Alexandre Marotti da Fonseca

NeuroNode:
A Visual Programming Tool to aid in neuroscience research and teaching

Recife, Pernambuco

2024

Alexandre Marotti da Fonseca

NeuroNode:
A Visual Programming Tool to aid in neuroscience research and teaching

Trabalho apresentado ao Programa de Graduação em Ciência da computação da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Universidade Federal De Pernambuco – UFPE

Centro de Informática

Orientador: Breno Miranda

Coorientador: Nivaldo Antonio Portela de Vasconncelos

Recife, Pernambuco

2024

Ficha de identificação da obra elaborada pelo autor,
através do programa de geração automática do SIB/UFPE

Temporal, Alexandre Marotti da Fonseca.
NeuroNode: A Visual Programming Tool to aid in neuroscience research and teaching / Alexandre Marotti da Fonseca Temporal. - Recife, 2024.
42 p. : il., tab.

Orientador(a): Breno Alexandro Ferreira de Miranda
Cooorientador(a): Nivaldo Antonio Portela De Vasconcelos
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Informática, Ciências da Computação - Bacharelado, 2024.
Inclui referências, apêndices.

1. Programação visual. 2. Python. 3. React. 4. Flask. 5. Neuroengenharia. I. Miranda, Breno Alexandro Ferreira de . (Orientação). II. Vasconcelos, Nivaldo Antonio Portela De. (Coorientação). IV. Título.

000 CDD (22.ed.)

Alexandre Marotti da Fonseca

NeuroNode:
A Visual Programming Tool to aid in neuroscience research and teaching

Trabalho apresentado ao Programa de Graduação em Ciência da computação da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Trabalho aprovado. Recife, Pernambuco, 23 de Agosto de 2024:

Profº Dr. Breno Miranda (Orientador)
Universidade Federal de Pernambuco

Profº Dr. Leopoldo Motta Teixeira
Universidade Federal de Pernambuco
(Examinador Interno)

Recife, Pernambuco
2024

Dedico este trabalho aos meus avós, que sempre quiseram me ver graduando. Sei que eles estão me vendo, apesar de não estarem mais conosco.

AGRADECIMENTOS

Agradeço primeiramente à minha mãe e à minha esposa, que sempre me acompanharam e apoiaram incondicionalmente ao longo de toda a minha jornada. Seu amor, paciência e incentivo foram fundamentais para a realização deste projeto, além de sempre estarem lá por mim nos momentos de dificuldade.

Agradeço também aos meus orientadores, Breno e Nivaldo, cuja orientação e expertise foram essenciais para o desenvolvimento deste trabalho. Sem eles, nada disso seria possível. Nivaldo me ensinou tanto sobre a área de neurociência, que sinto que estou saindo da faculdade com uma nova bagagem de conhecimento que não teria se não fosse por ele. E Breno me acolheu e guiou durante a criação desse TCC, e suas reuniões foram muito construtivas, pois pude olhar o trabalho de outros alunos e ver as mesmas dificuldades que estava passando, e aprender com as formas que eles lidavam com os problemas para gerar minhas próprias soluções.

A todos que contribuíram direta ou indiretamente para a concretização deste projeto, meu sincero agradecimento.

"É preciso tolerar
a presença de uma ou duas lagartas,
para poder conhecer as borboletas."
(Antoine de Saint-Exupéry, 1943)

RESUMO

Com o crescimento da quantidade de dados neuronais ao longo dos anos, aumentou também a demanda por ferramentas especializadas que possam auxiliar na interpretação desses dados. Em resposta a essa demanda crescente, novas linguagens de programação que propunham tornar mais fácil o processo de análise de dados e diminuir a labuta dese processo também surgiram e se tornaram populares, como python e R, que oferecem uma ampla gama de bibliotecas e ferramentas para análise estatística e visualização de dados. Porém, apesar da disponibilidade dessas ferramentas, ainda existem diversos pesquisadores de campos fora da computação que ainda enfrentam desafios, tendo em vista que muitos deles não possuem conhecimento em programação, ou possuem um conhecimento muito raso, o que pode limitar a sua capacidade de processar essa grande quantidade de dados. Essa dificuldade não se restringe apenas aos pesquisadores, mas também é uma realidade para muitos estudantes em áreas como medicina, ou engenharia biomédica que se encontram na mesma situação. Para enfrentar esse desafio e facilitar o trabalho desses pesquisadores, esta produção se propõe a criar uma ferramenta de programação visual para auxiliar pesquisadores e alunos das áreas de biologia, medicina e engenharia médica que não possuem conhecimento de programação a realizarem suas pesquisas de maneira mais simples, ou para facilitar o ensino por parte dos professores, para iniciar alunos no mundo da análise de dados neuronais sem que seja necessário conhecimento prévio de programação. Assim, reduzindo a barreira para a pesquisa científica e aprimorando o ensino nas disciplinas relacionadas.

Palavras-chave: Programação visual, python, react, flask, neuroengenharia, monografias.

ABSTRACT

As the amount of neuronal data has grown over the years, so has the demand for specialized tools that can help interpret this data. In response to this growing demand, new programming languages that set out to make the process of data analysis easier and to reduce the burden of this process have also emerged and become popular, such as python and R, which offer a wide range of libraries and tools for statistical analysis and data visualization. However, despite the availability of these tools, there are still a number of researchers from fields outside of IT who still face challenges, given that many of them have no or very little programming knowledge, which can limit their ability to process this large amount of data. This difficulty is not only restricted to researchers, but is also a reality for many students in areas such as medicine or biomedical engineering who find themselves in the same situation. To meet this challenge and facilitate the work of these researchers, this production aims to create a visual programming tool to help researchers and students in the fields of biology, medicine and medical engineering who have no knowledge of programming to carry out their research in a simpler way, or to facilitate teaching by teachers, to initiate students into the world of neural data analysis without the need for prior knowledge of programming. Thus, reducing the barrier to scientific research and improving teaching in related disciplines.

Keywords: Visual Programming, python, react, flask, neuroengineering, undergraduate thesis.

LISTA DE ABREVIATURAS E SIGLAS

IoT	Internet das coisas/Internet of Things
IFR	Taxa de Disparo Instantânea/Instant Firing Rate
CV	Coeficiente de Variação/Coefficient of Variation
IT	Tecnologia da Informação/Information Technology

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

Desde a antiguidade, nós sabemos da importância do sistema nervoso para a vida. Isso pode ser percebido por práticas como a trepanação, um procedimento cirúrgico que envolvia a perfuração e/ou corte do crânio, que possui registros datando de 7.000-10.000 anos atrás em todos os continentes(ANDRE, 2017) e pela maneira como o ser humano, diferentemente de outros animais, costuma atacar a cabeça, ao invés de, por exemplo, a garganta. Já no século XVIII, o experimento do sapo do Galvani mostrou a possível relação entre a eletricidade e a contração dos músculos e, portanto, sua capacidade de influenciar o sistema nervoso (VERKHRATSKY; KRISHTAL; PETERSEN, 2006). No entanto, foi com a doutrina neuronal proposta por Santiago Ramón y Cajal no final do século XIX que a neurociência começou a se desenvolver como uma disciplina científica. Cajal propôs que o cérebro era formado por células individuais chamadas neurônios.

Porém, foi com Cole e Curtis, em seu experimento com o axônio gigante de lula, em 1940, que começamos a realizar registros neuronais de potenciais de ação (VERKHRATSKY; KRISHTAL; PETERSEN, 2006). Essa parte do cérebro, que por muito tempo estava circulada de mistério, começou-se a ser revelada.

Agora, munidos do conhecimento sobre os neurônios e o mecanismo por trás de sua função, se tornou possível investigar de forma mais detalhada as diferentes etapas envolvidas no processamento sensorial e possíveis problemas que possam ocorrer nesse sistema. Dada a sua importância para a interação com o ambiente e a tomada de decisões em ambientes complexos(e.g calcular em tempo real a trajetória de um objeto vindo em sua direção), o processamento de estímulos visuais é de suma importância para o ser humano e qualquer deficiência nesse processamento podem impedir a independência e autodeterminação de um indivíduo.

Dessa maneira, a compreensão dos mecanismos neurais subjacentes ao processamento visual é essencial para o desenvolvimento de tratamentos para distúrbios visuais e outras condições relacionadas ao processamento sensorial.

Esses disturbios no processamento sensorial podem ocorrer em diferentes partes ao longo do caminho realizado pela informação sensorial no cérebro. Esse caminho começa nas células fotorreceptoras encontradas na retina, após receberem informação, estas células repassam o que receberam para o nervo óptico, percorrem através do quiasma óptico e seguem pelo trato óptico até chegar ao córtex visual. O córtex visual é a área do cérebro responsável pelo processamento das informações visuais.

A partir do córtex visual, as informações visuais são transmitidas para duas vias principais: a via dorsal e a via ventral. A via ventral é responsável pelo reconhecimento de objetos, enquanto a via dorsal é responsável pelo processamento de ação para com o

objeto e localização (MILNER; GOODALE, 1996).

Entender como e onde problemas podem ocorrer nessas vias e como elas afetam a percepção visual é crucial para entender os processor que geram distúrbios no processamento visual e desenvolver estratégias de tratamento para essas condições.

Uma das abordagens que podemos usar para podermos entender problemas no processamento visual é utilizar a atividade unitária de grandes populações de neurônios no cortex visual primário, que podem ou não ser combinadas com a atividade de outras áreas do cérebro, para analisar o funcionamento dessa região e como diferentes distúrbios se manifestam na atividade neuronal. Porém, dado o grande volume de dados registrados nessas áreas, nós precisamos de técnicas e tecnologias que nos permitam processar essa grande quantidade de dados.

