MINISTÉRIO DA EDUCAÇÃO
SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA FARROUPILHA
***CAMPUS* SANTO ÂNGELO**

AUTOMAÇÃO DE TESTE DO SISTEMA INTEGRADO DE GESTÃO DE ATIVIDADES ACADÊMICAS (SIGAA) - IFFAR

TRABALHO DE CONCLUSÃO DE CURSO

Diuli Bast Kuhn

Santo Ângelo, RS, Brasil.
2024

Automação de Teste do Sistema Integrado de Gestão de Atividades Acadêmicas (SIGAA) - IFFAR

por

Diuli Bast Kuhn

Trabalho de Conclusão de Curso apresentado ao Instituto Federal de Educação, Ciência e Tecnologia Farroupilha, como requisito parcial para obtenção do título de Tecnólogo em Sistemas para Internet.

Orientadora: Marta Breunig Loose

Santo Ângelo, RS, Brasil

2024

Ministério da Educação
Secretaria de Educação Profissional e Tecnológica
Instituto Federal de Educação Ciência e Tecnologia Farroupilha

A Comissão Examinadora, abaixo assinada,
aprova o Trabalho de Conclusão de Curso

AUTOMAÇÃO DE TESTE DO SISTEMA INTEGRADO DE GESTÃO DE ATIVIDADES ACADÊMICAS (SIGAA) - IFFAR

elaborada por
Diuli Bast Kuhn

como requisito parcial para obtenção do título de
Tecnólogo em Sistemas para Internet

COMISÃO EXAMINADORA

_____
Marta Breunig Loose, Me.
(Orientador)

_____
Alan Motta Schumacher, Esp. (IFFar)

_____
John Soldera, Dr. (IFFar)

Conceito Final: _____

Santo Ângelo, 12 de dezembro de 2024.

Dedico este trabalho a minha família e a todas
as pessoas que sempre me apoiaram.

AGRADECIMENTOS

Primeiramente, gostaria de agradecer à minha família, que sempre me apoiou incondicionalmente em todos os meus projetos. Agradeço aos meus pais pela educação que me proporcionaram e por terem me ensinado a importância da persistência, principalmente a minha mãe por sempre apoiar os meus sonhos e estar do meu lado. Agradeço também à minha orientadora Marta, pela paciência, dedicação e pelos ensinamentos valiosos que foram cruciais para a conclusão deste trabalho e também a todos os demais professores que contribuíram para minha formação acadêmica. Por fim, agradeço a todas as pessoas que me acompanharam durante essa jornada e sempre me incentivaram a seguir em frente.

"Investir em conhecimento rende os melhores juros."

Benjamin Franklin

RESUMO

A qualidade de software é uma característica fundamental para o sucesso de uma empresa, pois quando se tem boas práticas de qualidade, aumentam-se as chances de ter um produto que atenda as expectativas do cliente. Nesse contexto, o presente trabalho aborda o impacto de validar os requisitos do Sistema Integrado de Gestão de Atividades Acadêmicas do Instituto Federal Farroupilha (SIGAA), através da automação de testes para os principais processos do software, com a utilização do framework Cypress. Os principais objetivos deste trabalho consistem em atingir uma estratégia eficaz de automação de testes, possibilitando a melhora da qualidade do processo de desenvolvimento do software, garantir a eficiência dos testes, reduzindo os custos e o tempo de desenvolvimento. Para tanto, foi necessário realizar o levantamento dos casos de teste fundamentais para automação, implementar os testes definidos utilizando o Cypress nas funcionalidades mais utilizadas do SIGAA, dentre elas, autenticação de usuário, gestão de alunos, funcionalidades de matrícula e comunicação interna, garantindo a correta execução dos testes e a obtenção de resultados confiáveis e consistentes, além de apresentar os dados obtidos com a automação dos testes. Diante disso, ao seguir práticas padronizadas de desenvolvimento de código, espera-se aumentar a eficiência do processo de desenvolvimento e reduzir o risco de erros e falhas no software. As descobertas têm implicações significativas para o desenvolvimento e aprimoramento contínuo de sistemas de gestão educacional.

Palavras-chave: qualidade; typescript; software educacional; automação de testes.

ABSTRACT

Software quality is a fundamental characteristic for a company's success because good quality practices increase the chances of delivering a product that meets customer expectations. In this context, the present work addresses the impact of validating the requirements of the Integrated Academic Activities Management System of the Federal Farroupilha Institute (SIGAA) through test automation for its key processes, using the Cypress framework. The main objectives of this work are to achieve an effective test automation strategy to enhance the software development process quality, ensure test efficiency, and reduce development costs and time. To achieve this, it was necessary to identify essential test cases for automation, implement these tests using Cypress in the most used functionalities of the system, including user authentication, student management, enrollment features, and internal communication, ensuring correct test execution and obtaining reliable and consistent results. Additionally, the work presents the data obtained from test automation. By following standardized code development practices, the aim is to increase development process efficiency and reduce the risk of errors and software failures. These findings have significant implications for the continuous development and improvement of educational management systems.

Keywords: quality; TypeScript; educational software; test automation.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

E2E End-to-end

DevOps Dev (desenvolvimento) e Ops (operações)

SIGAA Sistema Integrado de Gestão de Atividades Acadêmicas

API Application Programming Interface

BDD Behavior Driven Development

UFRN Universidade Federal do Rio Grande do Norte

SIG Sistema Integrado de Gestão

SIPAC Sistema Integrado de Patrimônio, Administração e Contratos

SIGGP Sistema Integrado de Gestão de Pessoas

SIGPP Sistema Integrado de Gestão de Planejamento e de Projetos

SIGED Sistema Integrado de Gestão Eletrônica de Documentos

SIGAdmin Sistema de Administração dos Sistemas (Técnica e Gestão)

SOLID S- Single Responsibility Principle, O - Open/Closed Principle, L - Liskov Substitution Principle, I - Interface Segregation Principle, D - Dependency Inversion Principle

HTML Hyper Text Markup Language

IFFAR Instituto Federal Farroupilha

ID Identity

CI Integração Contínua

CD Entrega Contínua

SUMÁRIO

1 INTRODUÇÃO

O desenvolvimento de sistemas web pode tornar-se bastante complexo, pois depende muito das características e funcionalidade do sistema a ser desenvolvido. Sendo assim, este se torna suscetível a diversos tipos de erros, podendo resultar na obtenção de um produto diferente do esperado e especificado (DELAMARO; JINO; MALDONADO, 2013).

No início dos anos 80, surgiram os primeiros conceitos de qualidade de software, onde havia o trabalho conjunto entre desenvolvedores e testadores, durante todo o processo de desenvolvimento do software. Apenas nos anos 90 se deu início a produção de ferramentas de testes, as quais trariam alta produtividade e qualidade ao processo de teste (BARTIÉ, 2002).

No contexto do desenvolvimento de software, a busca por qualidade e eficiência é uma constante. A entrega de produtos robustos, confiáveis e que atendam às expectativas dos usuários é um desafio cada vez mais complexo, especialmente em um cenário em que os ciclos de desenvolvimento são cada vez mais curtos e as demandas por atualizações frequentes são constantes.

Nesse cenário, a automação de testes emerge como uma abordagem essencial para garantir a qualidade do software. Automatizar o processo de teste não apenas agiliza a identificação de defeitos, mas também proporciona uma cobertura mais abrangente e consistente dos casos de teste, reduzindo significativamente o tempo e os recursos necessários para validar a funcionalidade do software.

Este trabalho de conclusão de curso tem como objetivo explorar o processo de automação de teste aplicado a um software educacional específico SIGAA - Sistema Integrado de Gestão de Atividades Acadêmicas, que tem como finalidade, centralizar os processos de gestão dos docentes, discentes e servidores do Instituto Federal Farroupilha.

Serão abordados aspectos relacionados à seleção de ferramentas e técnicas de automação adequadas, bem como a implementação prática dessas estratégias no contexto do desenvolvimento de software educacional, incluindo testes de ponta a ponta (E2E), sendo uma metodologia crucial no desenvolvimento de software, garantindo que os sistemas funcionem como esperado do início ao fim. Esses testes simulam ações reais do usuário, percorrendo o software para verificar sua funcionalidade e desempenho.

Ao longo deste estudo, será destacada a importância de uma abordagem abrangente para a automação de teste, considerando não apenas a verificação de funcionalidades básicas,

mas também a avaliação de aspectos pedagógicos, como a adequação do conteúdo, a interatividade e a experiência do usuário.

Em suma, este trabalho visa contribuir para uma compreensão mais aprofundada do papel da automação de teste no contexto dos softwares educacionais, fornecendo *insights* valiosos para profissionais da área de desenvolvimento de software, educadores e demais interessados no uso de tecnologia para aprimorar processos de ensino e aprendizagem.

1.1 Objetivos

1.1.1 Objetivo Geral

Desenvolver e implementar uma estratégia eficaz de automação de testes no software educacional SIGAA (Sistema Integrado de Gestão de Atividades Acadêmicas), visando melhorar a qualidade do processo de desenvolvimento do software, aumentar a eficiência dos testes e reduzir custos e tempo de desenvolvimento.

1.1.2 Objetivos Específicos

Para atingir o objetivo principal deste trabalho os seguintes objetivos específicos são relacionados:

- Realizar o levantamento dos casos de teste fundamentais para automação;
- Implementar os testes definidos utilizando o Cypress nas funcionalidades mais utilizadas do SIGAA, dentre elas, autenticação de usuário, gestão de alunos, funcionalidades de matrícula e comunicação interna, garantindo a correta execução dos testes e a obtenção de resultados confiáveis e consistentes;
- Apresentar os dados obtidos com a automação dos testes.

1.2 Trabalhos Relacionados

Na análise dos trabalhos relacionados a esta pesquisa, recorreu-se às plataformas Google Scholar e pesquisa abrangente ao google de forma livre para identificar documentos pertinentes ao tema em questão. Entre os resultados obtidos, selecionaram-se três trabalhos para uma análise mais aprofundada sobre o assunto.

O propósito da análise dos trabalhos relacionados é identificar os principais temas abordados, visando estabelecer critérios para a avaliação das ferramentas de automação de testes.

A tabela 1 apresenta a lista dos trabalhos selecionados para análise e discussão, fornecendo uma base sólida para o desenvolvimento da pesquisa.

Tabela 1 - Lista de Trabalhos Relacionados

Referência	**Título**	**Tipo de produção**
Darielson, Ana Carolina. "Estudo Comparativo de Testes" [Comparative Study of Tests]. Revista Científica Multiversa, [12/2022].	Selenium, robot e cypress: Um estudo comparativo entre ferramentas de automação de testes.	Artigo
Teixeira, Eleonora. [Análise de ferramenta de automação de testes de software]. (Dissertação de Tecnologia), [Universidade Federal de Santa Maria (UFSM)], 2022).	Análise de ferramentas de automação de testes de software.	Trabalho de Conclusão de Curso
Lourenço, Rony. [Automação de testes para um sistema de e-commerce]. (Dissertação de Bacharelado), [Universidade Federal do Rio Grande do Norte(UFRN)], 2022)	Automação de testes para um sistema de e-commerce.	Trabalho de Conclusão de Curso

Fonte: Autoria Própria

1.2.1 Selenium, robot e cypress: Um estudo comparativo entre ferramentas de automação de testes.

O artigo de Darielson (2022) tem como objetivo avaliar, entre as ferramentas de teste atuais, qual se destaca quanto à velocidade e qualidade dos testes do lado do usuário em ambiente Web. Foi identificado neste trabalho que, para testes rápidos em páginas web, a ferramenta de automação Cypress se mostrou mais eficiente, rápida e intuitiva, reduzindo custos e tempo no processo de teste de qualidade de software.

Na figura 1 é possível visualizar um cenário de teste criado para o site da Faculdade Lourenço Filho.

Figura 1 - Cenários de Testes.

Teste	Descrição
1	Verificar o funcionamento de cada item do menu principal
2	Verificação dos campos do formulário de egresso
3	Verificação das publicações da flf
4	Verificação do campo pesquisar
5	Verificação dos botões desktop e mobile
6	Login no aluno online desktop
7	Login no aluno online mobile

Fonte: Ferreira (2022)

1.2.2 Análise de ferramentas de automação de testes de software.

Este estudo de Teixeira (2022), aborda a maneira como os testes automatizados podem aumentar a eficiência e a produtividade nos processos de teste de software, em comparação com os testes manuais, que tendem a ser mais dispendiosos e demorados. A automação dos testes proporciona uma abordagem mais rápida e eficaz, reduzindo a probabilidade de erros passarem despercebidos.

A implementação da automação de testes surge para minimizar o custo decorrente dos testes manuais, testando uma funcionalidade diversas vezes, de forma automática.

Na figura 2 é apresentado um exemplo de utilização do Cypress, onde há a interação com elementos através do método *get*, desta maneira há o preenchimento de uma lista e por fim são feitas interações de *check* e clique em tela.

Figura 2 - Exemplo de utilização do Cypress.

```
describe('TodoMVC', function () {
  beforeEach(function () {
    cy.visit('http://localhost:8888/')

    cy.get('.new-todo')
      .type('buy some cheese {enter}')
      .type('feed the cat {enter}')
      .type('book a doctors appointment {enter}')
  })

  it.only('hides "Clear Completed" with nothing checked', function () {
    cy.get('.todo-list li').eq(1).find('.toggle').check()
    cy.get('.clear-completed').should('be.visible').click()
    cy.get('.clear-completed').should('not.exist')
  })
})
```

Fonte: Teixeira (2022)

1.2.3 Automação de testes para um sistema de e-commerce.

Este trabalho desenvolvido por Lourenço (2022) tem por objetivo realizar a automação de testes no projeto Promofarma da Codeby usando o framework Cypress em combinação com o padrão Page Object. Com essa automação realizada, os testes passaram a ser utilizados como testes de regressão, onde a cada nova versão lançada, uma nova rodada dos testes automatizados é executada.

As principais contribuições deste trabalho são: um catálogo de casos de testes para sistemas de e-commerces; e um template para automação de casos de teste.

Na figura 3 é possível verificar as diferenças apresentadas entre as duas abordagens de Testes (manual e automatizada).

Figura 3 - Quadro comparativo entre as abordagens manual e automatizada

Teste Manual	Teste Automatizado
Alto risco de não cobertura	Baixo risco de não cobertura
É possível realizar testes exploratórios se aproximando de usuários reais	Só executa ações implementadas
Execução lenta	Execução rápida
Reutilização zero	Reutilização total
Interferências humanas na execução	Sem interferências humanas na execução
Alto custo de execução	Baixo custo de execução

Fonte: Lourenço (2022)

1.2.4 Diferencial do Trabalho

Quanto aos diferenciais do atual trabalho em relação aos trabalhos já existentes, apresenta-se as características relacionadas a garantir a qualidade de um sistema educacional com a ferramenta de automação já definida que neste trabalho será utilizado o framework Cypress, baseando-se na ideia de que em todos os casos citados obteve-se um bom resultado.

