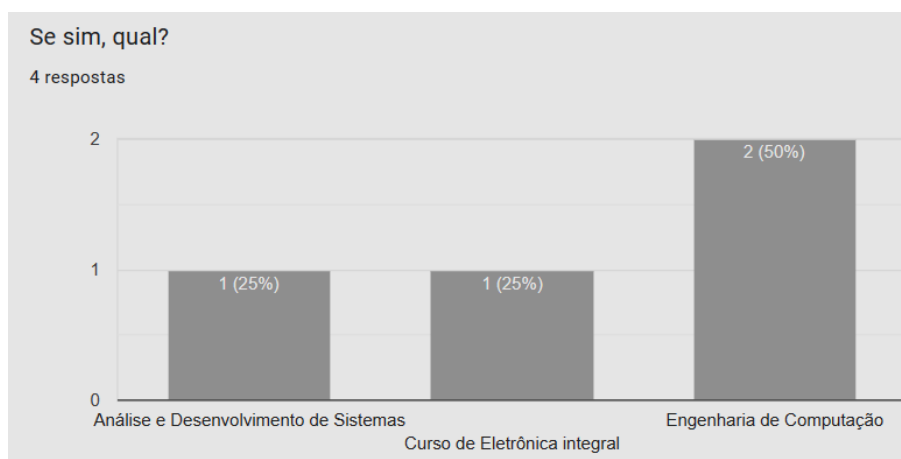
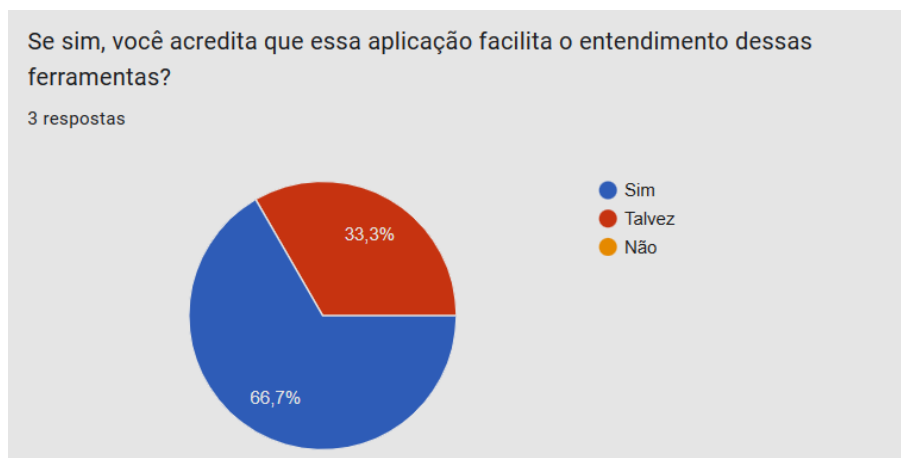


Figura 21 – Pergunta sobre a viabilidade do uso da plataforma para facilitar o entendimento de ferramentas como o NgSpice.



5.3 Discussão

Analizando os resultados apresentados, é possível afirmar que o 2AI2C atendeu ao seu propósito, mesmo que para uma amostra pequena de participantes. Foi intuitivo e fácil de utilizar, simulou o circuito de acordo com o esperado, exceto nos casos em que a simulação não pôde ser iniciada devido à alta demanda do servidor. As expectativas didáticas para a ferramenta foram atendidas e nos comentários os participantes também demonstraram a sua satisfação. Um participante disse “Muito bom o sistema, bem útil e fácil de se utilizar.” e outro reforçou essa ideia “Interativo e bem detalhado nos comandos, muito bom!!”.

Houveram comentários deixados na pesquisa que demonstram que a ferramenta, mesmo não sendo utilizada pelo seu público alvo, ainda demonstra capacidade de melhorar a didática, algo levantado no seguinte comentário: “Não sou estudante da área, por isso não entendo nada do assunto, mas achei o 2AI2C interessante. Acredito que com a solução de alguns bugs, seja um ótimo auxílio para a aprendizagem.”.