Para dar suporte a esse processamento, precisamos de uma base de hardware e software que nos auxilie nessa tarefa. Do lado dos instrumentos de hardware, é necessário um grande espaço de armazenamento para guardar tais dados - por exemplo, uma probe com 10 canais, taxa de amostragem de 30khz e 10 bits de resolução ADC vai gerar um arquivo de 1.35GB de dados por hora de medição, para cada um dos animais sendo estudados - além disso, é necessária uma boa quantidade de RAM para poder guardar cada arquivo na hora do processamento, dependendo das especificações da probe e do tempo de gravação. Já do lado de software, são utilizados algoritmos de filtros para a remoção de freqências não desejáveis, algoritmos de clustering para o agrupamento de potenciais de ação, técnicas estatísticas para análise do dado, entre outras.

Porém, à medida que as técnicas de registro neuronal se desenvolvem, há um aumento proporcional na quantidade e no tamanho dos dados gerados. Esse aumento é tratado na pesquisa realizada por Stevenson e Kording (2011), na qual se mostra que o número de neurônios registrados simultaneamente apresenta um crescimento exponencial, dobrando aproximadamente a cada 7 anos e meio.

Dessa forma, para lidar com o aumento na quantidade e tamanho dos dados gerados pelas técnicas de registro neuronal, torna-se imprescindível o desenvolvimento de novas estratégias de processamento. Tais estratégias podem envolver a implementação de novos algoritmos para o tratamento de dados, incluindo técnicas de machine learning, além da exploração de novas arquiteturas de hardware, como TPIs, que são capazes de realizar cálculos com matrizes de forma mais eficiente do que os processadores tradicionais.

À medida que as técnicas de registro neuronal se desenvolvem, a quantidade e o tamanho dos dados gerados aumentam proporcionalmente, apresentando um crescimento exponencial, conforme citado anteriormente. Consequentemente, o crescimento exponencial dos dados torna cada vez mais desafiador realizar pesquisas utilizando computadores pessoais.

Nesse contexto, a necessidade de computadores mais poderosos se torna imprescindível para processar e analisar grandes volumes de dados neurais gerados pelas técnicas de registro. No entanto, nem todos os pesquisadores têm acesso a tais recursos computacionais devido à verba disponível ou localização geográfica. Dessa forma, não seria interessante se fosse possível que os pesquisadores tivessem disponíveis a seu dispor computadores capazes de realizar o processamento desses dados de qualquer lugar?

Uma solução para esse problema seria terceirizar o processamento necessário para um computador com suficiente poder de processamento na rede, de modo que seu computador pessoal não precise fazer nenhum processamento complexo.

Perceba que tal descrição descreve o modelo cliente-servidor. Dessa forma, o processamento dos dados seria delegado a um servidor mais poderoso. Outro benefício desse modelo, além de resolver o problema de capacidade de processamento citado anteriormente, é o de permitir que diversos cientistas colaborem de maneira simultânea no mesmo projeto.

Para exemplificar um fluxo de trabalho de um aluno em uma cadeira de neurociência ou de um pesquisador analisando dados de algum experimento, podemos enumerar os passos que devem ser seguidos da seguinte forma(Essas funções serão as que farão parte do projeto, dessa forma, para ver a utilização do programa para a realização dessa tarefa, olhar no Apêndice):

1. Dado que ele tenha um dois arquivos com os dados do experimento, sendo um deles representando o tempo de um disparo neuronal e outro representando o cluster responsável pelo disparo.

2. Pode-se ler os dados dos arquivos e, baseado nos dados lidos, pode-se calcular a taxa de dísparo instantânea da população inteira ou de cada cluster. Essa taxa de disparo representa o quão rápido ocorreram disparos neurais em cada unidade de tempo.

3. Baseado na taxa de disparo calculada, pode-se calcular o coeficiente de variação. Com base nesse dado, dependendo de como ele se comporta e de seu valor, podemos ter conhecimento do estado cortical da criatura na qual foi realizado o experimento.

4. Baseado no coeficiente de variação, ou na taxa de disparo calculada, pode-se criar gráficos para comparar resultados de diferentes períodos do experimento, ou diferentes experimentos.

1.1 Arquitetura cliente-servidor

A arquitetura cliente-servidor é um modelo de design de sistemas de computação que divide a aplicação em duas partes distintas: o cliente e o servidor. Sendo o cliente o lado da aplicação executado no computador do usuário final e é responsável por enviar

solicitações para o servidor. Já o servidor é o lado da aplicação que executa as solicitações enviadas pelo cliente. Ela é amplamente utilizada na Internet e oferece uma série de benefícios, incluindo segurança, escalabilidade, flexibilidade e confiabilidade.

Outra das vantagens da arquitetura cliente-servidor é que ela pode ser usada para reduzir a carga nos computadores do lado cliente. Como o servidor é responsável por realizar a maior parte do processamento, o cliente pode ser projetado de forma mais simples e leve, sem a necessidade de executar tarefas complexas ou exigir recursos de hardware robustos.

Essa última vantagem traz um valor agregado grande para a nossa aplicação, tendo em vista o problema enfrentado comentado anteriormente. Dessa forma, uma aplicação que segue o modelo cliente-servidor centralizaria o processamento pesado da aplicação do lado servidor e permitiria que tanto o custo de processamento quanto o espaço de armazenamento fossem delegados para um computador terceiro. Assim, permitindo com que os pesquisadores trabalhem em seus computadores e laptops independente de onde estejam e da capacidade de suas máquinas. Outro benefício trazido é o de permitir a continuação do fluxo de trabalho de diferentes máquinas.

2 METODOLOGIA

Como dito na seção anterior, um framework é um conjunto de ferramentas, bibliotecas, padrões de codificação e convenções que são usados para desenvolver e criar aplicações de software. Utilizar um framework pode trazer diversos benefícios no processo de desenvolvimento, tais como melhora na robustez e consistência do programa, além de acelerar o processo de criação. No entanto, a escolha adequada do framework pode ser determinante para o sucesso ou falha do projeto. Visto isso, percebe-se a necessidade de uma análise comparativa objetiva para decidir o framework principal utilizado.

Para tal, foi utilizado o processo hierárquico analítico(AHP), desenvolvido por Thomas Saaty na década de 1970. Esse processo consiste em uma técnica matemática para tomada de decisões, que envolve a construção de uma estrutura hierárquica de critérios e subcritérios, bem como a atribuição de pesos para cada um deles, de acordo com sua importância relativa para a tomada de decisão. Apesar de ser um método antigo, ainda é muito relevante nos dias de hoje, sendo um dos métodos de tomada de decisão com maior quantidade de artigos publicados (WALLENIUS et al., 2008).

2.1 Seleção de framework/Biblioteca

Para selecionar o framework, foram definidas quatro métricas comparativas para aplicar o AHP. Essas métricas foram:

No processo de seleção foram considerados 4 diferentes bibliotecas/frameworks, sendo eles: ReactFlow, Blockly, Orange e Ryven.

As métricas utilizadas e as razões para que elas fossem usadas foram:

- Grau de aptidão da biblioteca para o modelo cliente-servidor: Muitos pesquisadores conduzem suas pesquisas em ambientes acadêmicos e/ou residenciais. Entretanto, embora os ambientes acadêmicos proporcionem uma maior capacidade de processamento, os pesquisadores frequentemente se deparam com limitações de recursos em seus domicílios para executar seus programas. Dessa forma, é desejável a terceirização do processamento de dados para um computador externo, a fim de contornar tais limitações. Além disso, a capacidade de continuar a realizar a pesquisa de qualquer ambiente também é desejável.

- Interoperabilidade: A capacidade de incorporar com facilidade novas funcionalidades tanto do código produzido durante o projeto quanto do código de outras bibliotecas e frameworks, sem a necessidade de um código de integração complexo ou soluções

improvisadas é desejável para melhorar o tempo de desenvolvimento do projeto(devido a menos código de integração necessário) e para diminuir a quantidade de código de integração, o que por sua vez diminui a chance de defeitos ocorrerem no código.

- Boa documentação(abundância e clareza): Bibliotecas e frameworks com documentação escassa ou críptica tornam o processo de desenvolvimento mais laborioso e aumentam a chance de erros.

- Facilidade de extensão de funcionalidade: como o âmbito ao qual se refere essa pesquisa é não normalmente explorada por ferramentas de programação visual, é importante que seja possível e fácil estender a funcionalidade para abarcar a área da neurociência sem que seja necessário muito código para realizar a alteração de função.

Tendo como base essas métricas para nos guiar, podemos aplicar o AHP para decidirmos qual dentre os frameworks escolhidos melhor trará um maior benefício para o projeto. Para isso, o primeiro passo é definir uma matriz de comparação par a par entre as métricas, avaliando-as entre 0-9, onde >1 significa mais importante, 1 significa igual importância e <1 significa menor importância (vale notar que dado uma comparação de valor x entre as métricas A e B implica numa comparação de 1/x para a comparação entre B e A).

Tabela de prioridades				
	Adequação para o modelo cliente-servidor	Facilidade de extensão do software	Facilidade de interoperabilidade	Qualidade e quantidade da documentação
Adequação para o modelo cliente-servidor	1	5.00	7.00	9.00
Facilidade de extensão do software	0.20	1	5.00	3.00
Facilidade de interoperabilidade	0.14	0.20	1	2.00
Qualidade e quantidade da documentação	0.11	0.33	0.50	1

Tabela 1 – Comparação entre as diretivas comparativas.