Com isso, será possível aprofundar o uso da ferramenta em um sistema único, o qual é utilizado atualmente por toda a rede de ensino do Instituto Federal Farroupilha e validar os processos, de forma que seja possível assegurar o funcionamento correto.

2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção, serão delineados os princípios teóricos fundamentais que baseiam-se na área de estudo do trabalho, juntamente com uma explanação das ferramentas, técnicas e tecnologias utilizadas em seu desenvolvimento.

2.1 Sistema Integrado de Gestão de Atividades Acadêmicas - SIGAA

Segundo o site do Instituto Federal Farroupilha, o Sistema Integrado de Gestão (SIG) foi adotado com o objetivo de facilitar o gerenciamento das informações e padronizar processos e estatísticas, colaborando para a integração das suas diversas unidades.

O SIG foi desenvolvido pela Universidade Federal do Rio Grande do Norte (UFRN) e a primeira versão do sistema foi implantada no ano de 2006. Desde então é usado por todos os discentes, docentes e servidores técnico-administrativos que apoiam as atividades de ensino.

O SIG se divide em subsistemas (SIGAA, SIPAC, SIGGP, SIGPP, SIGED, SIGAdmin), os quais se dividem em módulos, que disponibilizam um conjunto de funcionalidades relacionadas às necessidades/demandas de uma área/unidade específica.

De maneira específica, o Sistema Integrado de Gestão de Atividades Acadêmicas (SIGAA) informatiza os processos da área acadêmica através de módulos e disponibiliza portais específicos para professores, alunos, coordenações, etc.

> Segundo Souza e Monteiro (2015):
> O SIGAA traz um conjunto de unidades e serviços para a comunidade acadêmica, com o propósito de diminuir o tempo de operação das atividades mediante automação de atividades acadêmicas, entre estas, unifica os processos intrínsecos às atividades de ensino, pesquisa e extensão, além de outras atividades acadêmicas. (SOUZA e MONTEIRO, 2015, p. 615).

2.2 Garantia de Qualidade de Software

A implementação de um processo de qualidade de software tem por objetivo determinar um processo que gerencie o nível de qualidade do produto e processo de desenvolvimento de software para que atenda a todos os requisitos do projeto. O desenvolvimento de software inadequado eleva o custo total de desenvolvimento significativo, ampliando os riscos de insucesso do projeto e entregando muitas vezes com atraso (BARTIÉ,2002).

Ao buscar a qualidade de software em todas as atividades de desenvolvimento, há uma redução na quantidade de retrabalho, resultando em custos menores e diminuindo o tempo em que o produto será entregue e posto em produção (PRESSMAN; MAXIM, 2016).

No decorrer dos anos, é possível observar em grandes empresas a preocupação com a qualidade do produto final entregue ao cliente. Dessa forma, busca-se por novas tecnologias e ferramentas que possam atender essa necessidade. Ao inserir no seu dia a dia, as empresas conseguem ver grandes resultados positivos. Em resumo, a garantia de qualidade de software é um componente crítico do processo de desenvolvimento, certificando que o software entregue atenda aos requisitos de qualidade, confiabilidade e usabilidade esperados pelos usuários finais.

De acordo com Damazio, R. (2007), os principais conceitos associados à garantia de qualidade de software de acordo com seus componentes ativos:

Avaliação do Cliente (Operação):

- Corretitude: O software satisfaz as especificações e objetivos do cliente, atendendo às necessidades e facilitando sua utilização.
- Confiabilidade: O programa executa as funções esperadas com precisão, fornecendo informações consistentes para tomada de decisões.
- Eficiência: Requer uma quantidade mínima de recursos para alcançar os objetivos, garantindo uma fácil aprendizagem e utilização.
- Integridade: Apenas usuários autorizados podem acessar e controlar o software, especialmente importante para sistemas críticos.

Revisão do Software:

- Manutenibilidade: Minimiza o esforço para localizar e corrigir erros, promovendo testes e inspeções de código regulares.

- Flexibilidade: Facilita a alteração de funcionalidades ou regras de negócio, permitindo mudanças previstas e imprevistas.
- Testabilidade: Reduz o esforço de teste para garantir que o software atenda aos requisitos, integrando técnicas de teste em todos os processos de desenvolvimento.

Transição do Software:

- Portabilidade: Garante a independência do software em relação a hardware e sistemas operacionais, visando a autonomia através de padrões de desenvolvimento.
- Reusabilidade: Permite o uso modular de partes do software em outros sistemas, evitando retrabalho e duplicação de código.
- Interoperabilidade: Minimiza o esforço para integrar o software a outros sistemas, facilitando a troca de informações com segurança.

2.3 Processo de Desenvolvimento de Software

O processo de desenvolvimento de software refere-se à série de etapas, atividades e métodos utilizados para conceber, projetar, implementar, testar, entregar e manter um sistema de software. Existem várias metodologias e modelos de processo de desenvolvimento de software, cada um com suas próprias abordagens e ênfases.

O desenvolvimento de software é uma atividade que objetiva alcançar um propósito específico de negócio. Softwares profissionais, geralmente são desenvolvidos por equipes e são alterados e mantidos durante toda sua vida (SOMMERVILLE, 2011).

O processo de desenvolvimento de software, deve seguir uma série de atividades metodológicas, que não tem relação com o tamanho e complexidade do software a ser desenvolvido, para que assim ao final se tenha um produto de software. Uma metodologia genérica é composta de cinco atividades (PRESSMAN; MAXIM, 2016):

- Comunicação: entende os objetivos do projeto e reúne os requisitos que irão definir os
- recursos e funcionalidades do software;
- Planejamento: cria o plano de projeto de software, onde são descritas as tarefas a serem conduzidas, os riscos que possam surgir, os recursos necessários, o produto resultante e um cronograma de trabalho;
- Modelagem: cria os modelos que ajudarão no entendimento das necessidades do

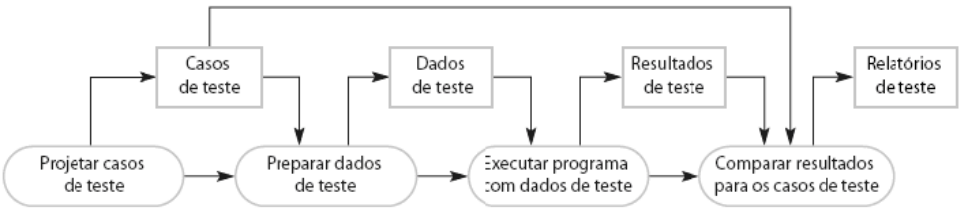
- software e o projeto que irá atender essas necessidades;
- Construção: gera o código e os testes necessários, a fim de revelar possíveis erros;
- Entrega: entrega ao cliente, que irá avaliar o produto e prover feedback, acerca do que
- foi entregue.

2.4 Processo de Automação de Testes

No desenvolvimento de software, as falhas geralmente surgem durante o processo de transformação dos dados em diferentes etapas. Mesmo com o uso das melhores metodologias, ferramentas e profissionais altamente qualificados, os testes permanecem indispensáveis.

Dessa forma, na Figura 4 é possível verificar um esboço de como o processo de testes de software funciona no decorrer do ciclo de desenvolvimento e utilização do sistema. Os retângulos representam os objetivos de cada etapa do teste, sendo que um caso de teste é o ponto de partida para determinar se o processo está funcional. Os quadros circulares ilustram as ações executadas em cada etapa do modelo, sendo necessário projetar os testes para que cada caso de teste se transforme em uma etapa válida.

Figura 4 - Modelo de Teste



Fonte: SOMMERVILLE (2011)

Complementando a utilização de testes surge a automação de testes dos processos utilizados pelos usuários do sistema com um framework de automação, inserindo esse ponto nesse trabalho em questão com a linguagem typescript.

A Figura 5 abaixo ilustra a organização de toda a codificação desses processos, levando em consideração todos os métodos utilizados e citados neste trabalho.O caso ilustrado refere-se a autenticação de usuário que serve como padrão aos demais criados posteriormente. Como breve resumo os arquivos são divididos em:

.ts: Escrita geral de todas as ações/passos que o usuário (automação) irá realizar em cada processo.

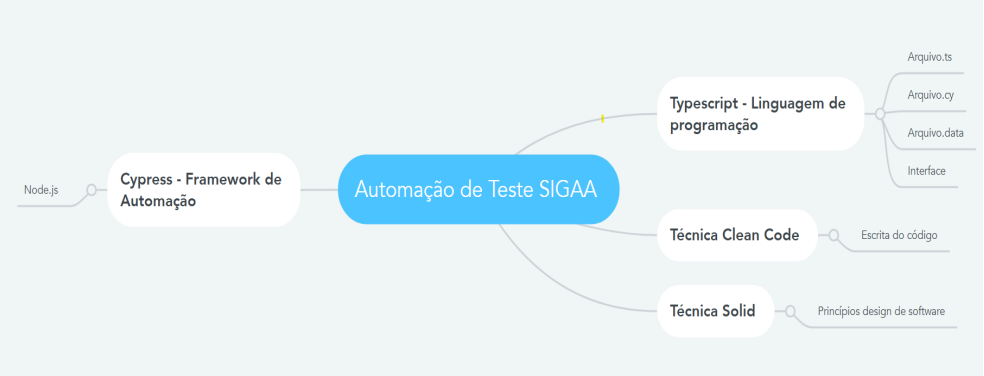
.cy: O arquivo executável, o qual chama as classes criadas no arquivo.ts e as apresenta

no framework de automação.

.data: Arquivo onde é informado os dados do teste, como por exemplo, nome, cpf, dentre outros.

Interface: Criada com o intuito de informar o tipo de valor que cada dado possui, exemplo, string, number.

Figura 5 - Mapa Mental Automação de Teste SIGAA



Fonte: Elaborado pela autora

Ao final desse trabalho, no apêndice A, são apresentadas as figuras ilustrativas que representam todos os arquivos mencionados no tópico typescript.

2.5 Testes de Software

Os softwares são ferramentas muito importantes e úteis para empresas que querem otimizar seus processos com tecnologia. Realizar o teste de software surge como um passo fundamental para garantir que esses programas funcionem da melhor maneira possível.

Os testes de software tem por finalidade a execução de um programa ou sistema com a intenção de encontrar defeitos ou verificar se o software atende aos requisitos especificados.

2.5.1 Casos de Teste

Tem como propósito um conjunto de condições ou variáveis sob as quais um testador determina se um sistema funciona corretamente. Cada caso de teste é projetado para testar uma funcionalidade específica do software.

Importante destacar que casos de teste bem definidos podem gerar um impacto muito mais positivo na entrega final do produto, pois o que será testado é aquilo em que está efetivamente descrito, por isso ao selecionar o responsável, também é importante que o

mesmo conheça do processo e consiga validar todos os pontos críticos de uma determinada situação.

2.5.2 Testes Funcionais Manuais

De acordo com Teixeira (2022), os testes funcionais são uma parte essencial do processo de garantia de qualidade de software e visam verificar se o sistema ou aplicativo está funcionando conforme o esperado em termos de suas funcionalidades. Basicamente o teste funcional, repete os processos que o usuário do sistema irá fazer no seu cotidiano e garante que o mesmo esteja funcional, sem apresentar falhas.

Os testes funcionais avaliam como o sistema responde a diferentes entradas. Eles verificam se as entradas fornecidas produzem as saídas esperadas de acordo com o comportamento previsto.

2.5.3 Testes Automatizados

Os testes automatizados desempenham um papel crucial na garantia da qualidade de software, proporcionando eficiência, cobertura abrangente e detecção precoce de defeitos. Eles são uma prática recomendada em equipes de desenvolvimento de software para garantir a entrega de produtos de alta qualidade.

Abaixo estão alguns pontos positivos para explicar o porque utilizar automação de testes de acordo com o site Sebrae (Conheça as vantagens e desvantagens da automação industrial, 2023):

- Os testes automatizados são mais eficientes em comparação com os testes manuais, pois podem ser executados de forma rápida e repetida, sem a necessidade de intervenção humana constante;
- Eles permitem uma cobertura mais abrangente dos casos de teste, pois podem ser configurados para testar uma grande variedade de cenários em um curto espaço de tempo;
- Desenvolvem a confiança da equipe nos processos operacionais, criando diretrizes específicas para executar atividades ou carregar/gerenciar documentos;
- Automatizar processos impacta o cliente, que ganha um melhor suporte de TI, atendimento responsivo e uma variedade de opções de itens ou serviços que deseja.

Dentre o processo de automação de testes existe o tipo de teste que denomina-se End-to-End (E2E), mais especificamente testes de ponta a ponta, que são executados percorrendo todo o ciclo do processo, ou seja, do início ao fim. Eles replicam o caminho que um usuário real seguiria através da aplicação, interagindo com todas as partes do sistema como um usuário faria, desde a entrada de dados até a visualização dos resultados.

Os testes E2E validam a integração entre diferentes componentes do sistema, incluindo o front-end, back-end, bancos de dados, serviços externos e outras partes do sistema.

Os testes E2E podem identificar problemas de usabilidade, como fluxos confusos, problemas de navegação e erros de validação de entrada, garantindo uma experiência de usuário mais suave.

2.6 Framework de Automação de Teste

São conjuntos de ferramentas, bibliotecas e padrões que são usados para automatizar o processo de teste de software. Eles fornecem uma estrutura organizada e reutilizável para desenvolver e executar casos de teste automaticamente.

Esses frameworks geralmente oferecem funcionalidades como:

- Abstração de elementos de interface do usuário: Permitindo que os testes interajam com os elementos da interface do usuário, como botões, caixas de texto e menus, de forma programática;
- Execução de testes em vários ambientes: Permitindo que os testes sejam executados em diferentes configurações de sistema operacional, navegadores da web, dispositivos móveis, etc;
- Reutilização de código: Permitindo que os casos de teste sejam reutilizados e compartilhados entre diferentes projetos ou equipes de desenvolvimento.

2.6.1 Cypress

De acordo com a documentação oficial do *Cypress* (Cypress 2024), é uma ferramenta de teste front-end de próxima geração desenvolvida para a web moderna. Cypress consiste em um aplicativo gratuito, de código aberto , instalado localmente e no *Cypress Cloud* para gravar seus testes .

Cypress permite que você escreva todos os tipos de testes:

- Testes de ponta a ponta;

- Testes de componentes;
- Testes de integração;
- Testes unitários.

Destaca-se algumas características do framework e o motivo pelo qual o mesmo deve ser utilizado:

- Simplicidade de uso: *Cypress* possui uma API intuitiva e uma sintaxe fácil de entender, o que facilita a criação e manutenção de testes automatizados;
- Tempo real: *Cypress* oferece a capacidade de visualizar o estado do aplicativo em tempo real durante a execução dos testes, permitindo uma depuração mais rápida e eficiente de problemas;
- Execução rápida: Devido à sua arquitetura única, Cypress é conhecido por sua rápida execução de testes. Ele executa testes em um navegador real em segundo plano, o que proporciona uma experiência mais rápida e confiável em comparação com outros frameworks que dependem de simulação de navegador.

