CENTRO PAULA SOUZA

ETEC TABOÃO DA SERRA

ENSINO MÉDIO E TÉCNICO EM DESENVOLVIMENTO DE SISTEMAS

João Nogueira Acioli Peixoto

Leonardo Ferreira Nascimento da Silva

Lucas Rafael Silveira Tamarindo

Max Keven de Jesus Silva

Rodrigo Ferreira Caldeira

Vinicius Saldanha da Silva

TRABALHO DE CONCLUSÃO DE CURSO: SEU ESTUDO COM LIBERDADE E FOCO – S.E.L.F

TABOÃO DA SERRA – SP

2022

João Nogueira Acioli Peixoto

Leonardo Ferreira Nascimento da Silva

Lucas Rafael Silveira Tamarindo

Max Keven de Jesus Silva

Rodrigo Ferreira Caldeira

Vinicius Saldanha da Silva

SEU ESTUDO COM LIBERDADE E FOCO - S.E.L.F

Trabalho de Conclusão de Curso apresentado ao Curso Técnico em Desenvolvimento de Sistemas da ETEC de Taboão da Serra, orientado pelos professores Fabiano Jorge Oliveira e Rodolfo Votto Filho, como requisito parcial para obtenção do Título de Técnico em Desenvolvimento de Sistemas.

Taboão da Serra – SP

2022

A Ronivon Lopes da Silva, nosso falecido professor que nos abriu portas ao mundo da computação concedendo-nos nossa primeira aula na ETEC de Taboão da Serra.

A Pierluigi Piazzi, cuja inspirou o presente trabalho.

"Falar de vocação é referir-se àqueles que pretendem fazer do trabalho intelectual sua própria vida, tanto os que têm todo seu tempo para dedicar ao estudo, como os que, empenhados em suas ocupações profissionais, reservam para si, como feliz suplemento e recompensa, o desenvolvimento profundo do espírito. Aplico-o entretanto especialmente àqueles que sabem que dispõem apenas de uma parte de sua vida, a menor, para se dedicar aos trabalhos da inteligência. Esses devem, mais que os outros, ser consagrados".

"Jovem, que compreende esta linguagem e que os heróis da inteligência parecem invocar misteriosamente, mas que teme estar despreparado, escute-me. Você tem duas horas por dia? Pode comprometer-se a preservá-las cuidadosamente, a empregá-las ardentemente, e depois, sendo também destinado ao Reino de Deus, poderá beber o cálice de sabor delicado e amargo que estas páginas pretendem que experimente? Se a resposta for sim, tenha confiança. Mais que isso, repouse numa absoluta certeza".

- P.E Antonin-Dalmace Sertillanges, 1921

Resumo

Ao longo deste trabalho, será apresentado o processo de aprendizagem e as bases neurais do cérebro humano, bem como tecnologias atuais e excessos de informação da modernidade que atrapalham este processo, utilizando essa base como fundamentação para a criação do aplicativo SELF, como neurologia e psicologia aplicada ao aprendizado, além de cumprir a função de espaço organizado e auxiliador do foco para o autodidatismo, sendo a aplicação um meio de amenizar as dificuldades de aprendizado e memorização autodidata, senda essa a principal problemática abordada e o foco do trabalho. As pesquisas direcionadas ao público-alvo (de faixa etária e ocupação profissional não específicas com tendências autodidatas) mostram a pertinência do projeto evidenciando os dificultadores do aprendizado para a maioria das pessoas sendo elas as tecnologias do dia a dia que prendem a atenção do indivíduo de forma viciante, a falta de acompanhamento do seu progresso trazendo uma desmotivação que atrapalha a continuidade do aprendizado e a falta de reforço periódico do conteúdo aprendido. O projeto utiliza do método Pomodoro, método da repetição espaçada e reforço de conteúdo por meio de *flashcards* como remediador dos problemas apresentados para melhorar seu desempenho autodidata e facilitando a entrada de novos estudantes à essa prática. Ao longo deste trabalho escrito será também colocado em evidência as tecnologias utilizadas no desenvolvimento do aplicativo SELF assim como sua lógica de funcionamento, seu embasamento científico para o desenvolvimento das funcionalidades e sua fundamentação em psicologia para a escolha de design, visando trazer uma base sólida para os conceitos de melhoramento efetivo de foco. Para uma maior visibilidade do projeto, foi criado em paralelo ao programa um *site* informativo e de divulgação do projeto, que, por sua acessibilidade multiplataforma e facilidade de compartilhamento, auxiliará o projeto a chegar em mais pessoas de diversas esferas da sociedade e tornar o acesso ao estudo focado e de maior produtividade o mais diversificado possível, o trabalho mostrará as tecnologias aplicadas para a criação do *website* e seu funcionamento como canal de divulgação.

Palavras-chave: foco, aprendizado, psicologia, desempenho, autodidata, estudante

Abstract

Throughout this paper, will be presented the learning process and the neural basis on the human brain, as well as the actual technologies and the informational overflows of the actual society who disturbs that process, using that as basis for the creation of the SELF application, as applied neurology and psychology on learning in addition to supply the function of focus helper and organized space for the self-learning practice, thus supplying the need of an application who ease those difficulties as principal problematic addressed and focus of the paper. The research done on the target audience (non-specific age group and professional occupation with self-taught tendencies) showed the project relevance pointing the learning difficulties for most of the people being them the lack of reinforcement on the past learned subject, the day-by-day technologies who trap people by addicting them and the lack of progress accompanying that brings a demotivation to the continuity of the study. The project uses the *Pomodoro* method, the spaced-repetition method and reinforcement of subjects by flashcards revisions, as presented problems relievers to improve the student's performance on the self-learning process and smoothing the entry of new students to this practice. Throughout this written work, the technologies used in the development of the SELF application will also be highlighted, as well as its operating logic, its scientific basis for the development of functionalities and its foundation in psychology for the choice of design, aiming to provide a solid basis for the concepts of effective focus improvement. For greater visibility of the project, a website for information and dissemination of the project was created in parallel with the program, which, due to its multi-platform accessibility and ease of sharing, will help the project to reach more people from different spheres of society and make access to the focused study and greater productivity as diversified as possible, the work will show the technologies applied for the creation of the website and its functioning as a advertising channel.

Keywords: self-taught, learning process, student, methods, psychology.

SUMÁRIO

1. INTRODUÇÃO

Estando o Brasil abaixo do *ranking* internacional do *Programme for International Student Assessment* (*PISA)* nas disciplinas de Matemática, Ciências e Leitura (SALINAS, DE MORAIS, e SCHWABE, 2018) da Organização para a Cooperação e Desenvolvimento Econômico (OCDE), se faz necessária a reavaliação dos paradigmas estudantis contemplando a nível individual quais ações podem ser tomadas para que sejam mitigadas as dificuldades. Para lidar com a problemática suscitada, os indivíduos - lesados ou não pelo sistema educacional - podem arrogar para si a responsabilidade de aprender. O autodidatismo[1], portanto, é a via independente a qual pode ser utilizada para que o estudante atinja seus objetivos[2], não importando de onde venha. É, em certo sentido, a resolução da demanda pela educação sem que propriamente existam tutores ou mestres disponíveis, de modo que seja possível a aprendizagem sem a presença física em salas de aulas, fóruns e seminários – ainda que o ensino tradicional e o autodidata não sejam inconciliáveis. Consiste no esforço pessoal e particular em descobrir o caminho a ser trilhado – o ordenamento dos conteúdos, isto é, se é necessário entender “y” antes de “x” –, as leituras obrigatórias etc.

Existe também no estudo solitário uma categoria de dificuldades que não concernem propriamente à subjetividade, ao ímpeto e aos objetivos do estudante, e podem ser chamadas de dificuldades práticas: os problemas da falta de organização no curso dos dias – que afetam a disciplina; a possível desmotivação em não conseguir observar progresso após o esforço; o sentimento de que, por falta de registros precisos, não se tenha feito nada durante um intervalo de tempo; o desconhecimento de como fazer anotações e

[1] “Autodidatismo”, para os fins deste trabalho, é compreendido no sentido em que o professor Pierluigi o apresenta, isto é: o estudo solitário e ativo onde o esforço cognitivo é voltado à reflexão; às associações lógicas; a elaboração sistemática dos conceitos; a postulação de suposições do estudante; as repetições e a escrita dos conceitos com as próprias palavras. Para uma apresentação prática, verificar: PIAZZI, 2014, pp.56-63

[2] Muitos autores destacaram o tema da independência intelectual por meio do autodidatismo; mas para citar três, temos: Antonin-Dalmace Sertillanges; Jean Guitton e Louis Riboulet; Respectivamente em “A Vida Intelectual”; “O Trabalho Intelectual” e “Conselhos sobre o Trabalho Intelectual”.

separar o estudo em seguimentos de tempo para evitar o cansaço e assim por diante.

As dificuldades práticas são passíveis de resolução igualmente de maneira prática, e é isso que *SELF* se propõe a fazer: ser uma interface centralizada em suas funcionalidades que sirvam de guia para que o estudante mude para melhor sua vida de estudos, possuindo respaldo científico em cada uma de suas recomendações.

1.2 O Funcionamento e Propostas de *SELF*

1.2.1 No que o projeto consiste?

SELF é uma aplicação *desktop* que permite aos usuários utilizarem técnicas de estudo de maneira digital sem a necessidade de alternar entre diversos aplicativos; de modo conciliatório com o papel e a caneta unindo a organização e facilidade da mídia digital e a eficiência dos métodos tradicionais, como mostrado na sessão “Referencial teórico”. O Público-alvo da aplicação pode ser definido de maneira generalizada como toda e qualquer pessoa interessada em utilizar o computador como um auxílio aos estudos[3]. As limitações da aplicação caminham lado a lado com a descrição do autodidatismo, onde o aprendizado depende principalmente do usuário e como ele irá seguir os métodos de estudo.

Algumas das funcionalidades presentes no projeto são uma transposição de métodos já existentes à plataforma; não cabendo aos desenvolvedores os créditos autorais quanto as técnicas de estudo implementadas. São elas: Método *Pomodoro, Flashcards* e *Task-tracker*[4]; Bem como a funcionalidade “ver progresso” que apresenta de maneira dinâmica os dados coletados por meio das outras funcionalidades. As recomendações de

[3] Mais detalhes em “Desenvolvimento – Pesquisas e Gráficos”

[4] A definição e aplicação exatas de cada um dos métodos apresentados e sua incorporação às funcionalidades do programa estão detalhadas em “Desenvolvimento - ‘Requisitos funcionais e não-funcionais’”.

SELF e sua filosofia de conciliar papel e caneta com o meio digital podem ser interpretadas, com certo exagero, em uma espécie de “método”. Contudo, não foi utilizada tal nomenclatura porque a equipe desenvolvedora não gostaria de creditar sobre si a eficiência daquilo que já existia; senão por juntá-las em um compêndio e sua divulgação que, em determinado sentido, pode ser categorizada como divulgação científica.

1.2.2 Objetivos Gerais

As metas do projeto podem ser definidas de maneira geral em dois tópicos: 1) tornar os estudos mais organizados e eficientes do ponto de vista neurológico, unindo para isso a neurociência e a psicologia com o propósito de dar às informações estudadas o maior grau de retenção, revisibilidade e foco quando obtidas; e 2) Divulgar os benefícios do autodidatismo de maneira conjunta à crença de que todos são capazes de aprender por conta própria possivelmente qualquer assunto, por meio dos resultados obtidos.

1.2.3 Objetivos Específicos

Os objetivos específicos no desenvolvimento de *SELF* para que fossem atingidas as metas gerais do programa podem ser destacados nos seguintes tópicos:

- Criação de uma interface modulada a não-dispersão em suas cores e seu comportamento[5], visando trazer aos olhos dos usuários uma menor incidência de luz azul (Pedir referência ao João) para que não cause cansaço ou distúrbios do sono[6]; bem como a centralização de funcionalidades que impelem o usuário a se manter na aplicação em vez de alternar entre várias e incorrer no risco de distração;

[5] Mais detalhes em Desenvolvimento – “Telas e Códigos”.
[6] Mais informações disponíveis no subcapítulo “Atenção, saturação informacional e os estudos”.

- Transposição dos métodos de estudo ao programa – transformando-os em funcionalidades – permitindo a troca de dados entre eles. Os usuários podem utilizá-los de maneira complementar um ao outro;
- Exibição de resumos diários e semanais dispostos de acordo com os dados coletados pelas funcionalidades – Os métodos de estudo;
- Exibição de um guia inicial sobre o funcionamento do programa juntamente de explicações acerca dos processos cerebrais envolvidos nos estudos para que os utilizem da maneira mais eficiente possível;

1.3 Bases teóricas

As explicações presentes na atual sessão constituem a nível fundamental toda a fundação do projeto, posto que sua concepção e execução foram totalmente baseados nas pesquisas que a compõem. Não há o propósito de esgotar o escopo dos assuntos, posto que o *Status quaestionis* é extenso e as informações aqui contidas são um pequeno fragmento do vasto universo das neurociências, psicologia e sociologia; sendo um recorte feito propositadamente a ilustrar o porquê certas convenções foram adotadas. O usuário receberá instruções baseadas nos processos aqui descritos para que use o programa de modo adequado[7].

1.3.1 Como o aprendizado ocorre?

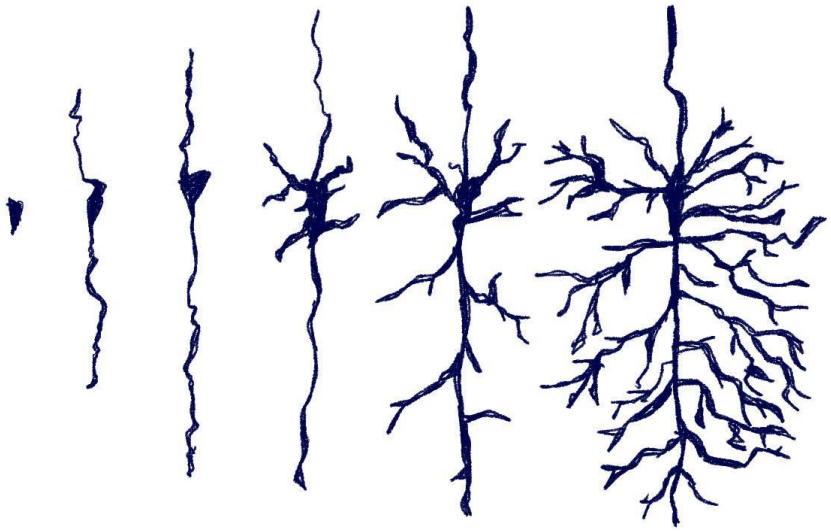
Do ponto de vista físico, aprender é modificar e criar redes neurais (*ibidem*, pp.35-43). Esse processo se dá por conta da neuroplasticidade que permite a criação de dendritos a partir dos quais novas informações obtidas se expandirão e criarão redes neurais durante o sono REM, se organizando física e especificamente de acordo com o conhecimento estudado (*ibidem*, pp. 64-73) (Figura 1)[8]. A importância de uma boa noite de sono, portanto, está permanentemente atrelada com o desempenho nos estudos; e essa foi a

[7] Como é possível ver na tela inicial do programa, disponível em Desenvolvimento – “Telas e códigos”

[8] Todas as Imagens estão disponíveis em maior qualidade ao fim do documento na sessão “Anexos”.

preocupação do design de SELF em utilizar cores que reduzem a emissão de luz azul juntamente dos avisos em sua tela inicial.

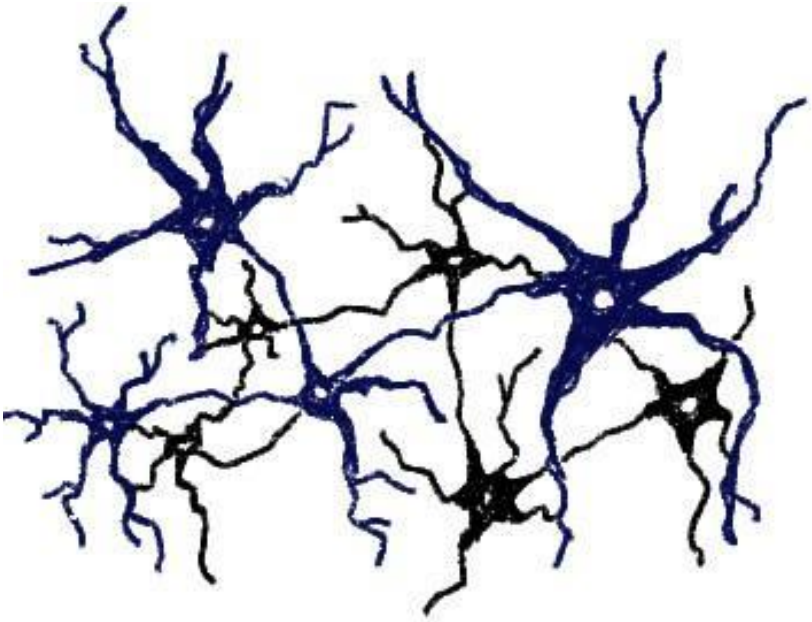
Figura 1 – Formação de novos dendritos se tornando uma rede neural



Adaptado de: “Inteligência em Concursos: Manual de Instrução do Cérebro para Concurseiros e Vestibulandos”, ilustração por Alan Carvalho.

As redes neurais que se formam analisam quimicamente seu entorno e determinam se uma rede neural previamente construída ainda tem relevância, incorrendo na possibilidade da mais nova tomar o espaço da antiga para si - provocando o esquecimento das informações (*ibidem*, pp.74-75). Esta dinâmica fundamenta o problema do esquecimento e, portanto, a necessidade de revisões ocasionais que contemplem os conhecimentos “mais antigos” para reforçá-los do ponto de vista neuronal (*ibidem*, pp. 75-81). As funcionalidades “Tarefas” e “*Flashcards*” correspondem ao conjunto de respostas que o usuário pode dar a essa problemática, juntamente da funcionalidade “Ver progresso” que o dá precisão sobre a direção de seus estudos.

Figura 2 - Rede de Neurônios Interconectados



Adaptado de: “Inteligência em Concursos: Manual de Instrução do Cérebro para Concurseiros e Vestibulandos”, ilustração por Alan Carvalho.

As informações armazenadas nas redes neurais estão também ligadas ao grau de estímulos cerebrais durante a obtenção da informação. Surge, portanto, a necessidade de tornar o estudo o mais estimulante e imersivo quanto for possível e, do ponto de vista neuromotor, só é possível com o esforço de se escrever à mão e desenhando (*ibidem*, pp. 168 – 182). A recomendação da tela inicial para que o usuário estude e utilize algumas funcionalidades sempre com papel e caneta se baseia nesse processo.

1.3.2 Atenção, saturação informacional e os estudos.

As mídias digitais representam quase a totalidade dos meios de informação e entretenimento no mundo contemporâneo (SPITZER, 2013a, seção 1ª). Aliado a isto, seu modelo de negócios, voltado a captar e capitalizar a atenção dos usuários, explora o sistema dopaminérgico cerebral a fim de causar um comportamento viciante, mantendo o usuário mais tempo atento à rede social, ao jogo, às séries e filmes. (SPITZER, 2013b, seção 3 – Vício)

Os estímulos assim recebidos modificam a estrutura física do cérebro, causando tolerância e exigindo um estímulo de maior intensidade para se atingir o mesmo grau de satisfação (L. HILTON, 2010). É correto afirmar que a atenção voltada aos estudos não sobreviverá nessas condições porque cerebralmente haverá todo um ciclo depressor que a sobrepujará (*SPITZER, 2013b,* seção 2 – Depressão).

Para além dos efeitos na atenção, o estado fisiológico provocado pelos estímulos luminosos inibe a secreção de melatonina[9] e muitas vezes induz à sensação de euforia. Nesse sentido, é necessário que o estudante se instale, no momento anterior e durante o estudo, em um ambiente especificamente modulado a evitar o contato com informações e estímulos não-essenciais; pois: 1) a exposição informacional tem, a longo prazo na atenção, efeitos similares ao uso da maconha; (WILSON, GLENN, 2009, par. sexto *apud* J LEVITIN, 2015) 2) uma distração no momento do estudo pode causar que a informação seja gravada no Corpo Estriado do cérebro em vez do Hipocampo, área que diz respeito à obtenção de habilidades manuais, não memórias (POLDRACK, par. sétimo apud J LEVITIN, 2015); 3) pois os estímulos – sobretudo canhões de cores e luz azul -, substituem a intensidade luminosa natural do círculo circadiano, levando à insônia.[10]

SELF destaca durante seu início a importância do usuário em se manter longe de estímulos antes de seus estudos, e seu caráter centralizador possui o propósito de fazer que o usuário não altere as telas e corra o risco de entrar em contato com informações não essenciais; bem como suas recomendações com o sono e sua relação com os estudos.

[9] Para um documento mais detalhado sobre Melatonina, acesse: https://www.endocrino.org.br/media/uploads/PDFs/posicionamento_sobre_melatonina_sbem.pdf

[10] Este processo se dá por meio do Núcleo Supraquiasmático. Para mais detalhes, é recomendada a leitura de “Neurociências: Desvendando o Sistema Nervoso (Mark F. Bear, Barry W. Connors etc.) “p. 676 par. “O Núcleo Supraquiasmático: um Relógio Encefálico”

2. DESENVOLVIMENTO

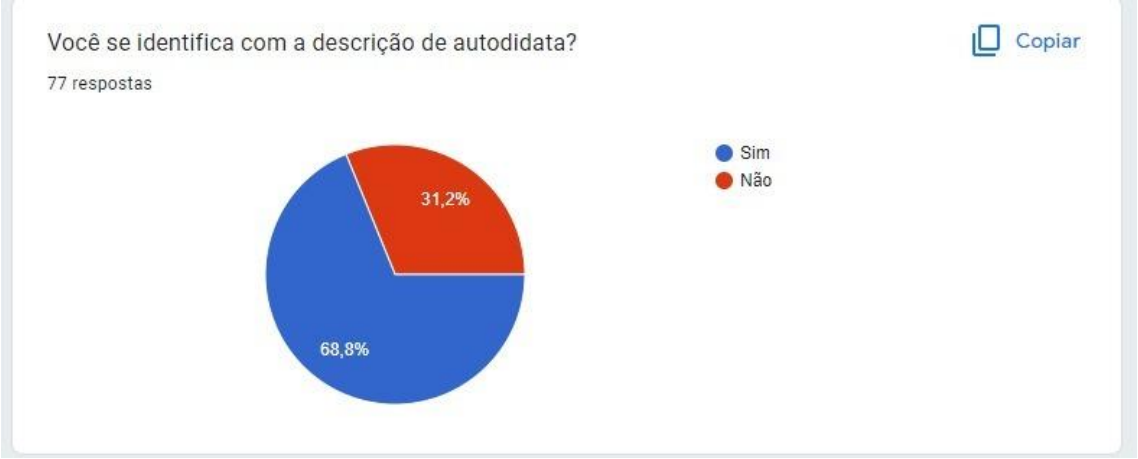
2.1.2 Pesquisas e gráficos

Visando estipular o público-alvo foram realizadas pesquisas e entrevistas para uma amostragem inicial. 75 pessoas foram consultadas e submetidas a um questionário no *Google Forms* com (quantidade) questões. O grupo de entrevistados foi escolhido de maneira proposital a ir de encontro com estudantes que utilizam computadores, constituindo-se de alunos do curso superior das áreas de Ciência da Computação, Comércio Exterior, Letras, Pedagogia, Psicologia, Química e Recursos Humanos; e alunos do Ensino Médio e ETIM, contemplando também os que já se formaram, mas que ainda não iniciaram o curso superior. Cada um dos entrevistados poderia responder apenas uma alternativa a cada pergunta. A aplicação dos questionários se deu do dia 18 de novembro de 2022 ao dia 22 do mesmo mês.

Visando garantir a total clareza e utilidade do questionário aos entrevistados, foram descritos os seguintes conceitos presentes nas perguntas:

- Definição de autodidatismo: “aprender algo por sua própria didática. Qualquer habilidade desenvolvida por conta própria, desde desenhar, tocar um instrumento ou estudar para um vestibular.”
- Definição de aparelhos eletrônicos: “Aparelhos eletrônicos envolvem quaisquer meios tecnológicos, acadêmicos ou de entretenimento.”

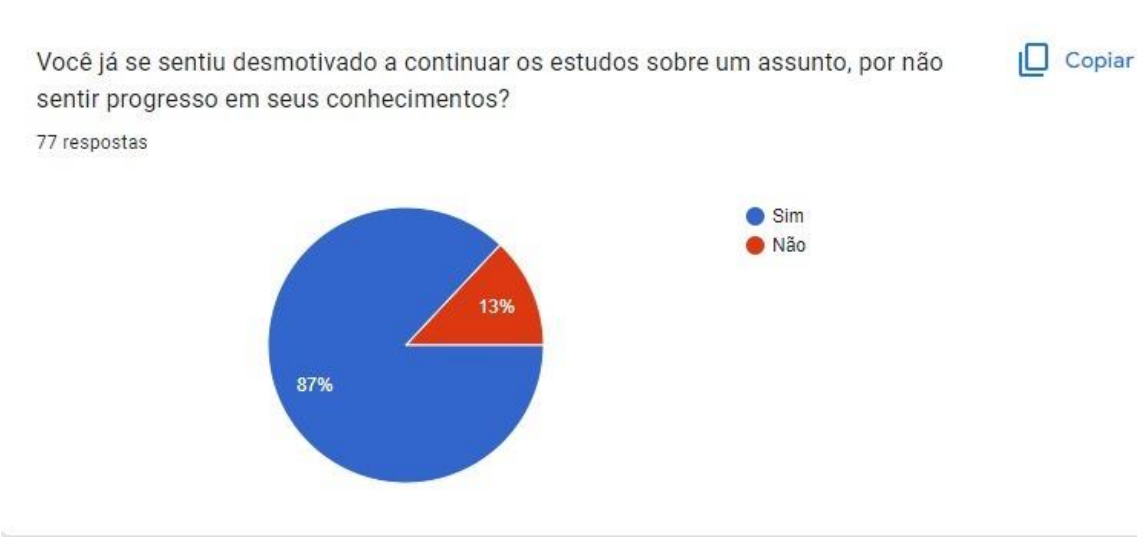
Figura 3 - Primeira pergunta da pesquisa



Fonte: do próprio autor, 2022.

Após a explicação sobre o que é ser um autodidata, foi feita a seguinte pergunta para se analisar quantos já se identificam com tal descrição, desta forma, podendo nós colhermos da melhor forma as dificuldades que um autodidata enfrenta, e então, atendermos essas demandas.

Figura 4 - Segunda pergunta da pesquisa



Fonte: do próprio autor, 2022

A segunda pergunta realizada teve o objetivo de verificar o quanto a desmotivação nos estudos afeta aqueles que trilham um caminho como autodidata. Tornando esse feedback importante para que se trabalhasse numa ferramenta capaz de gerenciar e exibir o progresso do usuário a respeito de sua jornada de estudos.