Dada a tabela 2.1, foi possível derivarmos o vetor de comparação, que vai ser utilizado, representado pela tabela 2.2

	Adequação para o modelo cliente-servidor	facilidade de extensão do software	facilidade de interoperabilidade	qualidade e quantidade da documentação
Vetor de comparação	0.64	0.22	0.08	0.06

Tabela 2 – Vetor de comparação para o processo AHP.

Agora, a última coisa necessária para tomar nossa decisão é dar uma nota para cada uma das métricas selecionadas nos passos anteriores de 1 a 9, com 1 sendo pouco apto e 9 sendo muito apto. As notas atribuidas podem ser vistas na tabela 2.3

	Adequação para o modelo cliente-servidor	facilidade de extensão do software	facilidade de interoperabilidade	qualidade e quantidade da documentação
ReactFlow	9	7	7	6
Blockly	9	4	2	6
Orange	3	3	2	3
Ryven	5	8	5	3

Tabela 3 – Nota das alternativas.

Agora, para obtermos a nota final de cada alternativa, nós normalizamos as colunas, para que sua soma seja 100% e realizamos uma multiplicação de matrizes entre o vetor de comparação transposto com a matriz de notas. Isso nos dá o seguinte resultado:

	ReactFlow	Blockly	Orange	Ryven
Notas	34.71%	29.14%	12.37%	23.78%

Tabela 4 – Nota final.

Com esse valor final, ficou decidido que a biblioteca utilizada seria a ReactFlow, que é uma biblioteca para representação de grafos para React. Porém, também é necessário de uma técnologia para a implementação do servidor. A escolha óbvia para isso é utilizar o python, devido às suas capacidades de processamento de dados. Portanto, ficou decidido que a implementação do cliente usaria React com ReactFlow e o servidor seria em python.

3 TRABALHOS RELACIONADOS

A área de programação visual no mundo acadêmico ainda é bem restrita e a maior parte dos seus trabalhos são na área de Internet das Coisas(IoT) e educação, porém o horizonte de utilizações de aplicações desse tipo vem aumentando (KUHAIL et al., 2021).

Ao mesmo tempo, no mundo corporativo, é bem comum encontrar ferramentas desse tipo para criar e modificar vizualizações de gráficos 2d e 3d. Entre essas, podemos citar o blender(utilizado para modelagem 3d) e o Godot(engine para criação de jogos), que oferecem a criação de materiais com programação visual, o Grasshopper3d(programa para ArchVis), que permite a criação de cenas 3d com programação visual e a Unreal Engine(engine para criação de jogos), que permite a criação de jogos 3d inteiros apenas com programação visual.

Voltando ao mundo acadêmico, é interessante citar os seguintes trabalhos relacionados:

- O primeiro é o Feng et al.(2017), cujo objetivo da aplicação desenvolvida no mesmo é o ensino de paralelismo, bem como facilitar o desenvolvimento de aplicações que portam dessa capacidade, se utilizando de programação visual para atingir este objetivo. Este projeto usa uma linguagem de programação visual já existente, chamada Snap, que é extremamente semelhante ao Blockly, descrito na seção 2.1, do capítulo de metodologia. Neste trabalho, os autores se propõe a adicionar a capacidade de multiprocessamento ao Snap. Quando comparamos o projeto do artigo ao nosso, podemos destacar a semelhança no fato de que os dois se utilizam de um ambiente web para poder realizar o desenvolvimento. Porém, a funcionalidade para desenvolvimento é bem limitada(por design, tendo em vista que o programa foi feito para ensino), porém a funcionalidade pode ser extendida se o usuário souber programar. Enquanto no nosso programa, existem mais funções relacionadas ao problema alvo, porém não adicionamos a capacidade nessa iteração do aplicativo, devivo ao seu foco em pessoas que não são da área de computação.

- O segundo é o trabalho de Valtolina et al.(2019). Este artigo cria uma aplicação para que funcionários de uma cidade possam controlar dispositivos de internet das coisas(IoT) através de uma interface visual. Assim como Feng et al.(2017), este artigo também apresenta uma interface web, semelhantemente ao nosso programa. Porém, este trabalho se assemelha mais ao projeto do nosso programa que o artigo de Feng et al.(2017), devido à forma como as informações do programa criado pelo usuário são apresentados, pois no artigo de Feng et al.(2017) as informações se dão por meio de uma distribuição de blocos, enquanto o de Valtolina et al.(2019) as informações são exibidas por meio de uma distribuição de diagrama(grafos).

- O terceiro artigo é o de Mei et al.(2018). Nesse, os autores criaram uma ferramenta para visualização de dados e criação de gráficos com interface web. De forma similar ao nosso programa e o de Valtolina et al.(2019), este artigo também apresenta os programas através de uma distribuição de diagrama. Porém, diferentemente do nosso, o programa desse artigo necessita de conhecimento prévio de programação para a realização de algumas funções dentro do programa.

- Por fim, temos o artigo de Johnsson e Magnusson(2020), cujo trabalho mais se distancia do nosso em forma, porém é interessante para mostrar as diferentes formas que um programa de programação visual podem tormar. Sendo um programa mobile com foco em internet das coisas(IoT), cujo desenvolvimento de aplicações se dá por forma de preenchimento de formulários.

4 ARQUITETURA

Agora falaremos um pouco sobre a arquitetura do projeto e como ele funciona. Primeiro descreveremos como o programa deverá funcionar, e a partir disso mostraremos como isso evolui para a arquitetura do programa e as decisões tomadas com relação ao desenvolvimento.

4.1 Descrição do problema e considerações iniciais

Dada a descrição do problema exibida na introdução, o funcionamento do programa se dará a partir da criação de "notebooks", que representarão um fluxo de trabalho que se dará por meio de um conjunto de nós, que são unidades de processamento, que poderão ser interligados através de arestas. Esses nós serão divididos entre "nós de entrada", "nós de processamento" e "nós de saída". A diferença entre eles é que os os de entrada não recebem dados de outros nós, os de saída não enviam dados para outros nós e os de processamento tanto recebem quanto enviam dados. Cada Nó pode possuir nós filhos(exceto os de saída). O usuário poderá executar o notebook, o que começará o processamento pelos nós de entrada e, conforme os nós forem sendo executados, os nós subsequentes que tiverem todos os pais executados, também serão. Assim sucessivamente, até que todos os nós tenham sido executados. O usuário também deve poder resetar a execução, que para a execução atual e reseta o estado dos nós. Os notebooks criados dever poder ser salvos e acessados depois. Como os fluxos de trabalho necessitam de arquivos com dados, o usuário também deve poder enivar arquivos que serão salvos para depois.

Devido à complexidade do problema e à diferente quantidade de funcionalidades que podem ser adicionadas, foi decidido trabalhar em apenas um fluxo de trabalho, completo do início ao fim, de modo a restringir o escopo do problema.

O fluxo de trabalho que foi decidido é o seguinte:

1. Ler dados de um arquivo previamente enviado contendo uma lista de números
2. Calcular a Taxa de Disparo Instantânea ao longo do tempo (que, a partir de agora, será chamado pelo seu nome em inglês, que é a Instant Firing Rate, ou IFR)
3. Calcular o Coeficiente de Variação
4. Criar gráficos
5. Salvar o notebook para uso futuro junto com as informações de nós que exibem informações ou possuem opções

6. Carregar salvos anteriormente

4.2 Arquitetura

Ao analisar o problema, é possível perceber uma clara divisão em duas partes. A primeira, corresponde à lógica de execução dos nós e dos notebooks. A segunda, a de criar contas, salvar arquivos, exibir os notebooks, entre outras responsabilidades gerais, não relacionadas à lógica de trabalho. Dessa forma, decidimos separar o problema em dois subsistemas que funcionarão de maneira independente um do outro. O primeiro corresponderia à lógica principal do programa, que apenas receberá requisições para executar/resetar notebooks. chamaremos essa parte de ProcessingSystem O segundo se preocupará com a parte de utilidade para o usuário, criação de contas, upload de arquivos, a renderização dos grafos e exibição de gráficos, e outras funcionalidades para a comodidade do usuário.

Devido à separação dessa maneira, é possível que outras pessoas interessadas criem outras aplicações que usem o sistema de execução de notebooks para diferentes sistemas.

4.2.1 Sistema para execução de notebooks

Dada a natureza do problema, foi decidido que a melhor forma de representar o estado do programa seria na forma de grafos direcionais, onde os nós realizariam os processamentos e poderiam - ou não - possuir uma entrada, um estado interno e uma saída(mas deverão possuir ao menos dois dos três). Ao executar o notebook, é possível realizar a execução dos nós através de uma versão modificada do algoritmo Breadth First Search (BFS), que será descrita no pseudo-código a seguir:

```
let nodes be a list with all nodes;
let border be a queue;
let executed be an empty set;
border ← {node|node ∈ nodes; x.pais = ∅};
while |border| > 0 do
    for node in border do
        execute node;
        add node to executed;
        remove node from border;
    end
    candidates ← {node|node ∈ nodes; node ∉ executed; {parent|parent ∈
     node.parent; parent ∉ executed} = ∅};
    append candidates to border;
end
```

Para representar o grafo do notebook, vamos utilizar o JSON(JavaScript Object Notation), devido à sua capacidade de interoperabilidade com diversos sistemas.

