

Universidade do planalto catarinense

Curso de sistemas de informação

(Bacharelado)

dienisson de almeida chinelatto

análise comparativa entre as diferentes abordagens da automação de testes

Lages (SC)

2019dienisson de almeida chinelatto

análise comparativa entre as diferenes abordagens de automação de testes

Trabalho de Conclusão de Curso submetido à Universidade do Planalto Catarinense para obtenção dos créditos de disciplina com nome equivalente no curso de Sistemas de Informação - Bacharelado.

Orientação: Prof.ª Thiago Sartor, M.Sc.

Lages (SC)

2018

dienisson de almeida chinelatto

análise comparativa entre as diferentes abordagens da automação de testes

Este relatório, do Trabalho de Conclusão de Curso, foi julgado adequado para obtenção dos créditos da disciplina de Trabalho de Conclusão de Curso, do 8º. semestre, obrigatória para obtenção do título de:

Bacharel em Sistemas de Informação

Lages (SC), 27 de novembro de 2019

|  |
| --- |
| Prof. Thiago Sartor, M.Sc.  **Orientador** |

Banca Examinadora:

|  |  |
| --- | --- |
| Prof. Sabrina Bet Sagaz Koerich, M.Sc.  **UNIPLAC** | Prof. Madalena Pereira da Silva, Dra.  **UNIPLAC** |

|  |
| --- |
| Prof. Sabrina Bet Sagaz Koerich, M.Sc / Madalena Pereira da Silva, Dra.  **Coordenadora de Curso / Professora de TCC** |

Lista de Ilustrações

FIGURA 1 - Black Box Testing Approach 22

FIGURA 2 - White Box Testing Approach 23

LISta de abreviaturas e Siglas

QA - *Quality Assurance*

Resumo

**Palavras-chave**:

Testes de Software; Qualidade de Software; Automação.

Abstract

**Keywords**:

Software Testing, Software Quality, Automation.

Sumário

1 INTRODUÇÃO 10

1.1 Apresentação 10

1.2 Descrição do problema 11

1.3 Justificativa 12

1.4 Objetivo geral 13

1.5 Objetivos específicos 13

1.6 Metodologia 14

2 REFERENCIAL TEÓRICO 15

2.1 Testes e qualidade de software 15

2.1.1 A necessidade do teste de software 15

2.1.2 A importância da QA nas Organizações 17

2.1.3 Definição de Teste 18

2.2 Técnicas de Teste de Software 20

2.2.1 Modelo Caixa-Preta 21

2.2.2 Modelo Caixa-Branca 22

2.3 Fases de teste de Software 23

2.3.1 Teste de Unidade 24

2.3.2 Teste de Integração 25

2.3.3 Testes de Sistema 26

2.3.4 Testes de Aceitação 27

2.3.5 Testes de Regressão 27

2.4 Outros tipos de teste de software 28

2.4.1 Testes de Desempenho 28

2.4.2 Testes de Carga 29

2.4.3 Testes de Stress 29

2.5 Automação de Testes 30

2.5.1 Propostas e Dúvidas da Automação de Testes 30

2.5.2 Quando não automatizar? 32

2.5.3 Casos para Automação de Testes 32

2.6 Métodos Ágeis 33

2.6.1 Scrum 34

3 MODELAGEM DE DADOS 35

*3.1 Product Backlog* 35

3.2 Histórias de usuários 35

3.3 Requisitos funcionais 37

3.4 Requisitos não funcionais 40

3.5 Modelagem conceitual 42

3.6 Modelagem lógica 43

3.7 Conclusão 43

4 IMPLEMENTAÇÃO DO SISTEMA 45

4.1 Funcionalidades 45

4.1.1 Apresentação do sistema 45

4.2 Conclusão 69

4.3 Validação 70

4.3.1 Teste do sistema 70

4.3.2 Resultado da pesquisa de avaliação 74

5 Considerações finais 83

Referências Bibliográficas 85

1. INTRODUÇÃO
   1. Apresentação

A crescente demanda por desenvolvimento de *softwares*, com prazos cada vez mais arrojados, leva as organizações a buscar maior qualidade dos sistemas desenvolvidos, sua conformidade, e cumprimento de prazos de entrega pré-estabelecidos. Em resposta a esta dinâmica, com o crescimento das metodologias ágeis, diversas técnicas e modalidades de testes de *software* surgiram, consolidando a prática como uma das fases essenciais do desenvolvimento de sistemas. Myers (2004, p. 2) define seu conceito como: [...] “O teste de *software* é um processo, ou uma série de processos, projetados para certificar-se de que o código do computador faça o que foi projetado para fazer e que ele não faça nada sem intenção”.

A execução de testes de *software* está fortemente relacionada com a qualidade final do produto desenvolvido. Os testes são, uma das alternativas que as equipes de desenvolvimento possuem para obter um produto com um número mínimo de defeitos. Entretanto, embora importante, as atividades de testes custam tempo, e se tratando de *software*, tempo pode ser igual a custos para a organização, além do mais, sistemas que são desenvolvidos utilizando um ciclo de vida incremental, normalmente trazem a necessidade de realização constante de testes em funcionalidades já entregues para os usuários. O objetivo desse ciclo é garantir que a implementação de novos requisitos não tenha inserido defeitos em funcionalidades existentes.

Atualmente, não podemos falar sobre testes sem falar sobre automação de teste. A aplicação da automação de teste dá um feedback rápido ao time e mantém a execução dos testes de regressão continuamente. Quando a automação de testes é usada corretamente pode trazer vários benefícios para uma organização. Segundo Cem Kaner, autor do livro "*Lessons Learned in Software Testing*", [...] “o propósito da automação de testes pode ser resumidamente descrito como a aplicação de estratégias e ferramentas tendo em vista a redução do envolvimento humano em atividades manuais repetitivas.”

Entretanto, é importante considerar que ela não pode ser usada em todos os casos. Automação de testes envolve maiores custos iniciais e deve ser tratado como um investimento a longo prazo. É preciso analisar cada projeto para definir quais processos realmente convém automatizar e investir esforço. E considerar que, mesmo se decidir usar um método de automação de teste, ainda será necessário utilizar o teste manual para diversas funcionalidades.

O TCC está organizado em capítulos. O primeiro capítulo aborda o projeto. Os demais capítulos serão definidos posteriormente.

* 1. Descrição do problema

Atualmente na área de desenvolvimento de *software* é comum testar o sistema com diferentes técnicas e em diferentes fases, mas partindo desse princípio, ainda há muitas incógnitas quanto à o que está sendo testado, qual método deveria ser usado, qual o seu real retorno, e o que se espera que ele retorne.