Figura 5- Terceira pergunta da pesquisa

Fonte: do próprio autor, 2022

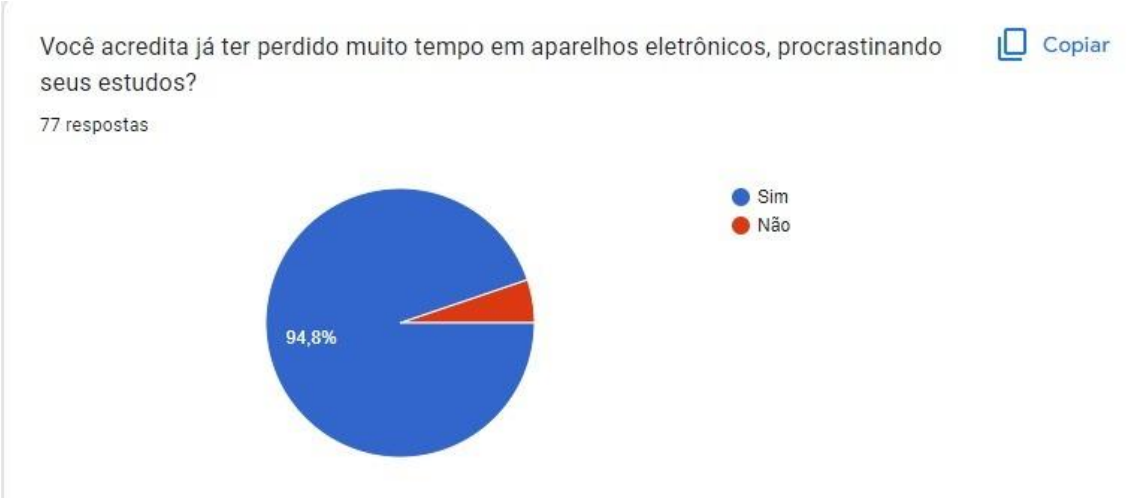
Para que se tivesse clareza a respeito da relevância de técnicas de memorização dentro da aplicação SELF, também foi realizada uma pergunta sobre o quanto o estudante possui dificuldade em se lembrar do conteúdo estudado.

Figura 6- Quarta pergunta da pesquisa

Fonte: do próprio autor, 2022

Sendo necessário saber o quanto os estudantes tinham dificuldade em filtrar as informações recebidas durante uma sessão de estudo, foi realizada uma pergunta sobre sua capacidade de manter o foco em relação aos aparelhos eletrônicos a sua volta, que poderiam estar sendo responsáveis por sua dispersão.

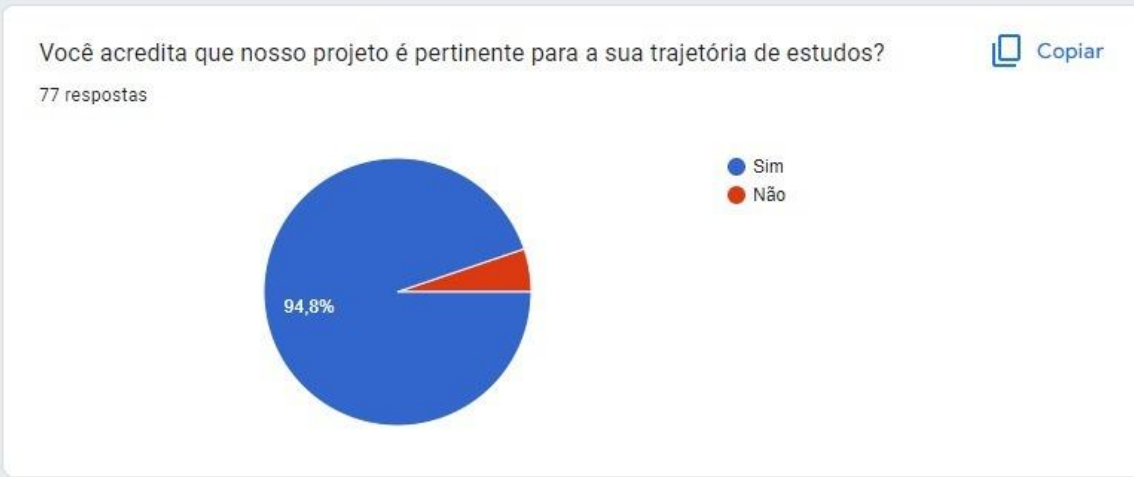
Figura 7- Quinta pergunta da pesquisa



Fonte: do próprio autor, 2022

Para que não houvesse dúvidas, foi realizada mais uma pergunta referente aos aparelhos eletrônicos, onde foi possível observar o quanto estes pode ser prejudiciais se não usados corretamente. Trazendo assim, um feedback importante para a relevância da tese no presente trabalho.

Figura 8- Sexta pergunta da pesquisa



Fonte: do próprio autor, 2022

E por último, após a leitura da descrição, uma breve apresentação do que é ser autodidata e uma sequência de perguntas direcionadas ao nível de dificuldade que os estudantes poderiam estar tendo durante seus estudos, foi realizada a presente pergunta para que se analisasse a demanda para criação do projeto SELF.

2.1.2.1 Discussão sobre a pesquisa

Sendo necessária que houvesse demanda para a pertinência do presente trabalho, foi realizada uma pesquisa via *Google Forms* com o intuito de colher o nível de dificuldade de estudantes que praticam o hábito do estudo independente. De antemão, foi feita uma breve descrição geral do projeto trabalhado, dando mais sentido às perguntas e conscientizando o entrevistado para que houvesse maior precisão em suas respostas.

Tendo em vista que o objetivo do presente trabalho é de fortalecer o foco, concertação e produtividade em meio a um excesso de informações muitas vezes vindas de aparelhos eletrônicos, a pesquisa foi um ponto essencial para que se comprovasse os problemas apresentados neste estudo e suas

respectivas possíveis soluções. Conforme pode-se observar acima, o resultado da pesquisa se encontra congruente com o que foi estudado para a criação de uma aplicação para autodidatas.

Como resultado, a presente pesquisa pode-se ser considerada um sucesso, já que ela foi capaz de comprovar um grande índice de dificuldade nos problemas apresentados e, também, um grande interesse por uma aplicação que reunisse ferramentas capazes de organizar um ambiente de estudos, proporcionando maior produtividade e menos distrações. Simultaneamente, a pesquisa possui muito potencial futuro, podendo se estender a grupos maiores de pessoas, de forma a serem encontradas novas dificuldades a serem trabalhadas.

2.2 Diagrama *Canvas*

Figura 9 – ilustração do modelo de negócios do projeto

Fonte: do próprio autor, 2022. Desenvolvido por meio da aplicação Web *Canva*.

A imagem acima oferece a descrição detalhada do modelo de negócios de *SELF*.

2.2.1 Licenciamento e modelo de negócios

SELF é uma aplicação de código-aberto e pode ser utilizada e distribuída livremente em ambientes domiciliares, acadêmicos e empresariais, acessível a todo tipo de pessoa e sem cobranças financeiras. O mantimento da aplicação se dará por meio de contribuições da comunidade via *GitHub*, bem como em doações financeiras dos usuários e apoiadores.

2.2.2 Site de divulgação do projeto

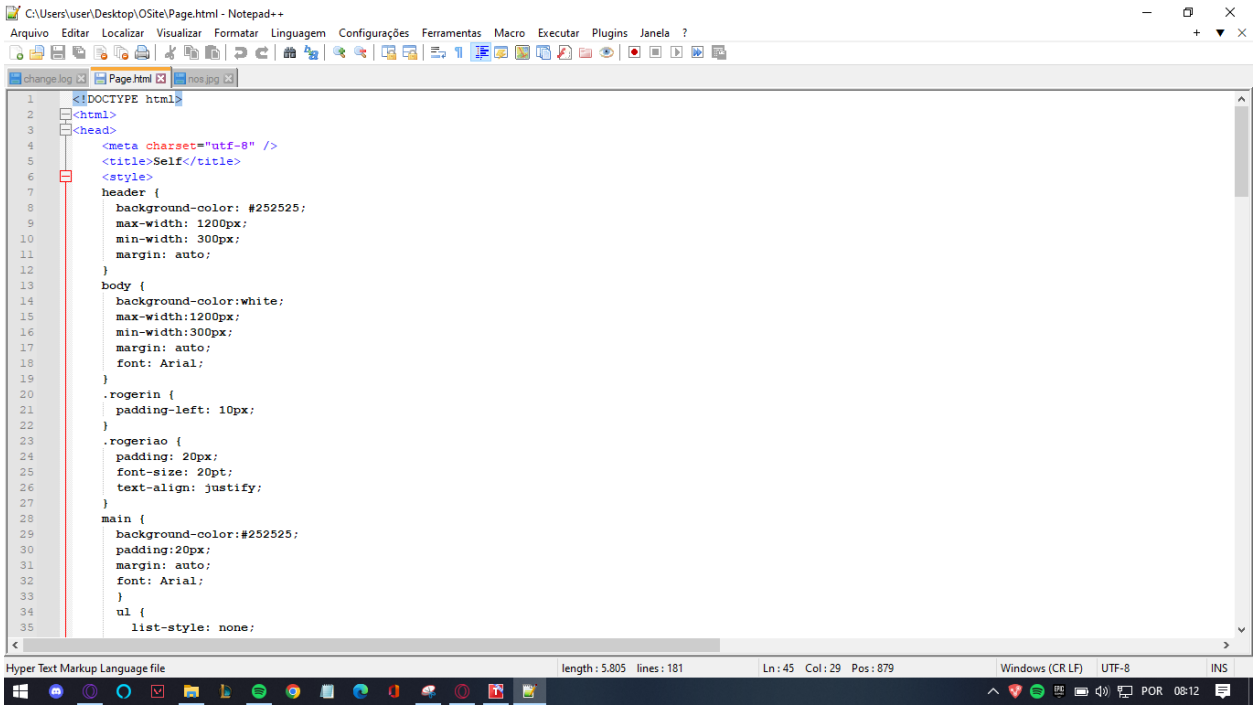
Os interessados poderão baixar e utilizar o *software* por meio do *GitHub* ou o site de divulgação oficial da aplicação – mais bem descrito na presente sessão. A opção por um site próprio se deu visando o conforto do usuário comum que não esteja habituado com a falta de intuitividade dos repositórios do *GitHub*.

2.2.2.1 Tela inicial do *Site*

Figura 10 - Tela inicial do *Site* de divulgação

Fonte: do próprio autor, 2022

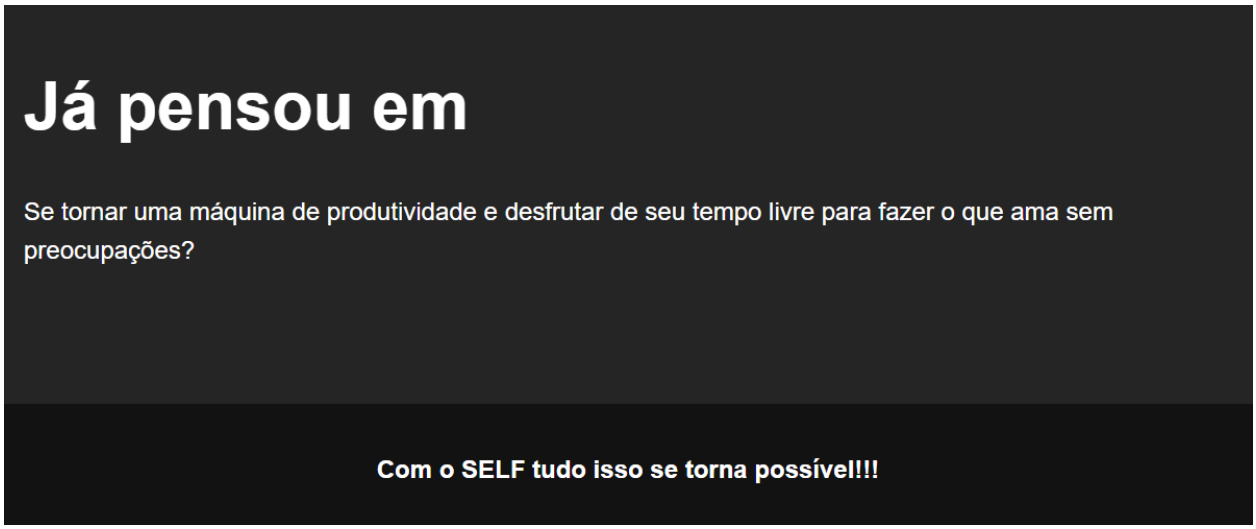
Figura 11 - Códigos da tela inicial



Fonte: do próprio autor, 2022

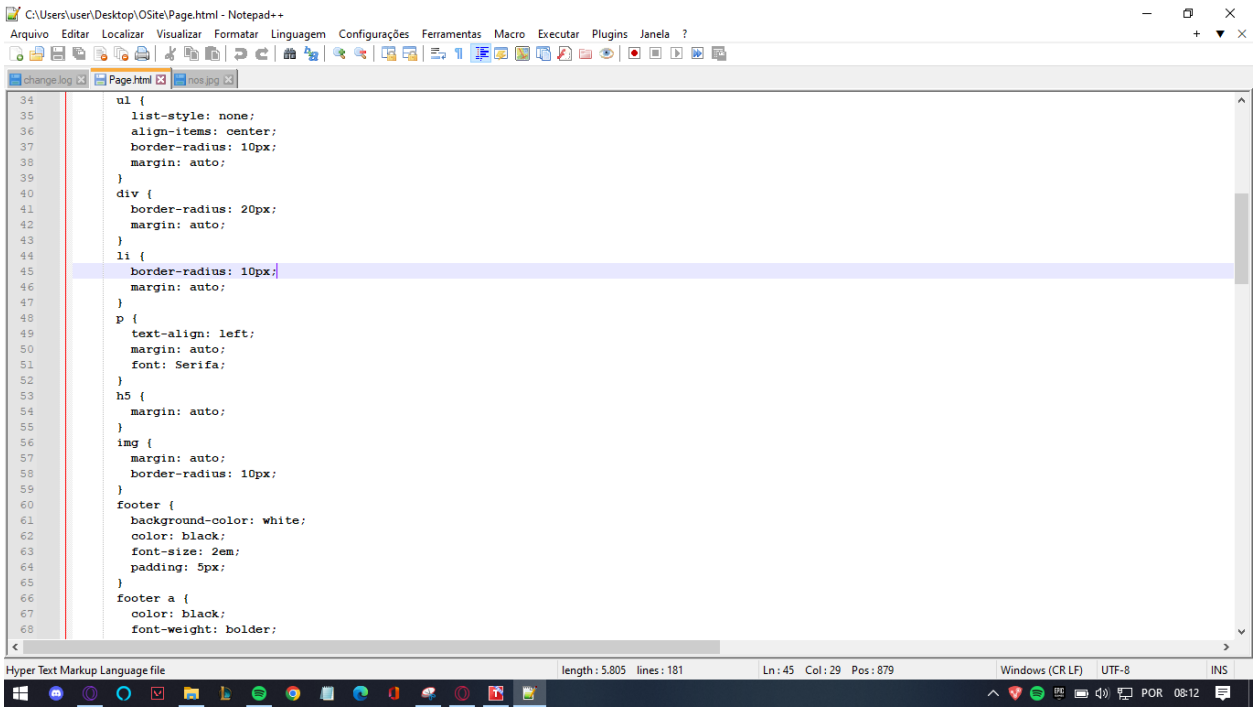
A tela inicial do site foi pensada em receber os visitantes do site, com uma imagem de fundo que facilmente pode-se ser associada aos estudos, produzindo de certa forma um ambiente imersivo congruente com a tematica do projeto.

Figura 12 - Introdução do site



Fonte: do próprio autor, 2022

Figura 13 - Tela de códigos segunda sessão do Site

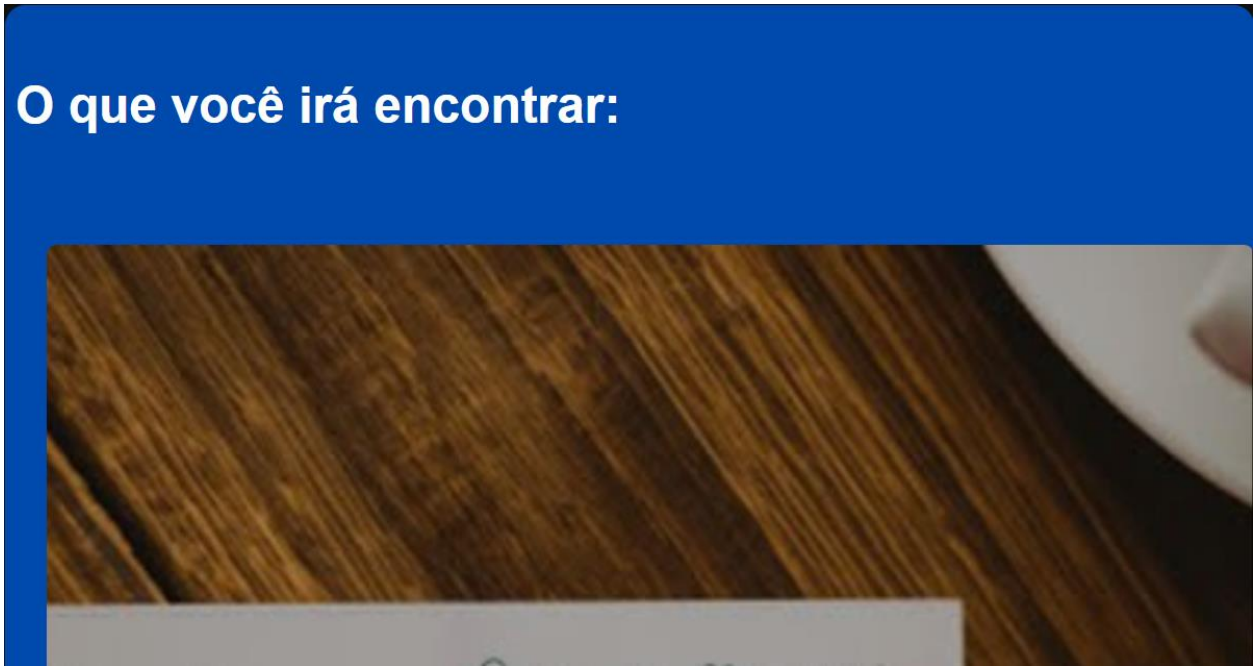


Fonte: do próprio autor, 2022

Pensando em introduzir o site ao visitante, foi feita essa breve apresentação para comunicar com clareza o que ele irá encontrar adiante.

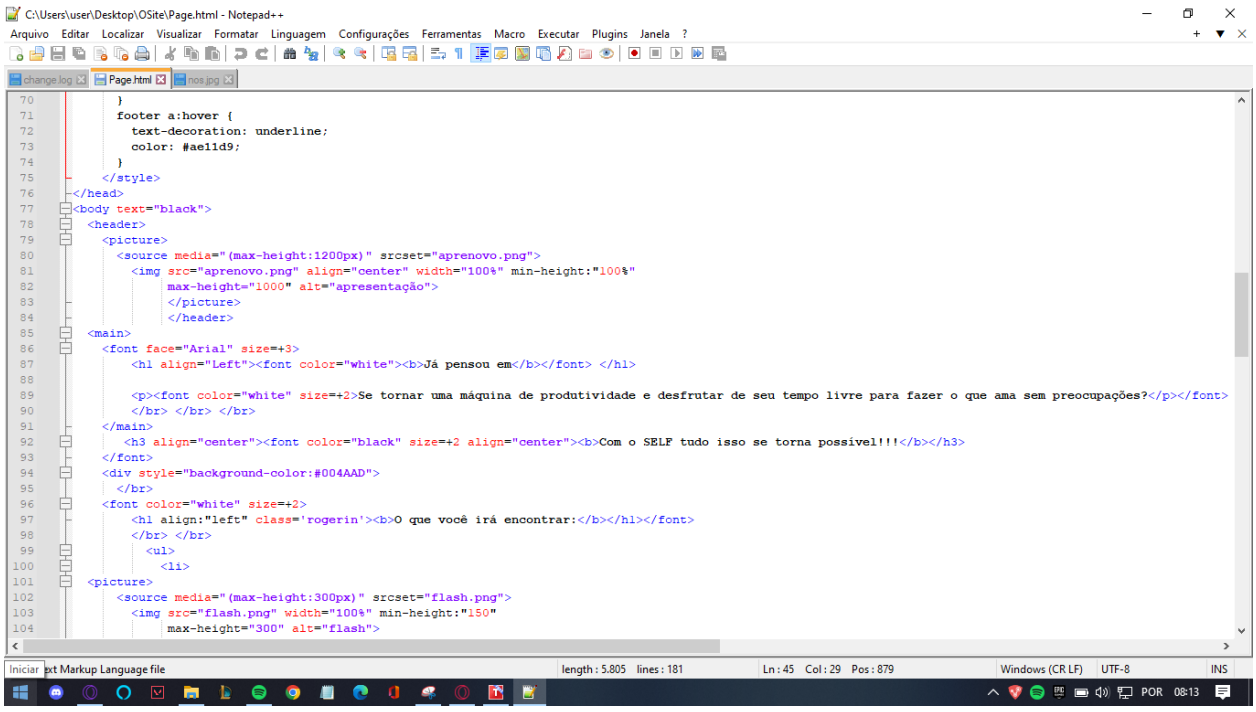
2.2.2.2 Tela de apresentação das funcionalidades do programa

Figura 14 - Introdução a sessão de apresentação das ferramentas



Fonte: do próprio autor, 2022

Figura 15 - Tela de códigos terceira sessão do site

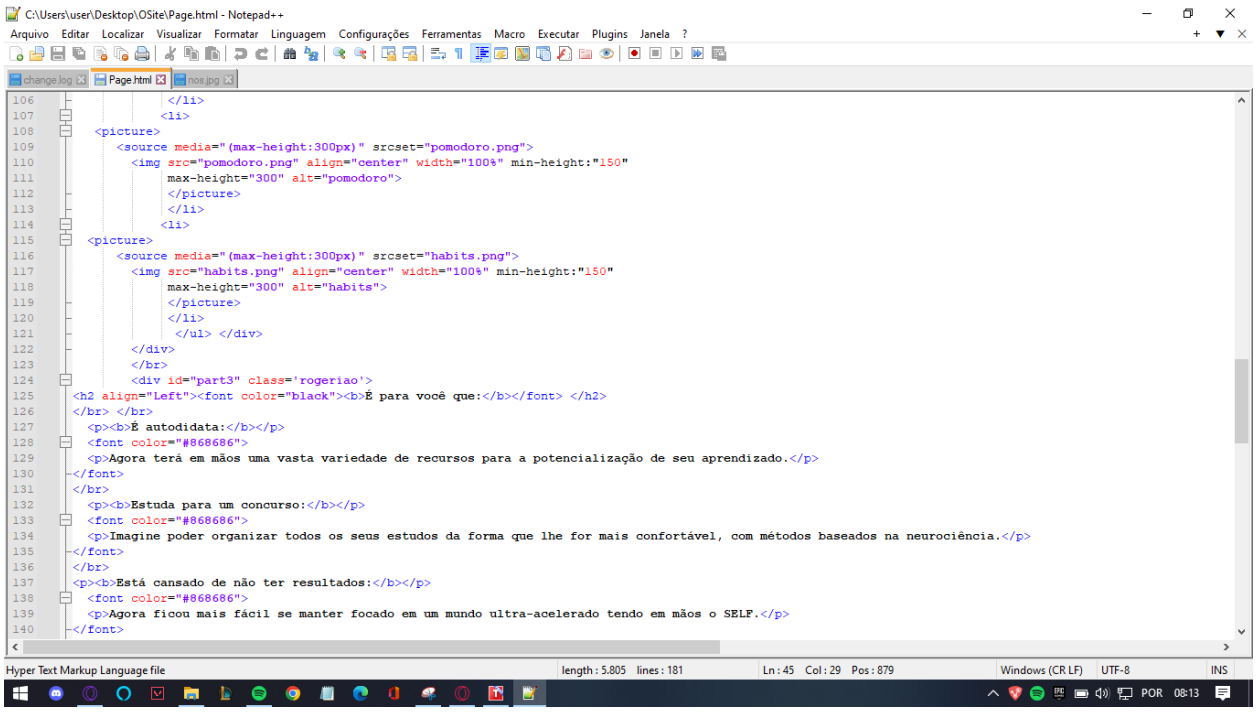


Fonte: do próprio autor, 2022

Figura 16 - Apresentação da funcionalidade Flashcards

Fonte: do próprio autor, 2022

Figura 17 - Tela de códigos quarta sessão do site



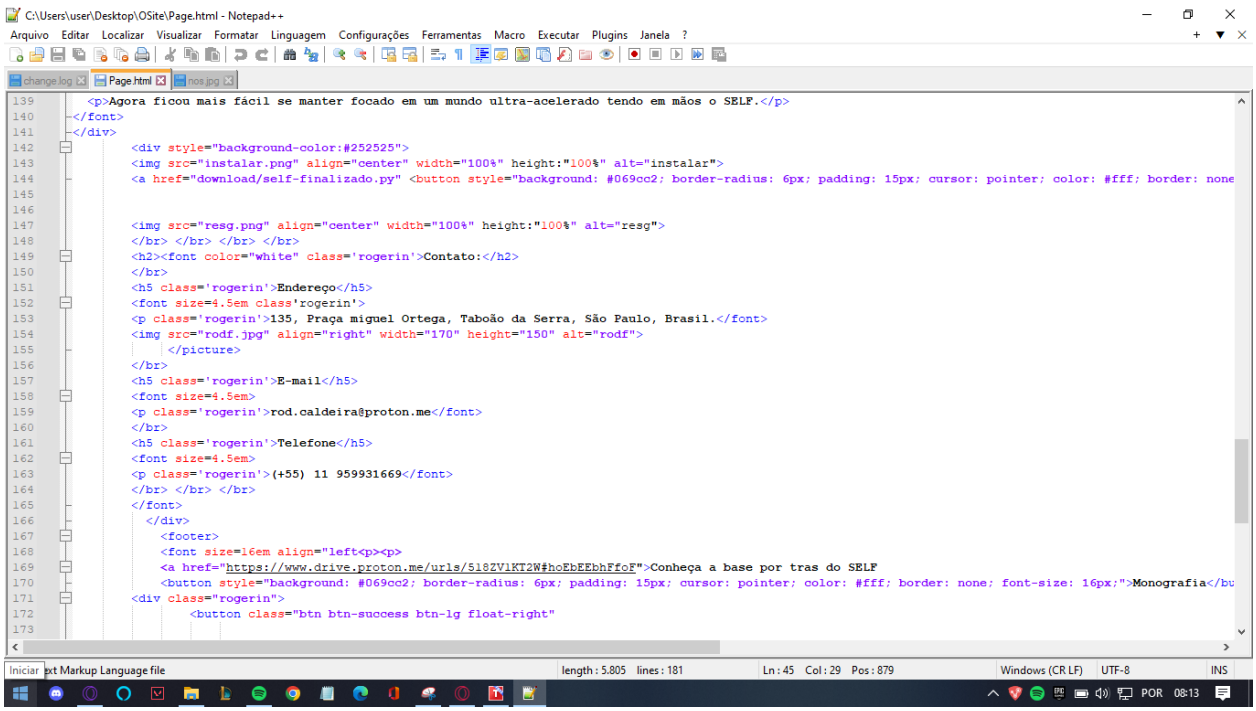
Fonte: do próprio autor, 2022.