A modelagem do grafo ficou definida da seguinte forma:

- O sevidor espera um objeto JSON com dois atributos; um chamado Nodes e um chamado Edges
- O atributo chamado Nodes possuirá uma lista de nós.
- Cada nó será um objeto JSON com os seguintes atributos:
  - id: um objeto textual que representa a identificação do nó
  - data: um objeto JSON que representa o estado interno do nó(cada tipo de nó pode necessitar de subatributos em "data" conforme sua função)
  - type: um objeto textual que representa o tipo do nó
- O atributo chamado Edges possuirá uma lista de arestas
- Cada aresta contem os seguintes atributos
  - source: atributo textual que representa o id do nó de origem
  - target: atributo textual que representa o id do nó de destino
  - sourceHandle: atributo textual que representa qual dos valores retornados pelo nó deve ser utilizado pro nó seguinte
  - targetHandle: atributo numérico que representa qual dos valores de entrada do nó destino será utilizado

4.3 Modelagem do sistema completo

Falaremos um pouco agora sobre a visão geral do sistema e como as partes se interconectam, além de mostrar diferentes diagramas para facilitar o entendimento do funcionamento do sistema.

O diagrama de caso de uso da figura 3.1 representa o problema descrito no inicio desse capítulo, porém de forma mais completa e já funcional. O ator chamado ProcessingSystem é o sistema de processamento de nós. Como o funcionamento dele já foi descrito superficialmente na seção 3.2.1, escolhemos apenas representá-lo aqui como um agente externo. Perceba que as duas únicas chamadas para as funcionalidades do ProcessingSystem são por meio do Run e Reset.

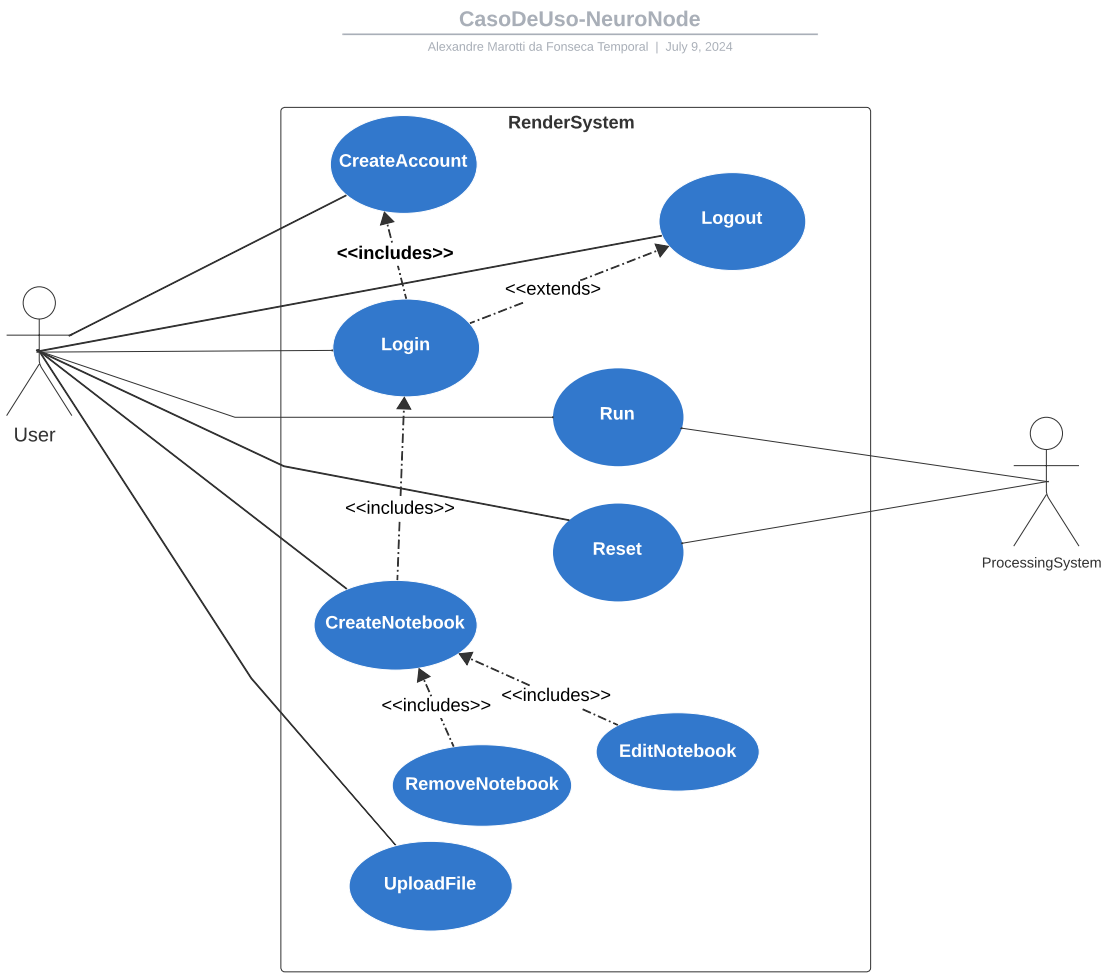

Figura 1 – Diagrama de caso de uso do sistema como um todo.

Dado o entendimento do diagrama de caso de uso, criamos um Package Diagram, que pode ser visto na figura 3.2. Devido ao grande número de pacotes externos usados pelo ProcessingSystem, escolhemos omití-los para manter a clareza do diagrama.

Explicando o que cada parte do sistema vai ser responsável por realizar, temos:

- RenderSystem:
  - notebookBuilder: se reponsabiliza por exibir o notebook para o usuário, permite a edição do mesmo e é responsável por construir o objeto JSON que será salvo ou eventualmente enviado para o ProcessingSystem para eventual execução.
  - API: responsável por garantir as funcionalidades básicas de funcionamento para o sistema, como criação de contas, salvamento de notebooks, edição de dados, etc... também é responsável por expor a funcionalidade do ProcessingSystem para o notebookBuilder
  - FileUploadHandler: Devido aos grandes tamanhos dos arquivos enviados, o FileUploadHandler se responsabiliza por quebrar o arquivo em pequenas partes, enviá-las para o servidor e remontá-las para o armazenamento dos arquivos.
  - DatabaseAPI: responsável por realizar as tarefas de interação com o banco de dados.
- ProcessingSystem:
  - ProgramManager: Responsável por receber a requisição de execução, criar uma nova thread para o notebook ser executado e instanciar o notebookManager para execução nessa nova thread.
  - notebookManager: Responsável por re-montar o notebook através do JSON, realizar sua execução, controlar o estado interno do notebook e seu progresso
  - Nodes: responsável por implementar os diferentes tipos de nós e suas funcionalidades
  - NeuroUtils: Responsável por oferecer funcionalidades úteis para os nós e a implementação de funções para calculos de resultados e gráficos

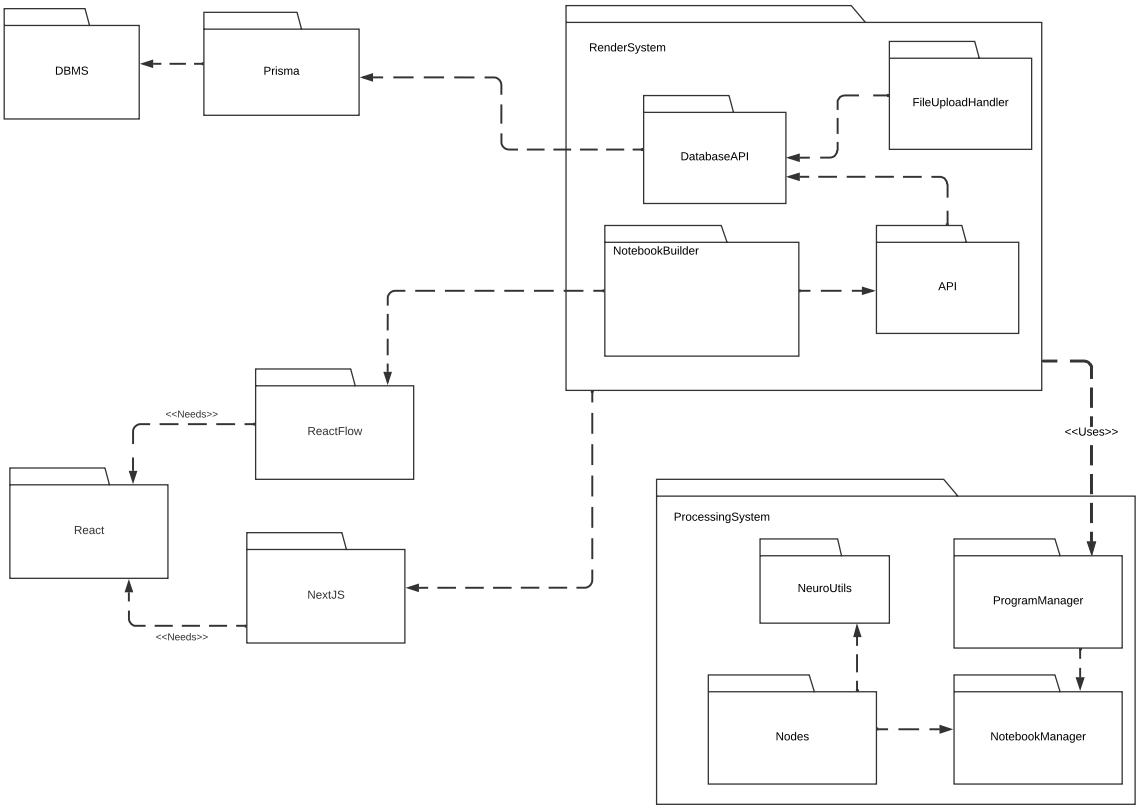

Figura 2 – Diagrama de pacotes do sistema.