Há riscos que precisam ser considerados antes de introduzir uma prática de teste na organização, a maioria desses riscos está associada com a falha na estimativa quanto ao esforço necessário para implementar e manter a automação. Existe a necessidade de apresentar quais os desafios e os benefícios da automação de testes nos seus diferentes níveis, auxiliando assim na tomada de decisão de quando vale a pena automatizar.

Na pirâmide de teste descrita pelo autor Mike Cohn (2009) ele cita: “[...] que deve-se concentrar maior esforço no nível de unidade e depois reduzir esse esforço nos próximos níveis.” Mas ainda assim não se pode chegar a uma conclusão de qual seria a carga de testes considerada ideal em cada nível. É realmente necessário automatizar tudo? Se uma funcionalidade já está coberta por um teste em um nível é necessário testar em outro?

* 1. Justificativa

Hoje no mercado de desenvolvimento de *software* com a popularidade das práticas ágeis cada vez mais se busca ter no final do processo um produto qualificado, com um número mínimo de falhas, em que suas características e funcionalidades cumpram os requisitos inicialmente estipulados pelo mesmo, tudo isso tentando reduzir cada vez mais os custos. Mas garantir a qualidade de *software* depende diretamente de qualificar o processo de desenvolvimento e suas abordagens.

Muitas empresas fracassam na implantação da automação de testes em virtude da inversão de prioridades causada pela busca da qualidade a qualquer preço. Neste cenário, as empresas adquirem uma ferramenta de automação de testes prematuramente sem, ao menos, possuírem um grau mínimo de maturidade no processo de testes. O processo de testes é informal, as responsabilidades não são bem definidas e os profissionais não possuem o perfil adequado ou não são qualificados. (Caetano, 2016).

De acordo com um estudo conduzido pelo NIST em 2002, os defeitos resultam num custo anual de 59,5 bilhões de dólares à economia dos Estados Unidos. Mais de um terço do custo poderia ser evitado com melhorias na infraestrutura do teste de *software*. Um dado um tanto quanto antigo em uma indústria que se atualiza muito rápido, contudo ainda há dados mais recentes que comprovam que ainda existe problemas relacionados a planejamento de testes. “Os gastos de TI em garantia de qualidade (QA) e testes aumentaram para uma média de 35% do gasto total de TI”, de acordo com o Relatório de Qualidade Mundial Capgemini (2015). Este é um sinal claro de que a abordagem de teste não está no nível de maturidade e eficiência necessária para suportar as complexas operações de TI de hoje.

Basicamente os testes devem provar que algo funciona ou não, e são essenciais para o sucesso do processo de qualidade de *software*. “Os testes unitários podem resolver entre 30% a 50% dos defeitos dos programas, os testes de sistemas podem remover entre 30% a 50% dos defeitos remanescentes, desse modo, os sistemas podem ir para produção ainda com aproximadamente 49% dos defeitos. Por último, as revisões de códigos podem reduzir entre 30% e 30% desses defeitos.” Myers in Bastos (2002).

De acordo com a Capgemini (2015) [...] “as organizações mais avançadas em sua jornada de transformação digital parecem gastar mais na QA e nos testes, indicando que estabeleceram uma conexão direta entre a qualidade e a obtenção de resultados comerciais”. Ou seja, organizações que estão focadas em tecnologias relativamente novas no mercado estão investindo fortemente em testes, consequentemente tendo custos mais elevados.

A rigor, o sucesso na implantação da automação de testes depende do entendimento de que a automação de testes ou uma ferramenta de automação, por si só, não vão melhorar ou organizar os testes existentes. É necessário antes, definir uma bateria de testes com um plano definido em cada nível e assim implantar um processo de testes formal para se obter sucesso.

* 1. Objetivo geral

Realizar uma análise comparativa, a fim de chegar em uma conclusão sobre qual é o esforço ideal e viável do investimento em testes automatizados nos seus diferentes níveis, assim gerando menores custos para as organizações.

* 1. Objetivos específicos

1. Propor uma solução que venha a sanar o problema do desperdício de esforço em testes automatizados.
2. Apresentar dados que auxiliem a automação de forma que seja uma prática para trazer ganhos no processo.
3. Traçar um perfil de investimento em testes automatizados, no qual mostrará o esforço ideal e viável.
   1. Metodologia

O desenvolvimento do projeto visa a formalização de uma estratégia de testes automatizados considerada ideal para um cenário de *software*. Sendo assim, buscou se o aprofundamento do assunto e conceitos através de estudos sobre testes e qualidade de *software*, a importância dos mesmos dentro do processo de desenvolvimento de *software*, definições, automação, diferenças entre tipos, técnicas e níveis de testes, bem como tecnologias usadas.

Após essas definições, iniciar-se-á o processo de desenvolvimento, um sistema que irá servir como base para o a definição dessa estratégia. Isso incluirá a especificação e desenvolvimento do sistema, sua arquitetura, bem como as regras de negócio nas quais guiarão a implementação dos testes.

Dando continuidade ao projeto, após a implementação do sistema, se iniciará a fase de implementação da estratégia de testes automatizados. Com base no estudos realizados, nas regras de negócio da aplicação será criado uma bateria de testes para esse cenário.

1. REFERENCIAL TEÓRICO
   1. Testes e qualidade de software
      1. A necessidade do teste de software

O objetivo do desenvolvimento de um *software* é a resolução de um problema real que existe no nosso mundo, através de meios tecnológicos, mais especificamente, através de comandos escritos que são interpretados por um computador. De acordo com Pressman (2011),

Software de computador é o produto que profissionais de software desenvolvem e ao qual dão suporte no longo prazo. Abrange programas executáveis em um computador de qualquer porte ou arquitetura, conteúdos, informações descritivas tanto na forma impressa como na virtual, abrangendo praticamente qualquer mídia eletrônica (2011, p. 29).

Ao analisar essa definição já é possível perceber que quando um desenvolvedor propõe-se a desenvolver um *software*, diversas falhas poderão surgir no andamento desse processo.

Em primeiro lugar, há casos em que a falha ocorre simplesmente na resolução do problema existente. O desenvolvedor pode vir a criar uma solução que resolva parte do problema analisado. Consequentemente, estaria ignorando outros cenários que, possivelmente, viriam a ocorrer durante a utilização da sistema por outros usuários.

Em segundo lugar, o erro pode ocorrer na escrita do código. As linguagens de programação, como o próprio nome define, são linguagens, e portanto envolvem ambiguidades. Muitos são os casos em que, no nosso dia-a-dia, nos deparamos com falhas no entendimento entre humanos por ambiguidade na comunicação. Portanto, é comum que isso ocorra também quando tentamos comandar um computador através desse mesmo meio.