2.7 Linguagem de programação - *Typescript*

De acordo com a documentação oficial (TypeScript 2024), é uma linguagem de programação desenvolvida pela Microsoft que é uma extensão do JavaScript. Ela adiciona recursos como tipagem estática, suporte a classes e módulos, entre outros, ao JavaScript padrão.

O objetivo do TypeScript é ser um verificador de tipo estático para programas JavaScript - em outras palavras, uma ferramenta que é executada antes da execução do código (estático) e garante que os tipos do programa estejam corretos.

- Tipagem estática: TypeScript permite a definição de tipos de dados para variáveis, parâmetros de função, retorno de função, entre outros;
- Integração com JavaScript: TypeScript é totalmente compatível com JavaScript, o que significa que você pode usar bibliotecas e frameworks JavaScript existentes em seus projetos TypeScript sem a necessidade de qualquer conversão;
- Classes e Interfaces: TypeScript suporta a definição de classes e interfaces, permitindo a aplicação de princípios de orientação a objetos de forma mais clara e estruturada;
- Módulos: TypeScript suporta módulos, que ajudam na organização e encapsulamento de código, facilitando a construção de aplicativos modulares e escaláveis.

Os sistemas de tipos estáticos descrevem as formas e comportamentos de quais serão os valores quando executa programas. Um verificador de tipo como o TypeScript usa essas informações e informa quando não está correto.

2.7.1 Técnica *Clean Code*

De acordo com o vídeo Clean Code // Dicionário do Programador (Código Fonte TV 2019), é uma abordagem de desenvolvimento de software que se concentra na escrita de código claro, legível e de fácil manutenção.

Algumas técnicas e princípios fundamentais da Clean Code incluem:

- Nomes significativos: Escolher nomes descritivos e significativos para variáveis, funções, classes e outros elementos do código, de modo que seu propósito e função sejam claros para qualquer pessoa que leia o código;
- Funções pequenas e focadas: Escrever funções curtas e com uma única responsabilidade, evitando funções longas e complexas que realizam múltiplas tarefas;
- Formato consistente: Manter um estilo de formatação consistente em todo o código, utilizando espaçamento, indentação e quebras de linha de forma uniforme para melhorar a legibilidade;
- Evitar duplicação de código: Identificar e eliminar duplicações no código, utilizando técnicas como extração de métodos, herança, polimorfismo, entre outros;
- Refatoração contínua: Constantemente revisar e refatorar o código para melhorar sua estrutura, legibilidade e manutenção, mantendo-o limpo ao longo do tempo.

2.7.2 Técnica *Solid*

De acordo com vídeo SOLID (O básico para você programar melhor) // Dicionário do Programador (Código Fonte TV 2020), o acrônimo SOLID representa um conjunto de cinco princípios de design de software que visam criar sistemas mais flexíveis, escaláveis e fáceis de manter. Esses princípios foram introduzidos por Robert C. Martin e são amplamente adotados na engenharia de software moderna.

- S - Single Responsibility Principle (Princípio da Responsabilidade Única): Este princípio estabelece que uma classe deve ter apenas uma razão para mudar, ou seja, deve ter apenas uma responsabilidade;
- O - Open/Closed Principle (Princípio Aberto/Fechado): Este princípio enfatiza que as entidades de software (classes, módulos, funções, etc.) devem ser abertas para extensão, mas fechadas para modificação;
- L - Liskov Substitution Principle (Princípio da Substituição de Liskov): Este princípio afirma que os objetos de um tipo base devem ser substituíveis por objetos de subtipos sem afetar a corretude do programa;
- I - Interface Segregation Principle (Princípio da Segregação de Interfaces): Este princípio declara que uma classe não deve ser forçada a depender de métodos que não utiliza;
- D - Dependency Inversion Principle (Princípio da Inversão de Dependência): Este princípio sugere que as classes de alto nível não devem depender de classes de baixo nível, mas sim de abstrações.

2.8 *Behavior Driven Development* (BDD)

Conforme o site DevMedia, visa integrar regras de negócios com linguagem de programação, focando o comportamento do software, isso permite que eles foquem em por que o código deve ser criado, ao invés de detalhes técnicos, e ainda possibilita uma comunicação eficiente entre as equipes de desenvolvimento e testes.

Vantagens em usar BDD:

- Comunicação entre equipes :possibilita essa integração porque os testadores podem escrever os cenários de testes para os desenvolvedores implementarem;
- Compartilhamento de conhecimento: com desenvolvedores e testadores trabalhando juntos, ao longo do tempo, um irá transferir o seu conhecimento para o outro, criando assim uma equipe multifuncional;
- Documentação dinâmica: algumas equipes ágeis afirmam que não documentam o sistema porque a manutenção destes artefatos é custosa. Usando os frameworks de BDD estes artefatos são gerados dinamicamente sem nenhum esforço adicional.

2.9 *Node.js*

Plataforma de software de código aberto, permite que os desenvolvedores criem aplicativos de rede e web escaláveis e de alta performance, usando JavaScript tanto no lado do cliente quanto no lado do servidor. Em essência, Node.js permite que execute JavaScript fora de um navegador da web, o que o torna ideal para criar aplicativos de servidor e outras aplicações de rede.

2.10 *Draw.io*

Oferece uma interface intuitiva para projetar e criar diagramas de alta qualidade com vários elementos, como formas, setas, tabelas, caixas de texto e imagens. Além disso, o software inclui uma ampla variedade de modelos incorporados para simplificar o processo de diagramação.

3 DESENVOLVIMENTO

Neste capítulo serão descritas as principais etapas do desenvolvimento do projeto. O foco principal está na documentação da proposta de validação do SIGAA, contendo a descrição dos requisitos e modelagem dos cenários de testes, modelagem de classes e telas da codificação dos cenários no framework de automação de testes.

3.1 Requisitos do Sistema

Realizar uma revisão detalhada da literatura sobre automação de testes, enfocando os conceitos fundamentais, metodologias e as melhores práticas relacionadas ao uso do Cypress.

Avaliar a eficácia da estratégia de automação de testes desenvolvida, analisando métricas como tempo de execução dos testes, taxa de detecção de defeitos e facilidade de manutenção dos casos de teste.

Propor recomendações e sugestões para aprimoramento contínuo da estratégia de automação de testes com Cypress, visando maximizar os benefícios obtidos e otimizar o processo de desenvolvimento de software como um todo.

3.2 Plano de Testes

Nesta seção são apresentados e descritos os cenários apropriados para testes que representam as funcionalidades selecionadas.

Planos de Testes focados nas funcionalidades:

- Validação de Login e Autenticação: Desenvolver testes automatizados para verificar a correta autenticação de usuários, como aluno;
- Testes de Edição de Alunos: Criar testes automatizados para garantir que a edição de informações de alunos estejam funcionando corretamente, incluindo validações de campos obrigatórios e formatos de dados;
- Verificação de Funcionalidades de Matrícula e Turmas: Implementar testes automatizados para validar o processo de matrícula de alunos em turmas, assegurando que as informações estejam sendo corretamente registradas e atualizadas no sistema;
- Validação de Funcionalidades de Notas: Desenvolver testes automatizados para garantir o correto funcionamento das funcionalidades relacionadas a notas, como acesso às notas de provas e trabalhos;
- Validação de Restrições de Acesso por Perfil: Criar testes automatizados para garantir que as restrições de acesso baseadas em perfis de usuário estejam funcionando

corretamente, garantindo que cada usuário tenha acesso apenas às funcionalidades e informações relevantes ao seu papel.

Planos de Testes focados no operacional:

- Testes de Comunicação: Criar testes automatizados para verificar a correta entrega de comunicações e notificações do sistema, como avisos de eventos, mensagens entre usuários e lembretes de prazos;
- Testes de Desempenho em Horários de Pico: Estabelecer testes automatizados para avaliar o desempenho do sistema em horários de pico, garantindo que ele seja capaz de lidar com um grande volume de acessos simultâneos sem comprometer a velocidade e estabilidade;
- Tempo de Resposta de Página: Os testes automatizados devem verificar se o tempo de carregamento das páginas do sistema acadêmico é aceitável, garantindo uma experiência de usuário fluente.

3.3 Cenários de Testes

Para a escrita dos cenários de teste, é utilizada a abordagem Behavior Driven Development (BDD) para que possam ser utilizados no projeto de automação Cypress.

As tabelas 2, 3, 4, 5, 6 e 7 abaixo apresentam a relação dos cenários de testes e suas definições para cada uma das *features* identificadas no plano de testes, ou seja, para cada funcionalidade específica do software.

Tabela 2 - Validação de Login e Autenticação

Feature	**Definição**	**Cenário**	
Autenticação de usuário	**Como** usuário do sistema **Quero** me autenticar **Para** acessar meu perfil de usuário	Inserir um usuário e/ou senha incorretos	**Dado** a tela de autenticação **Quando** informado um usuário inválido e/ou uma senha inválida **Então** ao entrar é apresentada a mensagem de validação: “Usuário e/ou senha inválidos”
		Autenticar	**Dad**o a tela de

		usuário com sucesso	autenticação **Quando** informado um usuário e senha válidos **Então** ao entrar redirecionado ao dashboard do sistema

Autenticação de usuário

Estruturação de como serão os cenários de teste da funcionalidade de Autenticação de Usuário. Onde o primeiro cenário é um esquema de cenário, que testará a variação de cenários inválidos, sendo esses o de cpf inválido ou senha inválida.

- Inserir um usuário incorreto:

Esquema do cenário: Autenticar usuário preenchendo informação de usuário inválido

Dado a tela de Autenticação

Quando informado um <usuario> e uma <senha>

E Entrar

Então é apresentado o erro <Usuário e/ou senha inválidos>

Exemplo:

|usuario |senha |mensagem de erro |

|'7878788787' |'09090909' |'Usuário e/ou senha inválidos'|

Tabela 3 - Testes de Edição de Alunos

Feature	**Definição**	**Cenário**	
Edição de dados do Aluno	**Como** usuário do sistema **Quero** editar os dados pessoais **Para** atualizar as informações do cadastro	Alterar os dados e confirmar	**Dado** o acesso a tela de edição de dados pessoais **Quando** alterar um dado **Então** é apresentada a mensagem de confirmação: “Atualização dos dados do discente realizada com sucesso!”
		Deixar campo	**Dado** a tela de

		obrigatório em branco	edição de dados pessoais **Quando** eliminado a informação de um campo **Então** é apresentada a mensagem de impedimento “Campo obrigatório não informado.”
		Deixar confirmação dos dados em branco	**Dado** a tela de edição de dados pessoais **Quando** não realizada a confirmação dos dados **Então** é apresentada a mensagem de impedimento: “Informe os dados de confirmação.”

Tabela 4 - Verificação de Funcionalidades de Matrícula e Turmas

Feature	**Definição**	**Cenário**	
Funcionalidade de Matrícula	**Como** usuário do sistema **Quero** selecionar turma de disciplinas **Para** realizar matrícula	Buscar turmas disponíveis	**Dado** a página de matrícula online **Quando** selecionada as turmas disponíveis **Então** é apresentada a mensagem de confirmação: “As seguintes turmas foram selecionadas com sucesso: XXXXX- Turma 00, XXXXX - Turma 00.”
		Selecionar turmas	**Dado** a página de matrícula online

		equivalentes as disciplinas do curso	**Quando** realizada a busca por turmas equivalentes **Então** é apresentada a mensagem de confirmação: “As seguintes turmas foram selecionadas com sucesso.”
		Realizar processo de matrícula completo	**Dado** a página de matrícula **Quando** realizado o processo de selecionar as turmas e confirmar matrícula **Então** é apresentada a mensagem de confirmação: “Confirmação de matrícula realizada com sucesso.”
		Confirmar matrícula sem turma selecionada	**Dado** a página de matrícula **Quando** realizar o processo de confirmar matrícula sem turma selecionada **Então** é apresentada a mensagem de impedimento: “É necessário selecionar no mínimo uma turma”
		Realizar matrícula fora do período	**Dado** a página inicial do SIGAA **Quando** selecionado o processo de iniciar matrícula online

			Então é apresentada a mensagem de impedimento: “Não está no período oficial de matrículas on-line”

Tabela 5 - Validação de Funcionalidades de Notas

Feature	**Definição**	**Cenário**	
Acesso às notas	**Como** usuário do sistema **Quero** acessar as notas de uma disciplina **Para** verificar a média da disciplina	Acessar uma disciplina e validar a nota	**Dado** o acesso a uma disciplina **Quando** consultar as notas **Então** a Nota deverá ser apresentada corretamente

Tabela 6 - Testes de Comunicação

Feature	**Definição**	**Cenário**	
Acesso à Caixa Postal	**Como** usuário do sistema **Quero** acessar a caixa entrada **Para** verificar as mensagens recebidas	Consultar a caixa de entrada	**Dado** o acesso a caixa postal **Quando** acessada a caixa de entrada **Então** exibe as mensagens recebidas
	Como usuário do sistema **Quero** acessar a caixa de saída **Para** verificar as mensagens enviadas	Consultar a caixa de saída	**Dado** o acesso a caixa postal **Quando** acessada a caixa de saída **Então** Exibe mensagens enviadas
	Como usuário do sistema **Quero** acessar a lixeira **Para** verificar as mensagens	Consultar a lixeira	**Dado** o acesso a caixa postal **Quando** acessada a lixeira **Então** exibe mensagens

	excluídas		excluídas

Tabela 7 - Validação de Restrições de Acesso por Perfil

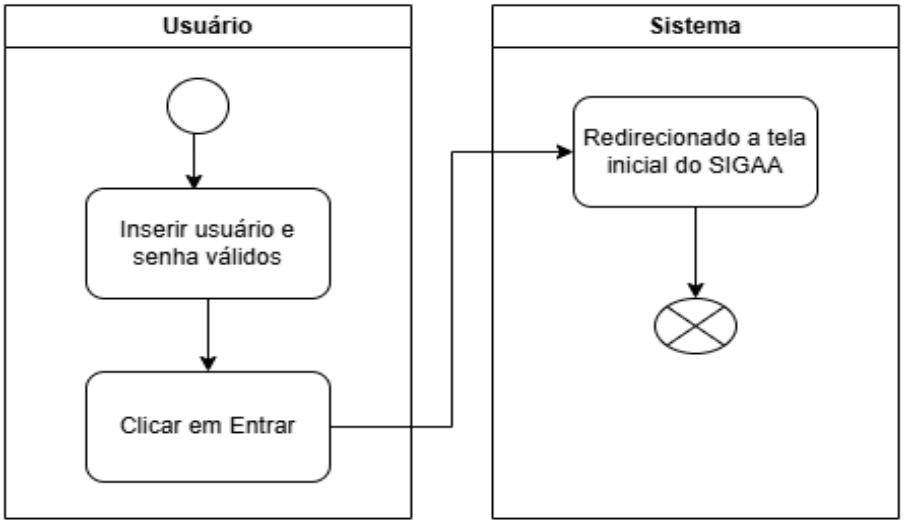
Feature	**Definição**	**Cenário**	
Restrição de acesso do usuário	**Como** usuário do sistema **Quero** autenticar o usuário em diferentes perfis **Para** validar restrição de acesso do usuário	Acessar com credenciais de aluno o perfil SIPAC	**Dado** a tela de autenticação **Quando** informado um usuário de aluno do perfil SIPAC **Então** será visualizado somente os processos liberados para perfil de aluno

3.4 Diagrama de Atividades

A próxima etapa envolve a apresentação do Diagrama de Atividades para visualizar e comunicar o fluxo de comportamentos do sistema. Dessa forma, foi realizada a separação entre as etapas que são de responsabilidade do usuário e do próprio sistema.