Sessão destinada a apresentar brevemente uma das ferramentas, flashcards. Com uma mensagem simples e rápida com o intuíto de chamar a atenção do estudante para o aplicativo.

Figura 18 - Apresentação da funcionalidade *Pomodoro*

Fonte: do próprio autor, 2022

Figura 19 - Tela de códigos quinta sessão do *site*



Fonte: do próprio autor, 2022

Sessão destinada à uma breve apresentação de mais uma das ferramentas contidas na aplicção *SELF*, *Pomodoro*. Com o intuìto de trabalhar no interesase do estudante pelo que poderá ser encontrado na aplicação *SELF*.

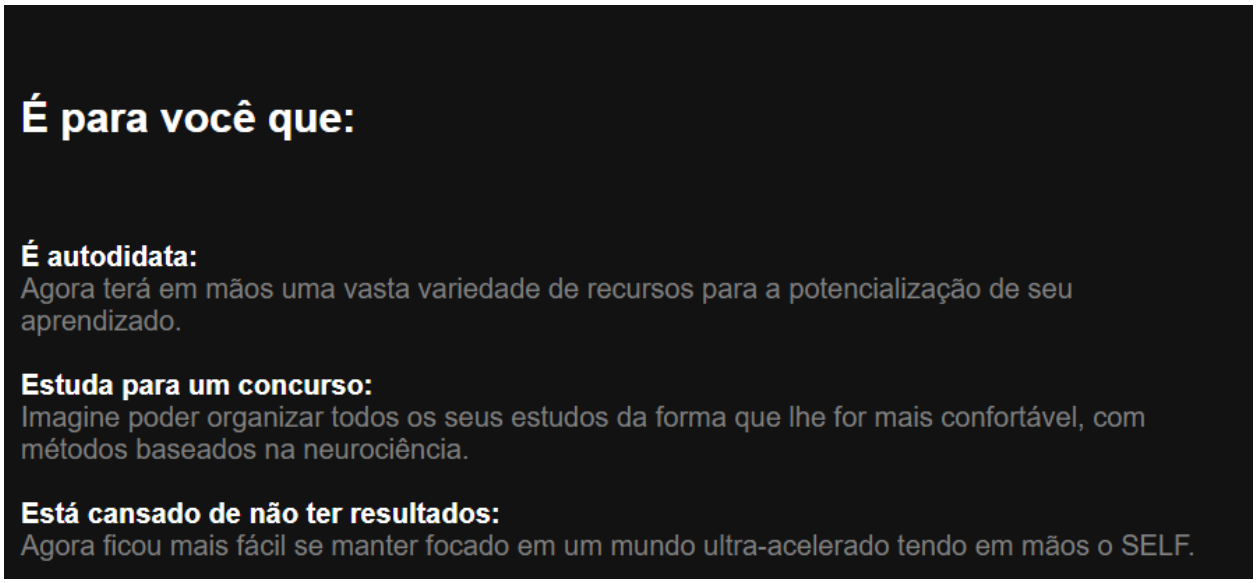
Figura 20 - Apresentação da funcionalidade "Tarefas"

Fonte: do próprio autor, 2022

Sessão destinada à uma breve apresentação de mais uma das ferramentas pertecentes dentro da aplicação *SELF*, *habit Tracker*. Seguindo a mesma linha das sessões anteriores, o objetivo é trabalhar no interesse do estudante

2.2.2.3 Marketing ao Usuário, contato e *Download*.

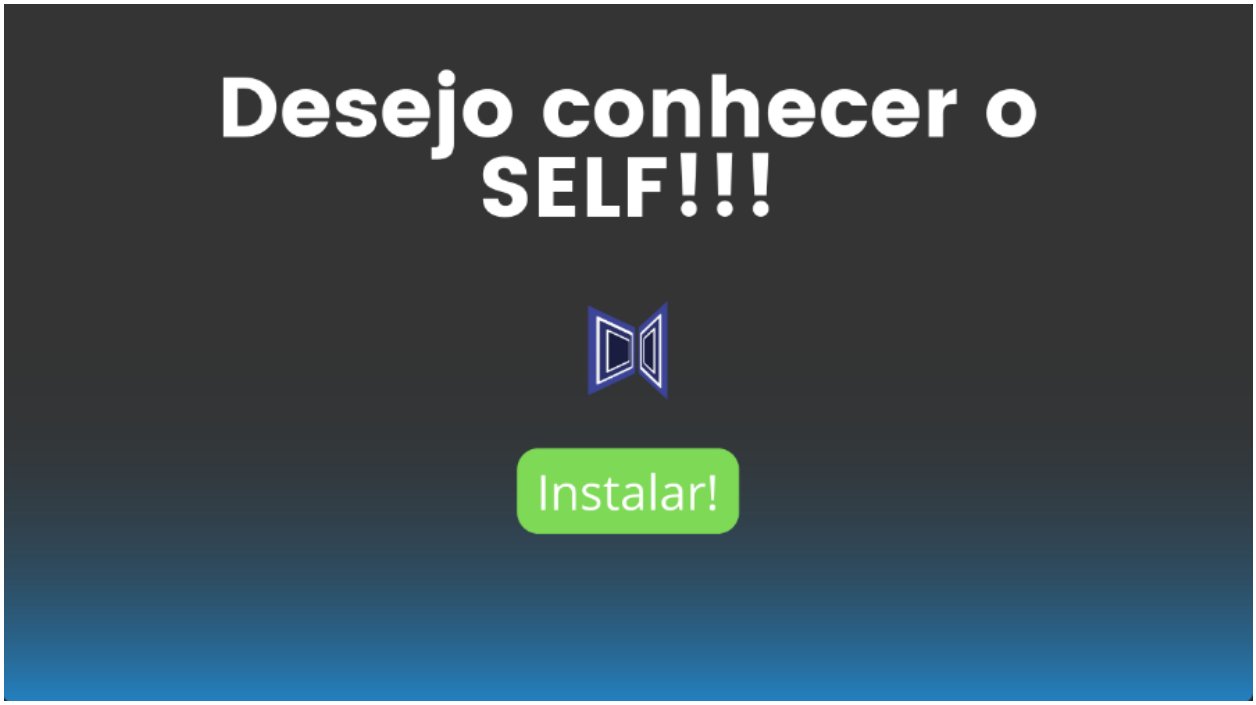
Figura 21- Identificação com o público alvo



Fonte: do próprio autor, 2022

Com o objetivo de proporcionar maior clareza a respeito dos beneficíos do *SELF* no cotidiano do estudante, foi escrita uma breve descrição do publico alvo ao qual as funcionalidades do projeto se destinam.

Figura 22- Redirecionamento para *download*



Fonte: do próprio autor, 2022

Para aqueles que leram até esta fase do site e se interessaram pela aplicação *SELF* foi feita numa tela de *download*, possuiindo um botão que direciona o usuário para a instalação do aplicativo *SELF*.

Figura 23- *Slogan Site*

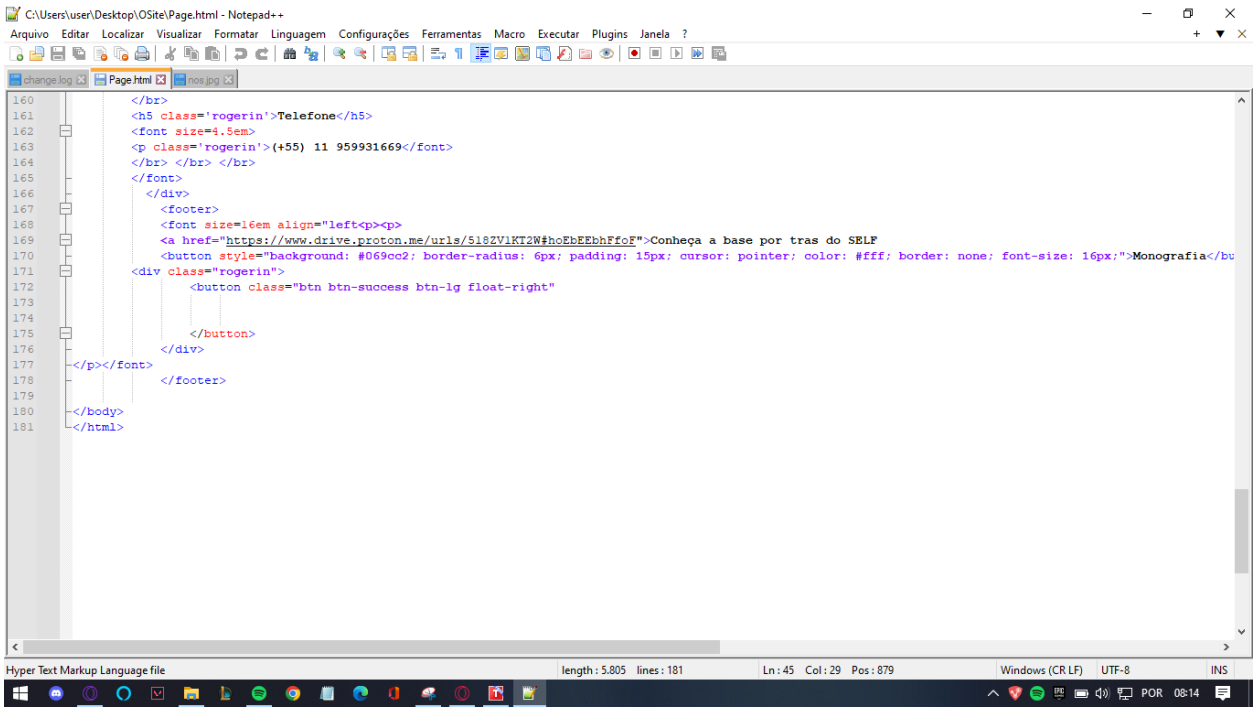
Fonte: do próprio autor, 2022

Tela feita com intuíto semelhante a tela inicial do *site*, onde a primeiro teve como objetivo receber o visitante com uma ambiente temático congruente com o tema do projeto, esta penúltima tela também teve como objetivo trabalhar com a sensação de um ambiente de estudos, trazendo também uma frase de efeito para encerrar esta sessão.

Figura 24- Rodapé do *Site*

Fonte: do próprio autor, 2022

Figura 25- Tela de códigos final do *site*



Fonte: do próprio autor, 2022

E por fim, a parte final do *site* contém informações de contato para quem se encontrar interessado e, também, um *link* de acesso à monografia.

2.3 Requisitos Funcionais e Não-funcionais

2.3.1 Descrição e Requisitos Funcionais da funcionalidade "*Flashcards*"

Os *Flashcards* são um método de memorização e revisão por meio de cartas. Em um lado da carta haverá uma pergunta e em seu verso a resposta correspondente a ela. Os *Flashcards* podem ser agrupados em conjuntos – chamados *decks* – de acordo com seus propósitos. Essa técnica retoma o assunto estudado e ajuda a amenizar o esquecimento no decorrer do tempo. O caráter neutro dos *Flashcards* permitem com que sejam aplicáveis a quaisquer disciplinas.

2.3.1.1 Requisitos funcionais da página inicial dos *Flashcards*

- Selecionar ou criar tópicos: O sistema deve providenciar a seleção e criação de tópicos, que serão utilizados para filtrar os *decks* do usuário. Representam o tema geral de um conjunto de *decks*.
- Criar um *deck* de *Flashcards*: O sistema deve disponibilizar áreas para a organização das cartas. Essas áreas serão chamadas de "*decks*" que possuirão um tópico e terão a função de alocar as cartas criadas pelo usuário;
- Adicionar cartas a determinado *deck***:** O sistema deve dispor ao usuário a liberdade de adicionar cartas dentro de seus *decks* de acordo com o planejamento de estudos desenvolvido por ele;
- Adicionar frente e verso das cartas: O sistema deve possuir ferramentas para que o usuário se sinta livre em preencher suas cartas com as suas respectivas questões e respostas sobre os temas que desejar abordar;
- Permitir a alteração e exclusão de cartas: O sistema deve viabilizar a função de alteração e exclusão de cartas onde desta forma o usuário poderá alterá-las, baseando-se em sua necessidade.

2.3.1.2 Requisitos funcionais da página de estudos dos *Flashcards*.

- Permitir a exibição das informações de um deck: O sistema deve ser capaz de exibir na página de estudo o nome do deck selecionado pelo usuário, bem como a quantidade de cartas, nome de seu respectivo tópico e uma barra de progresso referente ao rendimento do usuário.
- Permitir a visualização da frente e do verso de uma carta: O sistema deve permitir com que o usuário visualize inicialmente a frente, isto é, a pergunta de uma carta, e depois revele o verso com a então resposta correta.
- Inserir o *feedback* acerca do nível de acertos (negativo, intermediário, positivo): O sistema deve ser capaz de mensurar a performance do usuário, baseando-se no *feedback* pessoal do mesmo ao revelar a resposta de uma carta; esse sistema será composto por 3 *emojis*[2] que representam o desempenho do estudante:

Rosto feliz – O usuário lembrou com sucesso da resposta;

Rosto neutro – O usuário lembrou com dificuldade;

Rosto triste – O usuário falhou em se lembrar.

2.3.1.3 Descrição e Requisitos Funcionais da funcionalidade “Tarefas”

A funcionalidade “*Tarefas*” se trata de uma ferramenta de gestão para estipular metas e organizar assuntos a serem estudados. Ao adicionar uma nova tarefa, o usuário poderá visualizar um objetivo a ser concluído, e junto com a ferramenta “*Pomodoro*” – descrita no tópico seguinte – será capaz de definir um período para que a tarefa seja concluída. São seus Requisitos Funcionais:

- Permitir a adição de uma tarefa: O sistema deve disponibilizar uma área para a criação de novas tarefas, onde o usuário preencherá o nome e poderá alterar o status, data inicial, data final e o tópico da tarefa adicionada;
- Viabilizar a exclusão de uma tarefa: O sistema deve disponibilizar o botão para a exclusão de uma tarefa caso o usuário por algum motivo decida deletá-la;
- Permitir a manipulação das tarefas: A criação de tarefas será atrelada a: I) Edição do *status*, onde o usuário informará se ela está ativa ou já foi concluída; II) Alteração da data inicial; III) alteração da data de conclusão; IV) Adição de um tópico, onde representará o assunto da respectiva tarefa;
- Dispor a filtragem de tarefas via *status* ou tópico: O sistema deve disponibilizar para o usuário a opção de ordenar as tarefas que serão exibidas baseando-se em seu tópico e/ou *status*.

2.3.4 Descrição e Requisitos Funcionais da funcionalidade “*Pomodoro*”.

A funcionalidade “*Pomodoro*” é a transposição do “método *pomodoro”*[11], que consiste em gerir o tempo auxiliando o mantenimento do foco em uma tarefa até o momento de sua conclusão. Essa metodologia de estudo geralmente é eficaz para pessoas que possuem dificuldade em manter a

[11] Mais informações disponíveis em: https://geekbot.com/blog/pomodoro-technique-ultimate-guide-with-examples-tools-and-tips/

constância e foco na realização de seus objetivos. O estudante terá à sua disposição três estados de estudo: o tempo de trabalho, padronizado em vinte e cinco minutos; o tempo de descanso curto, possuindo por padrão cinco minutos; e o tempo de descanso longo, que é acionado na soma três tempos de trabalho concluídos em seu ciclo de estudos. O usuário poderá configurar os minutos de acordo com sua necessidade e atribuir tarefas a um ciclo de estudos, de modo com que poderá observar o tempo investido na resolução de seus ofícios. Os Requisitos Funcionais da funcionalidade são:

- Trocar estados de estudo: O sistema deve ser capaz de alternar entre os estados já listados acima, tempo de estudo, descanso curto e descanso longo. Após dado número de estudos (três por padrão), o sistema deve alterar para o descanso longo, permitindo ao usuário descansar por um determinado período;
- Dar um "*check"* nas tarefas e associá-las a um *pomodoro*: O sistema deve mostrar as tarefas ao usuário para que ele possa relacioná-la a um *pomodoro* (bloco de estudo). Desta forma, o usuário poderá dar um "*check"* em uma tarefa quando sentir que concluiu a mesma no tempo de estudo.

2.3.1.4 Descrição e Requisitos Funcionais da funcionalidade "Ver progresso"

Visando a conscientização do estudante sobre seu nível de aprendizado por meio da aplicação *SELF*, o usuário poderá medir, quantificar e acompanhar seu progresso de várias maneiras. Cada funcionalidade fornece um *feedback*. Desta forma, o sistema é capaz de se basear nesses *feedbacks* ao montar gráficos que serão exibidos ao usuário em uma tela específica para ver o seu progresso, oferecendo ao usuário uma visualização geral de seu desempenho, servindo como um sinal de que o estudante está progredindo ou que precisa mudar de rumo. São seus Requisitos Funcionais:

- Filtrar estatísticas dos *Flashcards* por tópico: O sistema deve ser capaz de filtrar as estatísticas baseando-se no tópico que o usuário escolher, bem como as estatísticas gerais de todas as cartas;

- Mostrar o total de *Flashcards*: O sistema deve ser capaz de mostrar a quantidade geral de *Flashcards* estudados ou específica para o tópico respectivo a filtragem.
- Mostrar os *Flashcards* separando-os pelo rendimento do estudo: O sistema deve ser capaz de mostrar, via texto, os *decks* baseando-se na porcentagem de desempenho; esses *decks* serão divididos entre categorias satisfatórias e insatisfatórias;
- Mostrar o desempenho dos estudos dos *decks* através de um gráfico: O sistema deve viabilizar a amostragem de dados da funcionalidade por meio de um gráfico "*pizza*" que contará com as informações de respostas obtidas pelo *feedback* pessoal do usuário.
- Filtrar estatísticas do *pomodoro* por tópico: O sistema deve ser capaz de filtrar as estatísticas baseando-se no tópico que o usuário escolher, caso opte por não visualizar as estatísticas gerais;
- Mostrar a quantidade de estudos do *pomodoro* realizados: O sistema deve dispor ao usuário, via texto, a quantidade de estudos inteiros efetivamente realizados.
- Mostrar o tempo total de estudo nos ciclos do *pomodoro*: O sistema deve dispor ao usuário, via texto, a quantidade de horas e minutos investidos nos estudos;
- Mostrar os dados de tempo e quantidade de estudo através de um gráfico: O sistema deve viabilizar a amostragem de dados da funcionalidade através de um gráfico de barras que contarão com as informações de estudo armazenadas previamente coletadas pela funcionalidade *Pomodoro*.

2.3.1.5 Requisitos Não-funcionais

2.3.1.6 Requisitos de Produtos

Requisitos que especificam o comportamento do produto.

2.3.1.7 Requisitos de portabilidade

- Nome: O sistema é portável para *Windows*, *Mac* e *Linux*.

Descrição: O usuário baixa no site da aplicação o arquivo referente ao seu sistema operacional.

2.3.1.8 Requisitos de Confiabilidade

- Nome: Confiabilidade é proporcionada pelo código aberto, isto é: transparência dos processos executados.

Descrição: O usuário tem a opção de analisar o código e as instruções realizadas durante seu funcionamento.

2.3.1.9 Requisitos de Eficiência

- Nome: O sistema deve se adequar ao processo de várias tarefas ao mesmo tempo.

Descrição: O timer do *pomodoro*, se ativo, deve funcionar mesmo que a funcionalidade não esteja visível. Tarefas e tópicos ao serem adicionados, devem ser também atualizadas nas outras funcionalidades do programa.

- Nome: O sistema deve mostrar os dados via gráfico de maneira efetiva.

Descrição: Quase que instantaneamente o usuário deve poder visualizar no sistema os dados de maneira fluída.

2.3.1.10 Requisitos organizacionais

Requisitos decorrentes de políticas e procedimentos corporativos. Ex. padrões, infraestrutura etc.

2.3.1.11 Requisitos de Entrega

- Nome: Um diário de bordo deve ser fornecido mensalmente.

Descrição: O arquivo deve ser entregue ao professor responsável para fins de avaliar o planejamento do grupo.

2.3.1.12 Requisitos de Implementação

- Nome: O sistema deve ser desenvolvido na linguagem *Python*.

Descrição: Linguagem na versão 3.9, para garantir a eficiência das versões mais recentes.

- Nome: O sistema deve utilizar a biblioteca *JSON*.

Descrição: Biblioteca para garantir o funcionamento adequado das funções de importação e exportação de materiais.

- Nome: O sistema deve utilizar a biblioteca *sqlite3*.

Descrição: Biblioteca para garantir o funcionamento do *CRUD* (*create*, *read*, *update* e *delete*), isto é, a manipulação dos materiais de estudo.

2.3.1.13 Requisitos de Padrões

- Nome: Uso de programação orientada a objetos.

Descrição: Objetos compõem a lógica de programação das funcionalidades.

- Nome: Uso de programação orientada a *Widgets*.

Descrição: *Widgets* compõem a construção da interface gráfica.

2.3.1.14 Requisitos externos

Requisitos decorrentes de fatores externos ao sistema e ao processo de desenvolvimento. Ex. requisitos de interoperabilidade, legislação, localização geográfica etc.

2.3.1.15 Requisitos de interoperabilidade

- Nome: O sistema deve interagir com uma base de dados *sqlite3*.

Descrição: Os dados e materiais de estudo devem ser compatíveis com as tabelas *SQL* criadas através do módulo *sqlite3*.

- Nome: O sistema deve interagir com arquivos do tipo *JSON*.

Descrição: Os dados de exportação e importação devem ser compatíveis e respectivamente armazenados e carregados em um arquivo de extensão “*. json*”.

2.3.1.16 Requisitos éticos

- Nome: Sistema preza pela privacidade de uso.

Descrição: O sistema não possui a demanda de coleta de qualquer dado pessoal vindo do usuário.

2.3.1.17 Requisitos Legais

O sistema deverá atender as normas legais da licença *GNU General Public License* utilizada nos *softwares*: *Modern GUI PyDracula*, *Pomodoro* e *PyQt5 Daily Task Planner App*. A licença permite usar, copiar, modificar, fundir, publicar e distribuir cumprindo as condições:

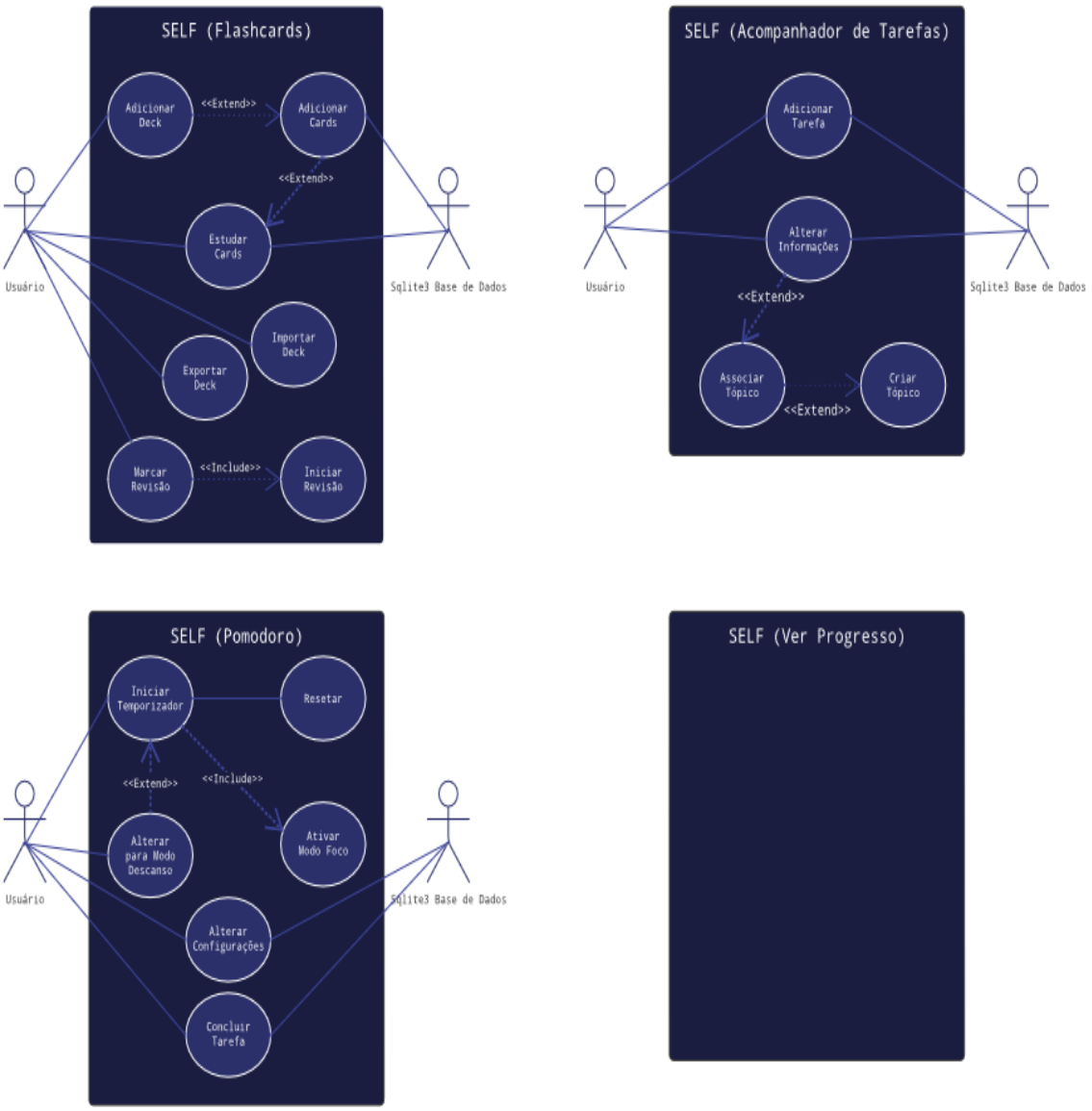
- O *software* é fornecido em sua forma original sem garantias de qualquer tipo. E não podendo efetuar a violação dos direitos autorais em nenhum caso, sempre expressando a devida autoria e serão responsáveis por qualquer reivindicação, danos ou qualquer outra responsabilidade que vier decorrer do uso do *software*.
- Utiliza-se a licença *GNU General Public License* para permitir que o usuário possa alterar o programa além de mantê-lo gratuito. Podendo usar o código-fonte, ou em partes para assim criar programas mantendo-os gratuito.