Por último, é importante salientar que ao se escrever um *software* para resolver um problema mais elaborado, o grau de complexidade do código também aumenta. Dessa forma, em muitas ocasiões, o próprio desenvolvedor perde o rastro de todo significado de todas as linhas de seu código. Isso pode vir a ser um problema grave, quando efetuar uma modificação em algum pedaço do seu código. Isso pode vir a ser um problema relevante, quando ele efetuar uma alteração em algum pedaço do seu código e esquecer o impacto da mudança nas outras funções que ele programou. Em consequência, ele estaria introduzindo novos erros, os quais ele desconhece, e que só virão a ser identificados posteriormente pelo usuário final.

Em resumo, é evidente o alto risco de se produzir um *software* com defeitos. Esse risco é o que justifica a aplicação de testes no processo de desenvolvimento. Testar reduz esse risco.

É pensando em reduzir esse risco que se criaram os processos de “*Quality assurance*” (QA). Tais processos tem como objetivo produzir *softwares* de melhor qualidade, isto é, com o menor de índice de erros possível.

É claro que sempre existirão defeitos em um código. Por mais que se teste exaustivamente, nunca é possível verificar todas as possibilidades de entradas e saídas de uma aplicação. Porém, processos de qualidade visam testar um *software* com o objetivo de se obter um padrão mínimo de qualidade. Assim, o número de defeitos que seria encontrado pelos usuários finais é reduzido, gerando um *software* melhor.

Em suma, todos os processos de qualidade almejam criar o melhor *software* possível dentro de um tempo aceitável. Esse *software* ideal é facilmente entendido citando-se Bruce Sterling em “*The Hacker Crackdown”: “The best software is that which has been tested by thousands of users under thousands of different conditions, over years. It is then known as "stable." This does NOT mean that the software is now flawless, free of bugs. It generally means that there are plenty of bugs in it, but the bugs are well-identified and fairly well understood.”*

* + 1. A importância da QA nas Organizações

Empresas de desenvolvimento de *software* estão cada vez mais preocupadas em garantir a qualidade de seus produtos, para que os mesmos sejam competitivos, tornem-se satisfatórios e confiantes do ponto de vista de seus clientes.

As organizações estão buscando eficiência para conseguir sobreviver em um ambiente cada vez mais hostil – o de um mercado cada vez mais competitivo. As empresas estão buscando a tecnologia para reduzir custos e ampliar sua forma de atuação. Estão sofisticando seus sistemas para tomar decisões cada vez mais complexas, com a intenção de ganhar eficiência e controle (BARTIÉ, 2002).

Para Sommerville (2003), atingir um alto nível de qualidade de produto ou serviço deve ser atualmente o maior objetivo das organizações, pois o mercado não admite mais que se entregue produtos de baixa qualidade.

Bartié (2002) define qualidade de *software* como “um processo sistemático que focaliza todas as etapas e artefatos produzidos com o objetivo de garantir a conformidade de processos e produtos, prevenindo e eliminando defeitos”.

Para que a qualidade seja garantida, é necessário definir e estabelecer uma estrutura de procedimentos e de padrões organizacionais que devem ser utilizados durante o desenvolvimento do *software* que conduzem ao produto final de alta qualidade.

Existe uma ampla gama de atributos de qualidade de software em potencial que devem ser considerados durante o processo de planejamento de qualidade. Em geral, não é possível para qualquer sistema ser otimizado em relação a todos esses atributos, e, assim, uma parte importante do planejamento de qualidade é selecionar os principais atributos de qualidade e planejar como eles podem ser obtidos (SOMMERVILLE, 2003).

A norma ISO/IEC 9126 identifica seis atributos-chave que um *software* de qualidade deve possuir (PRESSMAN, 2006 *apud* PINTO JÚNIOR, 2009):

* Funcionalidade – o *software* deve funcionar adequadamente e atender as necessidades especificadas;
* Confiabilidade – o *software* deve funcionar por determinado período de tempo livre de erros e falhas;
* Usabilidade – o *software* deve ser compreendido e utilizado facilmente pelo usuário;
* Eficiência – o *software* deve atingir seu melhor desempenho utilizando seus recursos da melhor forma;
* Manutenibilidade – o *software* deve realizar modificações e reparos sem grandes dificuldades;
* Portabilidade – o *software* deve ser transferido facilmente de um ambiente (plataforma) para outro.

Outros atributos que contribuem para se ter um *software* de qualidade são “profissionais experientes e bem treinados, metodologias e ferramentas adequadas, participação constante dos usuários finais, bom entendimento do problema e modelagem da solução flexível ao longo prazo” (BARTIÉ, 2002).

A qualidade deve ser garantida em todas as etapas do processo de desenvolvimento do *software*, para que não seja iniciada uma nova fase sem que a anterior tenha sido concluída adequadamente.

Segundo Bartié (2002), os erros ocorrem em todo o processo de desenvolvimento de um sistema, porém, a maior incidência dos erros está concentrada nas fases iniciais, devido à má especificação e entendimento dos objetivos a serem alcançados.

Com o surgimento de novas tecnologias, as aplicações estão se tornando cada vez mais complexas e, em resultado, o mercado e a comunidade de teste de *software* também está evoluindo de forma a buscar a qualidade dos sistemas.

* + 1. Definição de Teste

O teste é uma das etapas do processo de desenvolvimento de um *software* que mais contribui para a garantia da qualidade do *software*. Testar um *software* consiste em executa-lo de acordo como foi especificado, para determinar se ele irá se comportar como o esperado no ambiente para qual ele foi projetado. O profissional que é responsável por essa função é conhecido como *tester*, ou testador. Cabe ao testador a dura missão de garantir a qualidade do software antes que vá para o usuário final ou cliente.

O principal objetivo do teste é encontrar erros no software antes que o mesmo seja entregue ao cliente. Conforme Myers (1979, apud BARTIÉ, 2002),

[...] se o objetivo do teste fosse apenas provar a boa funcionalidade de um aplicativo, seriam encontrados poucos defeitos, uma vez que toda a energia do processo de testes seria direcionada apenas na comprovação desse fato. Porém, se o objetivo for identificar erros, um maior número de problemas será encontrado, uma vez que os profissionais de qualidade buscarão vários cenários para avaliar o comportamento do software.

Para Inthurn (2001, apud ZIMMERMANN, 2006), o “teste de *software* tem como objetivo aprimorar a produtividade e fornecer evidências de confiabilidade e da qualidade do *software*”.

Através dos testes pode se considerar se o sistema em desenvolvimento está sendo feito de maneira correta e conforme os requisitos estabelecidos pelo usuário.

De acordo com Tomelin (2001), os testes avaliam a qualidade do *software* em todas as etapas do processo de desenvolvimento, desde a análise de requisitos até a fase de manutenção do *software*.