5 FLUXO DE USO CORE DO APLICATIVO

Neste capítulo, exploraremos mais concretamente o fluxo de trabalho do programa, exibindo exemplos de tela e exibindo as funcionalidades implementadas, além de descrever os processos de cada uma das ações do usuário.

5.1 Funções gerais

Para o funcionamento do sistema, os usuários possuem algumas funções gerais que não vão ser descritas neste trabalho, devido à ubiquidade dessas funções e ao quão fácil é de encontrar implementações de funcionalidades semelhantes na internet, mas que são necessárias para o funcionamento correto do programa.

Essas funções são as seguintes:

- Criação de conta
- Login
- Logout
- Um CRUD simples para notebooks, que inclui a capacidade de criar um notebook, editar o nome de um notebook e excluir um notebook. E salvar as edições feitas. A funcionalidade de editar o conteúdo do notebook é essencial para o funcionamento do programa e não trivial, portanto será descrita em outra seção.

Capturas de telas que exibem essas funções serão disponibilizadas no Apêndice.

5.2 Funcionalidade core do sistema

Dado o fluxo de trabalho descrito na seção anterior, iremos aqui descrever mais concretamente o fluxo de trabalho do programa, exibindo exemplos de tela e como as informações fluem pelo sistema.

5.2.1 Upload e exclusão de arquivos

Após estar devidamente logado no sistema, uma nova seção é liberada para o usuário no menu, chamada de "User Area". Nela existem duas áreas disponíveis. A primeira diz

respeito à funcionalidade de notebooks, e permite ao usuário poder utilizar as funções de criação, exclusão e renomeação de notebooks, comentadas na seção anterior. A segunda área diz respeito à funcionalidade de upload e exclusão de arquivos. Devido ao grande tamanho dos arquivos que contem dados neuronais, a funcionalidade padrão de envio de arquivos fornecida pelo HTML e JavaScript se utilizando de um <input type="file"> para o envio de arquivos não é adequada.

Dessa forma, para o envio de arquivos, após selecionar o arquivo - arrastando-o para a área de arquivos(demonstrada na Figura 3) - o mesmo será dividido em partes de 1MB[1]. Após a divisão, o tamanho do arquivo é enviado para o servidor e então são calculados os hashs de cada uma das partes para verificar quais já foram enviadas. E em seguida, o arquivo é enviado, parte por parte, para o servidor, junto com sua posição geral no arquivo, tendo em vista que as partes podem ser enviadas em ordens diferentes.

Ao receber as partes, o servidor verifica a posição da parte recebida, separa o espaço no arquivo e escreve os dados no disco. Após receber todas as partes, a transferência é concluída e o diretório e nome do arquivo serão gravados no banco de dados.

Após a gravação do arquivo, o mesmo passará a ser exibido na lista de arquivos disponíveis e fica disponível uma opção para que o usuário possa excluir o arquivo.

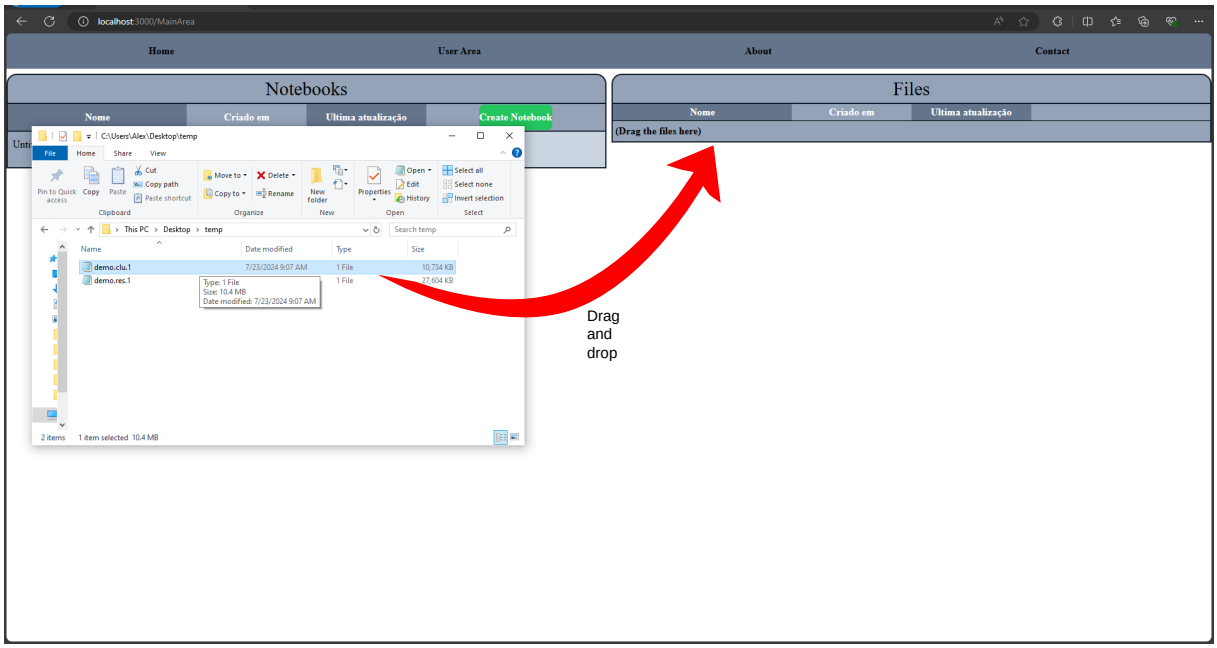

Figura 3 – Exemplo de upload de arquivos.

5.2.2 Edição de notebooks

[1] Com exceção da ultima parte, que terá tamanho de (tamanho do arquivo) mod$(1MB)$

Ainda na seção "User Area", o usuário pode selecionar um dos notebooks criados anteriormente para edição. Então, ele será redirecionado para a tela de edição de notebooks.

Caso o notebook seja recem criado, o usuário será apresentado com uma área vazia para edição e um menu para execução do notebook. Nesta área, é possível adicionar os nós que foram descritos no capítulo de Arquitetura.

Ao clicar em qualquer lugar dessa área com o botão direito, o usuário será apresentado um menu com os diferentes tipos de nós disponíveis, como demonstrado na figura 4. Cada um dos nós possui uma diferente função, que são descritas a seguir na seção de "Tipos de nós".

Após adicionar os nós, o usuário pode conectar nós através de pontos de entrada e saída chamados de handles, que permitem a passagem de informações entre os nós.

5.2.2.1 Tipos de nós

Os tipos de Nodes diferentes são descritos a seguir e capturas de tela que ilustram cada um dos nós são fornecidas no apêndice. Para cada tipo de nó, serão descritos tanto entradas, que são os pontos de input de cada um dos nós, que os permitem receber informações de nós anteriores, quanto saidas, que são os pontos de output, que permitem o envio de dados para outros nós. Por fim, o estado interno, que permite o armazenamento de dados gerados pelo usuário e seleções de opções para os nós. Caso os nós possuam estado interno, o mesmo será armazenado automáticamente entre cada uso do programa por parte do usuário, de modo a facilitar a retomada do uso entre diferentes sessões.

- File Input Node:
  - Entradas: Nenhuma
  - Estado interno: Armazena o path para o arquivo selecionado pelo usuário(selecionado através de uma lista de arquivos, baseado nos arquivos enviados anteriormente)
  - Saídas: O nó dá como saída o arquivo selecionado no formato de texto.
- IFR Node:
  - Entradas:
    - Timings: Uma lista com o tempo em que ocorreram spikes
    - Clusters: Uma lista com os clusters que realizaram os spikes
  - Estado interno: Esse nó armazena o tamanho de janela usado para calcular a taxa de disparo(IFR)
  - Saídas: O nó dá como saída a taxa de disparo(IFR) calculada.