2.4 Diagramas do Software[12]

[12] Todos os diagramas estão disponíveis em maior qualidade na sessão “Anexos” ao fim do documento.

2.4.1 Diagrama Casos de Uso

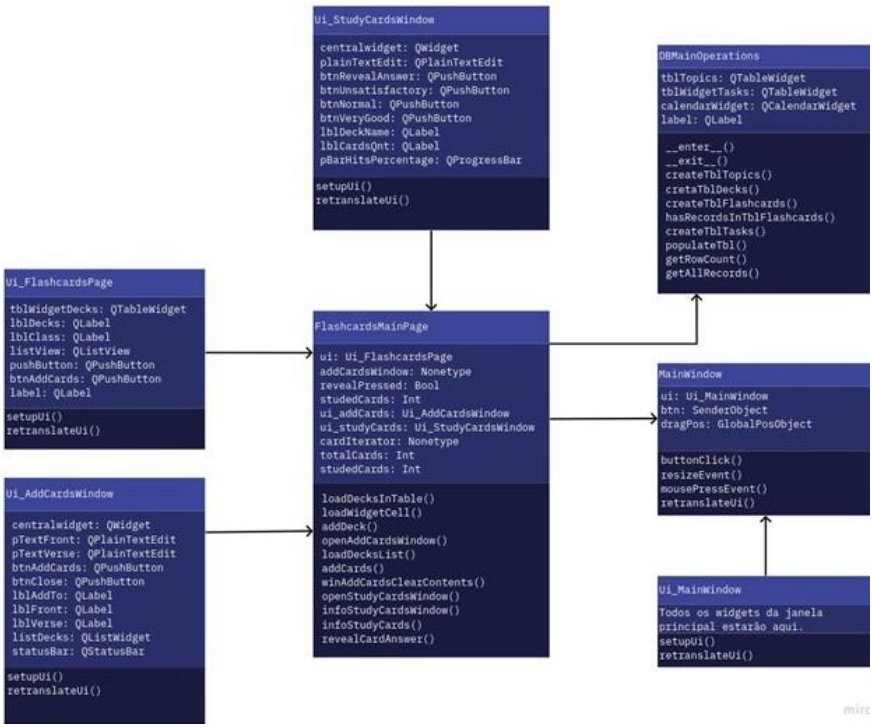
Figura 26- Diagrama de Caso de Uso



Fonte: do próprio autor, 2022. Construído por meio da aplicação Web *Miro*.

2.4.2 Diagramas de Classe

Figura 27 - Diagrama de Classe Flashcards



Fonte: do próprio autor, 2022. Construído por meio da aplicação Web *Miro*.

Figura 28 - Diagrama de Classe DailyTask

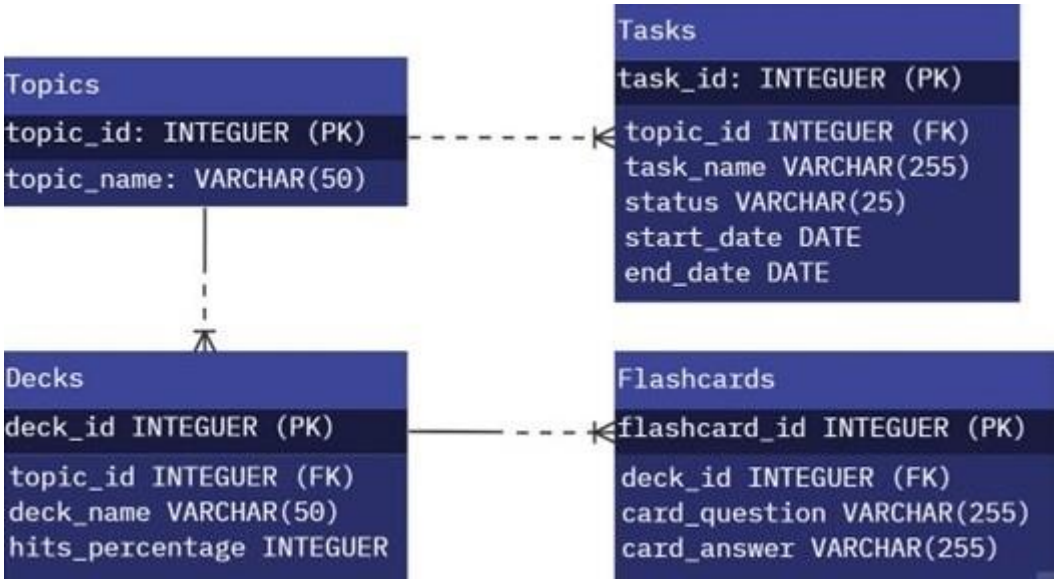
Fonte: do próprio autor, 2022. Construído por meio da aplicação *Web Miro.*

Figura 29 - Diagrama de Classe *Pomodoro*

Fonte: do próprio autor, 2022. Construído por meio da aplicação *Web Miro*.

2.4.3 Diagrama Entidade-Relacionamento

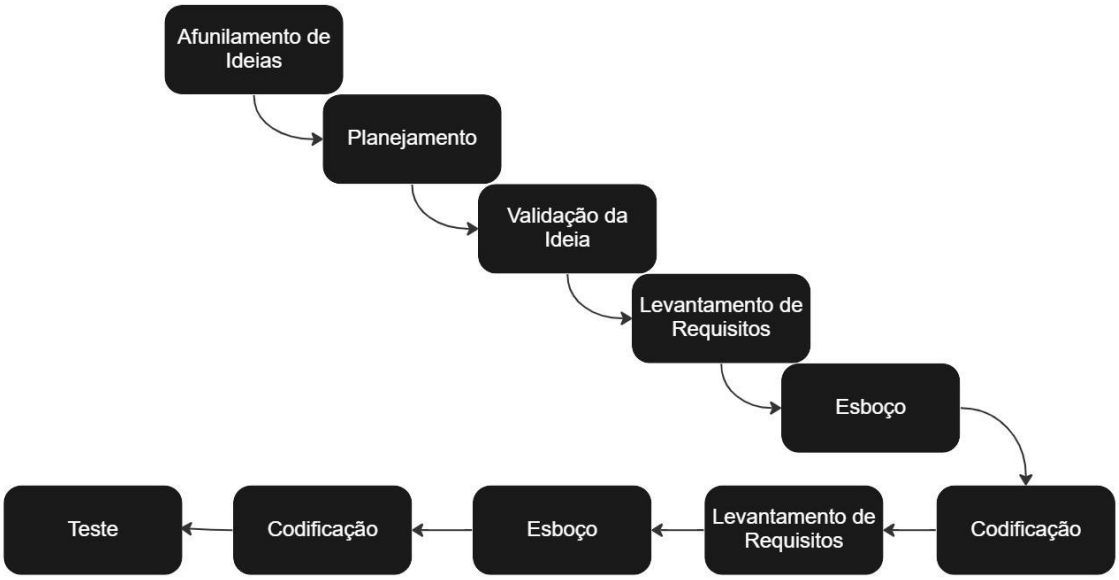
Figura 30 - Diagrama Entidade-Relacionamento



Fonte: Do próprio autor, 2022. Construído por meio da aplicação *Web Miro*.

2.4.4 Ciclo de vida do *Software*

Figura 31- Diagrama Ciclo de Vida do *Software*



Fonte: do próprio autor, 2022. Construído por meio da aplicação *Web Miro*.

2.5 *Softwares* Utilizados

2.5.1 Linguagem

- ***Python 3.9*** (Versões anteriores podem causar problemas de compatibilidade).

Python é a linguagem utilizada pelos desenvolvedores da aplicação *SELF* pois se demonstrou congruente com os propósitos do projeto, oferecendo portabilidade para diversos sistemas operacionais, tornando a aplicação mais acessível a qualquer tipo de pessoa; possuindo uma comunidade ativa em código aberto e suporte, tornando mais fácil o trabalho da pequena equipe de desenvolvimento; tendo uma sintaxe simples e muitas bibliotecas de desenvolvimento de interfaces gráficas.

2.5.2 Bibliotecas

2.5.2.1 Biblioteca "*PySide6*"

Toda a criação da interface gráfica do projeto *SELF* foi desenvolvida por meio da biblioteca *PySide6*; módulo utilizado em conjunto com a ferramenta *Qt Designer* para constituir interfaces de maneira prática mantendo a qualidade estética da aplicação. A biblioteca proporciona um ambiente orientado a "*widgets*" – componentes (botões, *frames*, tabelas etc.) que podem ser utilizados da maneira padrão, isto é, do modo já definido pela biblioteca, ou de maneira personalizada: cabendo ao programador a decisão de criar *widgets*.

2.5.2.2 Biblioteca "*JSON*" (*JavaScript Object Notation*)

A importação e exportação de *Flashcards* e tarefas para compartilhamento e/ou *backup* têm dependência de funcionamento fundamentada nos arquivos de extensão ". *json".* A biblioteca é responsável pela criação e manipulação dos arquivos que serão exportados ou importados. A conversão de uma base de dados para um arquivo menor e legível pelo programa é feita por esse módulo, de tal modo que o arquivo ". *json"* será utilizado para tal funcionalidade.

2.5.2.3 Biblioteca "*Sqlite3*"

Se o módulo *JSON* manipula um arquivo menor e legível, o módulo *Sqlite3* é responsável por tratar da base de dados do programa. Essa biblioteca providencia uma lógica compacta e eficiente para trabalhar com a engenharia *SQL*. Ao importar um arquivo ". *json"*, o módulo *Sqlite3* trabalha passando esses dados para o banco de dados real do projeto. Todas as funcionalidades necessitam do banco de dados providenciado por este módulo para funcionar adequadamente na próxima vez que o usuário utilizar o software.

2.5.3 *Softwares-base* sob licença *GPL*

2.5.3.1 “*Modern GUI PyDracula*”, por Wanderson Magalhães

Software conceito para o modelo de interface gráfica empregado neste trabalho; que se utiliza das lógicas de animação de menus e carregamento de paleta de cores empregadas pelo repositório em questão. Além disso, foi de sublime contribuição para o constante aprendizado da biblioteca *PySide6* bem como da implementação de lógicas para criação de *interface*.

Disponível em: https://github.com/Wanderson-Magalhaes/Modern_GUI_PyDracula_PySide6_or_PyQt6

2.5.3.2 “*Pomodoro*”, por Murak Martin

Software “inspiração” para a funcionalidade ”*Pomodoro”*. A implementação da funcionalidade, com temporizador, estados, tarefas e estatísticas, se deu através do estudo e compreensão majoritariamente baseados nesse repositório. Uma aplicação mais compacta e com menos diferenciais, contudo, que proporciona as noções mais básicas de estudo a partir dos ciclos de trabalho e descanso; imprescindível para a eficiência no desenvolvimento do projeto.

Disponível em: https://github.com/burakmartin/pomodoro

2.5.3.3 “*PyQt5 Daily Task Planner App*” por codefirstio.

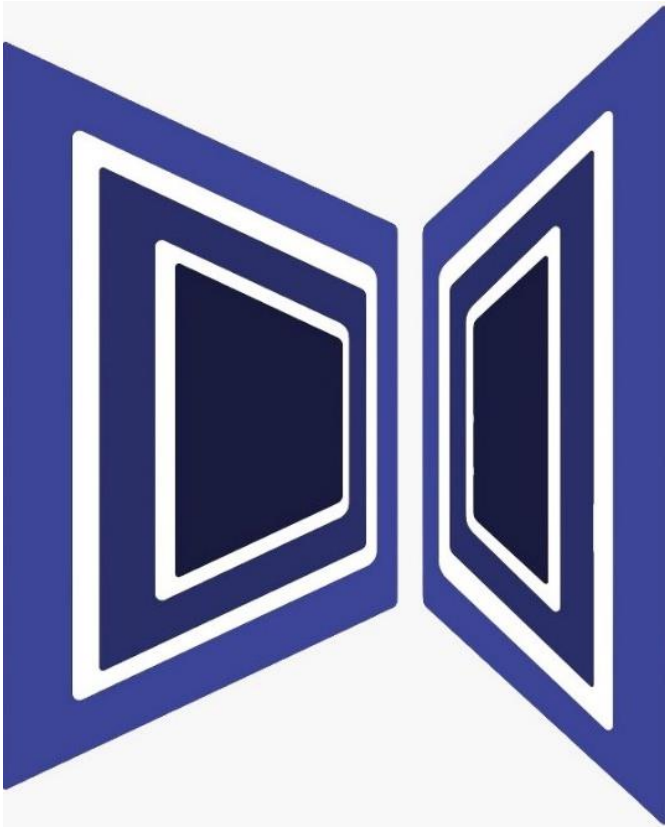
Projeto que se refere a um tutorial básico de como trabalhar com as tabelas da biblioteca *PyQt5*, bastante semelhante com a mecânica do *PySide6* aqui utilizado. Facilitou a compreensão tanto da parte lógica quanto estética de se trabalhar com as tabelas, componentes que foram predominantemente utilizados nas páginas do *software* para enriquecer a experiência do usuário com suas informações de estudo.

Disponível em: https://github.com/codefirstio/PyQt5-Daily-Task-Planner-App

2.6 *Design*, Telas e Códigos

2.6.1 Logotipo do Projeto

Figura 32 - Logotipo *SELF*



Fonte: do próprio autor, 2022. Construído por meio da aplicação *Web Canva.*

O logo do projeto foi pensado a partir do nome *SELF*. O nome, em sua essência, foi escolhido por representar uma introspecção positiva: o usuário ensinando a si mesmo o que desejasse. A partir deste conceito, a logotipo representa dois espelhos posicionados, um em frente ao outro, e deste modo, se refletindo infinitamente. O espelho é uma representação do usuário, que a partir da própria didática, consegue gerar uma quantidade infinita de conhecimentos, representado pelas reflexões. Foi utilizado da técnica minimalista para desenvolvimento de *design* do logo, de tal modo que não destoe

com o resto das aplicações que o usuário possua em seu *desktop*, permanecendo assim também, atemporal.

2.6.2 Cores utilizadas

SELF possui três temas de cores que transitam entre si conforme o usuário utiliza o programa. Cada tema escolhido possui uma graduação de cores a partir de sua cor principal.

Figura 33 - Cores Principais de cada tema

Fonte: do próprio autor, 2022. Construído por meio da aplicação *Krita*.

A primeira cor escolhida foi o roxo por ser sofisticada e convidativa; a primeira usada quando se abre o programa para que simbolize a junção de um estudo tradicional, por meios de organização tecnológica, de modo eficaz.

A segunda é o azul, por ser utilizada comumente como uma representação de confiança e profissionalismo, também consta em ambientes para maximizar a produtividade individual e é usada para captar a atenção e foco do usuário, quando assim se inicia sua jornada de estudos, e esse foco se vê necessário.

A terceira sendo o verde, usada no programa durante os intervalos entre os tempos de estudo focado, para que o usuário perceba o momento de descanso, e ao final, para que perceba que deve encerrar seus estudos para não se sobrecarregar. Escolhida por ser a cor que o olho humano menos precisa de adaptações para enxergar, e ser associada à tranquilidade de modo geral,

harmonia, para que o usuário seja capaz de sintetizar o conteúdo absorvido, e se mantenha calmo mesmo após uma grande rotina de estudos.[13]

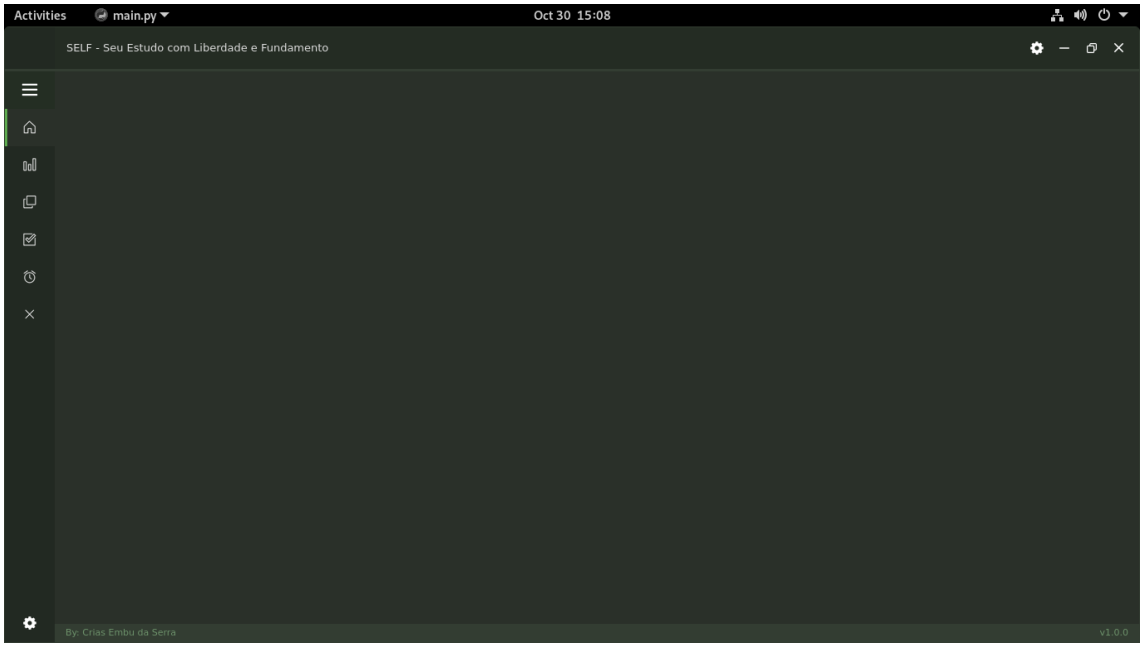
2.6.3 Telas e descrições

As imagens e descrições aqui colocadas correspondem fielmente ao funcionamento do programa tal qual o usuário o encontrará em seu uso, com exceção dos campos com informações adicionáveis que foram preenchidos com situações hipotéticas com fins meramente ilustrativos, não correspondendo às configurações padrões de primeira utilização.

Os *prints* foram tirados da versão final do programa rodando no sistema operacional *Debian 11* (*Bullseye*).

2.6.3.1 Tela principal

Figura 34- Tela Inicial da Aplicação

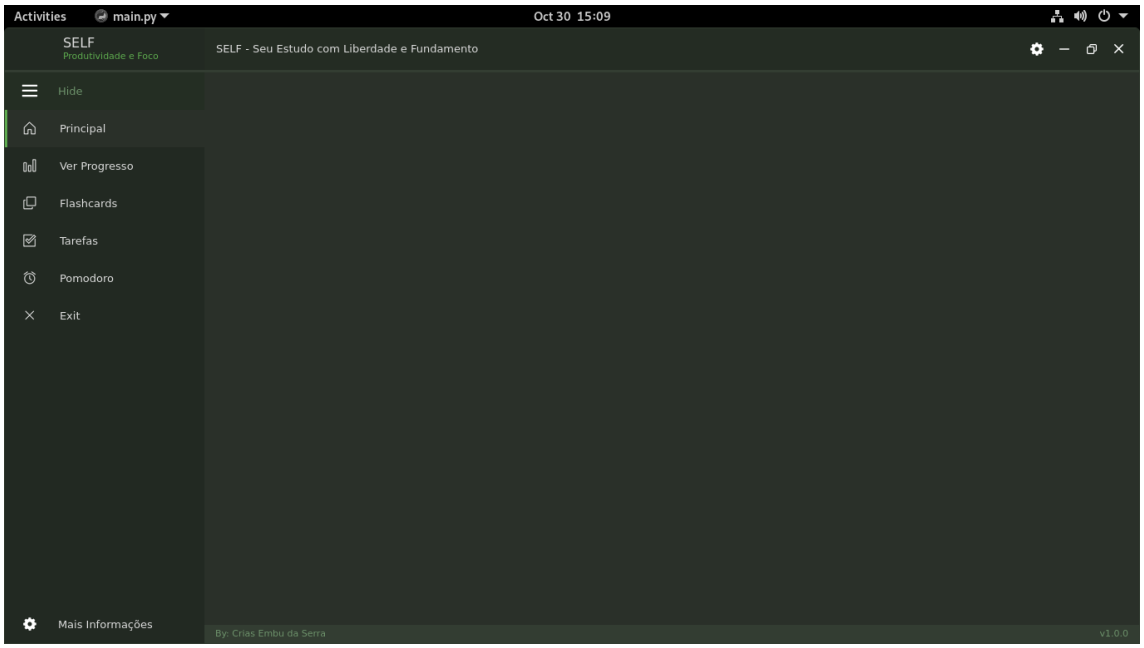


Fonte: do próprio autor, 2022

[13] Informações retiradas de “A Psicologia das Cores – Como as cores afetam a emoção e a razão”, de Eva Heller.

2.6.3.2 Tela principal ao expandir menu esquerdo

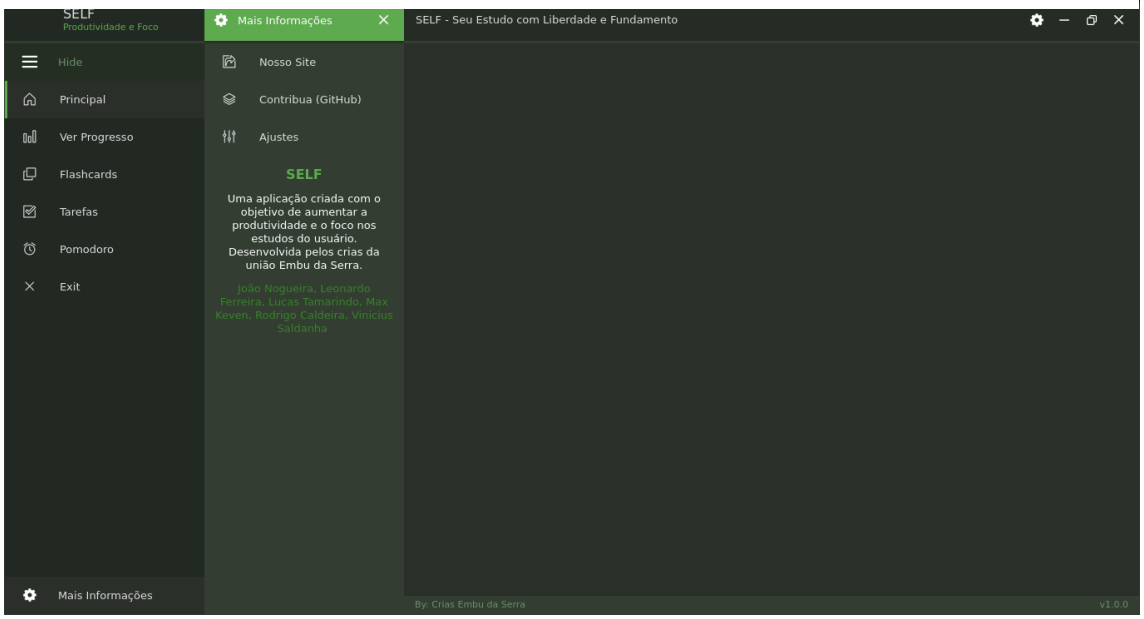
Figura 35 - Tela Inicial ao expandir menu esquerdo



Fonte: do próprio autor, 2022

2.6.3.3 Botão “Mais informações”

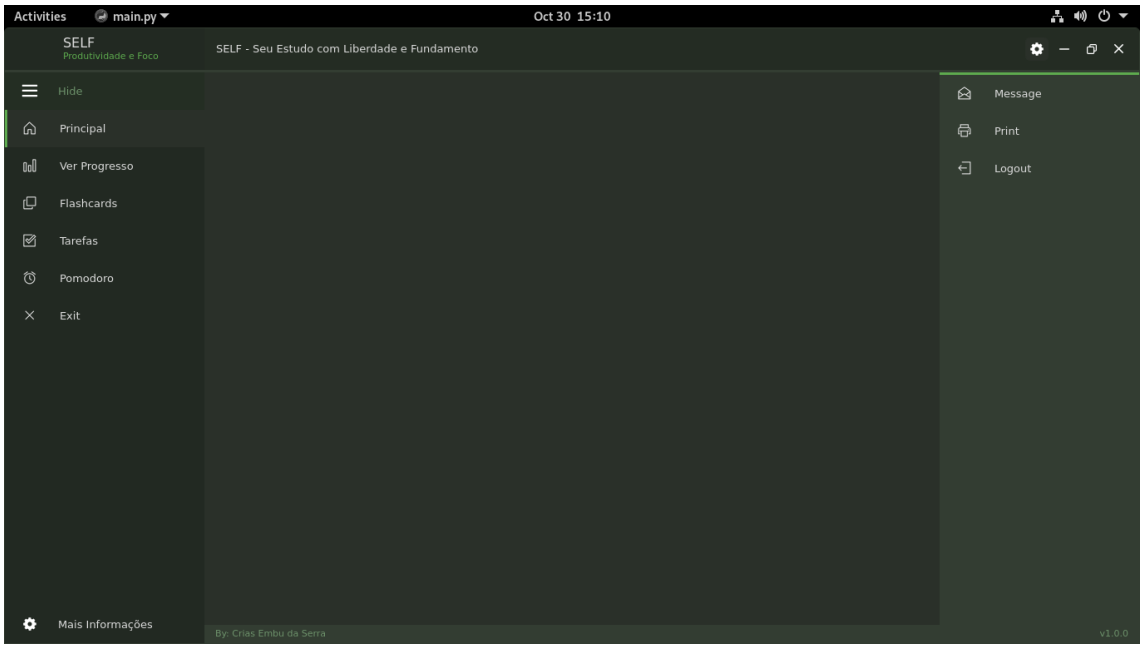
Figura 36 - Botão "Mais informações"



Fonte: do próprio autor, 2022

2.6.3.4 Botão “Configurações” canto superior direito

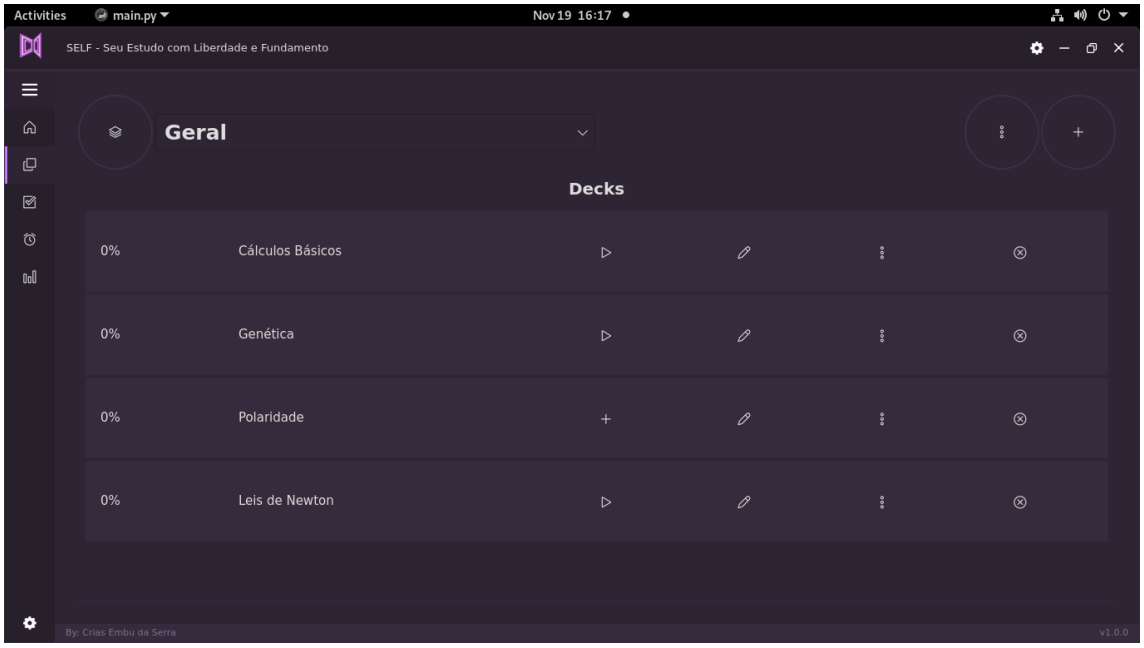
Figura 37 - Botão "Configurações" canto superior direito



Fonte: do próprio autor, 2022

2.6.3.5 Tela principal *Flashcards*

Figura 38- Tela Funcionalidade *Flashcards*



Fonte: do próprio autor, 2022

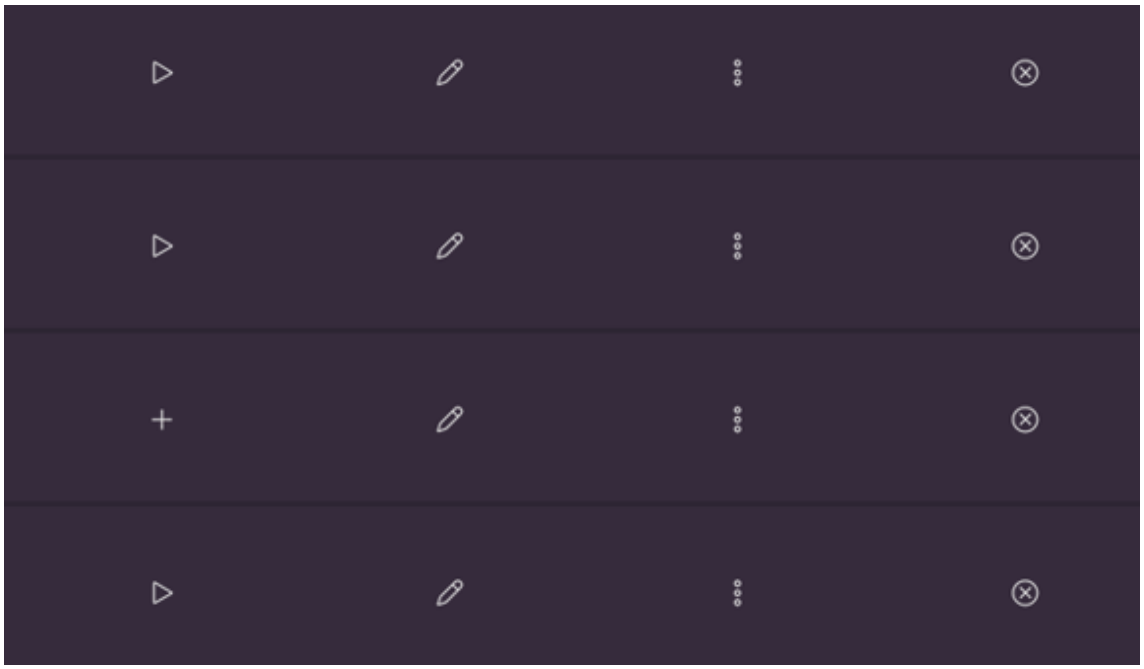
A tela principal da funcionalidade *Flashcards* dispõe por padrão uma visão geral dos *decks;* isto é, sem a separação por categorias.