Na Figura 6 é ilustrado o Diagrama de Atividade referente ao cenário de teste Autenticação de Usuário. É possível visualizar as ações do Usuário, que consistem em inserir os dados de acesso válidos de acesso, os quais o Sistema irá validar e redirecionar para a tela inicial.

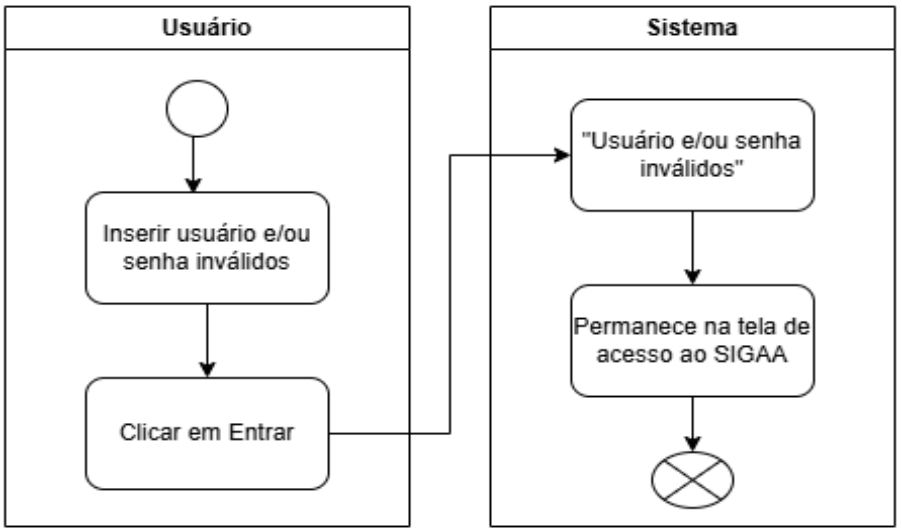
Figura 6 - Diagrama de Atividade com Autenticação de Usuário



Fonte: elaborado pelo autor (2024).

Na Figura 7 é ilustrado o Diagrama de Atividade referente ao cenário de teste Autenticação de Usuário Inválido. É possível visualizar as ações do Usuário, que consistem em inserir os dados inválidos para acesso ao sistema, retornando do sistema uma mensagem de validação.

Figura 7 - Diagrama de Atividade com Autenticação de Usuário Inválido

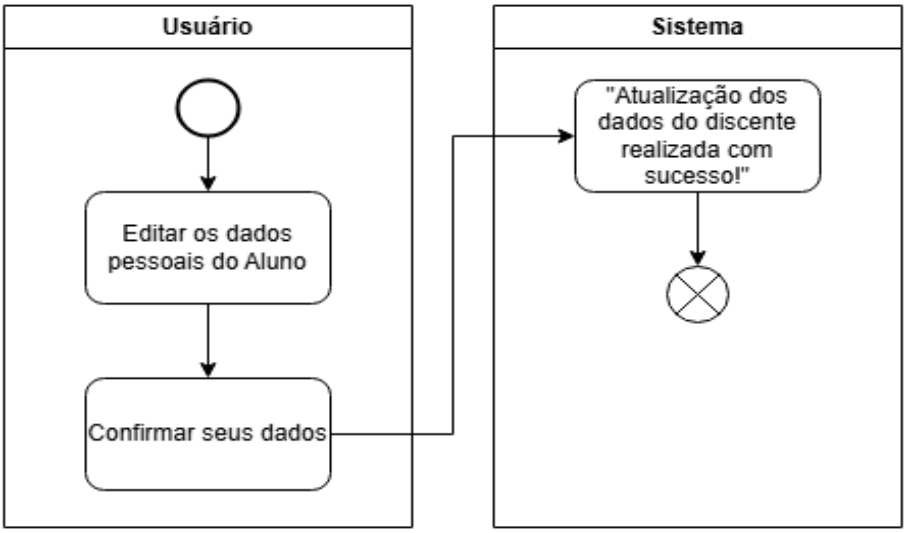


Fonte: elaborado pelo autor (2024).

Nas Figuras 8, 9 e 10 são ilustrados os Diagramas de Atividades referente ao cenário de teste Edição de Dados do Aluno.

Na Figura 8 é possível visualizar as ações do usuário, que consistem em editar os dados do aluno corretamente e receber o retorno do sistema de confirmação da edição.

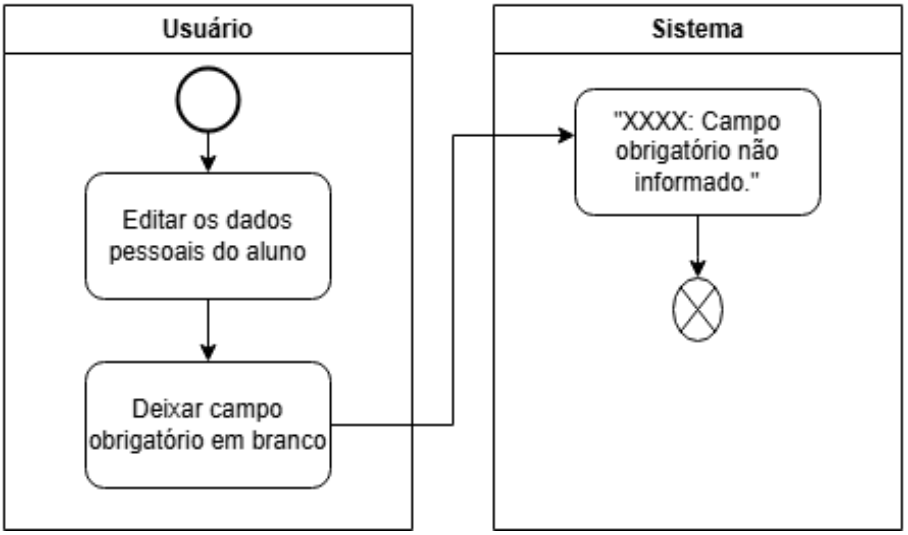
Figura 8 - Diagrama de Atividade Edição de dados do Aluno



Fonte: elaborado pelo autor (2024).

Na Figura 9 é possível visualizar as ações do Usuário, que consistem em não preencher campo obrigatório, o qual o sistema fará a validação de retorno.

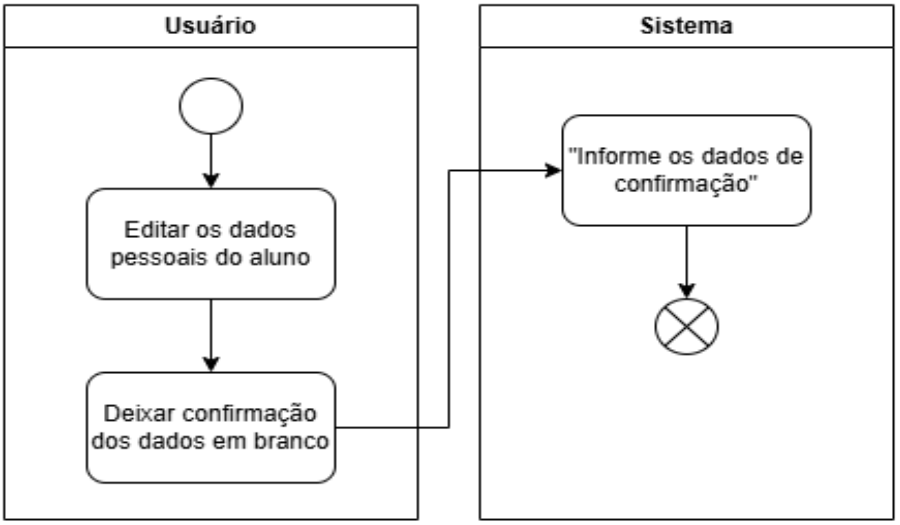
Figura 9 - Diagrama de Atividade Edição de dados do Aluno Campo obrigatório não informado



Fonte: elaborado pelo autor (2024).

Na Figura 10 é possível visualizar as ações do Usuário, que consistem em não realizar a confirmação dos dados, fazendo com o que sistema realize a validação e não permita a edição dos dados.

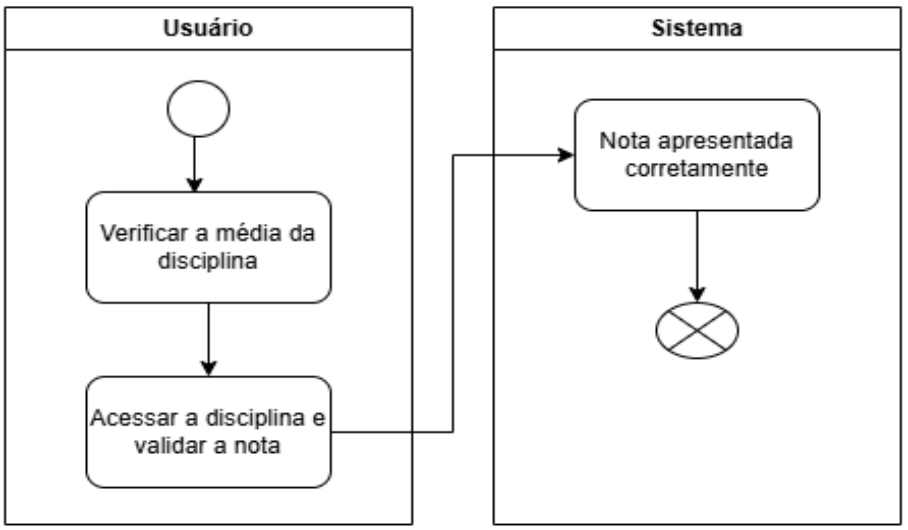
Figura 10 - Diagrama de Atividade Edição de dados do Aluno sem dados de confirmação



Fonte: elaborado pelo autor (2024).

Na Figura 11 é possível visualizar as ações do Usuário, que consistem em validar uma nota armazenada em uma disciplina e validar no sistema se equivale o valor.

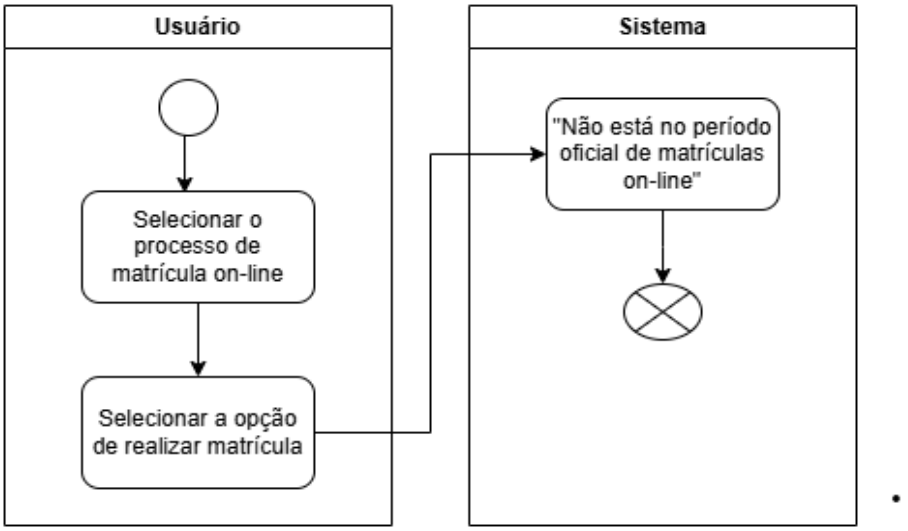
Figura 11 - Diagrama de Atividade Validação de Notas



Fonte: elaborado pelo autor (2024).

Na Figura 12 é possível visualizar as ações de validação do sistema quando o usuário realiza o processo de matrícula fora do período estabelecido pelo sistema, o sistema deve retornar a mensagem de impedimento, não permitindo iniciar a Matrícula.

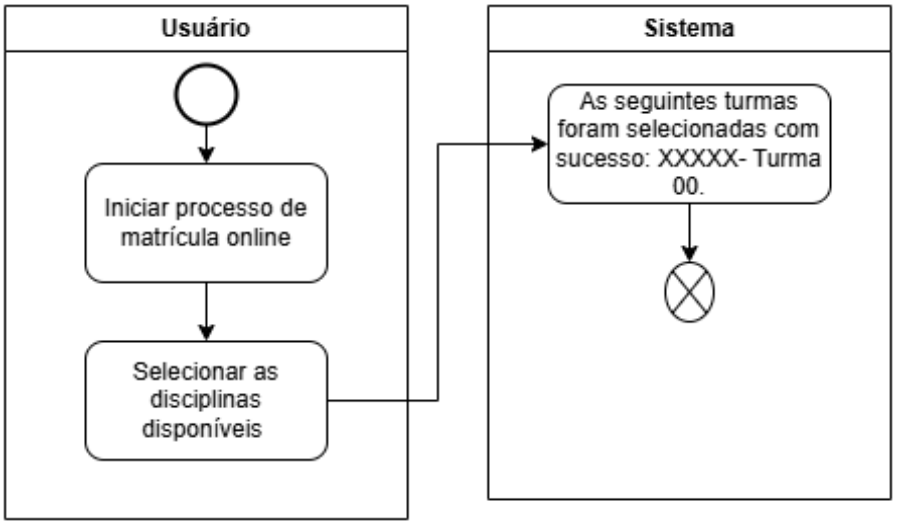
Figura 12 - Diagrama de Atividade Matrícula fora do Período



Fonte: elaborado pelo autor (2024).

Na Figura 13 o diagrama refere-se a validação de listar as turmas disponíveis para o aluno selecionar, dessa forma, ao selecionar as disciplinas, o sistema deve retornar a confirmação de que as mesmas foram selecionadas com sucesso.