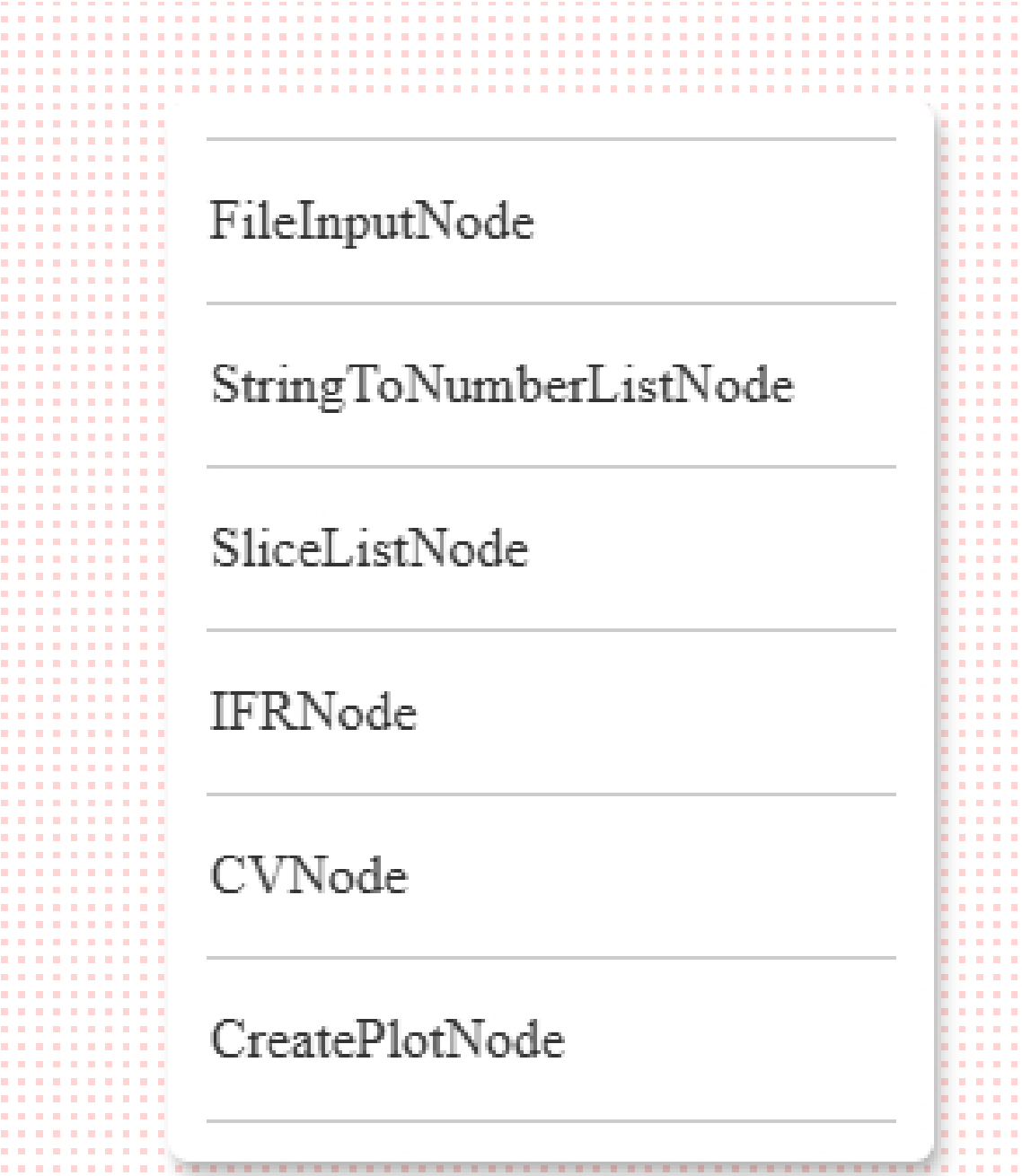

Figura 4 – Exemplo de Adição de nós.

- CV Node:
  - Entradas:
    - IFR: Uma lista com o tempo em que ocorreram spikes
  - Estado interno: Esse nó armazena o tamanho de janela usado para calcular o

coeficiente de variação

  - Saídas: O nó dá como saída o coeficiente de variação calculado.

- Plot Node:

  - Entradas:

    * Entrada 1: A "entrada 1" possui diferentes funções dependendo do estado interno. Caso o usuário tenha selecionado "1 entrada" na opção de quantidade de entradas, o nó cria um gráfico de linha onde a "entrada 1" é responsável tanto pelo eixo X, quanto pelo eixo Y. Caso o usuário tenha selecionado "2 entradas", o nó cria um gráfico de linha onde a "entrada 1" é responsável somente pelo eixo X.
    * Entrada 2(opicional): A "entrada 2" é responsável pelo eixo Y, caso a opção de entrada seja selecionada como "2 entradas". Caso contrário, a segunda entrada fica oculta.

  - Estado interno: Esse nó armazena a quantidade de entradas esperadas além de armazenar o gráfico criado durante a execução do programa.

  - Saídas: Nenhuma

- Nó avançado: String to number list Node:

  - Entradas: Texto com números

  - Estado interno: Nenhum

  - Saídas: Lista de números

  - Informações adicionais: Esse nó converte um texto com números em uma lista de números. Esse nó não é estritamente necessário para nada, no momento, tendo em vista que os nós que recebem uma lista de número, no momento, conferem se estão recebendo texto como entrada e tentam converter em lista de números. Porém, o uso desse nó torna mais fácil a identificação e exibição de erros.

- Nó avançado: Slice List Node:

  - Entradas: Lista

  - Estado interno: Esse nó armazena as posições de ínicio e/ou fim

  - Saídas: Sublista delimitada pelas posições de ínicio e fim

  - Informações adicionais: Esse nó permite obter uma sublista. As posições de início e fim podem ser posições, de fato, ou um número que representa segundos, caso a entrada seja um IFR ou um CV. Nesse último caso, todas as posições que caírem dentro do tempo delimitado serão retornadas.

5.2.3 Execução

Na seção de edição de notebook, o usuário pode selecionar a opção de executar o notebook. Ao selecionar a opção, o notebook é salvo e o notebook é marcado como "em execução". Ao fazer isso, as informações são transferidas do servidor RenderSystem para o ProcessingSystem para que o mesmo possa ser executado. Ao receber o notebook, o ProcessingSystem cria uma nova thread para o notebook, instancia o notebookManager, marca todos os Nodes como "não executados" e inicia a execução do notebook utilizando o algoritmo descrito na seção 4.2.1.

Durante a execução do notebook, o cliente continuamente requisita o estado de execução do notebook, a cada 1s. Enquanto estiver na tela de edição de notebook, mesmo antes de executar, o programa realiza um pooling da informação sobre execução do notebook e de sua execução - no momento isso não gera nenhuma alteração caso o notebook não esteja executando. Mas, permite que, no futuro, ao ser implementada a funcionalidade de mais de uma pessoa editar o notebook, o programa possa exibir corretamente as informações de execução, caso outra pessoa execute o notebook. Conforme cada no é executado, o sistema marca o nó com a flag de "executado".

Após todos os nós terem sido marcados como "executados", o sistema marca a execução do notebook como finalizada.

5.2.4 Reset

Durante ou após a execução do notebook, o usuário pode selecionar a opção de reset no menu. Ao selecionar essa opção, o cliente envia uma solicitação de parada para o notebookManager que, por sua vez, repassa o pedido para o ProcessingSystem. Ao receber o pedido, o ProcessingSystem busca a execução correspondente, remove da lista de execuções possíveis e envia um sinal de parada para a thread que está executando o notebook. Ao receber o sinal, a thread limpa os recursos utilizados e envia a confirmação de parada para o RenderSystem, que marca todos os nós como "não executados", e salva o estado do notebook.

6 AVALIAÇÃO

Para avaliar o sistema, usaremos o System Usability Scale(SUS), que é uma escala simples e eficaz composta por um questionário de 10 perguntas que avaliam a percepção geral dos usuários sobre a facilidade de uso e a satisfação com o sistema, desenvolvido por Brooke(1996), como uma forma de obter uma visão geral do sistema e também de identificar pontos fortes e áreas que precisam de melhorias.

Cada uma das 10 perguntas é respondida em uma escala Likert de cinco pontos, variando de "discordo totalmente" a "concordo totalmente". Após a coleta das respostas, a pontuação é calculada (existem diversas formas de calcular o SUS, porém nos utilizaremos da forma descrita a seguir), podendo então ser usada para auxiliar na tomada de decisões para aprimorar o sistema em questão.

$SUS = 2.5\,(\sum(q_{n+} - 1) + \sum(5 - q_{n-}))$ Onde "q_{n+}" é a questão positiva de número n(ou seja, aquela onde 5 é um bom resultado e 1 é um mal resultado) e "q_{n-}" é a questão negativa de número n(ou seja, aquela onde 1 é um bom resultado e 5 é um mal resultado). Essa pequena mudança, comparada à fórmula apresentada por Brooke (1996)[2], permite diferentes números de perguntas positivas ou negativas em diferentes ordens, enquanto ainda mantém o limite entre 0 a 100.

Para realizar a avaliação, realizamos o questionário SUS com 3 pessoas, sendo uma delas um pesquisador(que recebeu um conjunto de perguntas diferentes) e 2 pessoas sem conhecimento prévio tanto em programação quanto em neurociência, para simular um aluno que está começando a aprender agora. E, para as pessoas sem conhecimento prévio, realizamos um briefing rápido sobre o sistema, explicamos por cima o conceito do sistema, explicamos o que era taxa de disparo instantânea(sem explicar como isso seria feito no sistema) e pedimos para que eles tentassem realizar o cálculo do IFR e criassem um plot com base no IFR. O tempo de realização da tarefa foi cronometrada, e após a realização da tarefa, o questionário SUS foi aplicado.

A tabela 5 mostra o resultado da avaliação com as pessoas sem conhecimento prévio e o tempo médio de uso.

Devido ao difícil contato com pesquisadores da área de neurociência em um curto espaço de tempo, a pesquisa foi feita com apenas um pesquisador. As perguntas feitas e os resultados obtidos foram os seguintes:

Como pode ser verificado na tabela 6, o resutado SUS foi de 62.5, o que não é um valor ruim, porém também não é excepcional. Mas perceba que a maioria das respostas que receberam uma nota mais baixa, foram devido à pouca funcionalidade oferecida no

[2] A fórmula descrita por Brooke (1996) se dá por $SUS = 2.5\,(\sum(q_o - 1) + \sum(5 - q_e))$, onde "$q_o$" são as questões de número impar e "q_e" são as questões de número par.