Figura 39 Botão para alterar exibição dos Decks

Fonte: do próprio autor, 2022

O usuário pode alterar a exibição dos *decks* por meio do menu acima, onde são listadas as categorias dos *decks*.

Figura 40- Opções presentes nos *Decks*



Fonte: do próprio autor, 2022

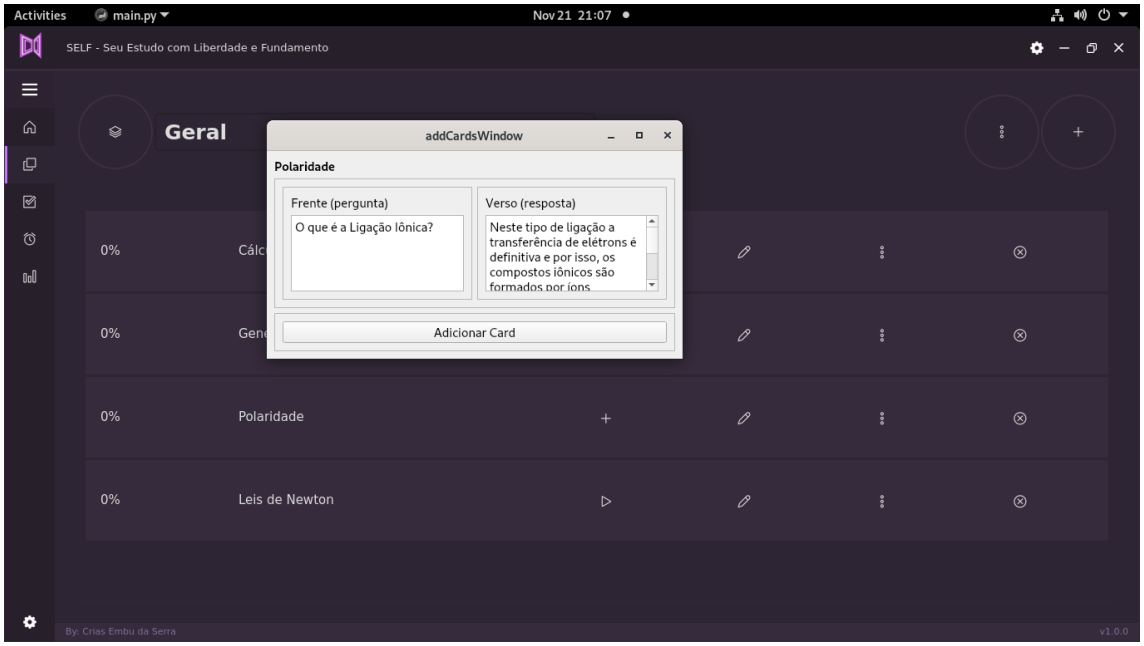
Os botões correspondem cada um deles a uma opção quanto ao *deck* de *Flashcards*. Respectivamente: 1) Para iniciar os estudos de um *deck*, com a variação do botão para criar *Flashcards* em um *deck* vazio; 2) Para editar os *Flashcards* de um *deck*; 3) para excluir um *deck*.

Figura 41 Botões configurações e adicionar novo *deck*

Fonte: do próprio autor, 2022

2.6.3.6 Botão “Adicionar *Cards*”

Figura 42- Botão "Adicionar *cards*"

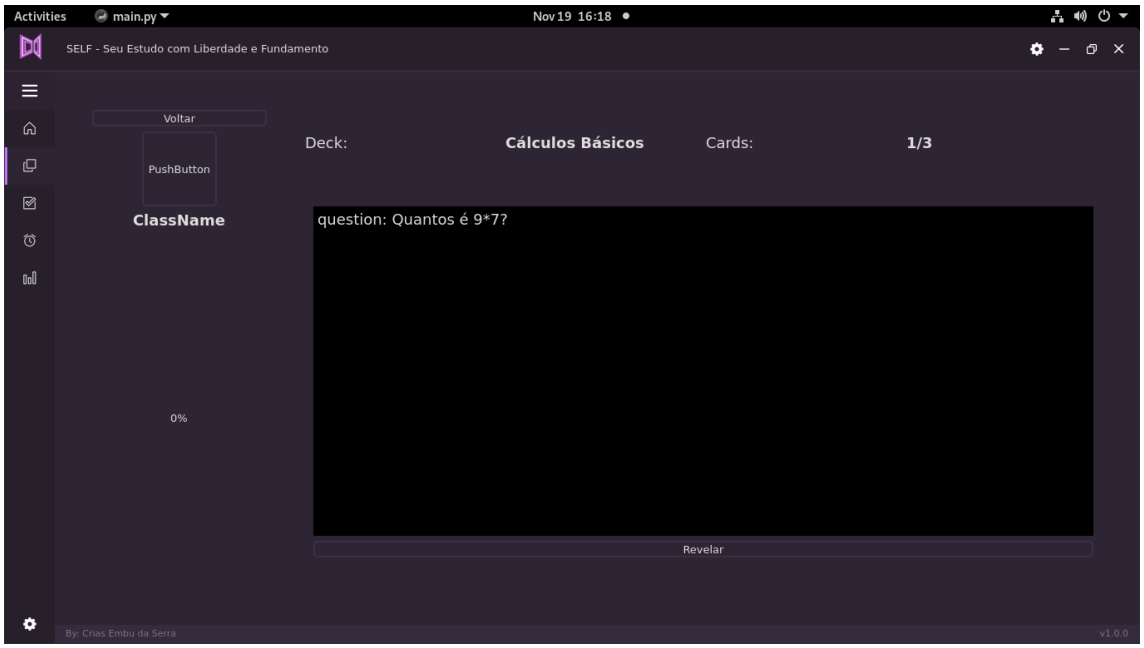


Fonte: do próprio autor, 2022

Ao clicar no botão “Adicionar *Cards*”, o usuário abre um *pop-up* onde poderá adicionar informações referente a pergunta e resposta do cartão.

2.6.3.7 Tela do botão “Iniciar”

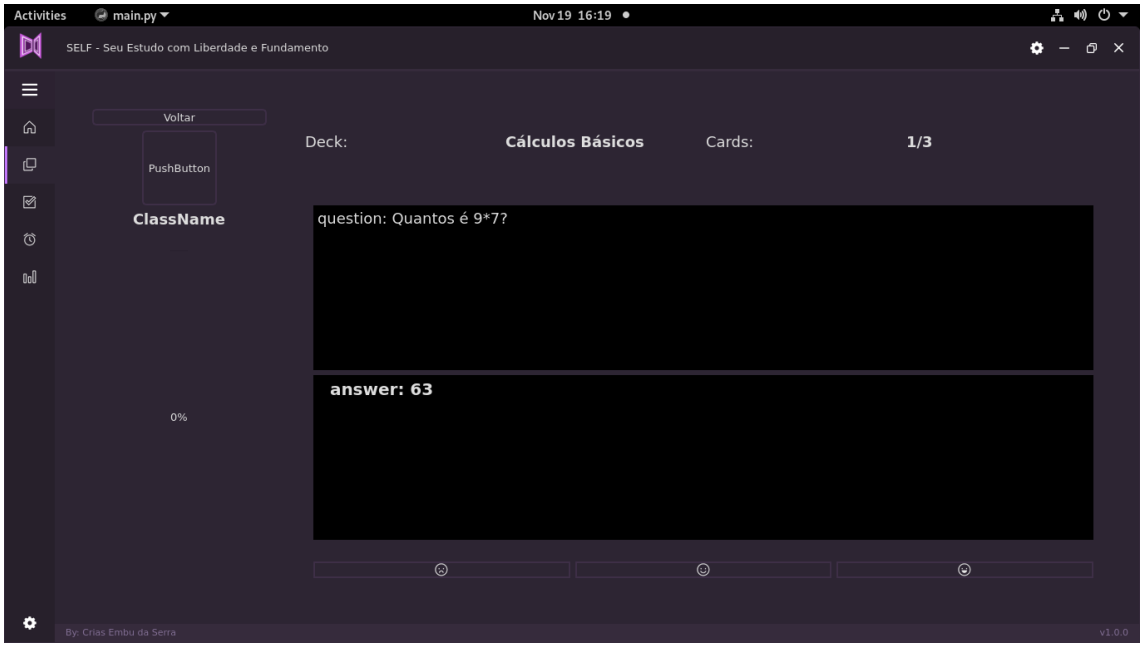
Figura 43 – Tela do botão "Iniciar"



Fonte: do próprio autor, 2022

A exibição do conteúdo do *Flashcards* é acompanhada do botão “Revelar” que ao clicado exibe o verso do cartão contendo a resposta.

Figura 44- Resposta do *Flashcards* e sistema de *feedback*



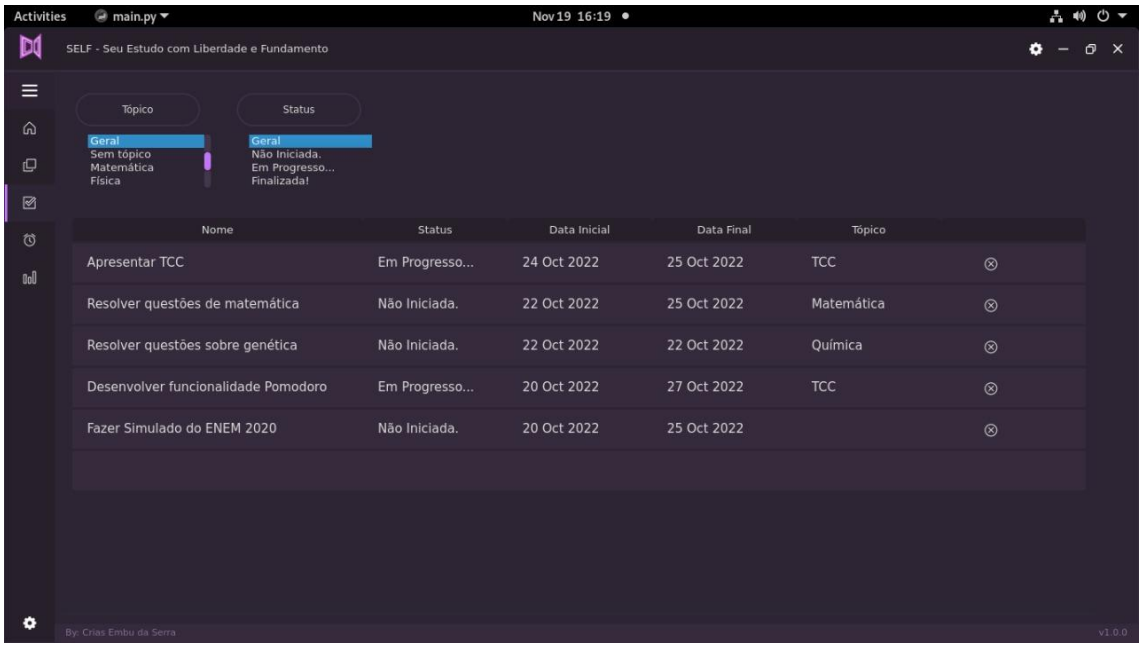
Fonte: do próprio autor, 2022

Abaixo da resposta há o sistema de *feedback* onde o usuário realizará uma autoavaliação correspondente ao grau de dificuldade de se recordar da resposta.

2.6.3.8 Tela da funcionalidade “Tarefas”

O acompanhador de tarefas permite que o usuário coloque seus afazeres de maneira visual em uma *interface* semelhante à de uma planilha, com integração a um sistema de calendário para que as distribua durante os dias de maneira eficiente. Por meio desta tela, o usuário pode inserir a data inicial e final a qual gostaria de terminá-la; junto de seu tópico, seu *status* atual e nome.

Figura 45- Tela funcionalidade "Tarefas"



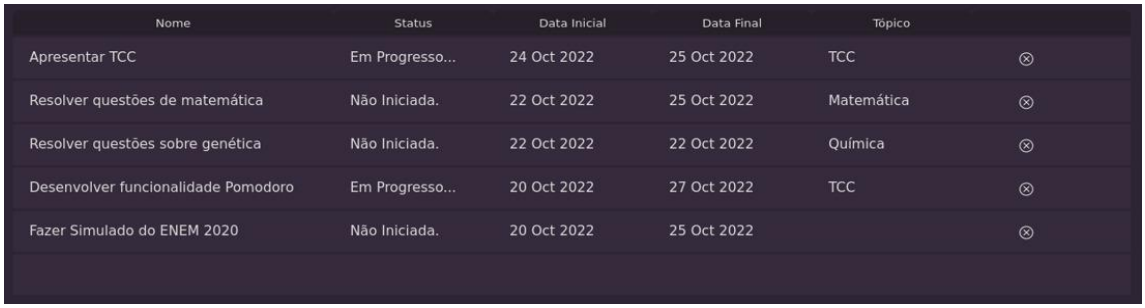
Fonte: do próprio autor, 2022

Figura 46- Botões para alterar exibição das tarefas

Fonte: do próprio autor, 2022

O usuário poderá alterar a exibição das tarefas por meio dos botões acima: Tópico; *Status*; onde o sistema será capaz de reunir as tarefas de acordo com a prioridade escolhida pelo próprio usuário, conforme mostra a imagem acima.

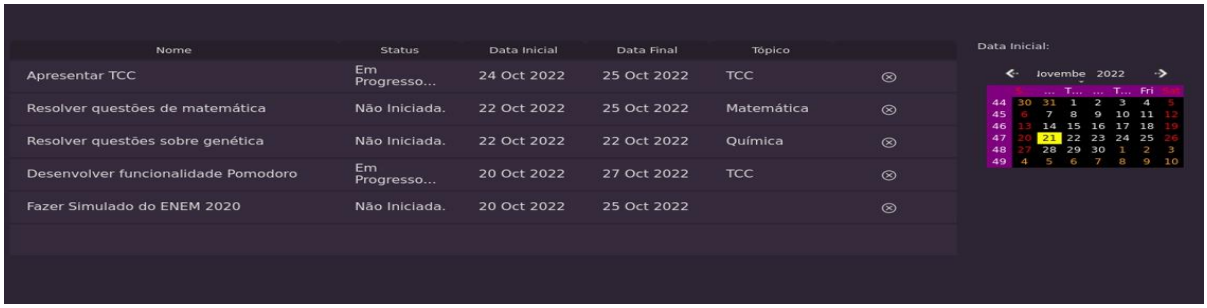
Figura 47 - Exibição de Tarefas



Fonte: do próprio autor, 2022

O usuário será capaz de criar uma tarefa ao clicar no espaço em branco e clicar em uma tarefa já existente para alterá-la.

Figura 48 - Exibição do calendário



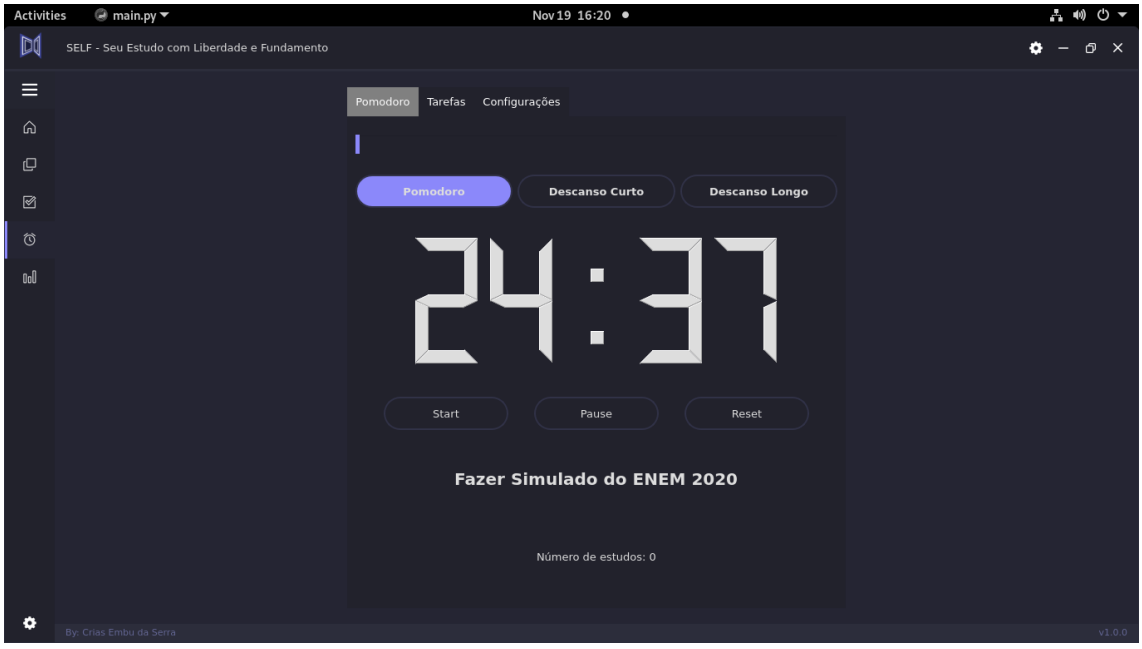
Fonte: do próprio autor, 2022

O usuário também poderá clicar na data dentro de uma tarefa, após isso, o sistema será capaz de exibir um calendário para o usuário.

2.6.3.9 Telas da funcionalidade “*Pomodoro*”

A funcionalidade *Pomodoro* permite que o usuário determine seu tempo total de estudo ativo e seus intervalos de descanso por meio de um *timer*, que pode ser iniciado, pausado e reiniciado. Sua finalidade é criar no usuário um mecanismo psicológico que o impelirá a se manter focado nas tarefas que se comprometeu a fazer sem que se sobrecarregue, mas divida-as em sessões.

Figura 49 - Tela inicial funcionalidade "*Pomodoro*"

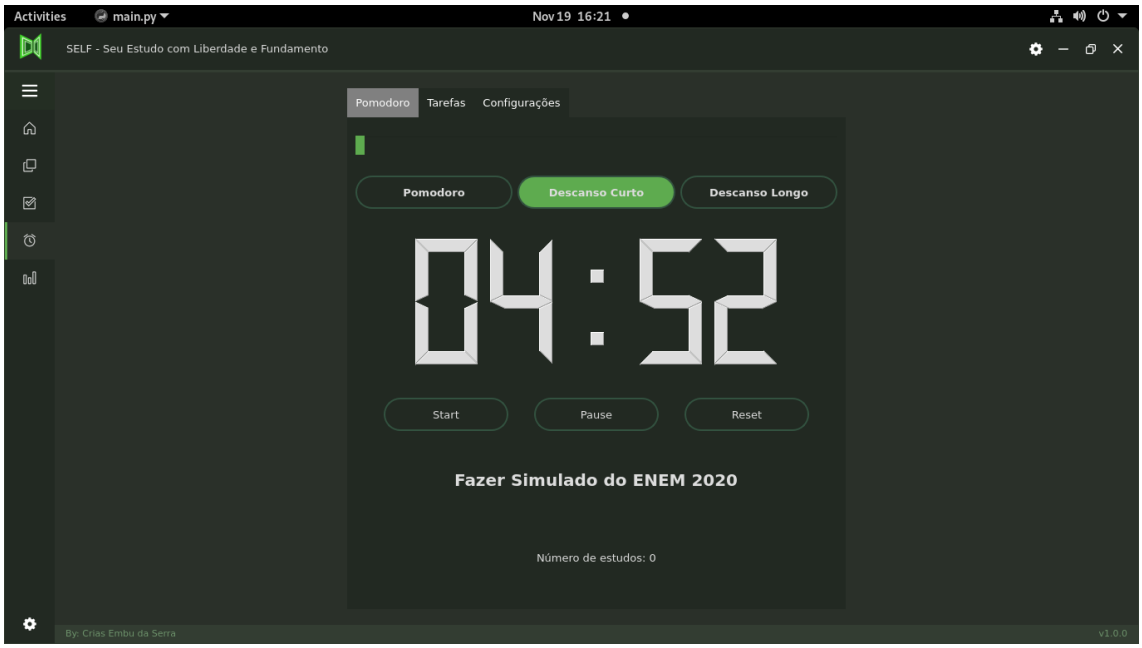


Fonte: do próprio autor, 2022

A imagem acima desmonstra a tela tela inicial da funcionalidade, com o modo “*Pomodoro”* ativado, implicando que o esquema de cores azul foi ativado. O usuário pode acompanhar a progressão do tempo de modo visual por meio de uma barra que indica o valor em porcentagem, definida de acordo com o tempo total previamente estabelecido.

2.6.3.10 Tela *Pomodoro* modo “Descanso Curto”

Figura 50 - Tela *Pomodoro* - Descanso curto

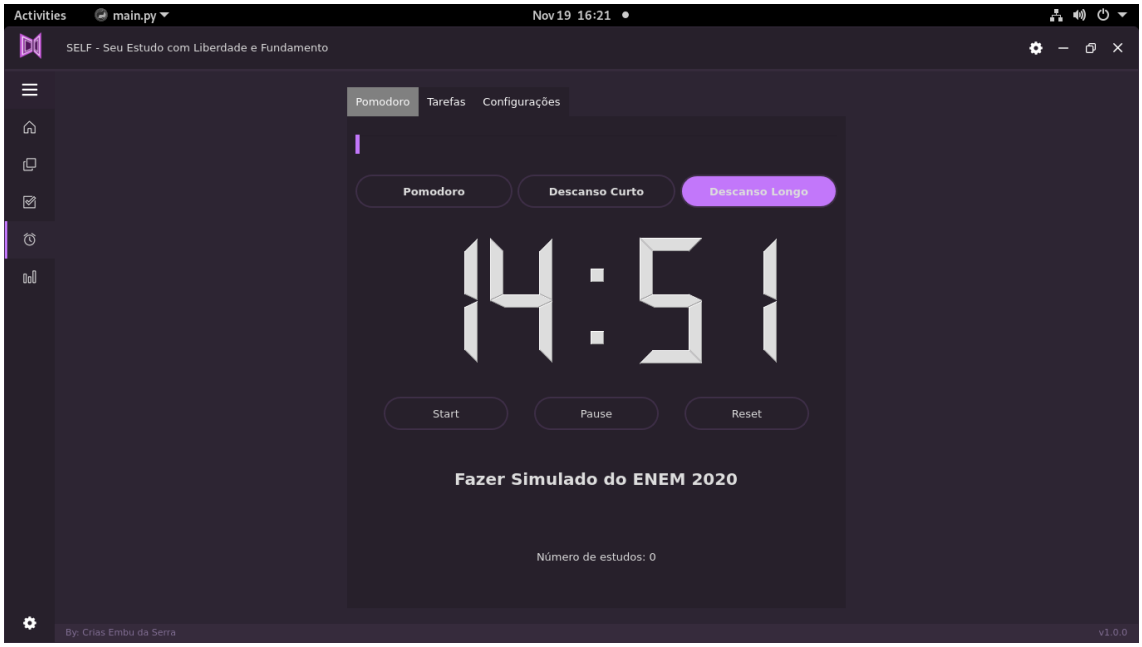


Fonte: do próprio autor, 2022.

A imagem acima demonstra o funcionamento do botão “Descanso Curto” que altera o timer e muda o esquema de cores para verde.