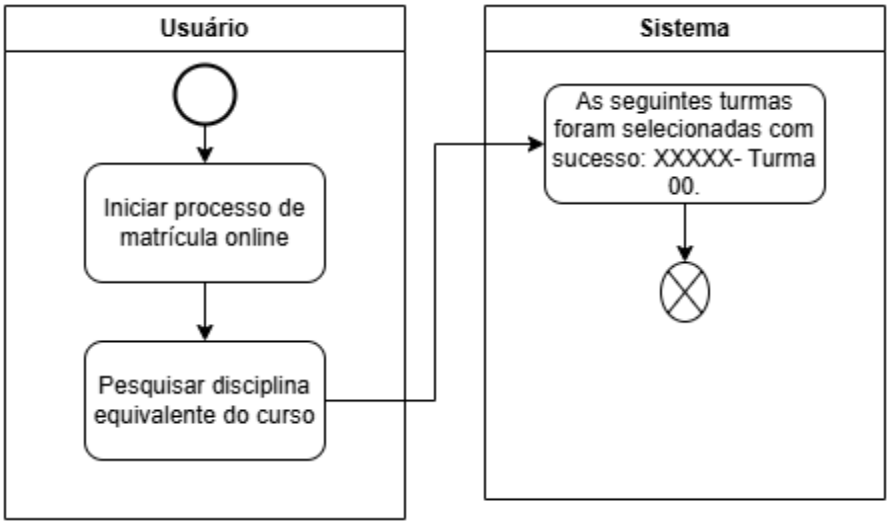
Figura 13 - Diagrama de Atividade Buscar turmas disponíveis



Fonte: elaborado pelo autor (2024).

A Figura 14 ilustra o processo de seleção das disciplinas equivalentes ao curso do aluno, onde o sistema retorna a mensagem de confirmação de seleção da turma equivalente selecionada.

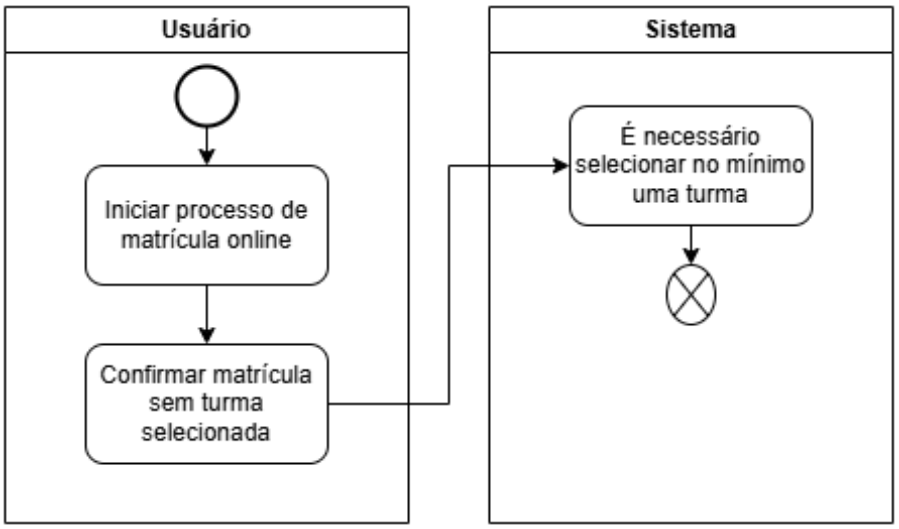
Figura 14 - Diagrama de Atividade Buscar turmas equivalentes disponíveis



Fonte: elaborado pelo autor (2024).

Na Figura 15 o usuário não deve selecionar nenhuma turma disponível para confirmação de matrícula e o sistema deve retornar a mensagem de validação com o impedimento de concluir o processo de matrícula.

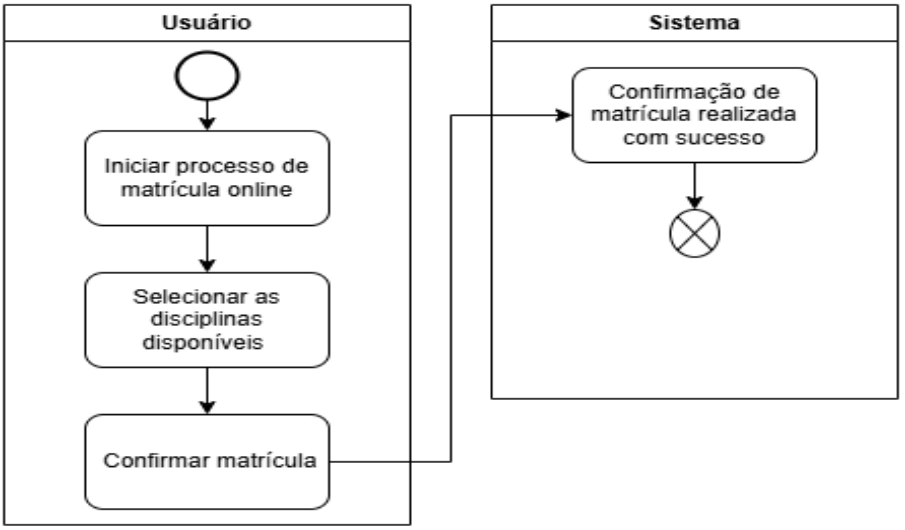
Figura 15 - Diagrama de Atividade Confirmar matrícula sem turma selecionada



Fonte: elaborado pelo autor (2024).

Na Figura 16 o diagrama de atividade ilustra o processo da ação de selecionar as turmas e realizar a matrícula sem nenhum impedimento.

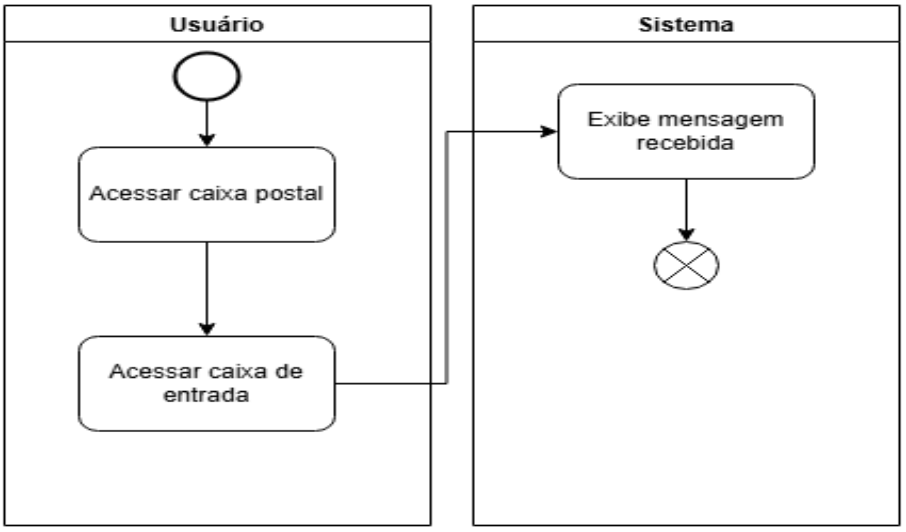
Figura 16 - Diagrama de Atividade Realizar matrícula com sucesso



Fonte: elaborado pelo autor (2024).

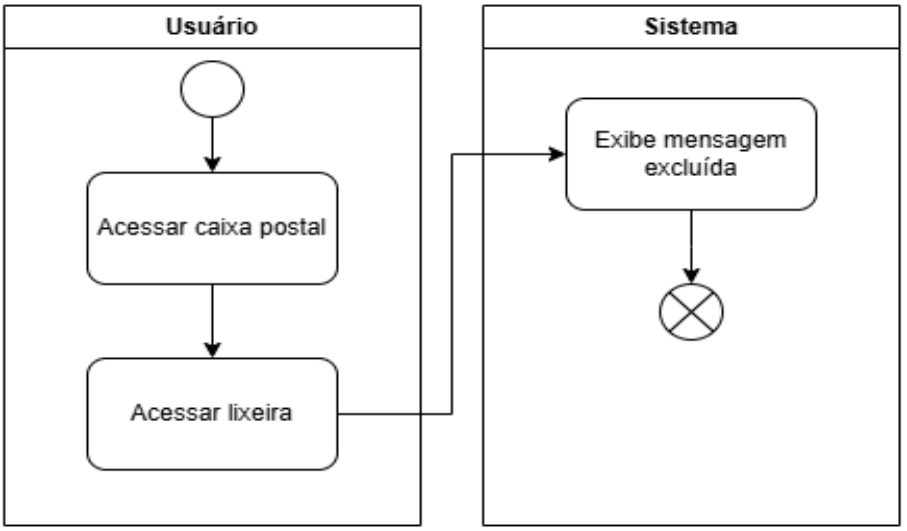
As Figuras 17, 18 e 19 exibem as ações do sistema referente a área de comunicação, onde realiza a validação da caixa postal com as três opções disponíveis, caixa de entrada, onde retorna as mensagens recebidas, caixa de saída, apresenta as mensagens enviadas e por fim, a lixeira, que salva as mensagens apagadas pelo aluno.

Figura 17 - Diagrama de Atividade Acessar Caixa de Entrada



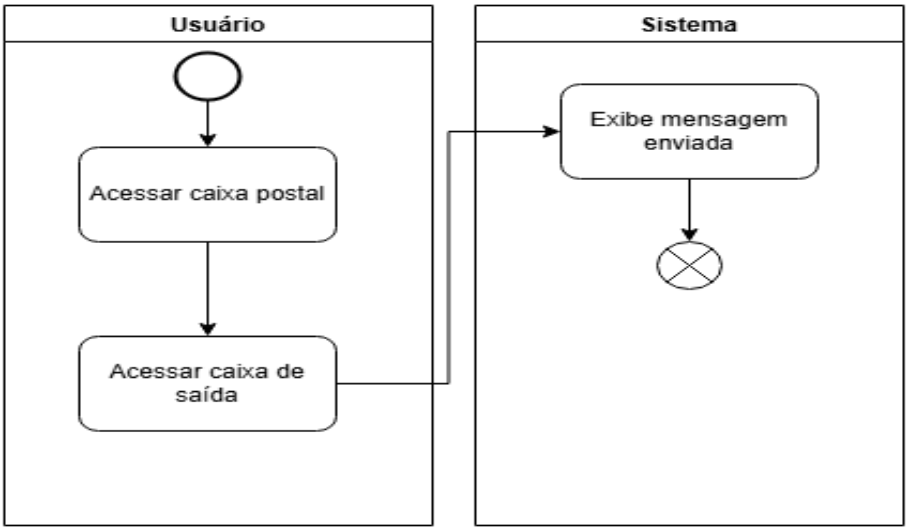
Fonte: elaborado pelo autor (2024).

Figura 18 - Diagrama de Atividade Acessar Lixeira



Fonte: elaborado pelo autor (2024).

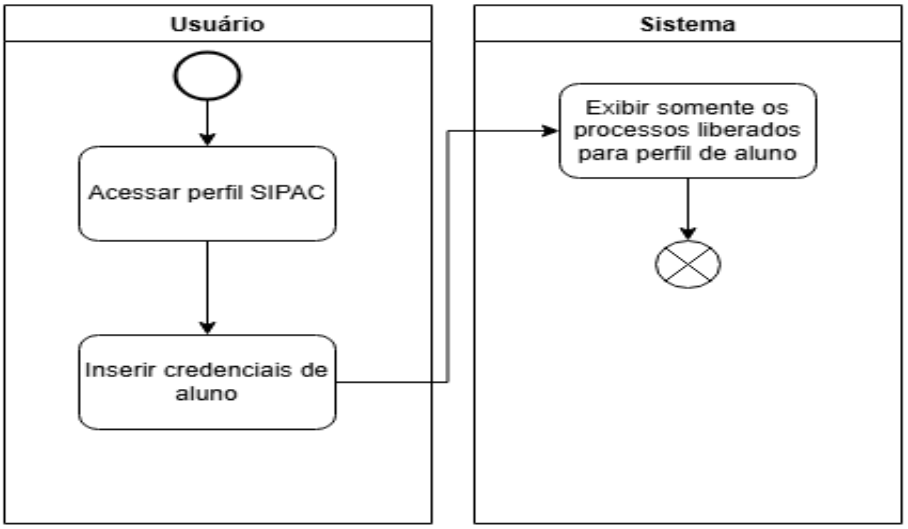
Figura 19 - Diagrama de Atividade Acessar Caixa de Saída



Fonte: elaborado pelo autor (2024).

Na Figura 20 é apresentada a validação de restrições de perfis de acordo com as suas credenciais, nesse caso o usuário insere os dados de aluno no perfil administrativo e deverá ter acesso liberado somente a opções de perfil de discente.

Figura 20 - Diagrama de Atividade Validação de Restrições de Acesso por Perfil



Fonte: elaborado pelo autor (2024).

3.6 Configuração do Projeto Prático

Para realizar a automação de testes de processos do software é necessário preparar o ambiente com todas as ferramentas inicializadas e instaladas. Com isso, será inserido abaixo uma breve introdução dessa etapa inicial.

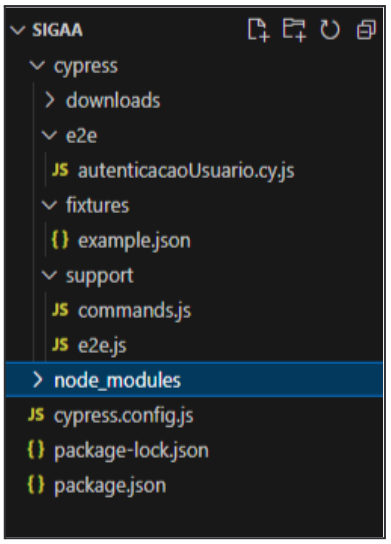
- Criar a pasta do projeto e abrir a mesma no Visual Studio Code;
- Inicializar um projeto Node através do comando npm init, que vai criar a pasta "package.json" com os dados do projeto;
- Instalar o cypress através do comando npm install -D cypress após finalizar a instalação será executado o Cypress, através do comando npx cypress open.

Na instalação são criados os arquivos:

- cypress.config.js: todas as configurações que podem ser realizadas com o framework;
- Pasta e2e: serão os arquivos de testes (spec).
- Pasta fixtures: arquivos que possuem script que geram dados para serem usados ao longo dos testes.
- Pasta support: para arquivos auxiliares, comandos ou abstração.

Na Figura 21 é possível visualizar todos os arquivos criados que foram citados acima após a correta inicialização e instalação do *node* e *cypress* respectivamente.