Para executar essa tarefa, um testador profissional precisa assumir uma atitude totalmente contraria a de um desenvolvedor. Enquanto o desenvolvedor tenta fazer seu *software* em construção funcionar perfeitamente, o *tester* possui um papel destrutivo, tentando de todas as formar, provar para o desenvolvedor que o programa possui falhas. Por esse motivo, não se recomenda que os testes sejam feitos pelo desenvolvedor.

Obviamente um desenvolvedor é capaz de assumir esse papel, porém ele assume uma visão otimista e conservadora ao criar seu *software*. Existe uma tendência lógica que faz com seu desenvolvedor acredite que suas modificações estão sempre aprimorando seu *software*, criando a solução perfeita para o problema inicial. Porém, isso é apenas uma suposição que se não for devidamente testada, não agrega nenhuma qualidade ao produto, resultando assim em usuários insatisfeitos com falhas no *software*. Além disso, estudos experimentais comprovam que 50% das modificações feitas no código pelo desenvolvedor na verdade inserem outros defeitos no código em inúmeros outros pontos do *software*.

Ao efetuar um teste, o profissional de teste parte da premissa que o software não está funcionando como deveria. Ele assume o fato de que o sistema está sujeito a vários defeitos e que seu papel é descobri-los antes que um usuário o faça.

Essas dois pontos de vistas citados (testador e desenvolvedor) são totalmente opostos e conflituosos, tornando-se muito complicado e inviável para apenas uma pessoa assumir os dois papeis no mesmo projeto.

De outro modo, apesar de não ser recomendável que o desenvolvedor não assuma papel de testador em um mesmo projeto, ele deve sim testar seu *software* desde o início da implementação. Isso significa, que por mais que tenhamos um profissional focado em testar, cabe ao desenvolvedor verificar seu código ao máximo, de modo a minimizar os defeitos existentes que viriam a ser encontradas posteriormente no processo de teste.

Segundo Barry Boehm, um defeito encontrado no desenvolvimento do design e requisitos da solução custa 10 vezes menos que um encontrado na codificação em si e 100 vezes menos do que um encontrado em produção. Portanto, o quanto antes os desenvolvedores e os testadores começarem a testar o produto, menos custoso ao projeto será e terá, absolutamente, muito mais qualidade.

Em suma, para atingir bons níveis de qualidade espera-se que no desenvolvimento de um *software* exista diversas etapas com diversos tipos de testes que virão a ser executados, não somente pelo testador, mas também pelo desenvolvedor.

* 1. Técnicas de Teste de Software

Segundo Pressman (2011), qualquer produto de *software* pode ser testado de duas maneiras diferentes:

* + - 1. Conhecendo a funcionalidade específica para o qual o *software* foi projetado para realizar, podem ser feitos testes que demonstram que cada uma das funcionalidades é totalmente operacional, embora ao mesmo tempo procurem erros em cada funcionalidade.
      2. Conhecendo o funcionamento interno do *software*, podem ser realizados testes para garantir que os componentes funcionam corretamente.

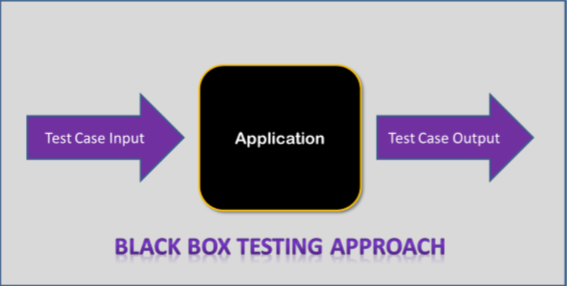
Dessa forma as abordagens de teste podem ser classificadas em: Modelo de teste de caixa branca e modelo de teste de caixa preta.

* + 1. Modelo Caixa-Preta

O modelo é conhecido por esse nome, “pelo fato de tratar o *software* como uma caixa cujo conteúdo é desconhecido e só é possível visualizar o lado externo” (MEIRELES, 2008). Consiste em testar sem que se possua nenhum conhecimento do funcionamento interior da aplicação que está sendo testada. O testador não possui conhecimento da arquitetura do sistema e não tem acesso ao código fonte. Normalmente, quando um testador está executando um teste como o modelo caixa-preta, ele interage apenas com a interface do sistema, provendo entradas e examinando as saídas sem saber como e onde essas entradas estão sendo utilizadas. Segundo Pressman (1995), o teste de caixa preta descobre “funções incorretas ou ausentes, erros de interface, erros nas estruturas de dados ou no acesso a banco de dados externos, erros de desempenho, e erros de iniciação e término”.

Este tipo de teste não se preocupa em “verificar como ocorrem internamente os processamentos no *software*, mas se o algoritmo inserido no *software* produz os resultados esperados” (BARTIÉ, 2002).

1. Black Box Testing Approach



Fonte: Bitware

* + 1. Modelo Caixa-Branca

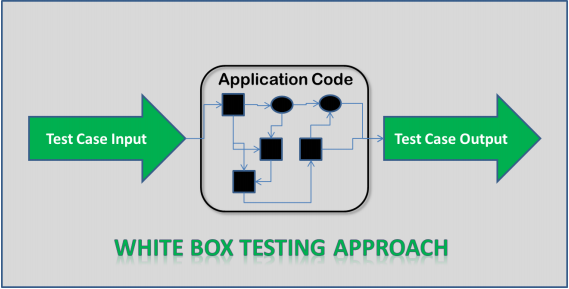
O modelo consiste em aplicar uma investigação da lógica interna e da estrutura do código fonte. O modelo de teste caixa-branca é também conhecido como “*glass testing*” (teste de vidro) ou teste caixa aberta.

É baseado na estrutura interna do sistema. Emprega métodos pra identificar erros nas estruturas internas dos programas através da simulação de situações que exercitem adequadamente todas as estruturas usadas na codificação (BARTIÉ, 2002).

Essa técnica de teste vista detectar “comandos incorretos, estruturas incorretas, variáveis não definidas e erros de inicialização e finalização de loops” (FURLAN, 2009).

O profissional responsável por realizar esse tipo de teste deve conhecer a tecnologia empregada pelo *software* e possuir conhecimento sobre a arquitetura interna da solução, para isso, ele deverá ter acesso a fontes e estrutura de banco de dados (BARTIÉ, 2002).

1. White Box Testing Approach



Fonte: Jobsandnewstoday.blogspot

Essa técnica te teste visa detectar “comandos incorretos, estruturas incorretas, variáveis não definidas e erros de inicialização e finalização de loops” (FURLAN, 2009).

Usando métodos de teste de caixa branca, podem-se derivar os casos de teste que garantam que todos os caminhos independentes dentro de um módulo tenham sido exercitados pelo menos uma vez; exercitem todas as decisões lógicas para valores falsos ou verdadeiros; executem todos os laços em suas fronteiras e dentro de seus limites operacionais; e exercitem as estruturas de dados internas para garantir a sua validade (PRESSMAN, 1995).