2.6.3.11 Tela *Pomodoro* modo “Descanso Longo”

Figura 51 - Tela *pomodoro* - modo Descanso Longo

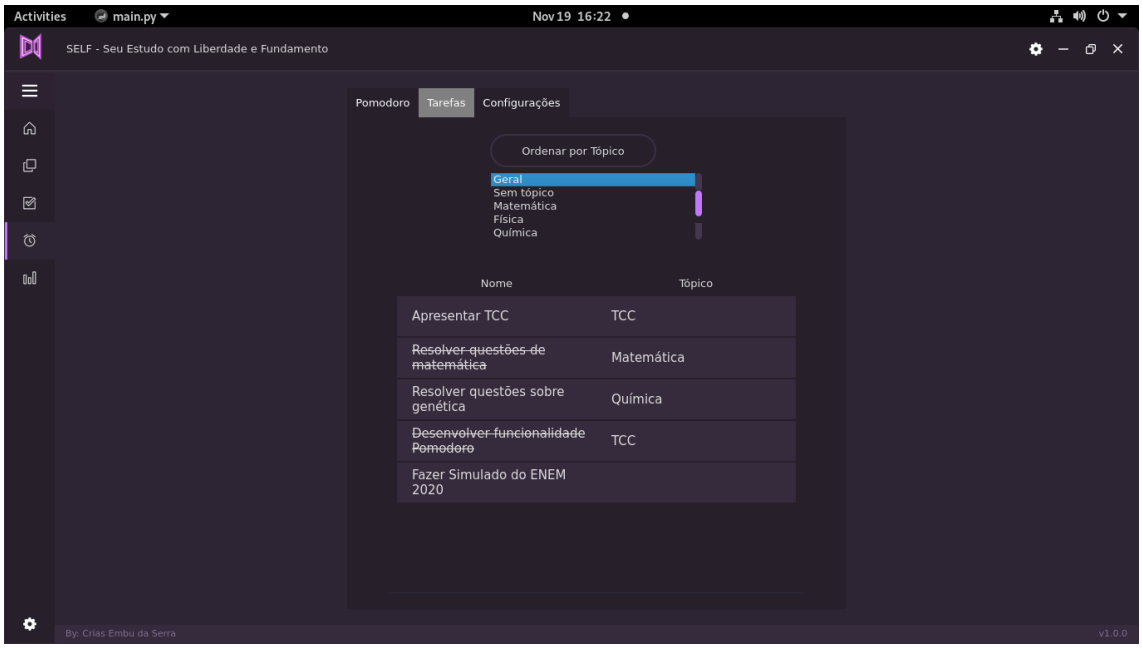


Fonte: do próprio autor, 2022

A imagem acima demonstra o funcionamento do botão “Descanso longo” que altera o timer e muda o esquema de cores para roxo.

2.6.3.12 Tela *Pomodoro* - aba “Tarefas”

Figura 52 - *Pomodoro* - aba "Tarefas"

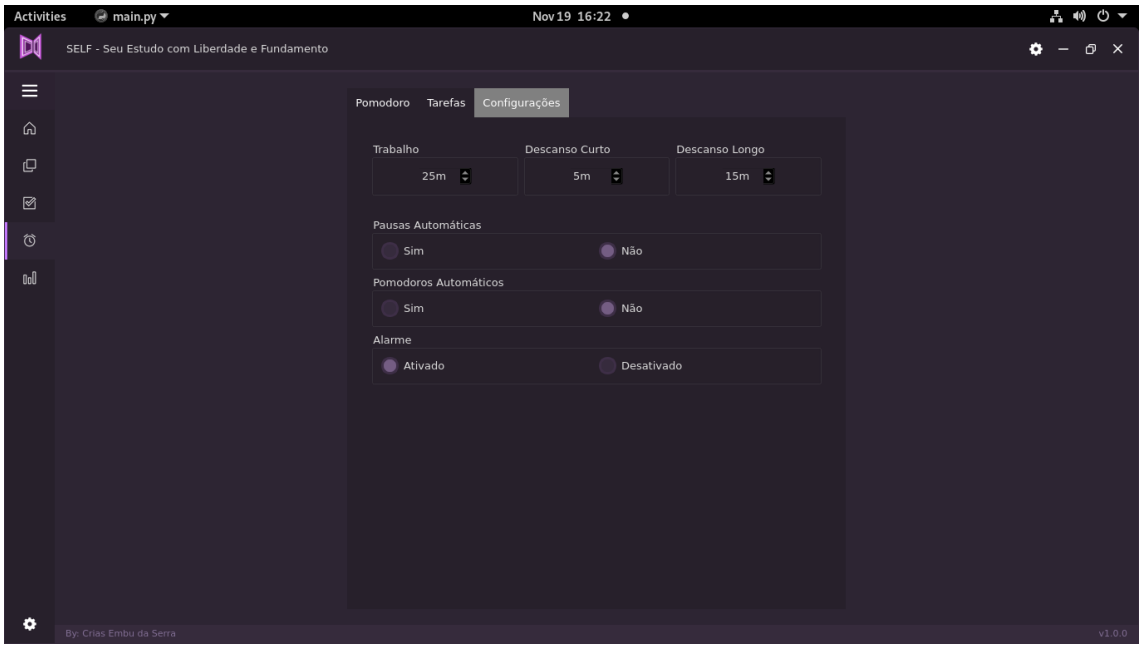


Fonte: do próprio autor, 2022.

A funcionalidade *Pomodoro* está associada à funcionalidade Tarefas, assim, o usuário pode atrelar o bloco de estudo a uma tarefa, fazendo com que ela seja marcada como “conluída” ao fim do ciclo. O usuário também terá disponível a opção de ordenar as tarefas por um tópico, assim como na tela “Tarefas”.

2.6.3.13 Tela *Pomodoro* – Aba “Configurações”

Figura 53 - Tela *Pomodoro* - Aba "Configurações"

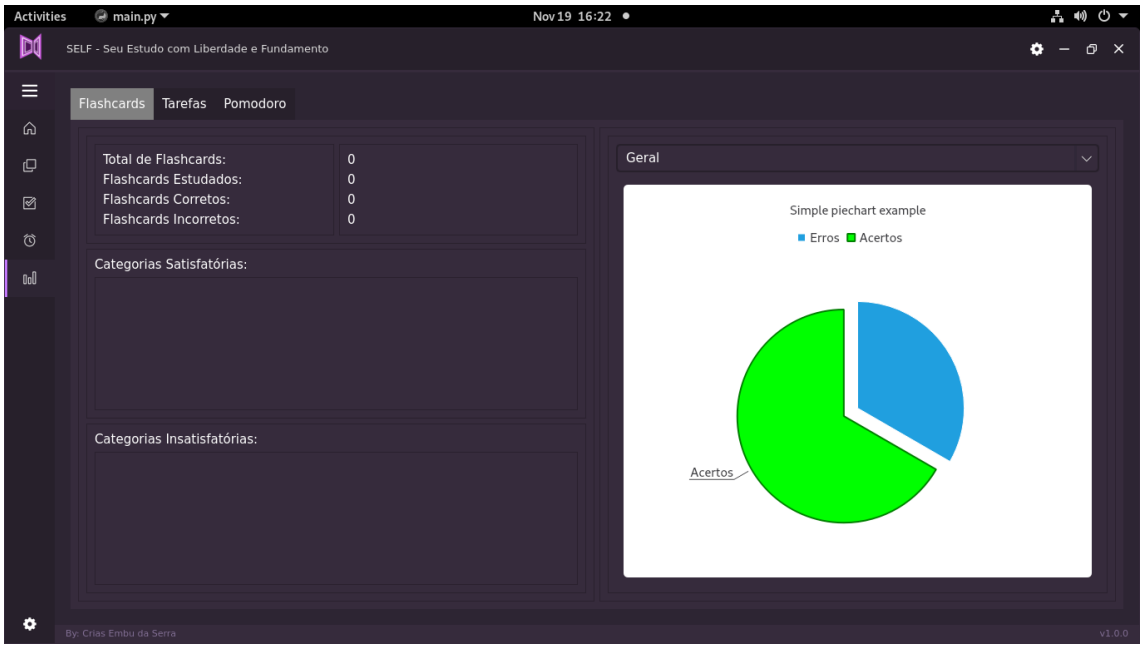


Fonte: do próprio autor, 2022

A tela de configurações permite que o usuário tenha total liberdade para definir o tempo total de cada bloco de estudo, o período de descanso curto e longo, podendo também definir se o sistema realizará paradas e início do *Pomodoro* automáticamente, com ou sem alarme.

2.6.3.14 Tela inicial da funcionalidade “Ver progresso” – Aba “*Flashcards*”

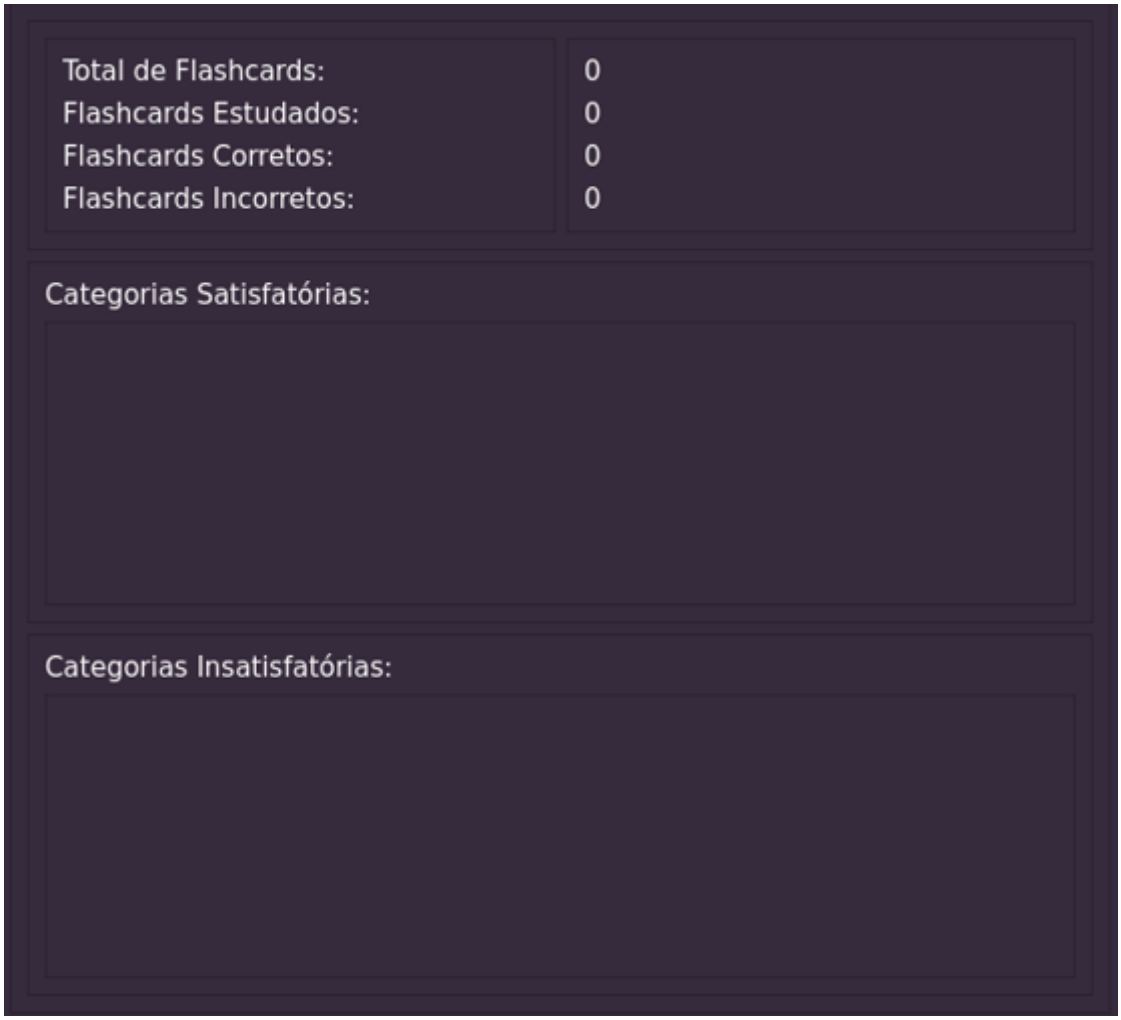
Figura 54 - Tela inicial da funcionalidade "Ver progresso" - aba “*Flashcards*”



Fonte: do próprio autor, 2022.

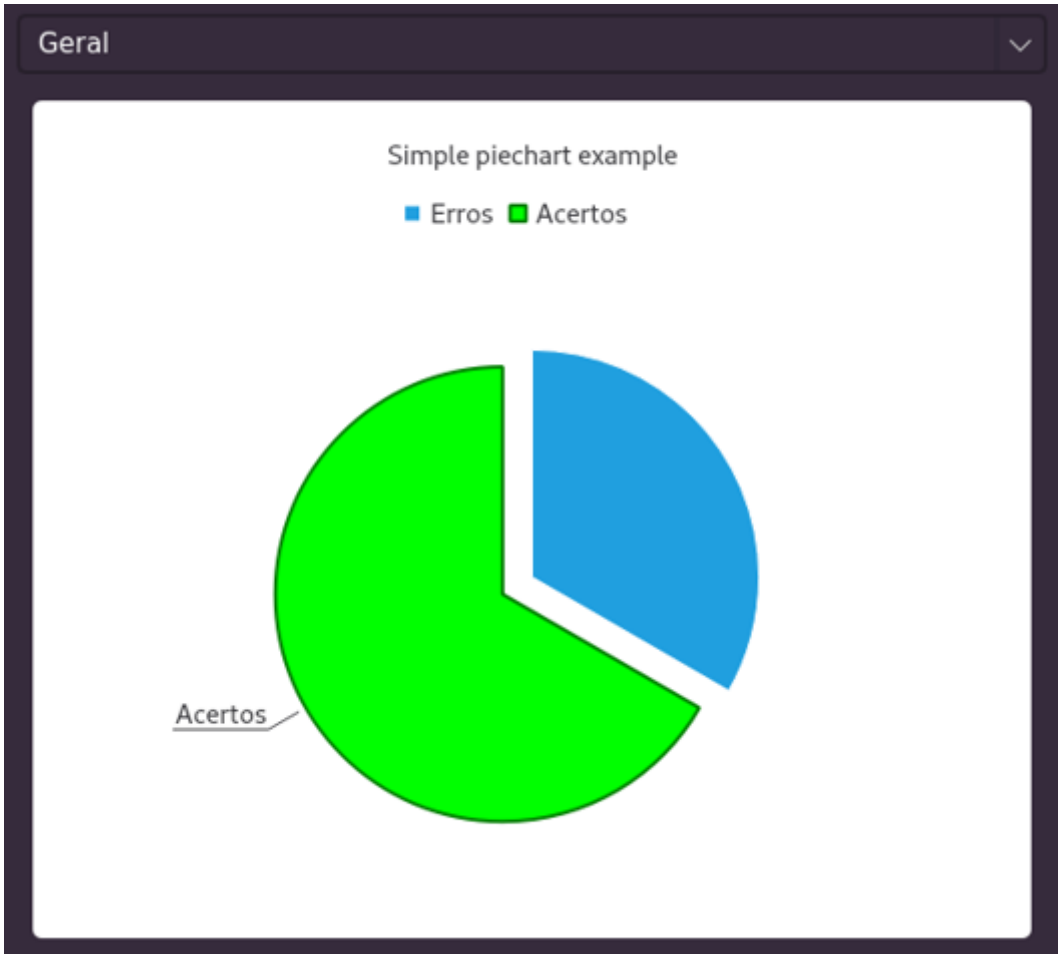
Visando a conscientização do estudante sobre seu nível de aprendizado através da aplicação *SELF*, o usuário ponderar medir, quantificar e acompanhar seu progresso de várias maneiras. Cada funcionalidade fornece um *feedback*. Desta forma, o sistema é capaz de se basear nesses *feedbacks* ao montar gráficos que serão exibidos ao usuário em uma tela específica para ver o seu progresso, oferecendo ao usuário uma visualização geral de seu desempenho, servindo como um sinal de que o estudante está progredindo ou que precisa mudar de rumo.

Figura 55 - Tabela com informações do usuário - "*Flashcards*"



Fonte: do próprio autor, 2022.

Figura 56 - Gráfico "*pizza*" com informações do usuário - *Flashcards*

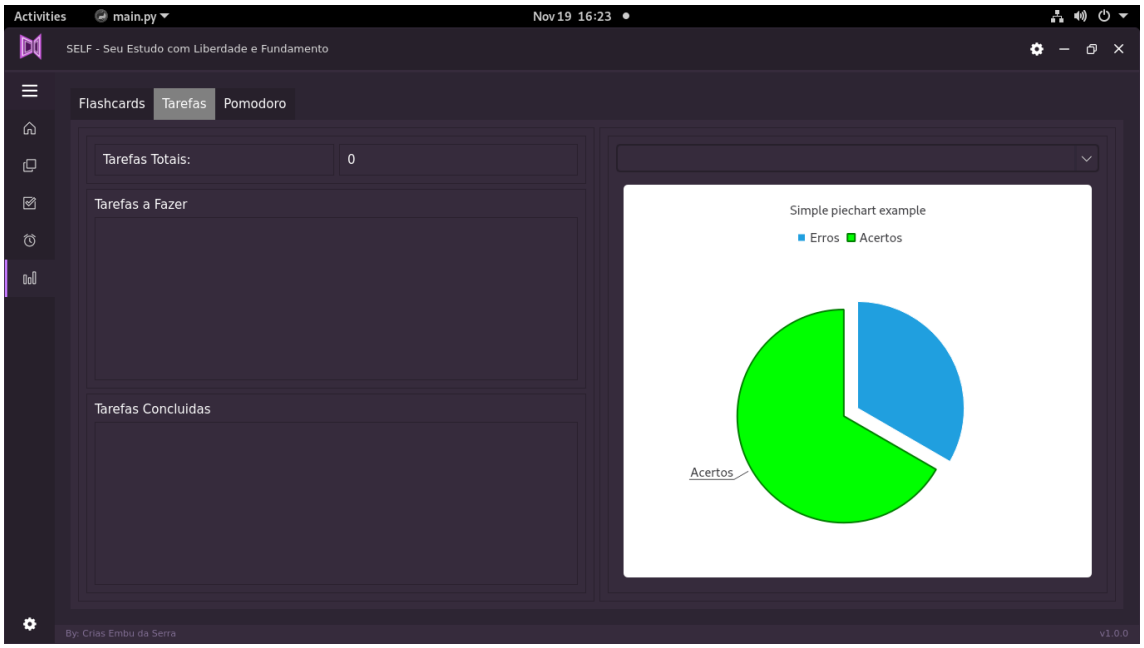


Fonte: do próprio autor, 2022.

Há também a exposição dos dados da aba em uma versão em gráfico para melhor visualização. Acima do gráfico o usuário pode alterar suas preferências de exibição para exibir alguma informação específica como as constadas na imagem anterior.

2.6.3.15 Tela funcionalidade “Ver progresso” – Aba “Tarefas”.

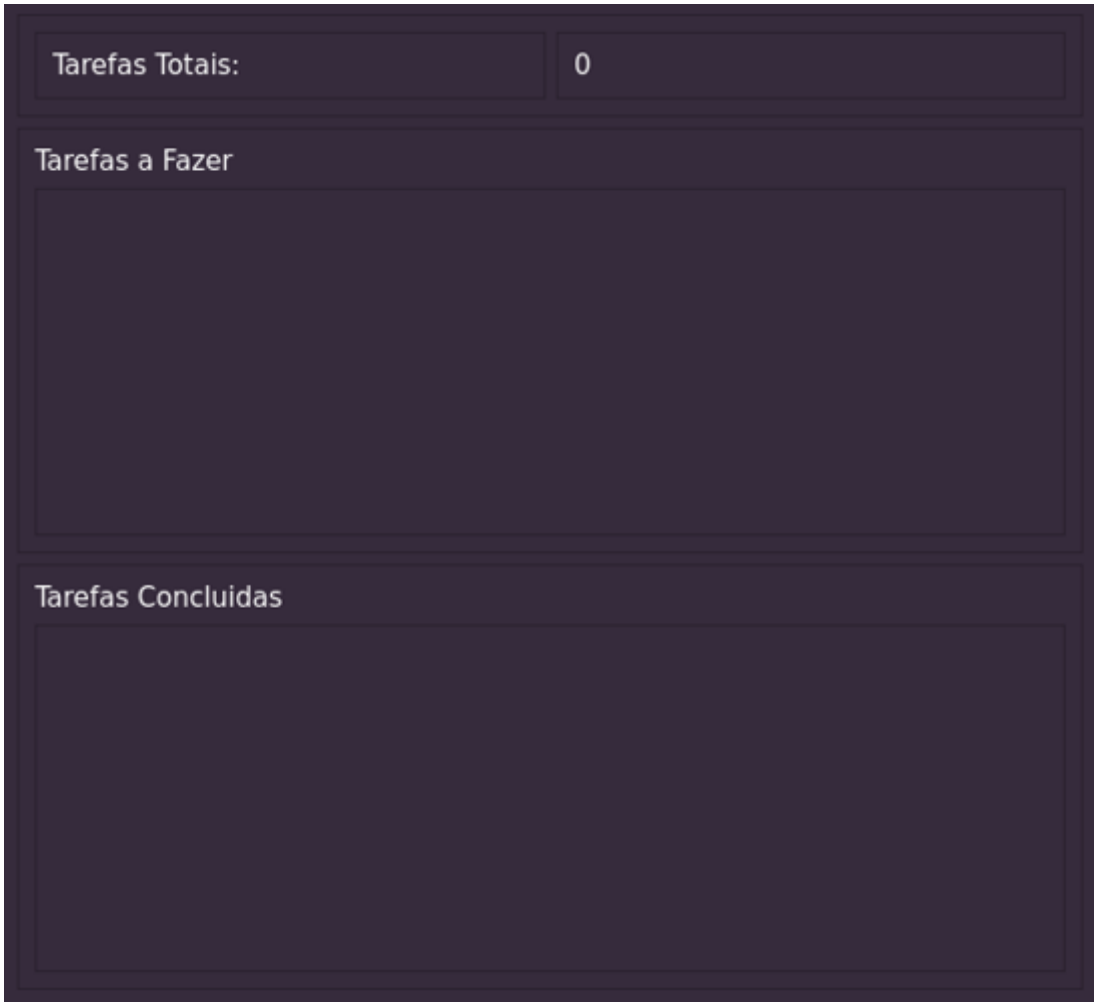
Figura 57 - Tela "Ver progresso" - Aba "Tarefas"



Fonte: do próprio autor, 2022.

Na tela acima há a aba contendo o resumo da funcionalidade “Tarefas”, onde são dispostas as informações do rendimento do usuário.

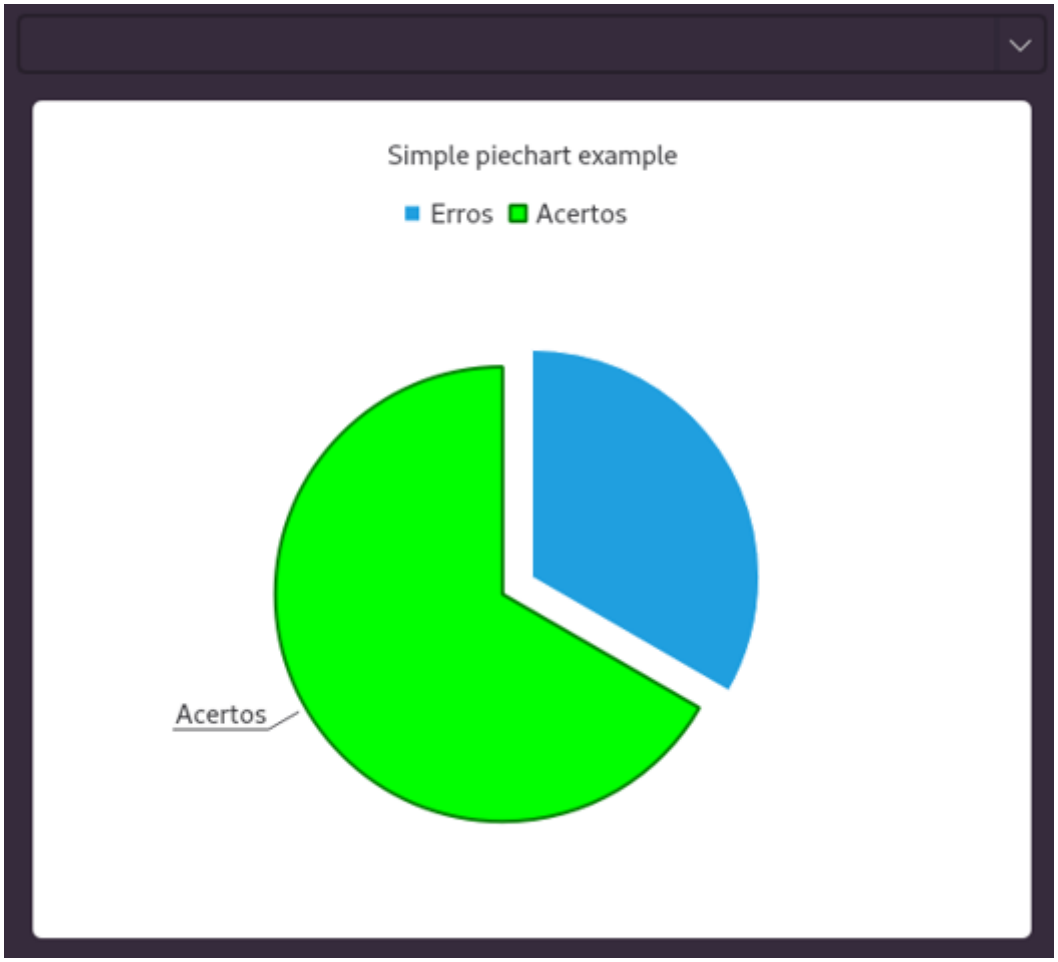
Figura 58 - Aba "Tarefas" com informações do usuário.



Fonte: do próprio autor, 2022

Nessa tela são exibidos os valores totais de tarefas pendentes e concluídas, coletados a partir da funcionalidade “Tarefas”.