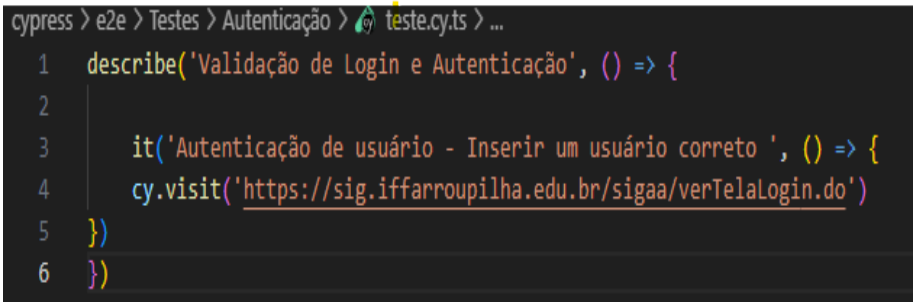
Figura 21 - Preparação do Ambiente



Fonte: Elaborado pela autora

Na Figura 22 abaixo, é ilustrado um caso de teste inicial, utilizando a linguagem JavaScript e comandos nativos do framework de automação.

Figura 22 - Exemplo de Criação de Teste



Fonte: Elaborado pela autora

De acordo com o projeto descrito neste trabalho, será realizada refatoração da linguagem nativa dos testes bem como do próprio framework, ou seja, serão convertidos os testes para Typescript e não utilizada os comandos nativos do framework diretamente, para que seja possível criar um projeto flexível a mudanças e ao uso de outras ferramentas.

Para mudar um projeto Cypress de JavaScript para TypeScript, é necessário realizar os seguintes passos:

- Instalar TypeScript no terminal do projeto:

npm install --save-dev typescript @types/node @types/cypress

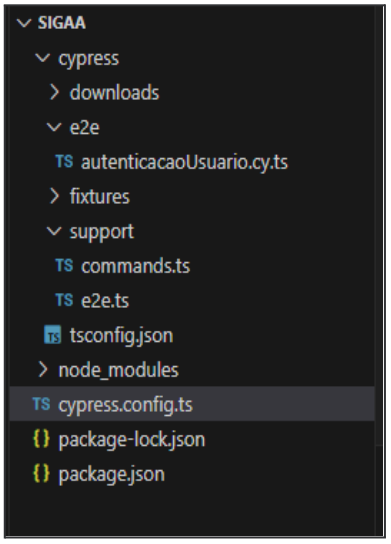
- Configurar o TypeScript: Criar um arquivo de configuração TypeScript tsconfig.json na raiz do projeto.

```
{
 "compilerOptions": {
  "target": "es5",
  "lib": ["es5", "dom"],
  "types": ["cypress"]
 },
 "include": [
  "**/*.ts"
 ]
}
```

- Renomear arquivos JS para TS;
- Converter código para TypeScript: Converta o código JavaScript existente para TypeScript, adicionando tipagem estática e ajustando a sintaxe conforme necessário.

Na Figura 23 é possível verificar a refatoração dos arquivos para a linguagem atual que será utilizada no decorrer do desenvolvimento.

Figura 23 - Projeto em Typescript



Fonte: Elaborado pela autora

É importante salientar que as ações sobre campos, combos e botões são informadas no código através da identificação do mesmo no html, o qual não possui um padrão para definição. Ou seja, em alguns casos, os mesmos podem ter um id específico, em outros, é necessário usar o *class*, *table*, *name*, *value*, entre outros.

O teste codificado na Figura 24 se refere a Validação de Login e Autenticação, o qual irá validar os dados de Usuário e Senha para acesso ao SIGAA.

Figura 24 - Codificação de Teste de Login e Autenticação

```
describe('Validação de Login e Autenticação', () => {

  it('Autenticação de usuário - Inserir um usuário incorreto ', () => {
    cy.visit('https://sig.iffarroupilha.edu.br/sigaa/verTelaLogin.do')
    cy.get('tbody > :nth-child(1) > td > input').type('03967013022')
    cy.get(':nth-child(2) > td > input').type('230914Diuli')
    cy.get('tfoot > tr > td > input').click()
  })
})
```

Fonte: Elaborado pela autora

Conforme citado neste projeto, para melhor a eficiência e qualidade da codificação e desenvolvimento do projeto são usadas técnicas de criação de interfaces, enumerados e ações pré definidas para serem usadas, conforme exemplo abaixo na Figura 25.

Figura 25 - Codificação de Teste de Login e Autenticação com otimização de código

```
import { AutenticacaoUsuario } from "../Login/autenticacaoUsuario"
import { acessoSIGAA } from "./autenticacaoUsuario.data"

describe('Validação de Login e Autenticação', () => {
  const autenticacaoUsuario: AutenticacaoUsuario = new AutenticacaoUsuario()

  it('Autenticação de usuário - Inserir um usuário incorreto ', () => {
    autenticacaoUsuario.autenticarUsuario({dados: acessoSIGAA})
  })
})
```

Fonte: Elaborado pela autora

4 RESULTADOS E DISCUSSÕES

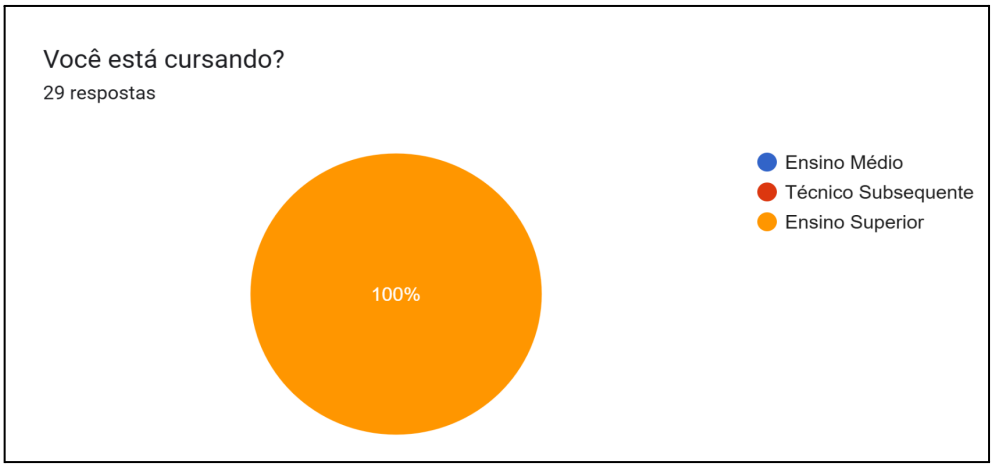
Nesta seção, serão discutidos os principais resultados do estudo, destacando-se as interpretações, suas implicações e possíveis limitações. Além disso, serão comparados os resultados encontrados com os de outros estudos relevantes da área, permitindo uma análise crítica e aprofundada. Através dessa abordagem, busca-se não apenas apresentar as evidências coletadas, mas também proporcionar uma reflexão que contribua para o entendimento do tema proposto.

Para facilitar a compreensão e complementação da pesquisa, foi desenvolvido um formulário com questões abrangentes relacionadas ao sistema educacional SIGAA. O mesmo visa coletar informações a fim de indicar a satisfação dos usuários com o software educacional utilizado na codificação dos casos de testes. Nele, os usuários inseriram as suas opiniões de forma a garantir a importância da qualidade e ajustes no SIGAA.

A pesquisa foi enviada para os estudantes do IFFAR campus Santo Ângelo, no período dos meses de Setembro e Outubro de 2024. Ao total foram obtidas 29 respostas e abaixo serão apresentados os resultados obtidos.

A primeira questão tem o objetivo de mapear os usuários do SIGAA de maneira geral, considerando que cada grau de escolaridade utiliza o sistema de formas diferentes, podendo ter percepções abrangentes para levar em consideração no resultado final. Porém foram obtidas respostas somente de alunos do Ensino superior conforme apresentado na Figura 26.

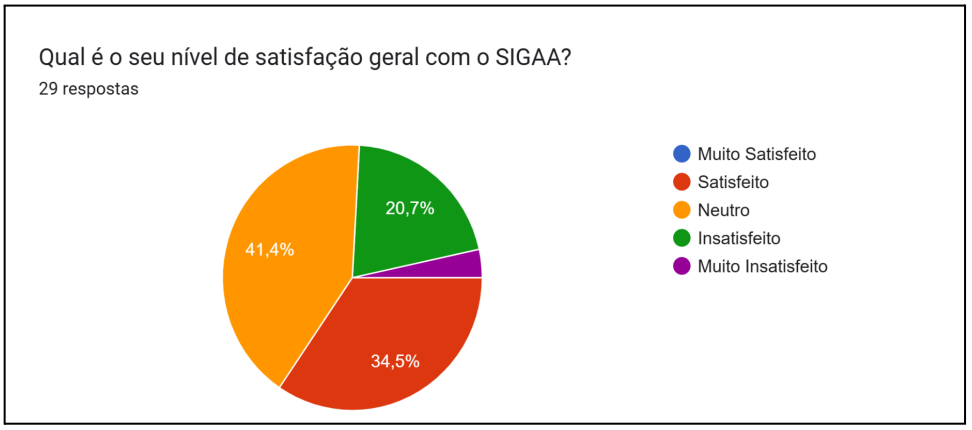
Figura 26 - Questão 1 - Grau de escolaridade



Fonte: Elaborado pela autora

Para validar a visão dos alunos referente ao SIGAA, na Figura 27 é exibida a questão relacionada ao nível de satisfação dos alunos com o sistema educacional, considerando as respostas, o maior percentual foi de 41,4% de alunos neutros em relação ao SIGAA e 34,5% de alunos satisfeitos.

Figura 27 - Questão 2 - Satisfação com o sistema educacional SIGAA

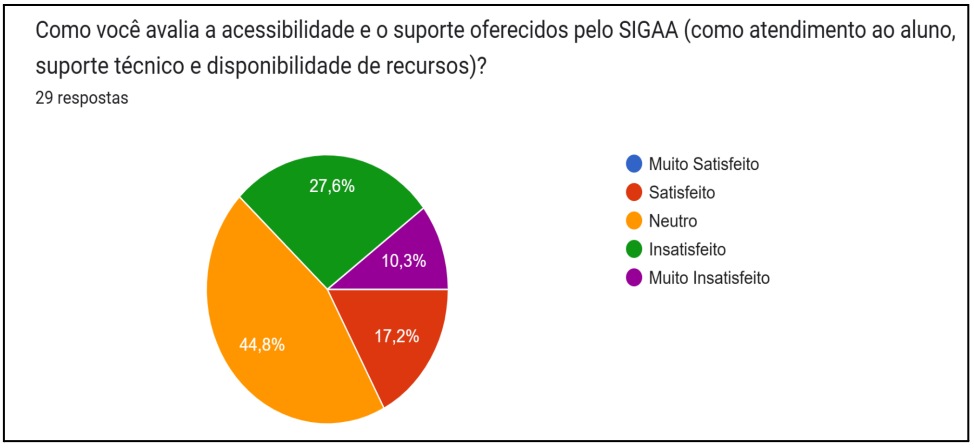


Fonte: Elaborado pela autora

As Figuras 28 e 29 tem ênfase em processos realizados dentro do sistema, os quais foram selecionados nesse projeto para realizar a automação. A ideia dessas duas questões surgiu para verificar se os casos de testes propostos iriam atender uma demanda a qual o aluno tem dificuldades dentro do SIGAA.

Na Figura 28 focada na acessibilidade e suporte do SIGAA, também obteve-se o maior número de resposta de alunos neutros com um total de 44,8% e em segundo lugar a opção de insatisfeito com 27,6% dos estudantes.

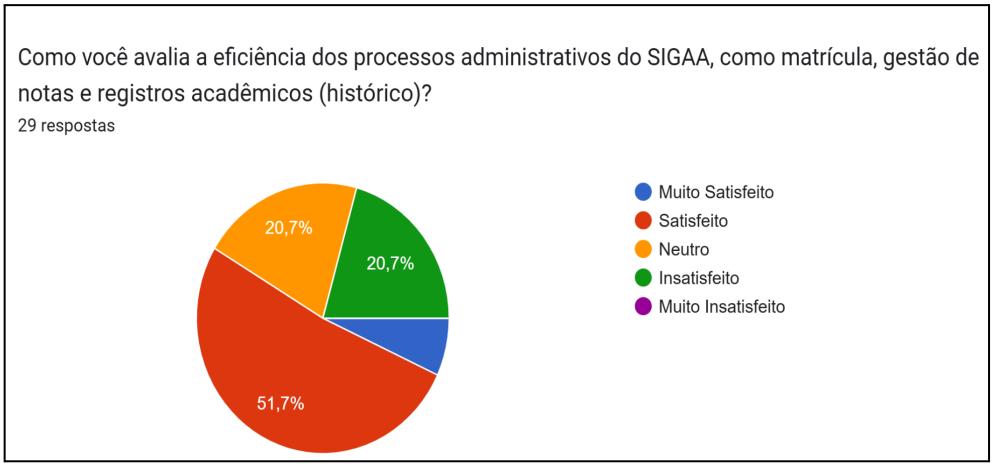
Figura 28 - Questão 3 - Avaliação de processos do SIGAA (Suporte)



Fonte: Elaborado pela autora

Na Figura 29 avaliando a eficiência em processos administrativos 51,7% dos alunos responderam como satisfeitos e 20,7% neutros e insatisfeitos.

Figura 29 - Questão 4 - Avaliação de processos do SIGAA (Administrativo)

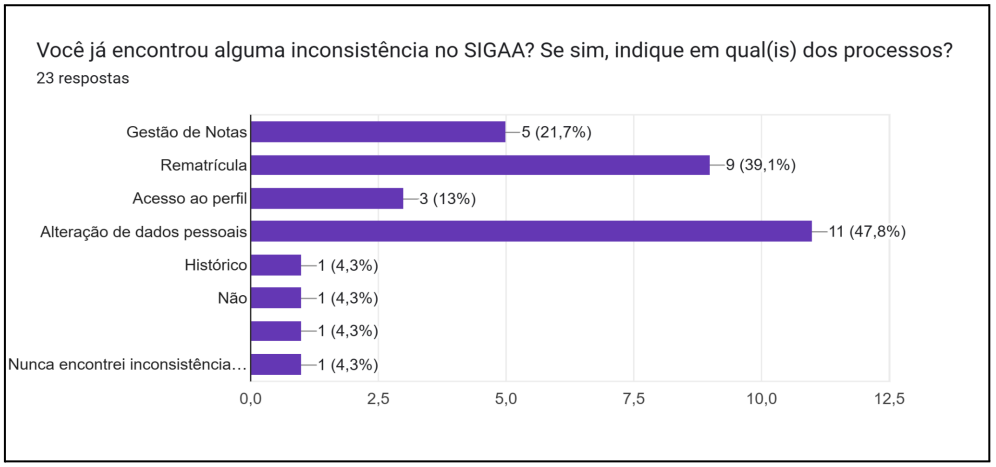


Fonte: Elaborado pela autora

Na Figura 30 são ilustradas as respostas da questão referente a inconsistências no sistema SIGAA. Nessa questão era possível indicar dentre os processos principais listados ou a opção de não encontrado nenhum problema ao utilizar o sistema.

Dentre os processos listados o maior percentual foi de problemas relacionados a alteração de dados pessoais com 47,8% e após a ação de rematrícula com 39,1%.