O profissional responsável por realizar esse tipo de teste deve conhecer a tecnologia empregada pelo *software* e possuir conhecimento sobre a arquitetura interna da solução, para isso, ele deverá ter acesso a fontes e estrutura de banco de dados (BARTIÉ, 2002).

* 1. Fases de teste de Software

Além da aplicação de técnicas e critérios para ampliar o nível de confiabilidade do *software* testado, quando o tamanho e a complexidade do projeto de teste aumentam, é necessário dividir a atividade de teste em várias fases ou níveis. Com esta divisão, os testes podem focar em diferentes aspectos do *software* testado, assim como diferentes tipos de erros.

As fases de teste de *software* são: teste de unidade, teste de integração, teste de sistema e teste de aceitação. Além dessas cinco, tem o teste de regressão que é uma fase especial de teste de *software*.

Uma estratégia empregada pela maioria das equipes de *software* está entre essas duas extremidades, ou seja, assume um visão incremental e decrescente dos testes, começando com os testes de unidade individuais, passando para os testes destinados a validar a integração de unidades e terminando com testes que o usam o sistema concluído (PRESSMAN, 2011).

Dessa forma pode-se considerar três níveis de testes: Teste de Unidade, Teste de Integração e Teste de Sistema (BARTIÉ, 2002).

Abaixo são definidos esses os tipos de testes que contemplam esses níveis.

* + 1. Teste de Unidade

O teste de unidade ou teste unitário é a fase do processo de teste em que se testam os menores componentes do código. Segundo Pressman (1995), “o teste de unidade concentra-se no esforço de verificação da menor unidade de projeto de *software* – o módulo”.

Os testes unitários tem como objetivo verificar cada unidade que compõe o *software*, isoladamente e independentemente para determinar se cada uma delas realiza o que foi estabelecido.

Segundo Bartié (2002), nessa fase de teste, é exercitado toda a estrutura interna de um componente do *software*, como os desvios condicionais, os laços de processamento e todos os possíveis caminhos alternativos de execução.

Para Inthurn (2001), o teste de unidade preocupa-se em exercitar a menor unidade de projeto de *software* com o intuito de verificar se as informações que entram e saem são consistentes; as condições de limites garantem a execução correta da unidade; todos os caminhos básicos estão corretos; e os caminhos de tratamento de erros sejam devidamente tratados.

O teste de unidade é o tipo mais importante de teste para a grande maioria das situações, já que é ele que deve testar se um algoritmo faz o que deveria ser feito e garantir que o código encapsulado por uma unidade deve produzir o efeito colateral esperado. Outro aspecto importante é que o teste de unidade é focalizado em um trecho específico do código, desta forma os erros encontrados são facilmente localizados, diminuindo o tempo gasto com depuração (BERNARDO E KON, 2008).

Sempre que possível os testes de unidade devem ser automatizados, desse modo um conjunto inteiro de testes regularmente pode ser executado em alguns segundos, possibilitando a reexecutar os testes a cada alteração realizada no sistema (SOMMERVILE, 2011).

* + 1. Teste de Integração

Após testar separadamente cada módulo, inicia-se a fase de integração, onde estes módulos são agrupados para compor os subsistemas, conforme a arquitetura do sistema definida no projeto preliminar. O objetivo desta fase é encontrar falhas de integração entre as unidades (FURLAN, 2009).

De acordo com Pressman (1995), o teste de integração é a fase para a construção da estrutura do programa e para descobrir erros associados a interfaces.

O teste de integração pode ter sua execução iniciada assim que alguns componentes ficarem prontos, ou seja, quando um componente estiver funcionando e tiver atingido seus objetivos (PFLEEGER, 2004 apud FURLAN, 2009).

A integração pode ser incremental ou não-incremental. Na integração não-incremental as unidades são testadas individualmente e depois integradas de uma só vez. Isso torna o teste menos eficaz, pois torna-se difícil isolar um erro e, quando esses são corrigidos, novos erros são gerados na estrutura do programa. Já na integração incremental, o programa construído é testado em pequenos segmentos, tornando o teste mais eficiente, pois os erros são mais fáceis de serem isolados e corrigidos e as interfaces tem maior probilidade de serem testadas corretamente (PRESSMAN, 1995).

O teste de integração é seguido por dois métodos de interação: top-down e bottom-up.

Na integração top-down,

[...] as unidades de maior nível hierárquico são criadas inicialmente, sofrendo um processo de refinamento e decomposição sucessivos, até que seja alcançado o menor nível estrutural do projeto e todas as unidades tenham sido criadas. À medida que as unidades são construídas e alteradas, os testes de integração avaliam se as interfaces com outras unidades continuam compatíveis (BARTIÉ, 2002).

Na integração bottom-up,

[...] os componentes de níveis mais básicos da arquitetura do aplicativo são conjuntamente testados por controladores construídos para simular as interações dinâmicas e suas interfaces. À medida que se desenvolve o processo de integração, componentes reais substituem os controladores anteriores e novos controladores de testes são criados para simular as interfaces de um maior nível da arquitetura. Esses níveis de integração seguem até que não existam mais níveis a serem alcançados, atingindo o nível máximo de integração que os componentes podem chegar (BARTIÉ, 2002).

A escolha de um método de integração depende das características do *software* e do cronograma do projeto. O ideal seria usar um método tio-down para os níveis superiores de estrutura do programa acoplada com método bottom-up para níveis subordinados (PRESSMAN, 1995).

* + 1. Testes de Sistema

Após a integração do *software*, aplica-se o teste de sistema, sendo que é esta a etapa em que o *software* é testado por completo.

A diferença entre um componente e um sistema é que sistemas funcionam por si só enquanto componentes só são utilizados como parte de um sistema. Mas na realidade esta diferença pode parecer vaga quando regularmente sistemas são integrados com outros para formar um sistema maior.

Esse teste “tem como objetivo determinar se o *software* e demais elementos que compõem o sistema, tais como hardware e banco de dados, combinam-se adequadamente e se atendem aos requisitos especificados” (FURLAN, 2009).

“Estruturar os testes de sistema não é uma tarefa simples, pois exige o perfeita compreensão dos requisitos não funcionais de um software, que na maioria das vezes não são claramente descritos ou representam cenários complexos” (BARTIÉ, 2002).

* + 1. Testes de Aceitação

O teste de aceitação é a etapa em que o teste é guiado por usuários finais do sistema. Essa fase serve para o cliente determinar se o produto é, ou não, aceitável de acordo com os critérios previamente acordados, e de maneira estabelecida em contrato.