Figura 59 – Gráfico “*Pizza*” – Aba “Tarefas”

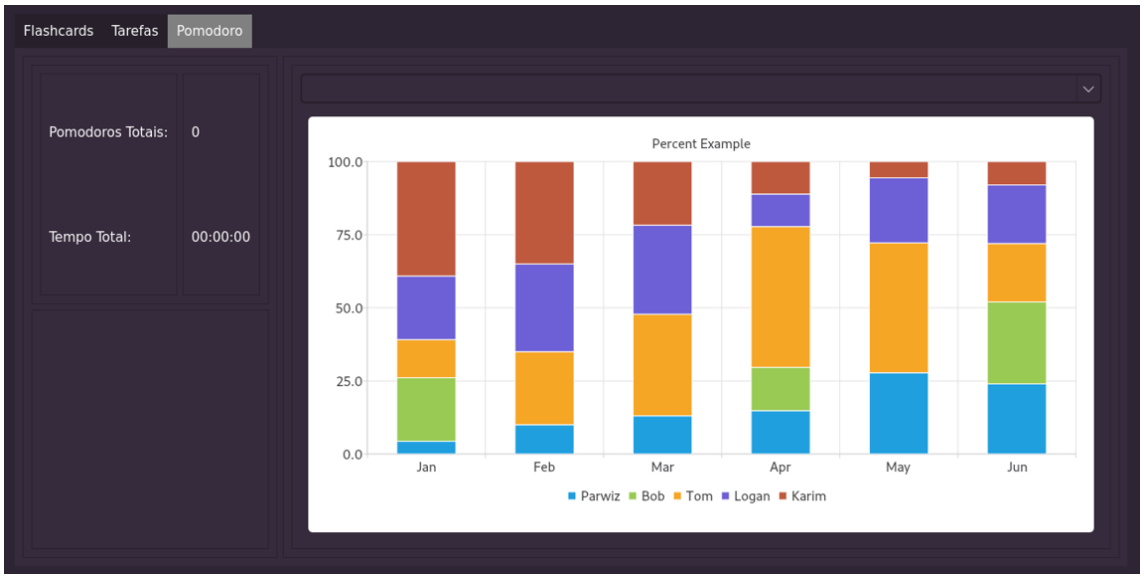


Fonte: do próprio autor, 2022.

Nessa tela é exibido um gráfico com as informações da imagem anterior. Não há a opção de alternar a exibição.

2.6.3.16 Tela Funcionalidade “Ver progresso” – Aba “*Pomodoro*”

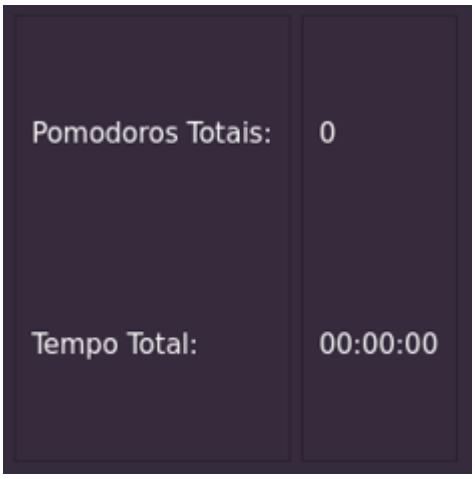
Figura 60 - "Ver progresso" - Aba "*Pomodoro*"



Fonte: do próprio autor, 2022

Nessa tela são exibidas informações coletadas a partir da funcionalidade “*Pomodoro*”.

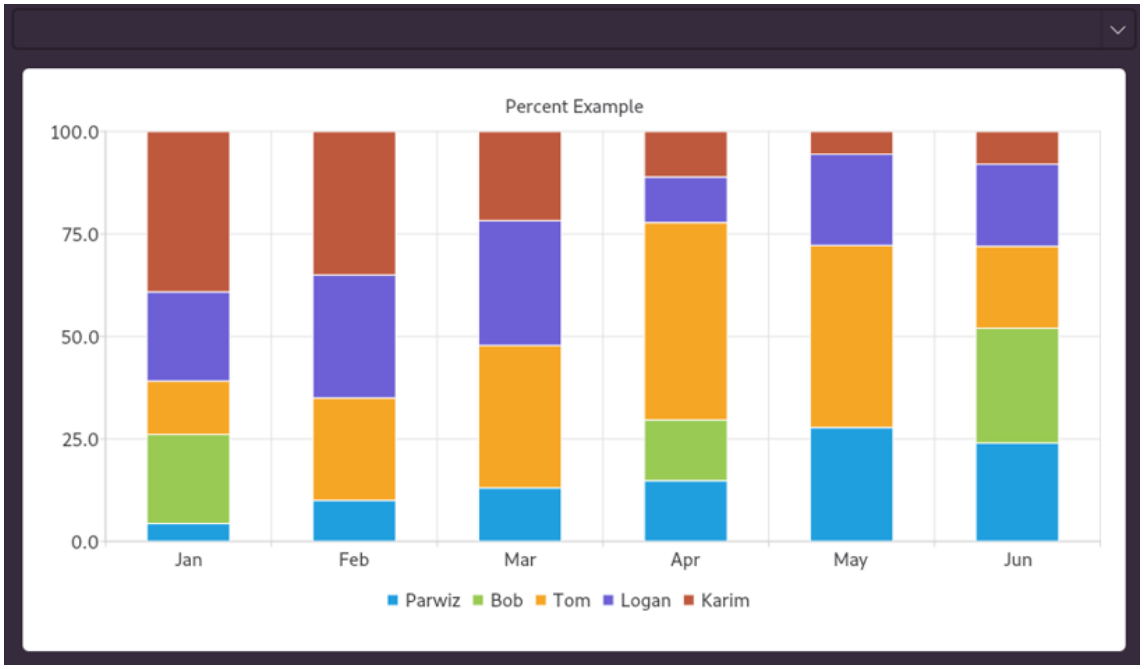
Figura 61 - aba "*Pomodoro*" informações exibidas ao usuário



Fonte: do próprio autor, 2022

Nessa tela são exibidos os dados referentes ao tempo total de estudo em modo *pomodoro* ativo e quantos ciclos completos foram realizados.

Figura 62 - Gráfico aba "*Pomodoro*"



Fonte: do próprio autor, 2022.

Nessa tela há também as informações da imagem anterior em forma de gráfico para melhor visualização do usuário.

2.6.4 Códigos do Programa

As prints de código a seguir representam apenas uma parte generalizada do arquivo oficial. Para tanto, em cada tópico há a disponibilidade de acesso ao link para melhor e significativa análise das lógicas implementadas.

Código principal (main.py). Disponível em:

https://github.com/Nbdyleto/PySide6_Self_TCC/blob/main/main.py

2.6.4.1 manipulação, alternância entre paleta de cores e navegação entre telas

Este código aborda tudo o que se refere a *interface* padrão, isto é, livre de funcionalidades. Possui conexões para que os botões do menu, ao serem clicados, carreguem e tornem visíveis a página respectiva; bem como a

alternância entre paletas de cores para a funcionalidade “*pomodoro”*, onde ocorre a lógica interna para que o programa mude as cores conforme essa função é executada.

Figura 63 - Código *interface* padrão

```
class MainWindow(QMainWindow):
    def __init__(self):

        QMainWindow.__init__(self)

        # SET AS GLOBAL WIDGETS
        # ///////////////////////////////////////////////////////////////
        self.ui = Ui_MainWindow()
        self.ui.setupUi(self)
        global widgets
        widgets = self.ui

        # USE CUSTOM TITLE BAR | USE AS "False" FOR MAC OR LINUX
        # ///////////////////////////////////////////////////////////////
        Settings.ENABLE_CUSTOM_TITLE_BAR = True

        # APP NAME
        # ///////////////////////////////////////////////////////////////
        title = "SELF - Produtividade e Foco"
        description = "SELF - Seu Estudo com Liberdade e Fundamento"
        # APPLY TEXTS
        self.setWindowTitle(title)
        widgets.titleRightInfo.setText(description)

        # TOGGLE MENU
        # ///////////////////////////////////////////////////////////////
        widgets.toggleButton.clicked.connect(lambda: UIFunctions.toggleMenu(self, True))
```

Fonte: do próprio autor, 2022.

Figura 64 - Código da *interface* padrão parte 2

```
# BUTTONS CLICK
# ////////////////////////////////////////////////////////////////

# LEFT MENUS
widgets.btn_home.clicked.connect(self.buttonClick)
widgets.btn_progress_page.clicked.connect(self.buttonClick)
widgets.btn_flashcards_page.clicked.connect(self.buttonClick)
widgets.btn_pomodoro_page.clicked.connect(self.buttonClick)
widgets.btn_daily_task_page.clicked.connect(self.buttonClick)

# PALLETE BUTTONS
widgets.btnPyJucoPurple.clicked.connect(self.changePallete)
widgets.btnPyJucoBlue.clicked.connect(self.changePallete)
widgets.btnPyJucoGreen.clicked.connect(self.changePallete)
widgets.newPomodoroPage.ui.btnPomodoro.clicked.connect(self.changePallete)
widgets.newPomodoroPage.ui.btnShortRest.clicked.connect(self.changePallete)
widgets.newPomodoroPage.ui.btnLongRest.clicked.connect(self.changePallete)
```

```
# BUTTONS CLICK
# Post here your functions for clicked buttons
# ////////////////////////////////////////////////////////////////
def buttonClick(self):
    # GET BUTTON CLICKED
    self.btnSelected = self.sender()
    btnName = self.btnSelected.objectName()

    # SHOW HOME PAGE
    if btnName == "btn_home":
        widgets.stackedWidget.setCurrentWidget(widgets.home)
        UIFunctions.resetStyle(self, btnName)
        self.btnSelected.setStyleSheet(UIFunctions.selectMenu(self.btnSelected.styleSheet()))

    # SHOW WIDGETS PAGE
    if btnName == "btn_progress_page":
        widgets.stackedWidget.setCurrentWidget(widgets.seeProgressPage)
        UIFunctions.resetStyle(self, btnName)
        self.btnSelected.setStyleSheet(UIFunctions.selectMenu(self.btnSelected.styleSheet()))

    # SHOW FLASHCARDS PAGE
    if btnName == "btn_flashcards_page":
        widgets.stackedWidget.setCurrentWidget(widgets.newFlashcardsPage) # SET PAGE
        UIFunctions.resetStyle(self, btnName) # RESET ANOTHERS BUTTONS SELECTED
        self.btnSelected.setStyleSheet(UIFunctions.selectMenu(self.btnSelected.styleSheet()))
        widgets.newFlashcardsPage.loadTopicsInComboBox()
```

Fonte: do próprio autor, 2022.

Figura 65- Código da *interface* padrão parte 3

```
        if btnName == "btn_pomodoro_page":
            widgets.stackedWidget.setCurrentWidget(widgets.newPomodoroPage) # SET PAGE
            UIFunctions.resetStyle(self, btnName) # RESET ANOTHERS BUTTONS SELECTED
            self.btnSelected.setStyleSheet(UIFunctions.selectMenu(self.btnSelected.styleSheet()))
            widgets.newPomodoroPage.loadDataInTable()

        # SHOW DAILY TASK PAGE
        if btnName == "btn_daily_task_page":
            widgets.stackedWidget.setCurrentWidget(widgets.newDailyTaskPage) # SET PAGE
            UIFunctions.resetStyle(self, btnName) # RESET ANOTHERS BUTTONS SELECTED
            self.btnSelected.setStyleSheet(UIFunctions.selectMenu(self.btnSelected.styleSheet()))
            widgets.newDailyTaskPage.loadDataInTable()
            widgets.newDailyTaskPage.hideAll()

        # PRINT BTN NAME
        print(f'Button "{btnName}" pressed!')
```

```
    def changePallete(self):
        if widgets.newPomodoroPage.allowChangeModeManually:

            btn = self.sender()
            btnName = btn.objectName()

            UIFunctions.resetButtonsStyle(self, widget=None, resetall=True) # RESET ALL MENU BUTTONS SELECTED
            UIFunctions.resetStyle(self, widget=None, resetall=True)    # RESET ALL POMODORO BUTTONS SELECTED

            if btnName == "btnPomodoro" or btnName == "btnPyJucoBlue":
                UIFunctions.theme(self, self.blueFile, True)
                AppFunctions.setThemeBlue(self)
            if btnName == "btnShortRest" or btnName == "btnPyJucoGreen":
                UIFunctions.theme(self, self.greenFile, True)
                AppFunctions.setThemeGreen(self)
            if btnName == "btnLongRest" or btnName == "btnPyJucoPurple":
                UIFunctions.theme(self, self.purpleFile, True)
                AppFunctions.setThemePurple(self)

if __name__ == "__main__":
    app = QApplication(sys.argv)
    app.setWindowIcon(QIcon("icon.ico"))
    window = MainWindow()
    sys.exit(app.exec_())
```

Fonte: do próprio autor, 2022.

2.6.4.2 *Interface* gráfica (*ui.main.py*)

Este arquivo contém a lógica principal para implementação da interface gráfica com componentes e layouts. Toda a parte visual do software é compreendida e implementada por este módulo, criado previamente pelo editor de interface gráfica Qt Creator, e adaptado posteriormente via programação manual.

Figura 66 - Código implementação de componentes e *layouts*

```
class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        if not MainWindow.objectName():
            MainWindow.setObjectName(u"MainWindow")
        MainWindow.resize(1280, 720)
        MainWindow.setMinimumSize(QSize(940, 560))
        self.styleSheet = QWidget(MainWindow)
        self.styleSheet.setObjectName(u"styleSheet")

        self.leftMenuFrame = QFrame(self.leftMenuBg)
        self.leftMenuFrame.setObjectName(u"leftMenuFrame")
        self.leftMenuFrame.setFrameShape(QFrame.NoFrame)
        self.leftMenuFrame.setFrameShadow(QFrame.Raised)
        self.verticalMenuLayout = QVBoxLayout(self.leftMenuFrame)
        self.verticalMenuLayout.setSpacing(0)
        self.verticalMenuLayout.setObjectName(u"verticalMenuLayout")
        self.verticalMenuLayout.setContentsMargins(0, 0, 0, 0)
        self.toggleBox = QFrame(self.leftMenuFrame)
        self.toggleBox.setObjectName(u"toggleBox")
        self.toggleBox.setMaximumSize(QSize(16777215, 45))
        self.toggleBox.setFrameShape(QFrame.NoFrame)
        self.toggleBox.setFrameShadow(QFrame.Raised)
        self.verticalLayout_4 = QVBoxLayout(self.toggleBox)
        self.verticalLayout_4.setSpacing(0)
        self.verticalLayout_4.setObjectName(u"verticalLayout_4")
        self.verticalLayout_4.setContentsMargins(0, 0, 0, 0)
        self.toggleButton = QPushButton(self.toggleBox)
        self.toggleButton.setObjectName(u"toggleButton")
        sizePolicy = QSizePolicy(QSizePolicy.Expanding, QSizePolicy.Fixed)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(self.toggleButton.sizePolicy().hasHeightForWidth())
        self.toggleButton.setSizePolicy(sizePolicy)
        self.toggleButton.setMinimumSize(QSize(0, 45))
        self.toggleButton.setFont(font)
        self.toggleButton.setCursor(QCursor(Qt.PointingHandCursor))
        self.toggleButton.setLayoutDirection(Qt.LeftToRight)
        self.toggleButton.setStyleSheet(u"background-image: url(:/icons/images/icons/icon_menu.png);")

        self.verticalLayout_4.addWidget(self.toggleButton)
```

Fonte: do próprio autor, 2022.

Figura 67- Código implementação de componentes e *layouts* parte 2

```
    def retranslateUi(self, MainWindow):
        MainWindow.setWindowTitle(QCoreApplication.translate("MainWindow", u"MainWindow", None))
        self.titleLeftApp.setText(QCoreApplication.translate("MainWindow", u"SELF", None))
        self.titleLeftDescription.setText(QCoreApplication.translate("MainWindow", u"Produtividade e Foco", Non
        self.toggleButton.setText(QCoreApplication.translate("MainWindow", u"Hide", None))
        self.btn_home.setText(QCoreApplication.translate("MainWindow", u"Principal", None))
        self.btn_home.setToolTip(QCoreApplication.translate("MainWindow", u"Principal", None))
        self.btn_progress_page.setText(QCoreApplication.translate("MainWindow", u"Ver Progresso", None))
        self.btn_progress_page.setToolTip(QCoreApplication.translate("MainWindow", u"Ver Progresso", None))
        self.btn_flashcards_page.setText(QCoreApplication.translate("MainWindow", u"Flashcards", None))
        self.btn_flashcards_page.setToolTip(QCoreApplication.translate("MainWindow", u"Flashcards", None))
        self.btn_pomodoro_page.setText(QCoreApplication.translate("MainWindow", u"Pomodoro", None))
        self.btn_pomodoro_page.setToolTip(QCoreApplication.translate("MainWindow", u"Pomodoro", None))
        self.btn_daily_task_page.setText(QCoreApplication.translate("MainWindow", u"Tarefas", None))
        self.btn_daily_task_page.setToolTip(QCoreApplication.translate("MainWindow", u"Tarefas", None))
        self.toggleLeftBox.setText(QCoreApplication.translate("MainWindow", u"Mais Informações", None))
```

Fonte: do próprio autor, 2022.

2.6.4.3 Operações com Banco de Dados (*functions/db_main_operations.py*)

As operações de banco de dados do software passam diretamente pelo uso da classe “*DBMainOperations*()” contida neste arquivo; contendo a responsabilidade pela manipulação das práticas de *CRUD* (*create, read, update e delete*).

São algumas das operações destes blocos de código: criação das tabelas para tópicos, decks, flashcards e tarefas; verificação se existe ou não registros para estas tabelas; obtenção de todas as informações solicitadas; contagem de registros.

Possui ainda funções para lidar com o início, entrada e saída de conexões de um banco de dados, proporcionando a legibilidade e eficiência quanto a essas práticas. Disponível em:

https://github.com/Nbdyleto/PySide6_Self_TCC/blob/main/functions/db_main_operations.py

Figura 68 - Códigos das operações com banco de dados

```
class DBMainOperations:
    __DB_LOCATION = 'functions/db_main_operations.db'

    def __init__(self):
        self.conn = sqlite3.connect(DBMainOperations.__DB_LOCATION)
        self.cursor = self.conn.cursor()

    def __enter__(self):
        return self

    def __exit__(self, ext_type, exc_value, traceback):
        self.cursor.close()
        if isinstance(exc_value, Exception):
            self.conn.rollback()
        else:
            self.conn.commit()
        self.conn.close()
```

Fonte: do próprio autor, 2022.

Figura 69- Códigos das operações com banco de dados parte 2

```
def createTblTopics(self):
    # Parent Table
    self.cursor.execute("DROP TABLE IF EXISTS topics")
    qry_topics = """ CREATE TABLE topics (
        topic_id INTEGUER PRIMARY KEY,
        topic_name VARCHAR(50) NOT NULL
    );"""
    self.cursor.execute(qry_topics)
    print('table topics is ready!')

##########################
# Flashcards. DB Functions

def createTblDecks(self):
    # Parent Table of flashcards, Child Table of topics.
    self.cursor.execute("DROP TABLE IF EXISTS decks")
    qry_decks = """ CREATE TABLE decks (
        deck_id INTEGUER PRIMARY KEY,
        deck_name VARCHAR(50),
        hits_percentage INTEGUER NOT NULL,
        topic_id INTEGUER NOT NULL,
        FOREIGN KEY (topic_id)
            REFERENCES topics (topic_id)
    );"""
    self.cursor.execute(qry_decks)
    print('table decks is ready!')
```

```
def createTblFlashcards(self):
    # Child Table
    self.cursor.execute("DROP TABLE IF EXISTS flashcards")
    qry_flashcards = """ CREATE TABLE flashcards (
        card_id INTEGUER PRIMARY KEY,
        card_question VARCHAR(255) NOT NULL,
        card_answer VARCHAR(255) NOT NULL,
        deck_id INTEGUER NOT NULL,
        FOREIGN KEY (deck_id)
            REFERENCES topics (deck_id)
        );"""
    self.cursor.execute(qry_flashcards)
    print('table flashcards is ready!')

def hasRecordsInTblFlashcards(self, id):
    qry = f"SELECT COUNT(*) FROM flashcards WHERE (deck_id = ?)"
    recordscount = self.cursor.execute(qry, str(id)).fetchall()[0][0]
    if recordscount > 0:
        return True
    return False
```

Fonte: do próprio autor, 2022.

Figura 70 - Códigos das operações com banco de dados parte 3

```
#########################
# Daily Task. DB Functions

def createTblTasks(self):
    # Child Table
    self.cursor.execute("DROP TABLE IF EXISTS tasks")
    qry_tasks = """ CREATE TABLE tasks (
        task_id INTEGUER PRIMARY KEY,
        task_name VARCHAR(255) NOT NULL,
        status VARCHAR(25) NOT NULL,
        start_date DATE NOT NULL,
        end_date DATE NOT NULL,
        topic_id INTEGUER NOT NULL,
        FOREIGN KEY (topic_id)
            REFERENCES topics (topic_id)
    );"""
    self.cursor.execute(qry_tasks)
    print('table tasks is ready!')

########################
# General. DB Functions

def populateTbl(self, tbl, params):
    qry = f"INSERT INTO {tbl} VALUES {params};"
    self.cursor.execute(qry)
    self.conn.commit()
```

```
def getRowCount(self, tbl, whclause=None):
    if whclause is None:
        return self.cursor.execute(f"SELECT COUNT(*) FROM {tbl}").fetchone()[0]
    return self.cursor.execute(f"SELECT COUNT(*) FROM {tbl} WHERE ({whclause})").fetchone()[0]

def getAllRecords(self, tbl, specifcols='*', fetchall=True, whclause = None):
    if whclause is None:
        self.cursor.execute(f"SELECT {specifcols} FROM {tbl}")
    elif whclause is not None:
        self.cursor.execute(f"SELECT {specifcols} FROM {tbl} WHERE ({whclause})")
    if fetchall:
        return self.cursor.fetchall()
    return self.cursor
```

Fonte: do próprio autor, 2022.

2.6.4.4 Funções do *App (modules/app_functions.py)*

Este Arquivo possui a finalidade de alterar a paleta de componentes mais complexos como textos da aba “mais informações” e configurações estéticas para quando um botão é selecionado nos menus. Disponível em: https://github.com/Nbdyleto/PySide6_Self_TCC/blob/main/modules/app_functions.py

Figura 71 - Código funções do *App*

```
class AppFunctions(MainWindow):
    def setThemeBlue(self):
        Settings.BTN_LEFT_BOX_COLOR = "background-color: rgb(37, 37, 51);"
        Settings.BTN_RIGHT_BOX_COLOR = "background-color: rgb(37, 37, 51);"
        Settings.MENU_SELECTED_STYLESHEET = MENU_SELECTED_STYLESHEET = """
        border-left: 22px solid qlineargradient(spread:pad, x1:0.034, y1:0, x2:0.216, y2:0, stop:0.499 rgb(139,
        background-color: rgb(37, 37, 51);
        """
        Settings.POMODORO_SELECTED_STYLESHEET = POMODORO_SELECTED_STYLESHEET = """
        background-color: rgb(139, 136, 250);
        """

        self.ui.textEdit.setHtml(QCoreApplication.translate("MainWindow", u"<!DOCTYPE HTML PUBLIC \"-//W3C//DTD
"<html><head><meta name=\"qrichtext\" content=\"1\" /><meta charset=\"utf-8\" /><style type=\"text/css\">\n"
"p, li { white-space: pre-wrap; }\n"
"</style></head><body style=\" font-family:'Segoe UI'; font-size:10pt; font-weight:400; font-style:normal;\">\n"
"<p align=\"center\" style=\" margin-top:12px; margin-bottom:12px; -qt-block-indent:0; text-indent:0px;\"><span
"<p align=\"center\" style=\" margin-top:12px; margin-bottom:12px; -qt-block-indent:0; text-indent:0px;\"><span
"<p align=\"center\" style=\" margin-top:12px; margin-bottom:12px; -qt-block-indent:0; text-indent:0px;\"><span
```

Fonte: do próprio autor, 2022.

Figura 72 - Código funções do *App* parte 2

```
    def setThemePurple(self):
        Settings.BTN_LEFT_BOX_COLOR = "background-color: rgb(45, 37, 51);"
        Settings.BTN_RIGHT_BOX_COLOR = "background-color: rgb(45, 37, 51);"
        Settings.MENU_SELECTED_STYLESHEET = MENU_SELECTED_STYLESHEET = """
        border-left: 22px solid qlineargradient(spread:pad, x1:0.034, y1:0, x2:0.216, y2:0, stop:0.499 rgb(194,
        background-color: rgb(45, 37, 51);
        """
        Settings.POMODORO_SELECTED_STYLESHEET = POMODORO_SELECTED_STYLESHEET = """
        background-color: rgb(194, 119, 250);
        """

        self.ui.textEdit.setHtml(QCoreApplication.translate("MainWindow", u"<!DOCTYPE HTML PUBLIC \"-//W3C//DTD
"<html><head><meta name=\"qrichtext\" content=\"1\" /><meta charset=\"utf-8\" /><style type=\"text/css\">\n"
"p, li { white-space: pre-wrap; }\n"
"</style></head><body style=\" font-family:'Segoe UI'; font-size:10pt; font-weight:400; font-style:normal;\">\n"
"<p align=\"center\" style=\" margin-top:12px; margin-bottom:12px; -qt-block-indent:0; text-indent:0px;\"><span
"<p align=\"center\" style=\" margin-top:12px; margin-bottom:12px; -qt-block-indent:0; text-indent:0px;\"><span
"<p align=\"center\" style=\" margin-top:12px; margin-bottom:12px; -qt-block-indent:0; text-indent:0px;\"><span
```

Fonte: do próprio autor, 2022.

Figura 73 - Código funções do *App* parte 3

```
    def setThemeGreen(self):
            Settings.BTN_LEFT_BOX_COLOR = "background-color: rgb(42, 48, 41);"
            Settings.BTN_RIGHT_BOX_COLOR = "background-color: rgb(42, 48, 41);"
            Settings.MENU_SELECTED_STYLESHEET = MENU_SELECTED_STYLESHEET = """
            border-left: 22px solid qlineargradient(spread:pad, x1:0.034, y1:0, x2:0.216, y2:0, stop:0.499 rgb(9
            background-color: rgb(42, 48, 41);
            """
            Settings.POMODORO_SELECTED_STYLESHEET = POMODORO_SELECTED_STYLESHEET = """
        background-color: rgb(94, 171, 79);
        """

            self.ui.textEdit.setHtml(QCoreApplication.translate("MainWindow", u"<!DOCTYPE HTML PUBLIC \"-//W3C//
    "<html><head><meta name=\"qrichtext\" content=\"1\" /><meta charset=\"utf-8\" /><style type=\"text/css\">\n"
    "p, li { white-space: pre-wrap; }\n"
    "</style></head><body style=\" font-family:'Segoe UI'; font-size:10pt; font-weight:400; font-style:normal;\"
    "<p align=\"center\" style=\" margin-top:12px; margin-bottom:12px; -qt-block-indent:0; text-indent:0px;\"><s
    "<p align=\"center\" style=\" margin-top:12px; margin-bottom:12px; -qt-block-indent:0; text-indent:0px;\"><s
    "<p align=\"center\" style=\" margin-top:12px; margin-bottom:12px; -qt-block-indent:0; text-indent:0px;\"><s
```

Fonte: do próprio autor, 2022.

2.6.4.5 Funções da *Interface* (*modules/ui_functions.py*)

Arquivo referente às funções gráficas de *interface*. Possui acesso a manipulação dos componentes da janela, portanto é indispensável ao arquivo "*main.py*" e "*ui_main.py*".