Figura 30 - Questão 5 - Inconsistência no SIGAA



Fonte: Elaborado pela autora

A Tabela 8 listada abaixo refere-se à última questão do formulário na qual o aluno poderia responder sem obrigatoriedade sobre erros ou observações gerais sobre o SIGAA.

Foram obtidas respostas importantes, sendo que as mesmas complementam situações dos casos de testes e inconsistências encontrados no decorrer do projeto e descritos no mesmo. Por exemplo, a funcionalidade de matrícula/rematrícula foi citada como problemática, sendo que os testes realizados neste trabalho também indicaram dificuldades em encontrar tais processos como selecionar turmas de demais cursos ofertados e também pelo processo somente ser executado em períodos pré definidos do ano letivo.

Tabela 8 - Questão 6 Opcional

<u>Questão opcional:</u> Nesse espaço você pode relatar brevemente o(s) erro(s) encontrado(s) ou deixar alguma observação sobre o SIGAA.
Difícil localização ao fazer a matrícula de cursos seletivos.
Percebo que o SIGAA tem uma demora muito grande para a aprovação perante aos novos professores, já estamos na metade do semestre e tem professores sem acesso ao SIGAA, o que atrapalha a forma com a qual os professores disponibilizam os conteúdos de aula e acabam optando por outros serviços, também vejo a falta de informações presentes no modo mobile, e em aspecto visual não vejo o SIGAA bem estruturado visualmente sendo de certo modo um visual "poluído", com muitos textos e informações não filtradas em uma única página.
Usava o SIGAA para acessar os materiais das aulas, rematrícula e emissão de histórico e alguns outros documentos, por isso não encontrei maiores problemas no meu uso.
Erro ao tentar adicionar uma foto ao perfil do aluno.
Tive problemas com a rematrícula, onde eu enviava mas dava erro. É também o SIGAA é muito pouco intuitivo e arcaico.
Ele tem um visual horrível e desatualizado. Não dá vontade de usar. Precisa urgentemente de uma revisão em termos de usabilidade e de experiência de usuário. A única coisa positiva nele é que guarda histórico das matérias e materiais de aula. Além das notas.
Pequeno erro mas ao informar, foi prontamente corrigido.
Travando a todo momento.
Ainda não foi alterada a foto do aluno.

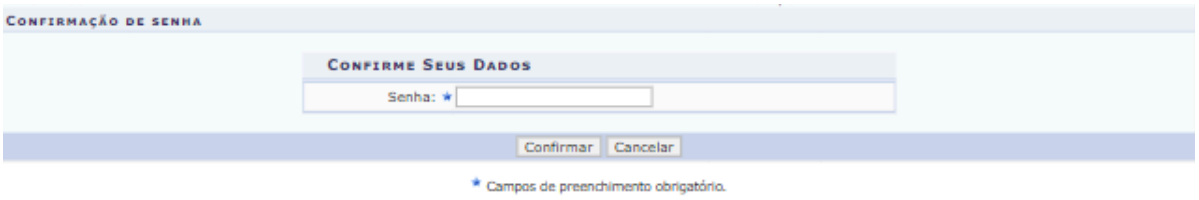
Fonte: Elaborado pela autora

4.1 Resultados Inconsistentes Validação Final

Neste tópico será relatado os resultados obtidos de forma errônea, os quais foram codificados planejados com um resultado esperado e ao executar foram exibidas diferentes apurações.

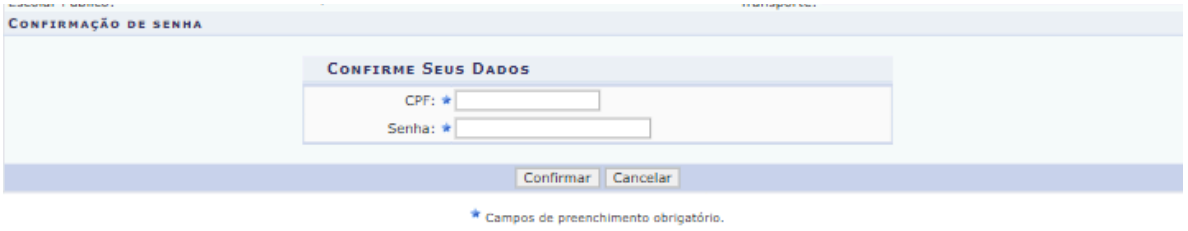
Nas Figuras 31,32 e 33 é possível observar que não possui um resultado válido para execução do Caso de Teste, pois o processo de confirmar os dados para edição de alunos não segue o padrão, solicitando sempre CPF e senha, em alguns casos pode solicitar somente a senha, ou até mesmo Data de Nascimento e senha. Considerando esses pontos, para a automação ele não seria um teste válido na execução, pois somente um deles iria apresentar sucesso na execução.

Figura 31 - Inconsistência no caso de teste Edição de dados do Aluno - Somente senha



Fonte: SIGAA

Figura 32 - Inconsistência no caso de teste Edição de dados do Aluno - CPF e senha



Fonte: SIGAA

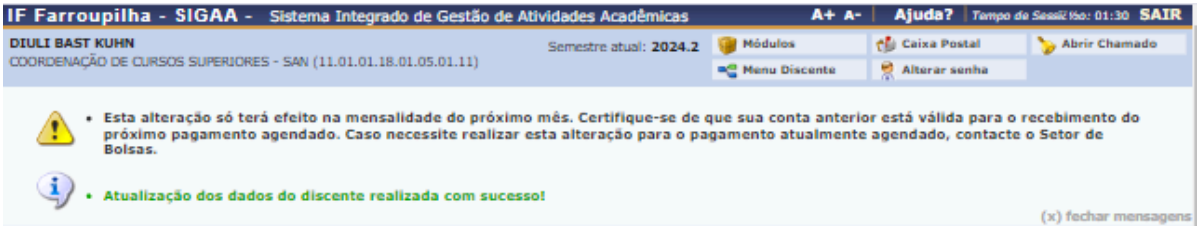
Figura 33 - Inconsistência no caso de teste Edição de dados do Aluno - Data de Nascimento e senha

Fonte: SIGAA

Na Figura 34 é possível verificar que ao realizar a confirmação da Edição dos Dados do Aluno é apresentada a mensagem de confirmação da edição, porém também é exibida uma mensagem equivocada:

"Esta alteração só terá efeito na mensalidade do próximo mês. Certifique-se de que sua conta anterior está válida para o recebimento do próximo pagamento agendado. Caso necessite realizar esta alteração para o pagamento atualmente agendado, contacte o Setor de Bolsas."

Figura 34 - Inconsistência no caso de teste Confirmação de Edição de dados do Aluno



Fonte: SIGAA

4.1.2 Casos de Teste com Pendências na Execução

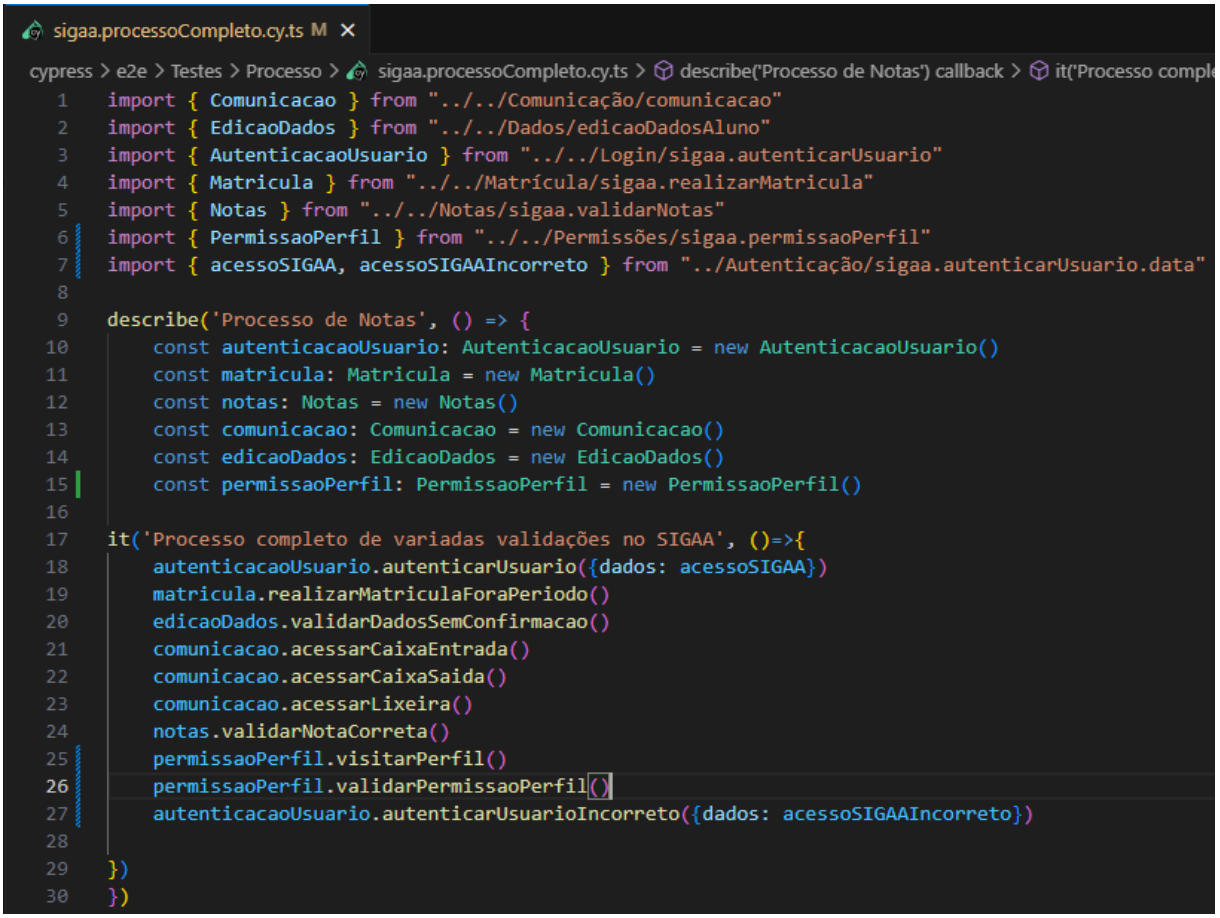
Além dos testes que apresentam falhas durante a execução, existem aqueles que enfrentam impedimentos em determinados períodos. Por exemplo, os testes de matrícula/rematrícula, que só poderão ser validados novamente quando o prazo para esse processo for aberto. Caso contrário, ao serem executados fora desse período, os testes falham, pois não encontram as validações desenvolvidas.

Dessa forma, para ter uma validação referente a esse processo, foi codificado a validação de mensagem de impedimento de matrícula, para que não seja permitido acessar o processo em períodos diferentes dos estabelecidos pela instituição.

4.2 Resultados Consistentes Validação Final

Dentre os quinze casos de testes definidos para codificação, a maior parte obteve uma boa execução e um resultado satisfatório. Abaixo na Figura 35 será apresentado um exemplo de teste a ser executado com a inclusão de vários casos de testes de diferentes processos, os quais apresentam sucesso ao final da sua execução e validam uma grande parcela da quantidade total.

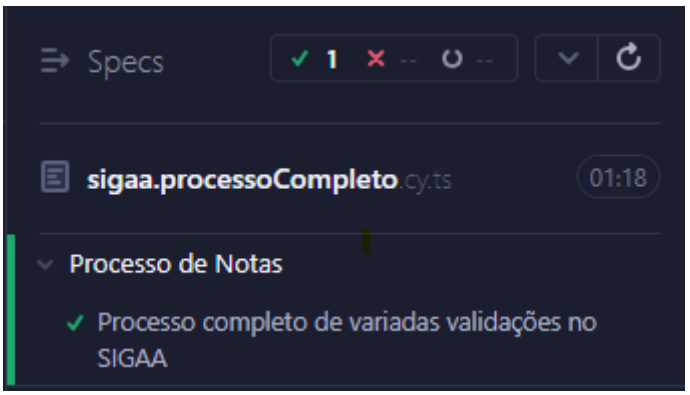
Figura 35 - Processo Completo



Fonte: Elaborado pela autora

Na Figura 36 é possível visualizar o resultado da execução do teste codificado na Figura 35, o qual possui a validação de vários processos que foram definidos no projeto.

Figura 36 - Execução do Processo Completo



Fonte: Elaborado pela autora

4.3 Avaliação dos testes

Conforme os requisitos estabelecidos na seção de desenvolvimento deste projeto, as análises a seguir abordam os aspectos que atenderam às expectativas propostas.

4.3.1 Tempo de Execução

Em relação ao tempo de execução dos casos de teste, foram realizadas análises em datas previamente definidas, considerando a possibilidade de variação na quantidade de usuários logados no SIGAA. Os testes foram, então, executados durante as datas dos conselhos de classes (Conselho intermediário 30/09/2024 - 04/10/2024 e Conselho de classe 14/10/2024 - 18/10/2024) para verificar se haveria alguma alteração no tempo de execução.

Como resultado, a diferença observada nos tempos de execução foi mínima, variando entre 2 a 3 segundos, em comparação aos dias de uso normal e horários de menor movimento. Essa variação pode ser atribuída, também, a fatores internos do framework utilizado. Dessa forma, conclui-se que o SIGAA apresenta uma qualidade notável, mantendo sua performance consistente, mesmo com um maior número de usuários simultâneos, sem comprometer significativamente os tempos de execução.

4.3.1.2 Manutenção dos Casos de Teste

Após a finalização da codificação dos casos de teste, foram atendidos de forma significativa os princípios de clean code e SOLID. O código foi devidamente comentado, com a separação de arquivos e criação de métodos específicos para garantir a reutilização e modularização.

Em alguns casos, não foi possível utilizar os métodos e funções previamente codificados. Nesses casos, foi necessário recorrer exclusivamente ao código nativo da ferramenta para garantir que os casos de teste fossem executados corretamente, sem falhas.

4.3.1.3 Recomendações e Sugestões

Durante o desenvolvimento dos testes, a maior dificuldade encontrada foi a ausência de identificadores (IDs) específicos nos campos do sistema, além da presença de múltiplos submenus e opções agrupadas em um mesmo menu. Como resultado, em alguns casos foi necessário adotar outras alternativas para validar os processos, já que a estrutura inicial não atendia de forma ideal os objetivos do projeto.

Com base nas observações feitas, as principais recomendações para aprimorar o sistema incluem:

- Melhoria na Organização dos Menus:

Realizar uma melhor divisão das opções nos menus, com uma organização mais clara e separada, para evitar confusão e facilitar a navegação.

- Ajustes em Processos:

De acordo com o feedback dos usuários, há dificuldades em localizar disciplinas de outros cursos e problemas com a inclusão da foto de perfil. Ajustes nesses processos poderiam melhorar a experiência do usuário.