Nesta fase o *software* é fornecido ao cliente para que este interaja com o sistema validando se as funcionalidades requisitadas no início do projeto funcionam como deveriam (BARTIÉ, 2002).

Segundo o mesmo autor, se essa fase mostrar muitas falhas, é prenuncio de que os processos de detecção de erros anteriormente executados não estão sendo implementados corretamente (BARTIÉ, 2002).

O teste de aceitação pode ser:

* Teste Alfa: Onde o teste de aceitação é efetuado pelo cliente nas instalações do desenvolvedor. O *software* é utilizado num ambiente real com a presença do desenvolvedor que registra erros e problemas de uso (PRESSMAN, 1995);
* Teste Beta: Onde o teste de aceitação é efetuado pelo usuário final no ambiente de produção, sem a presença ou controle do desenvolvedor. O usuário registra os problemas encontrados e relata-os ao desenvolvedor (PRESSMAN, 1995).
  + 1. Testes de Regressão

O teste de regressão ou teste regressivo é um tipo especial de teste, que propõe assegurar que alterações feitas em parte do sistema não afetem as partes já testadas. A cada nova versão do sistema, são empregados todos os testes que ocorreram nas versões prévias do *software*.

O teste de regressão é a ato de reexecutar o mesmo subconjunto de testes que foram executados, para assegurar que as alterações não tenham introduzidos efeitos colaterais indesejados. Cada vez que um novo módulo é acrescentado, o sistema muda, novos caminhos de fluxo de dados são determinados, podem ocorrer novas entradas e saídas e novas lógicas podem ser chamadas. Essas modificações podem causar problemas em funcionalidades que antes funcionavam corretamente. Sempre que um *software* é modificado ou incrementado resulta em alteração de algum aspecto de configuração do *software* (PLFEEGER, 2004).

O teste regressivo ajuda a garantir que as modificações (acréscimo de novos componentes, correções ou qualquer tipo de alteração) não introduzam comportamentos indesejados ou erros. A medida que o teste de integração evolui o número de casos de testes pode crescer muito (PRESSMAN, 2011).

O teste de regressão pode ser executado de forma manual, reexecutando um grupo de casos de teste ou aplicando a automação de testes utilizando ferramentas apropriadas. Normalmente o número de casos de teste é grande, por essa razão é indicado que os testes sejam automatizados, desse modo é possível contemplar um número maior de casos de teste em um curto espaço de tempo (SOMMERVILLE, 2011).

* 1. Outros tipos de teste de software
     1. Testes de Desempenho

Testes de desempenho, como o próprio nome já diz, têm como finalidade validar se o sistema funciona em situações mais próximas na prática a qual ele será exposto. O teste tem como meta avaliar se o *software* reagirá nos tempos esperados mediante pedidos e acessos de mais de um usuário. Além disso, mede-se como o sistema responderia em condições mais extremas de falhas de conexão, inconstância na rede.

Testes de desempenho podem ser realizados e dentro de todos as fases de de testes, mas somente quando todos os módulos do sistema estão completamente integrados é que se pode ter uma medida real do desempenho do sistema (PRESSMAN, 1995).

* + 1. Testes de Carga

Testes de carga baseiam-se na simples verificação do comportamento do sistema por intermédio da aplicação de uma carga máxima de acessos e manipulações de dados de entrada.

Podem ser realizados em condições normais, simulando a realidade em que se deseja introduzir no sistema, ou em condições de pico, simulando condições extremas que podem vir a ocorrer devido a alguma exceção.

Esses tipos de testes podem ser executados da seguinte forma (BARTIÉ, 2002):

* Elevando e reduzindo sucessivamente o número de transações simultâneas;
* Aumentando e reduzindo o tráfego de rede;
* Aumentando o número de usuários simultâneos;
* Combinando todos esses elementos.
  + 1. Testes de Stress

Testes de stress visam assegurar o comportamento do sistema sob condições anormais. Durante a execução de um teste de stress retiram-se recursos aplicando uma carga acima dos limites definidos pelos requisitos.

O principal objetivo é testar o *software* em condições extremas de modo a buscar o ponto no qual o *software* deixa de funcionar, conhecida como “breaking point”.

Testes de stress podem ser realizados em diferentes cenários, abaixo se destacam alguns deles:

* Conexão intermitente com o banco de dados.
* Conexão intermitente na rede.
* Aplicando carga através de diferentes processos que consomem memória, processamento no servidor.
* Aplicando carga através de diferentes processos que consomem memória, processamento no cliente.
  + - 1. Testes de Segurança
  1. Automação de Testes

Automação de testes, por definição, consiste em utilizar um *software* externo (que não seja o *software* que está sendo testado) para controlar a execução dos testes e a comparação dos resultados encontrados com os resultados esperados. Por meio desse processo, podem-se automatizar algumas tarefas de teste repetitivas, no entanto fundamentais, que fazem parte do fluxo de teste de um sistema ou adicionar novos testes que seriam muito trabalhosos, e consequentemente custosos para serem executados manualmente.

* + 1. Propostas e Dúvidas da Automação de Testes

A automação pode auxiliar a carga de trabalho de desenvolvedores e testadores de *software* e dispensa-los de tarefas repetidas. Isso apresenta a capacidade de agregar na qualidade do sistema e diminuir o tempo da atividade de teste.

Poder implementar testes previamente criados sem nenhum trabalho extra sobre novas versões de sistema claramente torna a fase de testes mais efetiva. A automação, além de dar condições da executar os testes de forma mais dinâmica, o que significa mais e mais rodadas de testes, também torna a atividade de criar novos *test cases* mais fácil e mais rápida. Isto é, automatizar permite executar testes com maior regularidade. Tudo isso, interiormente, possibilita um melhor aproveitamento de recursos.

Além disso, o reaproveitamento de testes e a diminuição do tempo de execução dos mesmos, torna mais rápido o ciclo de feedback para os desenvolvedores. Isso, acrescentado ao ganho de confiança que toda a equipe obtém por ter uma rotina de testes automatizados executando frequentemente de forma segura e em diferentes ambientes, acaba por reduzir o tempo que o produto leva para chegar até o cliente.

Porém, apesar da automação de testes trazer diversas vantagens, ela não é uma tarefa fácil, apresenta alguns problemas e deve ser implementada com muito cuidado.

Problemas como expectativas ilusórias, onde gestores acreditam que a automação acabará com todos os problemas relacionados a testes e milagrosamente fará o *software* ter mais qualidade. Esperar que testes automatizados acharão muitos novos defeitos de *software*. Falsa sensação de segurança, onde só porque o sistema não falhou nos testes, não indica que o sistema esteja 100% livre de falhas.