Ainda observando o comentário anterior, é possível notar que a plataforma precisa de melhorias. Outro comentário retrata bem a realidade, mas também exalta os pontos positivos do aplicativo, “Apesar de apresentar alguns bugs e ainda necessitar de algumas melhorias, o aplicativo cumpre com o que é dele esperado, com um passo a passo claro e objetivo e uma interface simples e intuitiva, que o tornam de fácil utilização.”.

6 Conclusão



O presente trabalho se propôs a relatar o desenvolvimento da aplicação 2AI2C, destinada a auxiliar no ensino e aprendizagem de circuitos elétricos/eletrônicos, bem como na diminuição da curva de aprendizado para os alunos. O seu desenvolvimento chegou em fase de MVP, onde até então foram desenvolvidos um *wrapper* em volta da biblioteca compartilhada do *NgSpice* a aplicação separada em *front-end* e *back-end*.

Atualmente, a plataforma se encontra online¹, com o formulário de pesquisa ainda disponível, junto com o roteiro. A intenção é coletar mais informações para adquirir um repertório mais rico de melhorias e definir melhor o caminho que o 2AI2C deve seguir daqui em diante. Além disso, para que a aplicação pudesse ser idealizada, um pacote Go foi desenvolvido e publicado como código livre².

Para trabalhos futuros, estão previstos requisitos não-funcionais, melhorias e correção de erros. Uma parte essencial que foi planejada, mas ainda não teve seu desenvolvimento concluído, é a plataforma do educador. Embora já esteja em desenvolvimento, as melhorias no sistema de simulação precisam ser priorizadas para garantir o pleno funcionamento do ambiente. No entanto, o aplicativo cumpriu seu objetivo de tornar a curva de aprendizado menor, transformando a criação de circuitos elétricos em uma experiência mais interativa e compreensível, com visualizações e simulações controladas, em um ambiente web que pode ser utilizado como OA.



¹ Você pode acessar no link <http://2ai2c.com.br>

² Disponível em <https://github.com/Patrick-Varela/gongs>



Referências

DANTAS, A.; MAIA, D.; SCAICO, P. Objetos de aprendizagem: da definição ao desenvolvimento, passando pela sala de aula. In: _____. [S.l.: s.n.], 2021. p. 1. ISBN 978-65-87003-34-4. Citado na página 10.

GO. **Case Studies - The Go Programming Language**. s.d. Disponível em: <<https://go.dev/solutions/case-studies>>. Acesso em: 2023-06-24. Citado na página 16.

GO. **cgo**. S.d. Disponível em: <<https://github.com/golang/go/wiki/cgo>>. Acesso em: 2023-06-25. Citado na página 23.

GO. **cgo command - cmd/cgo - Go Packages**. s.d. Disponível em: <https://pkg.go.dev/cmd/cgo#hdr-Passing_pointers>. Acesso em: 2023-06-25. Citado na página 23.

GOFIBER. **gofiber/fiber**. Fiber, 2023. Disponível em: <<https://github.com/gofiber/fiber>>. Acesso em: 2023-06-25. Citado na página 17.

HOLTON, D.; VERMA, A.; BISWAS, G. Assessing student difficulties in understanding the behavior of ac and dc circuits. 01 2008. Citado na página 10.

MONGODB. **O Que É O MongoDB?** s.d. Disponível em: <<https://www.mongodb.com/pt-br/what-is-mongodb>>. Acesso em: 2023-06-25. Citado na página 19.

NGSPICE. **Ngspice circuit simulator**. s.d. Disponível em: <<https://ngspice.sourceforge.io/presentation.html#>>>. Acesso em: 2023-06-25. Citado na página 20.

PAPER.JS. **Paper.js — About**. s.d. Disponível em: <<http://paperjs.org/about/>>. Acesso em: 2023-06-24. Citado na página 18.