- Inclusão de IDs Específicos no HTML:

Para facilitar a automação e a codificação de futuros casos de teste, seria de grande importância que fossem incluídos identificadores exclusivos nos campos HTML, permitindo uma seleção mais precisa e eficiente.

5 CONCLUSÃO

A implementação de uma estratégia de automação de testes no SIGAA demonstrou ser uma abordagem importante para otimizar o processo de desenvolvimento do software. Ao realizar o levantamento dos casos de teste essenciais e focar em funcionalidades críticas, como autenticação, gestão de alunos, matrícula e comunicação interna, foi possível garantir a cobertura de testes nas áreas mais utilizadas e sensíveis do sistema. Além disso, a aplicação do questionário aos usuários do sistema educacional, complementou a abordagem, pois os mesmos relataram problemas nas funcionalidades escolhidas.

A utilização do Cypress como ferramenta de automação proporcionou uma execução de testes mais eficiente e padronizada, permitindo a obtenção de resultados consistentes e confiáveis. A coleta e a análise dos dados obtidos ao longo do processo forneceram insights importantes sobre o desempenho e a estabilidade do SIGAA, contribuindo para melhorias contínuas no ciclo de desenvolvimento.

Com isso, este trabalho alcança seu objetivo principal ao demonstrar que a automação de testes é uma estratégia eficaz para assegurar a qualidade do software educacional, tornando o desenvolvimento mais ágil e assertivo.

Com base nos resultados e na experiência adquirida com a implementação da automação de testes no SIGAA, algumas sugestões para trabalhos futuros incluem, aumentar a cobertura de testes para abranger outras funcionalidades críticas do SIGAA, como gestão administrativa, controle de frequência e geração de relatórios acadêmicos. Isso fortaleceria a robustez do sistema, incluindo áreas adicionais que estejam devidamente testadas.

Além disso, também sugere-se implementar a automação de testes como parte de um pipeline de Integração Contínua (CI/CD), facilitando a execução automática dos testes em cada atualização ou alteração no código. Essa integração proporciona feedback rápido para os desenvolvedores, permitindo a detecção e a correção de falhas mais cedo no processo de desenvolvimento.

Por fim, ainda como trabalho futuro, pode-se citar a avaliação de outras ferramentas de automação além do Cypress, como Playwright ou Selenium, por exemplo. Nessa avaliação seria possível explorar as potencialidades e identificar alternativas que possam atender a requisitos específicos do SIGAA ou se integrar melhor ao fluxo de trabalho da equipe, já que em alguns casos o Cypress não atendeu os requisitos dos casos de testes como foram descritos inicialmente.

REFERÊNCIAS

BARTIÉ, Alexandre. **Garantia da Qualidade de Software.** Rio de Janeiro: Elsevier, 2002
Autor: Robert C. Martin. Título do Livro: Clean Code: A Handbook of Agile Software Craftsmanship. Disponível em: https://github.com/jnguyen095/clean-code/blob/master/Clean.Code.A.Handbook.of.Agile.Software.Craftsmanship.pdf. Acesso em: [Abril 2024]

BARTIÉ, Alexandre. **Garantia da Qualidade de Software.** Rio de Janeiro: Elsevier, 2002
SOMMERVILLE, Ian. Engenharia de Software. São Paulo: Person, 2011.Acesso em: [Março 2024]

BDD **(Desenvolvimento orientado por comportamento).** Disponível em: <https://www.devmedia.com.br/desenvolvimento-orientado-por-comportamento-bdd/21127#1>. Acesso em: [Junho 2024].

CYPRESS. **JavaScript End to End Testing Framework.** Disponível em: <https://www.cypress.io/>.
Why Cypress? Disponível em: <https://docs.cypress.io/guides/overview/why-cypress>. Acesso em: [Maio 2024].

DAMAZIO, R. (2007). **[Requisitos de Métodos de Garantia da Qualidade no Desenvolvimento de Software].** Disponível em: https://homepages.dcc.ufmg.br/~rodolfo/dcc823-2-07/Entrega4/Damazio4.pdf. Acesso em: [Março 2024].

DARIELSON, Ana Carolina. **"Estudo Comparativo de Testes" [Comparative Study of Tests].** Revista Científica Multiversa, [12/2022]. Acesso em: [Abril 2024].

DELAMARO, Marcio; JINO, Mario; MALDONADO, José. **Introdução a Teste De Software.** Rio de Janeiro: Elsevier, 2013.

FRÓES, Gabriel e Weber, Vanessa. CÓDIGO FONTE TV. **Clean Code // Dicionário do Programador.** Disponível em: <https://www.youtube.com/watch?v=ln6t3uyTveQ>. Acesso em: [Março 2024].

FRÓES, Gabriel e Weber, Vanessa. CÓDIGO FONTE TV. **SOLID (O básico para você programar melhor) // Dicionário do Programador.** Disponível em: <https://www.youtube.com/watch?v=mkx0CdWiPRA>. Acesso em: [Março 2024].

LOURENÇO, Rony. **[Automação de testes para um sistema de e-commerce].** (Dissertação de Bacharelado), [Universidade Federal do Rio Grande do Norte(UFRN)], 2022). Acesso em: [Agosto 2024].

PRESSMAN, Roger S.; MAXIM, Bruce R. **Engenharia de software: uma abordagem profissional.** 8 ed. Porto Alegre: AMGH, 2016. Acesso em: [Março 2024].

SOUZA, Maria Naires Alves de; MONTEIRO, André Jalles. **Os docentes da Universidade Federal do Ceará e a utilização de alguns dos recursos do sistema integrado de gestão de**

atividades acadêmica (SIGAA). Ensaio: aval. pol. públ. Educ., Rio de Janeiro, v. 23, n. 88, p. 611-630, jul./set. 2015. SIG - IFFar. Disponível em: <https://www.iffarroupilha.edu.br/tecnologia-da-informacao-dti/sig#vis%C3%A3o-geral>. Acesso em: [Abril 2024].

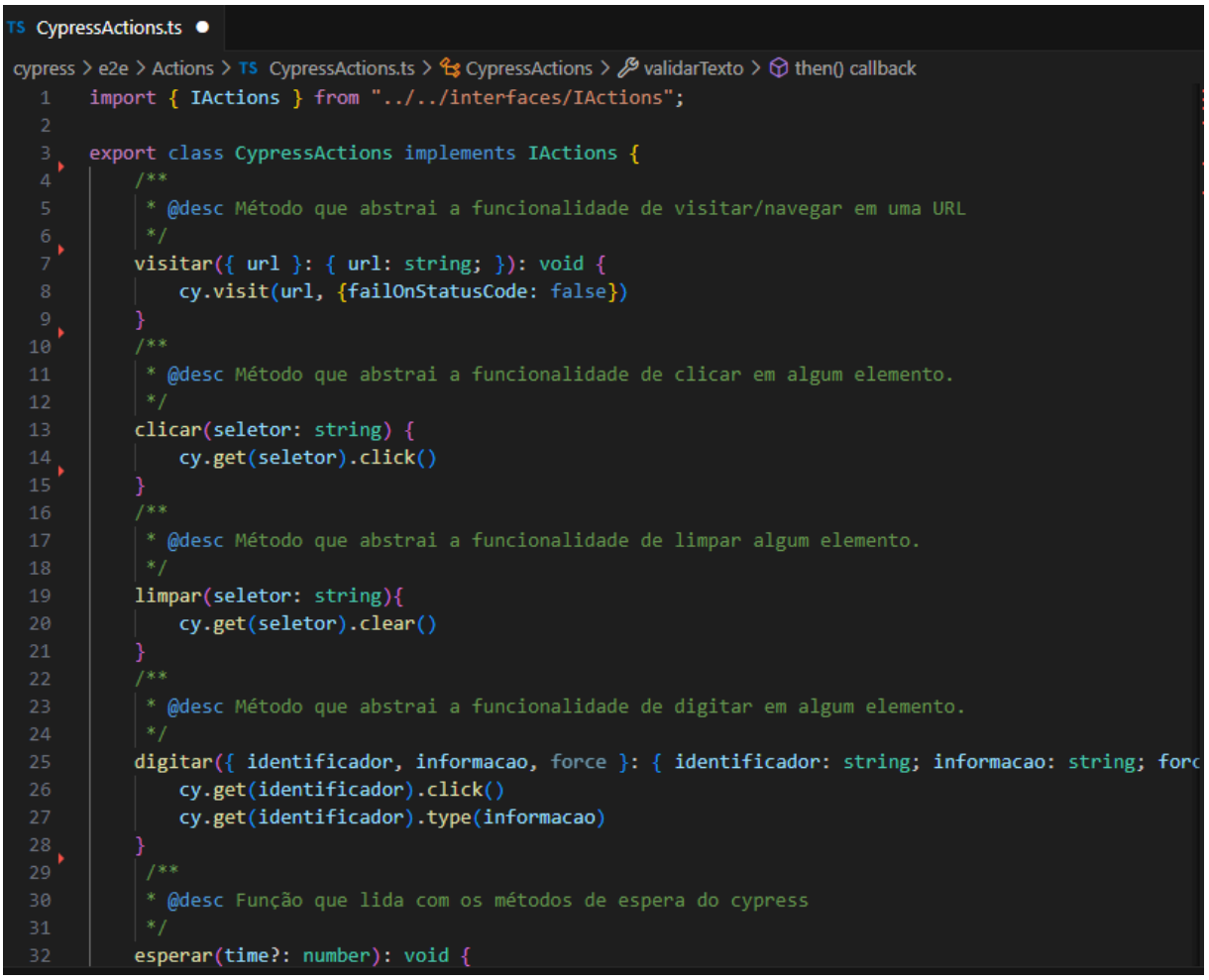
TEIXEIRA, Eleonora. **[Análise de ferramenta de automação de testes de software].** (Dissertação de Tecnologia), [Universidade Federal de Santa Maria (UFSM)], 2022). Acesso em: [Abril 2024].

TYPESCRIPT. **TypeScript Documentation.** Disponível em:<https://www.typescriptlang.org/docs/>. Acesso em: [Maio 2024].

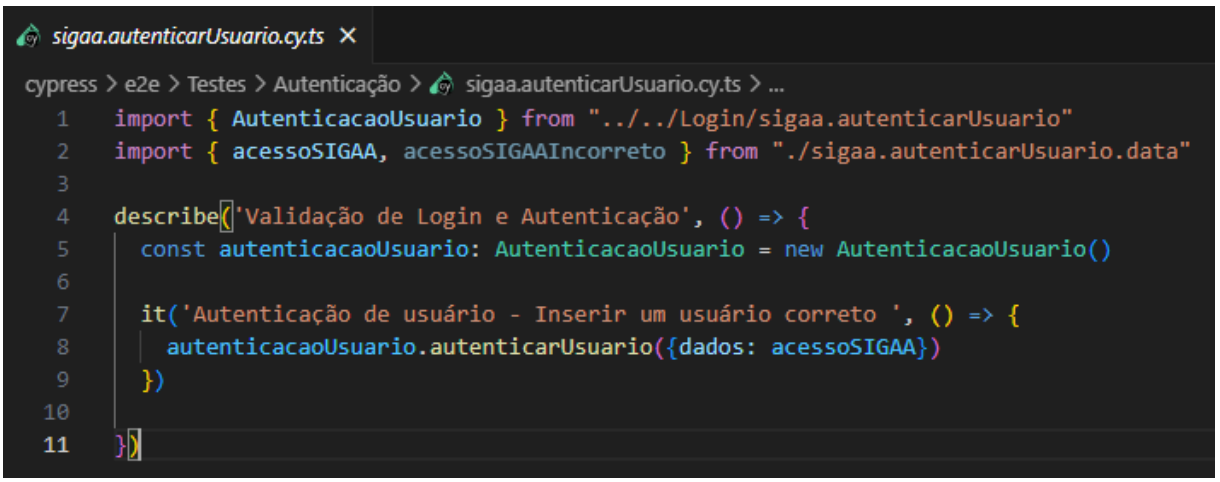
WAZLAWICK, R. S. **Metodologia de pesquisa para ciência da computação**. Rio de Janeiro: Elsevier, 2009. 124 p. Acesso em: [Junho 2024].

APÊNDICE A : CODIFICAÇÃO CASOS DE TESTES

Ações - arquivo com funções úteis para utilizar nos testes em geral.



Autenticação de Usuário - arquivo executável



Autenticação de Usuário - arquivo .data (armazenar dados estáticos)

sigaa.autenticarUsuario.data.ts

cypress > e2e > Testes > Autenticação > sigaa.autenticarUsuario.data.ts > acessoSIGAAIncorreto

```ts
import { IAutenticacaoUsuario } from "../../../interfaces/IAutenticacaoUsuario"

export const acessoSIGAA: IAutenticacaoUsuario = {
    login: '03967013022',
    senha: '**********'
}
export const acessoSIGAAIncorreto: IAutenticacaoUsuario = {
    login: '1213551515131',
    senha: '54654654'
}
```

Autenticação de Usuário - classe para descrever as ações

sigaa.autenticarUsuario.ts M

cypress > e2e > Login > sigaa.autenticarUsuario.ts > AutenticacaoUsuario > autenticarUsuarioIncorreto

```ts
import { IAutenticacaoUsuario } from "../../interfaces/IAutenticacaoUsuario";
import { CypressActions } from "../Actions/CypressActions";

export class AutenticacaoUsuario extends CypressActions{
//método definido dentro de um objeto ou classe para descrever uma função que está associada.
    autenticarUsuario({dados}:{dados: IAutenticacaoUsuario}){
        this.visitar({url:'https://sig.iffarroupilha.edu.br/sigaa/verTelaLogin.do'})
        this.informarDadosAutenticacao({dados})
        this.clicar('tfoot > tr > td > input')
    }
    private informarDadosAutenticacao({dados}:{dados: IAutenticacaoUsuario}){
        this.clicar('tbody > :nth-child(1) > td > input')
        this.digitar({identificador: 'tbody > :nth-child(1) > td > input', informacao: dados.login})
        this.clicar(':nth-child(2) > td > input')
        this.digitar({identificador: ':nth-child(2) > td > input', informacao: dados.senha})
    }

    autenticarUsuarioIncorreto({dados}:{dados: IAutenticacaoUsuario}){
        this.visitar({url:'https://sig.iffarroupilha.edu.br/sigaa/verTelaLogin.do'})
        this.informarDadosAutenticacao({dados})
        this.clicar('tfoot > tr > td > input')
        cy.get('[style="color: #922; font-weight: bold;"]').contains('Usuï¿½rio e/ou senha invï¿½li
    }

}
```

Autenticação de Usuário - Interface (formato de objetos e tipos de dados utilizados)

IAutenticacaoUsuario.ts

cypress > interfaces > IAutenticacaoUsuario.ts > IAutenticacaoUsuario

```ts
export interface IAutenticacaoUsuario {
    login: string;
    senha: string;
}
```