Em suma, o processo de automatização encontra mais erros durante a implementação do que quando os testes são executados. Uma vez que os testes estiverem executando, eles só devem quebrar quando houver mudanças nos requisitos funcionais que provoquem alterações de interface ou de ações de usuários.

Essa particularidade implica na capacidade de manutenção dos testes, pois uma vez que a aplicação sofre alteração, os testes também sofrem. Desenvolvedores e testadores testando uma aplicação de forma manual são capazes de se adaptarem a grandes mudanças sem problemas, já os testes automatizados podem falhar mesmo nas pequenas mudanças. E se o esforço da manutenção nos testes automatizados for maior que o de testar manualmente, a pratica de automação de testes certamente será abandonada. O mesmo pode acontecer se a atividade de adicionar novos recursos for muito complexa.

Os problemas relacionados a testes automatizados parece estar na falta de percepção que qualquer projeto de automação de testes pode se tornar grande o suficiente para ser tratado como um projeto de *software* por si só. Sistemas de Informações falham se não tiverem a devido tratamento, e testes automatizados não são diferentes.

Ainda que todas as propostas fazem a automação de testes parecer muito atraente, alcançar tal automação na prática exige muito trabalho. De forma que, se a automação não for bem implementada, sela será rapidamente abandonada e as propostas não serão alcançadas (FEWSTER; GRAHAM, 1999).

* + 1. Quando não automatizar?

A automação de testes tem como objetivo resumir esforços em tarefas repetitivas, logo nem todos os cenários de teste devem ser automatizados. Antes de introduzir essa abordagem em um projeto deve-se analisar qual é a carga de testes que seu sistema precisa e o tempo que é levado para executar esses planos de testes. Se o projeto em pauta tiver um nível de complexidade baixo de modo que o volume de teste para verificar a qualidade do *software* é muito baixo, o esforço para inserir essa metodologia de testes pode ser mais custoso do que a manutenção dos testes manuais. Com isso, o tempo para a execução dos testes automatizados não seriam, nesse cenário, tão menores que os manuais.

* + 1. Casos para Automação de Testes
       1. Testes de Carga e Desempenho

Testes automatizados são um pré-requisito para esses tipos de teste. É inviável financeiramente utilizar 100 ou mais usuários, durante um teste, para acessar um aplicação simultaneamente com o propósito de avaliar a qualidade do sistema sob tais condições.

* + - 1. Testes de Regressão

É um dos melhores casos para se aplicar testes automatizados. Testes de regressão custam muito tempo para serem executados de forma manual e são muito propensos a erro humano pelo fato, do nível de frequência. Ao automatizar, diminui-se consideravelmente a possibilidade de o sistema ir para produção com a inclusão de um novo defeito em uma funcionalidade antiga. Além disso, o ganho em tempo de execução dos testes normalmente justifica a sua utilização.

* + - 1. Testes manuais repetitivos

Nesses casos, o grande benefício do teste automatizado se deve a diminuição de erros propriamente.

* 1. Métodos Ágeis

As metodologias ágeis são uma série de práticas e métodos, criados com o intuito de tomar o processo de desenvolvimento de *software* mais rápido e com custo acessível.

Nessas metodologias é dado maior valor aos envolvidos do projeto e suas interações ao invés de processos formais e ferramentas de desenvolvimento. Além de que, existe a preocupação de passar menos tempo com documentação e mais com implementação (SOARES, 2004).

Outro aspecto das metodologias ágeis é que é oferecido ao desenvolvedor ou testador total flexibilidade. Dessa forma, é possível se adaptar facilmente a novas mudanças, ao contrário de procurar analisar previamente tudo o que pode acontecer ao longo do desenvolvimento (SOARES, 2004).

Além disso, são entregues frequentemente partes operacionais do *software*. Assim, o cliente não precisa esperar muito para ver o sistema funcionando, como nas metodologias tradicionais (SOARES, 2004)

* + 1. Scrum

O *Scrum* é um framework para organizar e gerenciar trabalhos complexos, tal como projetos de *software*.

O *Scrum* é baseado em pequenos ciclos de tempo, normalmente 4 semanas, chamados *Sprints*, onde se trabalha para alcançar objetivos bem definidos (SANCHEZ, 2007).

Os principais elementos do *Scrum* são o Time (geralmente entre 5 e 9 pessoas) que são responsáveis por entregar o projeto de forma auto gerenciada. O *Product* *Owner*, responsável pela visão de negócio do projeto. É quem define o que é prioridade. E também o *Scrum* *Master* que é uma combinação de gerente, facilitador e mediador, deve identificar e remover os obstáculos e assegurar que as práticas *Scrum* serão executadas (SANCHEZ, 2007).

Na metodologia *Scrum* são efetuadas reuniões diárias de 30 minutos e cada membro do time precisa responder as questões: o que foi feito ontem? O que planeja fazer amanhã? Tem algum impedimento? (MATSUDO, 2009).

O intuito da reunião é que todos os integrantes da equipe se mantenham informados do que cada um está executando e se existe algum impedimento que o *Scrum Master* deverá resolver (MATSUDO, 2009).

No fim de cada *Sprint* é efetuada uma reunião para que o Prodcut Owner verifique se o que foi desenvolvido atende aos requisitos inicialmente propostos. Também é feito uma reunião para encontrar os pontos positivos e negativos que aconteceram durante o *Sprint* e levar como experiência, melhorando o processo, o entrosamento do time e o projeto em desenvolvimento. (MATSUDO, 2009).

1. MODELAGEM DE DADOS

Neste capítulo será apresentado o projeto do sistema. Este capítulo analisa os requisitos para obtenção dos dados a serem utilizados na implementação, com base nas análises realizadas sobre o tema do trabalho.

* 1. *Product Backlog*

Neste tópico é apresentado as principais funcionalidades que serão implementadas. O Quadro 1 demonstra o planejamento, utilizando uma priorização de 0 a 10, sendo que quanto maior o valor, maior a prioridade.

1. *Product Backlog*

|  |  |  |
| --- | --- | --- |
| ID | DESCRIÇÃO | PRIORIDADE |
| 1 | Desenvolvimento de um layout para o sistema. | 8 |
| 2 | Realizar cadastro, edição e exclusão de usuários. | 9 |
| 3 | Realizar cadastro, edição e exclusão de turmas. | 9 |
| 4 | Realizar vínculo, entre turmas e regentes | 10 |
| 5 | Realizar cadastro, edição e exclusão de alunos. | 9 |
| 6 | Realizar vínculo, entre alunos e turmas | 10 |
| 7 | Realizar cadastro, edição e exclusão de disciplinas. | 9 |
| 8 | Realizar vínculo, entre disciplinas e turmas | 10 |
| 9 | Realizar cadastro, edição e exclusão de avaliações. | 9 |
| 10 | Realizar vínculo entre avaliação e disciplina. | 10 |
| 11 | Realizar vínculo entre avaliação e turmas | 10 |
| 12 | Permitir a função de habilitar e desabilitar as avalições cadastradas | 10 |

(FONTE: PRÓPRIO AUTOR, 2018)

* 1. Histórias de usuários

O Quadro 2 apresenta as histórias de usuário que serão empregadas no desenvolvimento do projeto, aplicando uma priorização de 0 a 10, sendo que quanto maior o valor, maior a prioridade, e a estimativa da realização de determinada história de usuário em horas.