As funções de maximizar, minimizar, selecionar botão e alterar paleta de cores, amplamente requisitadas pelo arquivo principal (*main.py*), são efetuadas pela execução desse código. Disponível em: https://github.com/Nbdyleto/PySide6_Self_TCC/blob/main/modules/ui_functions.py

Figura 74- Código funções da *Interface*

```
class UIFunctions(MainWindow):
    # MAXIMIZE/RESTORE
    # ///////////////////////////////////////////////////////////////
    def maximize_restore(self):
        global GLOBAL_STATE
        status = GLOBAL_STATE
        if status == False:
            self.showMaximized()
            GLOBAL_STATE = True
            self.ui.appMargins.setContentsMargins(0, 0, 0, 0)
            self.ui.maximizeRestoreAppBtn.setToolTip("Restore")
            self.ui.maximizeRestoreAppBtn.setIcon(QIcon(u":/icons/images/icons/icon_restore.png"))
            self.ui.frame_size_grip.hide()
            self.left_grip.hide()
            self.right_grip.hide()
            self.top_grip.hide()
            self.bottom_grip.hide()
```

Fonte: do próprio autor, 2022.

Figura 75- Código funções da *Interface* parte 2

```
        else:
            GLOBAL_STATE = False
            self.showNormal()
            self.resize(self.width()+1, self.height()+1)
            self.ui.appMargins.setContentsMargins(10, 10, 10, 10)
            self.ui.maximizeRestoreAppBtn.setToolTip("Maximize")
            self.ui.maximizeRestoreAppBtn.setIcon(QIcon(u":/icons/images/icons/icon_maximize.png"))
            self.ui.frame_size_grip.show()
            self.left_grip.show()
            self.right_grip.show()
            self.top_grip.show()
            self.bottom_grip.show()
```

Fonte: do próprio autor, 2022.

Figura 76 - Código funções da *Interface* parte 3

```
# TOGGLE MENU
# ///////////////////////////////////////////////////////////////
def toggleMenu(self, enable):
    if enable:
        # GET WIDTH
        width = self.ui.leftMenuBg.width()
        maxExtend = Settings.MENU_WIDTH
        standard = 60

        # SET MAX WIDTH
        if width == 60:
            widthExtended = maxExtend
        else:
            widthExtended = standard

        # ANIMATION
        self.animation = QPropertyAnimation(self.ui.leftMenuBg, b"minimumWidth")
        self.animation.setDuration(Settings.TIME_ANIMATION)
        self.animation.setStartValue(width)
        self.animation.setEndValue(widthExtended)
        self.animation.setEasingCurve(QEasingCurve.InOutQuart)
        self.animation.start()
```

Fonte: do próprio autor, 2022.

Figura 77 - Código funções da *Interface* parte 4

```
# IMPORT THEMES FILES QSS/CSS
# ///////////////////////////////////////////////////////////////
def theme(self, file, useCustomTheme):
    if useCustomTheme:
        str = open(file, 'r').read()
        self.ui.styleSheet.setStyleSheet(str)
```

Fonte: do próprio autor, 2022.

2.6.4.6 Acompanhador de Tarefas (*functions/daily_task/daily_task.py*)

Código responsável pela manipulação da tabela de tarefas e filtragem de tópicos para visualização. Para tanto são utilizadas conexões que armazenam os eventos efetuados pelo usuário, onde o programa reage de acordo com esses eventos e possibilita a alteração dos dados e seleção de datas. A filtragem ocorre no carregamento da tabela, onde é baseada nos valores que o usuário seleciona nas listas de filtragem. Disponível em:

https://github.com/Nbdyleto/PySide6_Self_TCC/blob/main/functions/new_daily_task/dailytask.py

Figura 78 - Código acompanhador de tarefas

```
class DTaskMainPage(QtWidgets.QWidget):
    def __init__(self):
        super(DTaskMainPage, self).__init__()
        self.ui = Ui_DailyTaskPage()
        self.ui.setupUi(self)
        self.setupVariables()
        self.setupConnections()
        self.setupWidgets()

    def setupConnections(self):
        widgets.tblTasks.cellClicked.connect(self.rowClickedFunctions)
        widgets.tblTasks.itemChanged.connect(self.updateDBRecord)
        widgets.tblLists.cellClicked.connect(self.updateStatusOrTopic)
        widgets.qCalendar.selectionChanged.connect(self.updateCalendarDate)
        widgets.btnOrderByTopic.clicked.connect(self.loadTopicsInList)
        widgets.btnOrderByStatus.clicked.connect(self.loadStatusInList)
        widgets.listByTopic.itemClicked.connect(self.checkTopicFilter)
        widgets.listByStatus.itemClicked.connect(self.checkStatusFilter)
```

Fonte: do próprio autor, 2022.

Figura 79 - Código acompanhador de tarefas parte 2

```
# LOAD DATA FUNCTIONS
# ////////////////////////////////////////////////////////////////

def loadDataInTable(self):
    widgets.tblTasks.clearContents()
    qry = f"""SELECT * FROM TASKS"""    # Default
    showemptyrow = True
    ### FILTER OPTIONS
    if self.filterByTopic and not self.filterByStatus:
        print("FILTERING JUST BY TOPIC")
        qry = f"""SELECT * FROM TASKS
                  WHERE (topic_id = {self.activeTopicId})"""
    elif self.filterByStatus and not self.filterByTopic:
        print("FILTERING JUST BY STATUS")
        qry = f"""SELECT * FROM TASKS
                  WHERE (status = '{self.activeStatus}')"""
        showemptyrow = False
    elif self.filterByStatus and self.filterByTopic:
        print("FILTERING BY TOPIC AND BY STATUS")
        qry = f"""SELECT * FROM TASKS
                  WHERE (topic_id = {self.activeTopicId}) AND
                      (status = '{self.activeStatus}')"""
        showemptyrow = False
    else:
        pass
    print(f'\nQUERY: {qry}')
```

Fonte: do próprio autor, 2022.

Figura 80 - Código acompanhador de tarefas parte 3

```
try:
    with DBMainOperations() as db:
        tasks = db.cursor.execute(qry).fetchall()
        print(f'{tasks}')
        rowcount = len(tasks)+1 if showemptyrow else len(tasks)
        widgets.tblTasks.setRowCount(rowcount)
    tablerow = 0
    for row in tasks:
        #print('\n', row)
        widgets.tblTasks.setRowHeight(tablerow, 50)
        taskid, taskname, status = row[0], row[1], row[2]
        startDate, endDate = self.formatDate(row[3], row[4])
        topic = self.getTopicName(row[5])
        btnRemoveTask = self.buttonToPutInRow(tablerow, taskid)
        widgets.tblTasks.setItem(tablerow, 0, QtWidgets.QTableWidgetItem(taskname))
        widgets.tblTasks.setItem(tablerow, 1, QtWidgets.QTableWidgetItem(status))
        widgets.tblTasks.setItem(tablerow, 2, QtWidgets.QTableWidgetItem(startDate))
        widgets.tblTasks.setItem(tablerow, 3, QtWidgets.QTableWidgetItem(endDate))
        widgets.tblTasks.setItem(tablerow, 4, QtWidgets.QTableWidgetItem(topic))
        widgets.tblTasks.setCellWidget(tablerow, 5, btnRemoveTask)
        tablerow += 1
    widgets.tblTasks.setRowHeight(tablerow, 50)
except Exception as e:
    print('ERROR: ', e)
```

Fonte: do próprio autor, 2022.

2.6.4.7 *Flashcards* (*functions/flashcards/flashcards_page.py*)

Arquivo referente a lógica de programação da funcionalidade de flashcards, englobando tanto a parte de ressaltar os decks e manipulá-los, quanto a de proporcionar o estudo destes. Possui conexões para adicionar um deck ao apertar no botão respectivo, e obter o tópico selecionado pelo usuário para filtrar suas informações de cartas. Assim como a função acompanhador de tarefas, os decks são mostrados baseando-se no tópico selecionado; portanto, ao carregar a tabela a filtragem é imediatamente executada. Referente a parte de estudo, possui a lógica de cálculo da porcentagem aproximada do rendimento de estudos do usuário, além de tornar visível ou não as respostas para que o utilizador aproveite seu ciclo de estudos adequadamente. Disponível em: https://github.com/Nbdyleto/PySide6_Self_TCC/blob/main/functions/new_flashcards/flashcards_page.py

Figura 81 - Código *Flashcards*

```
class MainFlashcardsPage(QtWidgets.QWidget):
    def __init__(self):
        super(MainFlashcardsPage, self).__init__()
        self.ui = Ui_FlashcardsPage()
        self.ui.setupUi(self)
        self.setupVariables()
        self.setupConnections()
        self.setupWidgets()
```

Fonte: do próprio autor, 2022.

Figura 82 - Código *Flashcards* parte 2

```
def setupConnections(self):
    # Main Page Connections
    widgets.btnAddDecks.clicked.connect(self.addNewDeck)
    widgets.classComboBox.currentIndexChanged.connect(self.selectTopicInComboBox)
    # Study Page Connections
    widgets.btnBackPage.clicked.connect(lambda: widgets.stackedWidget.setCurrentWidget(widgets.MainPage))
    widgets.btnBackPage.clicked.connect(self.resetPage)
    widgets.btnRevealAnswer.clicked.connect(self.revealAnswer)
    widgets.btnRevealAnswer.clicked.connect(lambda: widgets.btnRevealAnswer.setVisible(False))
    widgets.btnRevealAnswer.clicked.connect(lambda: widgets.btnBadFeedback.setVisible(True))
    widgets.btnRevealAnswer.clicked.connect(lambda: widgets.btnOkFeedback.setVisible(True))
    widgets.btnRevealAnswer.clicked.connect(lambda: widgets.btnGoodFeedback.setVisible(True))
    widgets.btnBadFeedback.clicked.connect(lambda: self.nextFlashcard(value=10))
    widgets.btnBadFeedback.clicked.connect(lambda: widgets.btnRevealAnswer.setVisible(True))
    widgets.btnOkFeedback.clicked.connect(lambda: self.nextFlashcard(value=40))
    widgets.btnOkFeedback.clicked.connect(lambda: widgets.btnRevealAnswer.setVisible(True))
    widgets.btnGoodFeedback.clicked.connect(lambda: self.nextFlashcard(value=60))
    widgets.btnGoodFeedback.clicked.connect(lambda: widgets.btnRevealAnswer.setVisible(True))
```

Fonte: do próprio autor, 2022.

Figura 83 - Código Flashcards parte 3

```
def selectTopicInComboBox(self):
    idx = widgets.classComboBox.currentIndex()
    print('IDX É O SEGUINTE', idx)
    topicname = widgets.classComboBox.currentText()
    if idx == 0:    # Show geral
        self.topicID = -1
        self.loadDecksInTable(showall=True, topicid=self.topicID)
    elif idx == widgets.classComboBox.count()-1 and idx != -1: # Add topic
        self.topicID = -1
        print('last row!')
        self.addNewTopic()
    else:   # Show specific decks
        with DBMainOperations() as db:
            self.topicID = db.getAllRecords(tbl='topics', specifcols='topic_id',
            whclause=f'topic_name = "{topicname}"')[0][0]
        self.loadDecksInTable(showall=False, topicid=self.topicID)
```

Fonte: do próprio autor, 2022.

2.6.4.8 *Pomodoro* (*functions/pomodoro/pomodoro_page.py*)

Arquivo que engloba lógica referente à: temporizador do pomodoro e sua eventual relação com a funcionalidade de acompanhador de tarefas; funções básicas de execução; configurações em tempo real solicitadas pelo usuário. Expondo de maneira sucinta, é por meio deste código que as operações de iniciar, pausar, resetar, carregar tarefas, alterar configurações e mostrar o temporizador são efetivamente executadas. Disponível em: https://github.com/Nbdyleto/PySide6_Self_TCC/blob/main/functions/new_pomodoro/new_pomodoro_page.py

Figura 84 - Código *Pomodoro*

```
class Mode(Enum):
    work = 1
    short_rest = 2
    long_rest = 3

class Status(Enum):
    workFinished = 1
    shortRestFinished = 2
    longRestFinished = 3
    repetitionsReached = 4

class NewPomodoroMainPage(QtWidgets.QWidget):
    def __init__(self):
        super(NewPomodoroMainPage, self).__init__()
        self.ui = Ui_Widget()
        self.ui.setupUi(self)
        self.setupVariables()
        self.setupConnections()
        self.setupWidgets()
        self.displayTime()
```

Fonte: do próprio autor, 2022.

Figura 85 - Código *Pomodoro* parte 2

```
def setupVariables(self):
    global widgets
    widgets = self.ui
    # QTime Variables
    self.settings = QtCore.QSettings()

    self.workEndTime = QtCore.QTime(
        int(self.settings.value(workHoursKey, 0)),
        int(self.settings.value(workMinutesKey, 25)),
        int(self.settings.value(workSecondsKey, 0)),
    )
    self.shortRestEndTime = QtCore.QTime(
        int(self.settings.value(shortHoursKey, 0)),
        int(self.settings.value(shortMinutesKey, 5)),
        int(self.settings.value(shortSecondsKey, 0)),
    )
    self.longRestEndTime = QtCore.QTime(
        int(self.settings.value(longHoursKey, 0)),
        int(self.settings.value(longMinutesKey, 15)),
        int(self.settings.value(longSecondsKey, 0)),
    )
    self.timeFormat = "hh:mm:ss"
    self.time = self.workEndTime
    self.workTime = QtCore.QTime(0, 0, 0, 0)
    self.restTime = QtCore.QTime(0, 0, 0, 0)
    self.totalTime = QtCore.QTime(0, 0, 0, 0)
    self.currentMode = Mode.work
    self.currentPomodoros = 0
    self.maxPomodoros = 3
```

Fonte: do próprio autor, 2022.

Figura 86 - Código Pomodoro parte 3

```
        self.timeFormat = "hh:mm:ss"
        self.time = self.workEndTime
        self.workTime = QtCore.QTime(0, 0, 0, 0)
        self.restTime = QtCore.QTime(0, 0, 0, 0)
        self.totalTime = QtCore.QTime(0, 0, 0, 0)
        self.currentMode = Mode.work
        self.currentPomodoros = 0
        self.maxPomodoros = 3
        # Progress Bar Variables
        self.workSecondPercent = 1/(int(self.settings.value(workMinutesKey, 25))*60/100)
        self.shortRestSecondPercent =  1/(int(self.settings.value(shortMinutesKey, 5))*60/100)
        self.longRestSecondPercent = 1/(int(self.settings.value(longMinutesKey, 15))*60/100)
        self.progressValue = 0
        # Pallete Variables
        self.allowChangeModeManually = True
        # Filter Variables
        self.filterByTopic, self.filterByDate = False, False
        self.activeTopicId, self.activeDate = None, None
```

Fonte: do próprio autor, 2022.

Figura 87 - Código *Pomodoro* parte 4

```
def setupConnections(self):
    widgets.tblTasks.cellDoubleClicked.connect(self.markTaskAsFinished)
    widgets.tblTasks.cellClicked.connect(self.showTaskInLabel)

    widgets.btnPomodoro.clicked.connect(lambda: self.updateCurrentMode(Mode.work))
    widgets.btnShortRest.clicked.connect(lambda: self.updateCurrentMode(Mode.short_rest))
    widgets.btnLongRest.clicked.connect(lambda: self.updateCurrentMode(Mode.long_rest))

    widgets.btnStartTimer.clicked.connect(self.startTimer)
    widgets.btnStartTimer.clicked.connect(lambda: widgets.btnStartTimer.setDisabled(True))
    widgets.btnStartTimer.clicked.connect(lambda: widgets.btnPauseTimer.setDisabled(False))
    widgets.btnStartTimer.clicked.connect(lambda: widgets.btnResetTimer.setDisabled(False))
    widgets.btnPauseTimer.clicked.connect(self.pauseTimer)
    widgets.btnPauseTimer.clicked.connect(lambda: widgets.btnStartTimer.setDisabled(False))
    widgets.btnPauseTimer.clicked.connect(lambda: widgets.btnPauseTimer.setDisabled(True))
    widgets.btnPauseTimer.clicked.connect(lambda: widgets.btnResetTimer.setDisabled(False))
    widgets.btnResetTimer.clicked.connect(self.resetTimer)
    widgets.btnResetTimer.clicked.connect(lambda: widgets.btnStartTimer.setDisabled(False))
    widgets.btnResetTimer.clicked.connect(lambda: widgets.btnPauseTimer.setDisabled(True))
    widgets.btnResetTimer.clicked.connect(lambda: widgets.btnResetTimer.setDisabled(False))

    widgets.workMinutesSpinBox.valueChanged.connect(self.updateWorkValue)
    widgets.shortMinutesSpinBox.valueChanged.connect(self.updateShortRestValue)
    widgets.longMinutesSpinBox.valueChanged.connect(self.updateLongRestValue)

    widgets.listByTopic.itemClicked.connect(self.checkTopicFilter)
```

Fonte: do próprio autor, 2022.

Figura 88 - Código *Pomodoro* parte 5

```
def updateTime(self):
    self.time = self.time.addSecs(-1)
    self.totalTime = self.totalTime.addSecs(-1)
    if self.currentMode is Mode.work:
        self.workTime = self.workTime.addSecs(1)
        self.progressValue += self.workSecondPercent
    elif self.currentMode is Mode.short_rest:
        self.restTime = self.restTime.addSecs(1)
        self.progressValue += self.shortRestSecondPercent
    elif self.currentMode is Mode.long_rest:
        self.restTime = self.restTime.addSecs(1)
        self.progressValue += self.longRestSecondPercent
    else:
        pass

    widgets.progressBar.setValue(self.progressValue)
    self.displayTime()
```

Fonte: do próprio autor, 2022.

2.6.4.9 Ver Progresso (*functions/see_progress/see_progress.py*)

Arquivo que engloba lógica referente à: temporizador do pomodoro e sua eventual relação com a funcionalidade de acompanhador de tarefas; funções básicas de execução; configurações em tempo real solicitadas pelo usuário.

Expondo de maneira sucinta, é por meio deste código que as operações de iniciar, pausar, resetar, carregar tarefas, alterar configurações e mostrar o temporizador são efetivamente executadas. Disponível em: https://github.com/Nbdyleto/PySide6_Self_TCC/blob/main/functions/see_progress/see_progress.py

Figura 89 - Código ver progresso

```
class SeeProgressMainPage(QtWidgets.QWidget):
    def __init__(self):
        super(SeeProgressMainPage, self).__init__()
        self.ui = Ui_SeeProgressPage()
        self.ui.setupUi(self)
        self.setupVariables()
        self.setupWidgets()
        self.setupCharts()

    def setupVariables(self):
        global widgets
        widgets = self.ui


    def setupCharts(self):
        # flashcards temp
        series1 = QtCharts.QPieSeries()
        series1.append('Erros', 1)
        series1.append('Acertos', 2)
        slice1 = series1.slices()[1]
        slice1.setExploded()
        slice1.setLabelVisible()
        slice1.setPen(QtGui.QPen(QtCore.Qt.darkGreen, 2))
        slice1.setBrush(QtCore.Qt.green)
        chart1 = QtCharts.QChart()
        chart1.addSeries(series1)
        chart1.setTitle('Simple piechart example')
        chart1.legend().show()
        self._chart_view1 = QtCharts.QChartView(chart1)
        self._chart_view1.setRenderHint(QtGui.QPainter.Antialiasing)

        widgets.verticalLayout_18.addWidget(self._chart_view1)    # verticalLayout_18: Flashcards Frame
```

Fonte: do próprio autor, 2022.

Figura 90 - Código ver progresso parte 2

```
#pomodoro temp
set0 = QtCharts.QBarSet("Parwiz")
set1 = QtCharts.QBarSet("Bob")
set2 = QtCharts.QBarSet("Tom")
set3 = QtCharts.QBarSet("Logan")
set4 = QtCharts.QBarSet("Karim")
set0 << 1 << 2 << 3 << 4 << 5 << 6
set1 << 5 << 0 << 0 << 4 << 0 << 7
set2 << 3 << 5 << 8 << 13 << 8 << 5
set3 << 5 << 6 << 7 << 3 << 4 << 5
set4 << 9 << 7 << 5 << 3 << 1 << 2
series3 = QtCharts.QPercentBarSeries()
series3.append(set0)
series3.append(set1)
series3.append(set2)
series3.append(set3)
series3.append(set4)
chart3 = QtCharts.QChart()
chart3.addSeries(series3)
chart3.setTitle("Percent Example")
chart3.setAnimationOptions(QtCharts.QChart.SeriesAnimations)
categories = ["Jan", "Feb", "Mar", "Apr", "May", "Jun"]
axis = QtCharts.QBarCategoryAxis()
axis.append(categories)
chart3.createDefaultAxes()
chart3.setAxisX(axis, series3)
chart3.legend().setVisible(True)
chart3.legend().setAlignment(QtCore.Qt.AlignBottom)
self._chart_view3 = QtCharts.QChartView(chart3)
self._chart_view3.setRenderHint(QtGui.QPainter.Antialiasing)
widgets.verticalLayout_21.addWidget(self._chart_view3)
```

Fonte: do próprio autor, 2022.

3 CONCLUSÃO

No decorrer deste estudo, observou-se que a aplicação desenvolvida pôde servir como um recurso útil para aqueles que assumiram para si a responsabilidade de lidar com suas dificuldades estudantis, muitas vezes sem um tutor para os guiar. O autodidatismo consiste em se instruir de forma independente, escolhendo e gerenciando seus próprios métodos de aprendizagem, entretanto, essa pode não ser uma tarefa fácil. Os obstáculos encontrados por um estudante solitário são diversos, como a falta de organização, a possível desmotivação por não conseguir observar um progresso significativo e o desconhecimento de técnicas para manter o foco e a concentração durante o período de estudo.

Visando a resolução deste dilema, foi bem-quista a criação de uma plataforma que reunisse métodos de aprendizagem baseados na psicóloga e na neurociência, métodos esses que já existem, porém, espalhados em diversas aplicações diferentes. O *SELF*, plataforma desenvolvida, foi capaz de reunir em uma só aplicação o que havia de essencial em tais métodos, com o objetivo de tornar os estudos mais organizados, fortalecer a retenção, foco e recuperar a confiança intelectual do indivíduo, o auxiliando para a realização de seus próprios objetivos.

Para que houvesse o máximo de precisão quanto a relevância deste projeto, foi realizada uma pesquisa via *Google Forms*, em que, com êxito, foram obtidas respostas que confirmaram o interesse pelo estudo independente, assim como, também, as dificuldades encontradas pelos estudantes em arriscar uma jornada de aprendizagem solitária. Todavia, há muito que poderá ser explorado no futuro, podendo ser encontradas novas demandas advindas de uma pesquisa englobando um público maior. Além do mais, às áreas da psicóloga e neurociência são extensas demais para serem abrangidas em todas as nuances no atual documento; havendo sempre espaço para o contraditório e futuras implementações.

Através dos resultados obtidos foi possível a construção de uma aplicação que melhor atendesse às necessidades do usuário. Como resultado, o *SELF* se tornou uma ferramenta capaz de auxiliar estudantes, vestibulandos e

profissionais a trabalharem em um ambiente adequado para estudos. Este projeto conseguiu ultrapassar as simples expectativas de trazer conforto para o estudante, por ter sido fundamentado em cima da psicologia e neurociência, o *SELF* se tornou capaz de se ajustar ao ciclo circadiano e a sobrecarga gerada pelo excesso de informações, por fim, trazendo maior produtividade com o mínimo de dispersão possível.

4 BIBLIOGRAFIA

DONALD L. HILTON, J. . M. D. O Mestre da Escravidão: Como a pornografia Droga e muda nossos Cérebros. **Medium Recuse A Clicar**, 2018. Disponivel em: <https://medium.com/recuse-a-clicar/o-mestre-da-escravid%C3%A3o-2012d22d58e3>. Acesso em: 25 Setembro 2022.

GUITTON, J. **O Trabalho Intelectual - Conselhos para os que estudam e para os que escrevem**. Tradução de Lucas Félix de Oliveira Santana. 1ª. ed. Campinas: CEDET - Centro de Desenvolvimento Profissional e Tecnológico, v. I, 2018.

HELLER, E. **A Psicologia das Cores - Como as cores afetam a emoção e a razão**. Tradução de Maria Lúcia Lopes da Silva. 1ª. ed. São Paulo: Editora Garamond Ltda, 2014.

LEVITIN, D. J. Why the modern world is bad for your brain. **The Guardian**, 18 Janeiro 2015. Disponivel em: <https://www.theguardian.com/science/2015/jan/18/modern-world-bad-for-brain-daniel-j-levitin-organized-mind-information-overload>. Acesso em: 3 Outubro 2022.

METABOLOGIA, S. B. D. E. E. Posicionamento sobre melatonina. **www.sbem.org.br**, 2016. Disponivel em: <https://www.endocrino.org.br/media/uploads/PDFs/posicionamento_sobre_melatonina_sbem.pdf>. Acesso em: 3 Outubro 2022.

PIAZZI, P. **Aprendendo Inteligência:** Manual de Instruções do cérebro para estudantes em geral. 1ª. ed. São Paulo: Aleph, v. I, 2007.

PIAZZI, P. **Ensinando Inteligência:** Manual de Instruções do Cérebro de Seu Aluno. 3ª. ed. São Paulo: Aleph, v. III, 2009.

PIAZZI, P. **Estimulando Inteligência:** Manual de Instruções do Cérebro de Seu Filho. 2ª. ed. São Paulo: Aleph, v. I, 2015.

PIAZZI, P. **Inteligência em Concursos:** Manual de instruções do cérebro para concurseiros e vestibulandos. 3ª. ed. São Paulo: Aleph, v. IV, 2015.

RIBOULET, L. **Conselhos sobre o Trabalho Intelectual**. Tradução de Karleno Bocarro. 1ª. ed. Campinas: CEDET - Centro de Desenvolvimento Profissional e Tecnológico, v. I, 2019.

SERTILLLANGES, A.-D. **A vida Intelectual - Seu espírito, suas condições, seus métodos**. Tradução de Roberto Mallet. 1ª. ed. Campinas: CEDET - Centro de Desenvolvimento Profissional e Tecnológico, v. I, 2019.

SPITZER, M. Demência Digital: Introdução. **Medium**, 2013a. Disponivel em: <https://medium.com/helkein-filosofia/dem%C3%AAncia-digital-introdu%C3%A7%C3%A3o-671a93de69c3#_ftnref6>. Acesso em: 20 Novembro 2022.

SPITZER, M. Demência Digital: Insônia, Depressão, Vícios e consequências físicas. **Medium**, 2013b. Disponivel em: <https://medium.com/helkein-filosofia/dem%C3%AAncia-digital-ins%C3%B4nia-depress%C3%A3o-v%C3%ADcios-e-consequ%C3%AAncias-f%C3%ADsicas-ccb06c236746>. Acesso em: 3 Outubro 2022.