Perguntas e pontuação		
Pergunta	usuário 1	usuário 2
Eu sinto que o programa seria mais fácil de aprender que aprender a programar	5	5
Eu achei o sistema desnecessariamente complexo	1	1
Eu achei o sistema fácil de usar	4	4
Eu acho que precisaria de ajuda de uma pessoa com conhecimentos técnicos para usar o sistema	4	4
Eu acho que as funções do sistema estão bem integradas	3	4
Eu acho que o sistema apresenta muita inconsistência	3	2
Eu imagino que as pessoas aprenderão como usar esse sistema rapidamente	4	5
Eu achei o sistema atrapalhado de usar	1	1
Eu me senti confiante ao usar o sistema	4	4
Eu precisei aprender várias coisas novas antes de conseguir usar o sistema	3	1
Resultado SUS:	70	82.5
Tempo para execução da tarefa:	6m7s52ms	9m57s04ms

Tabela 5 – Resultado SUS pesquisador.

momento, e não necessariamente devido a uma baixa qualidade de uso.

Perguntas e pontuação	
Pergunta	Pontuação
Eu usaria esse sistema no ensino dos meus alunos	5
Eu usaria esse sistema no meu processo de pesquisa	3
Se houvessem novas funcionalidades adicionadas, eu usaria esse sistema no meu processo de pesquisa	4
Eu achei o sistema desnecessariamente complexo	1
Eu recomendaria esse sistema para outros pesquisadores	2
Eu achei o sistema fácil de entender	3
Se houvessem novas funcionalidades adicionadas, eu recomendaria esse sistema para outros pesquisadores	4
Eu acho que o sistema apresenta muita inconsistência	3
Eu imagino que as pessoas aprenderão como usar esse sistema rapidamente	4
Eu achei o sistema atrapalhado de usar	1
Resultado SUS:	62.5

Tabela 6 – Resultado SUS pesquisador.

7 CONCLUSÕES E TRABALHOS FUTUROS

Este programa desenvolvido no presente trabalho é apenas uma prova de conceito, projetada para demonstrar a capacidade de criar uma ferramenta para programação visual para neurologia utilizando-se do modelo cliente-servidor que servirá como base para implementar as funcionalidades em cima da base desenvolvida aqui. Dessa forma, ainda existem diversas funcionalidades que podem ser úteis, mas que não foram implementadas. Porém, acreditamos que o trabalho que foi desenvolvido aqui atinge o objetivo esperado de projetar essa base de forma satisfatória e robusta.

Entre elas, uma das mais importantes para uso futuro é a capacidade de compartilhar notebooks entre pessoas e edição simultânea dos notebooks por parte dos usuários. Isso facilitará o compartilhamento de informações entre pesquisadores durante suas pesquisas, além de permitir que professores possam avaliar o desempenho dos seus alunos.

Outra necessidade importante é a de implementar novos fluxos de trabalho para ampliar o horizonte de uso do sistema, para que ele tenha uma capacidade de ser utilizado em uma variedade maior de tarefas.

Também vale a pena destacar a importância da capacidade de gerar gráficos mais complexos, tendo em vista que o único gráfico que pode ser gerado atualmente é um line chart simples. Então, é interessante que outros tipos de gráfico possam ser gerados, além de permitir a alteração de opções do gráfico, como cores, tamanho do gráfico, legenda, etc. Além da possibilidade de combinar diferentes gráficos em apenas um plot.

Outra capacidade que não foi implantada, mas que seria muito útil é a de realizar análises estatísticas em cima dos dados recebidos de outros nós, como média, desvio padrão, teste de hipótese etc... sendo uma capacidade relativamente fácil de implementar, porém devido ao curto tempo de execução, não foi possível a realização.

Tamém seria útil introduzir a capacidade de simular dados neuronais para testes. Assim, haverá a possibilidade de realizar simulações com dados novos, para testes rápidos. Mesmo que esses dados não possam ser utilizados para pesquisa, devido ao fato de ter sido gerado automáticamente, eles ainda podem ser utilizados para testes, pois tanto a taxa de disparo(IFR) quanto o CV podem ser moldados de acordo com o desejo do pesquisador. E para facilitar o ensino em ambientes de estudo, permitindo que os alunos sejam capazes de aprender as funcionalidades do sistema com conjuntos variados de dados, mesmo que esses não estejam disponíveis.

Por fim, vale notar que é interessante introduzir um sistema de localização para o aplicativo, tendo em vista que ele está todo em inglês e diversos alunos falam apenas o português, ou falam outras linguas estrangeiras, que não são inglês.

7.1 Bugs conhecidos

Durante o uso do sistema, foram encontrados alguns bugs que ainda não foram solucionados. Esses bugs são relativamente pequenos e podem ser contornados pelo usuário durante o uso, porém ainda podem causar incômodos ao usuário durante o uso do sistema. Estes bugs foram:

- Ao salvar o notebook, sair da tela de edição de notebook e voltar para a tela de edição, caso o primeiro nó que o usuário adicionar for um novo nó do mesmo tipo do último nó adicionado antes de salvar o notebook, o nó recem criado vai substituir o antigo, ao invés de adicionar um novo e manter o antigo.
- Ao selecionar um plot com duas entradas, caso o usuário conecte dois nós nas entradas do plot, ao mudar para o plot de uma entrada, a segunda entrada não será excluída. Isso não afeta o funcionamento do programa, sendo apenas um bug visual. O plot gerado ainda será gerado corretamente e exibido normalmente, considerando apenas a primeira entrada.

REFERÊNCIAS

ANDRE, Charles. Evolving story: trepanation and self-trepanation to enhance brain function. Arquivos de Neuro-Psiquiatria, v. 75, p. 307–313, 5 mai. 2017. ISSN 0004-282X. DOI: 10.1590/0004-282x20170040.

BROOKE, John. SUS – a quick and dirty usability scale. In: [s.l.: s.n.], jan. 1996. P. 189–194.

FENG, Annette; GARDNER, Mark; FENG, Wu-chun. Parallel programming with pictures is a Snap! Journal of Parallel and Distributed Computing, v. 105, p. 150–162, jul. 2017. ISSN 07437315. DOI: 10.1016/j.jpdc.2017.01.018.

JOHNSSON, Björn A.; MAGNUSSON, Boris. Towards end-user development of graphical user interfaces for internet of things. Future Generation Computer Systems, v. 107, p. 670–680, jun. 2020. ISSN 0167739X. DOI: 10.1016/j.future.2017.09.068.

KUHAIL, Mohammad Amin et al. Characterizing Visual Programming Approaches for End-User Developers: A Systematic Review. IEEE Access, v. 9, p. 14181–14202, 2021. ISSN 2169-3536. DOI: 10.1109/ACCESS.2021.3051043.

MEI, Honghui et al. VisComposer: A Visual Programmable Composition Environment for Information Visualization. Visual Informatics, v. 2, p. 71–81, 1 mar. 2018. ISSN 2468502X. DOI: 10.1016/j.visinf.2018.04.008.

MILNER, David; GOODALE, Mel. The Visual Brain in Action. [S.l.]: Oxford University Press, out. 1996. ISBN 9780198524724. DOI: 10.1093/acprof:oso/9780198524724.001.0001.

STEVENSON, Ian H; KORDING, Konrad P. How advances in neural recording affect data analysis. Nature Neuroscience, v. 14, p. 139–142, 2 fev. 2011. ISSN 1097-6256. DOI: 10.1038/nn.2731.

VALTOLINA, Stefano et al. Facilitating the Development of IoT Applications in Smart City Platforms. In: [s.l.: s.n.], 2019. P. 83–99. DOI: 10.1007/978-3-030-24781-2_6.

VERKHRATSKY, Alexei; KRISHTAL, O. A.; PETERSEN, Ole H. From Galvani to patch clamp: the development of electrophysiology. Pflügers Archiv - European Journal of Physiology, v. 453, p. 233–247, 3 nov. 2006. ISSN 0031-6768. DOI: 10.1007/s00424-006-0169-z.

WALLENIUS, Jyrki et al. Multiple Criteria Decision Making, Multiattribute Utility Theory: Recent Accomplishments and What Lies Ahead. Management Science, v. 54, p. 1336–1349, 7 jul. 2008. ISSN 0025-1909. DOI: 10.1287/mnsc.1070.0838.

Apêndices

APÊNDICE A – FUNÇÕES GERAIS DO SISTEMA

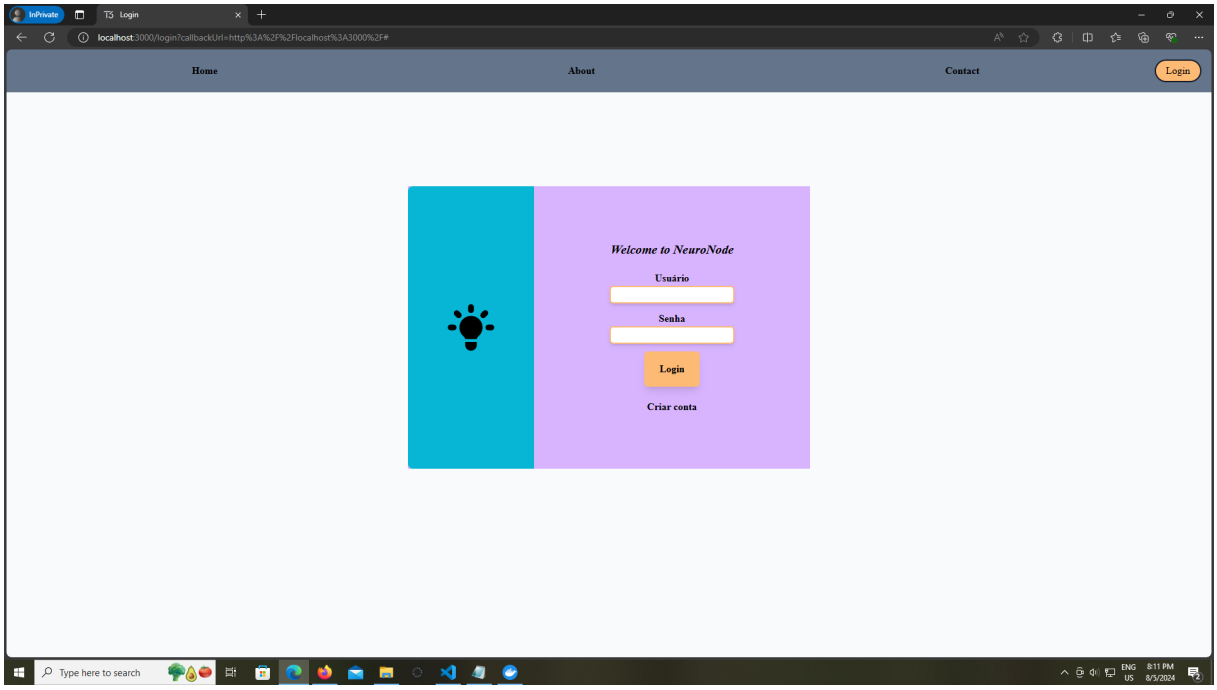

Figura 5 – Exemplo de página de login.

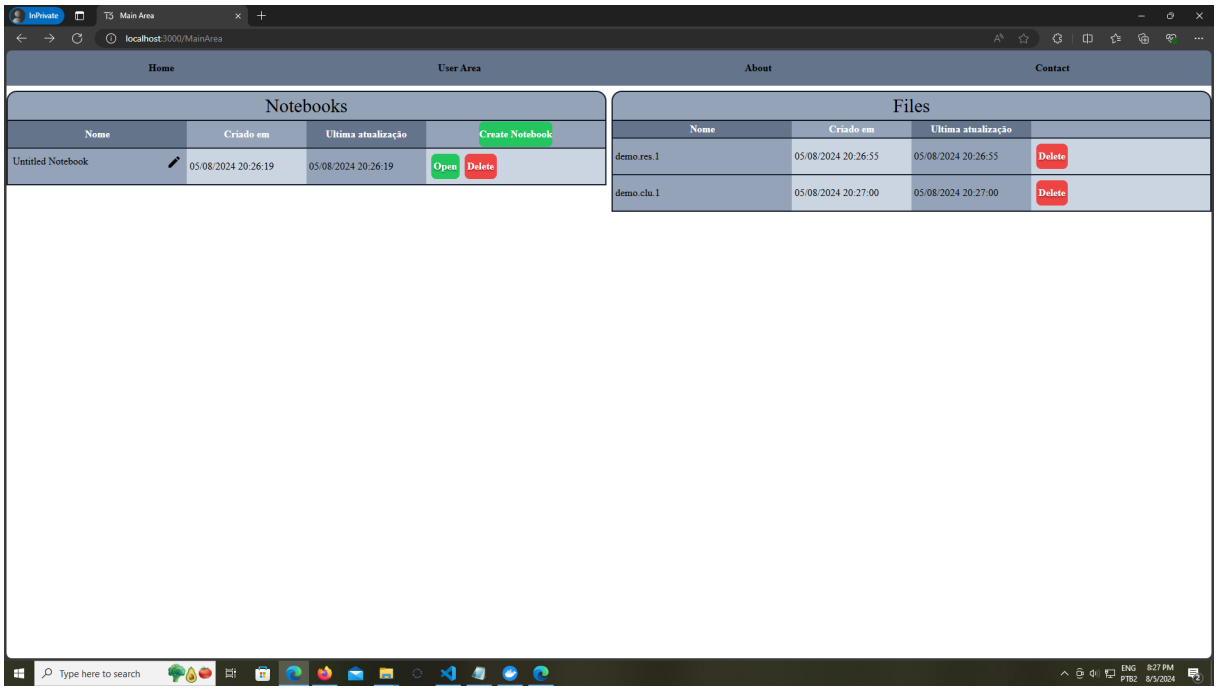

Figura 6 – Exemplo de dos botões de delete.

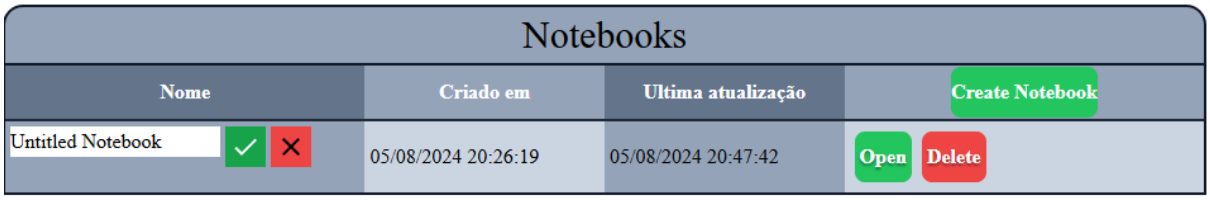

Figura 7 – Exemplo de quando você seleciona para renomear o notebook.

APÊNDICE B – FLUXO DE TRABALHO COMPLETO DO SISTEMA

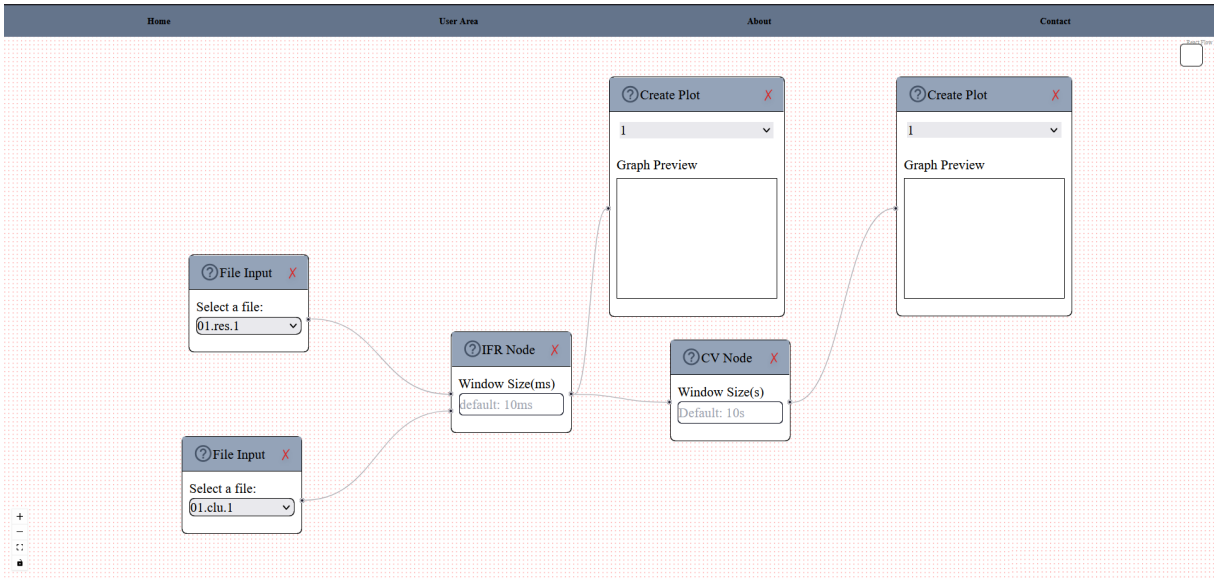

Figura 8 – Exemplo: Fluxo de trabalho antes de executar o sistema

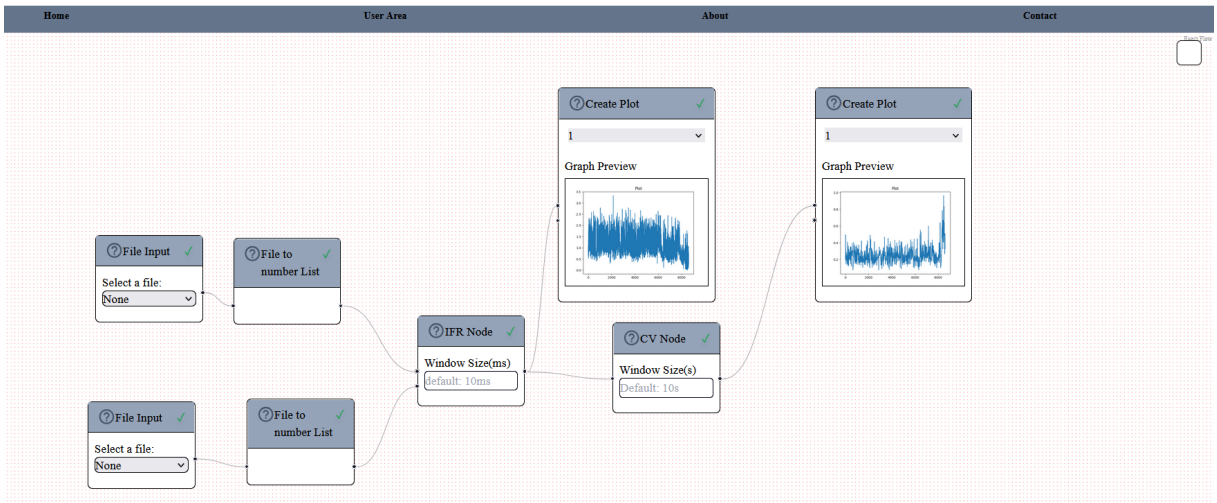

Figura 9 – Exemplo: Fluxo de trabalho após executar o sistema