1. Histórias de usuário

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **DESCRIÇÃO** | **PRIORIDADE** | **ESTIMATIVA** |
| 001 Cadastro de usuário | Como coordenador de curso quero poder cadastrar os usuários, podendo ser classificados por níveis de coordenador, regente e professor, para que o acesso seja diferenciado para cada nível. | 10 | 15 |
| 002 Edição de usuário | Como coordenador de curso quero poder editar um usuário cadastrado, para que na eventualidade de algum erro ou mudança, eu possa alterar algum dado. | 9 | 15 |
| 003 Exclusão de usuário | Como coordenador de curso quero poder excluir um usuário. | 9 | 5 |
| 004 Visualização de usuário | Como coordenador de curso quero poder visualizar os usuários já cadastrados. | 8 | 5 |
| 005 Cadastro de turma | Como coordenador de curso quero poder cadastrar as turmas, para que eu tenha um registro permanente de todas as turmas. | 10 | 15 |
| 006 Vínculo de turma com usuário regente | Como um coordenador de quero poder vincular uma turma a um regente já cadastrado como usuário, para que este possa cadastrar posteriormente os alunos. | 10 | 5 |
| 007 Edição de turma | Como coordenador de curso quero poder editar uma turma cadastrada, para que na eventualidade de algum erro ou mudança, eu possa alterar algum dado. | 9 | 15 |
| 008 Exclusão de turma | Como coordenador de curso quero poder excluir uma turma, desde que a mesma não possua alunos e/ou usuários com nível de regente vinculados. | 9 | 5 |
| Visualização de turma | Como coordenador de curso quero poder visualizar as turmas já cadastradas | 8 | 5 |
| 009 Cadastro de aluno | Como coordenador ou regente de curso eu quero poder cadastrar um aluno, para que eu tenha um registro permanente de todos os alunos. | 10 | 15 |
| 010 Vínculo de aluno com turma | Como um coordenador ou regente de curso quero poder vincular um aluno a mais de uma turma existente, para que o mesmo possa acessar as turmas tanto de nível médio quanto de profissionalizante que este faz parte. | 10 | 5 |
| 011 Edição de aluno | Como coordenador ou regente de curso quero poder editar um aluno cadastrado, para que na eventualidade de algum erro ou mudança, eu possa alterar algum dado. | 9 | 15 |
| 012 Exclusão de aluno | Como coordenador ou regente de curso quero poder excluir um aluno caso este não faça mais parte do quadro de alunos da instituição. | 9 | 5 |
| 013 Visualização de aluno | Como coordenador de curso quero poder visualizar os alunos já cadastradas | 8 | 5 |
| 014 Cadastro de disciplina | Como coordenador de curso eu quero poder cadastrar uma disciplina. | 10 | 15 |
| 015 Vínculo de disciplina com turma | Como coordenador de curso quero poder vincular uma disciplina com uma turma existente. | 10 | 5 |
| 016 Edição de disciplina | Como coordenador de curso quero poder editar uma disciplina cadastrada, para que na eventualidade de algum erro ou mudança, eu possa alterar algum dado. | 9 | 15 |
| 017 Exclusão de disciplina | Como coordenador de curso quero poder excluir uma disciplina. | 9 | 5 |
| 018 Visualização de disciplina | Como coordenador de curso quero poder visualizar as disciplinas já cadastradas. | 8 | 5 |
| 019 Cadastro de avaliação | Como coordenador, regente ou professor eu quero poder cadastrar uma avaliação. | 10 | 15 |
| 020 Vínculo de avaliação com disciplina | Como coordenador, regente ou professor quero poder vincular uma avaliação a uma disciplina. | 10 | 5 |
| 021 Vínculo de avaliação com turma | Como coordenador, regente ou professor quero poder vincular uma avaliação a uma turma. | 10 | 5 |
| 022 Habilitar/desabilitar avaliação | Como coordenador, regente ou professor quero poder habilitar e desabilitar uma avaliação já cadastrada. | 10 | 5 |
| 023 Edição de avaliação | Como coordenador, regente ou professor quero poder editar uma avaliação cadastrada, para que na eventualidade de algum erro ou mudança, eu possa alterar algum dado. | 9 | 15 |
| 024 Exclusão da avaliação | Como coordenador, regente ou professor quero poder excluir uma avaliação. | 9 | 5 |
| 025 Visualização de avaliações | Como coordenador, regente ou professor quero poder visualizar as avaliações já cadastradas. | 8 | 5 |
| 026 Corrigir avaliações respondidas | Como coordenador, regente ou professor quero poder selecionar uma avaliação para efetuar a correção. | 10 | 40 |
| 027 Editar correção da avaliação | Como coordenador, regente ou professor quero poder editar uma avaliação que já tenha passado por correção, para que na eventualidade de algum erro ou mudança, eu possa alterar algum dado. | 10 | 40 |
| 028 Exportar a avaliação já corrigida | Como coordenador, regente ou professor quero poder exportar essa avaliação em um arquivo, para que possa compartilhar de forma externa caso necessário. | 9 | 40 |
| 029 Excluir avaliação já corrigida | Como coordenador, regente ou professor quero poder excluir uma avaliação que já tenha sido corrigida. | 8 | 5 |
| Estimativa total de horas | | | 355 |

(FONTE: PRÓPRIO AUTOR, 2018)

* 1. Requisitos funcionais

O Quadro 3 apresenta os requisitos funcionais, que serão primordiais na fase de implementação para que o sistema tenha um correto desempenho, definidos por uma prioridade de alta, média e baixa, eles foram descritos baseados nas histórias de usuários descritas no Quadro 2, que serão de grande importância no desenvolvimento e andamento do projeto.

1. Requisitos funcionais

|  |  |
| --- | --- |
| R.F Requisitos Funcionais | |
| R.F 01 | |
| Nome: | Validação de ações sobre botões |
| Descrição: | Quando um usuário clicar sobre um botão que tenha uma ação de sair, concluir, iniciar, finalizar ou excluir no sistema, este deve ter confirmações de clique, para que se acidentalmente haja o clique sobre algum destes, ele não venha a ter um aborrecimento sobre o sistema. |
| Prioridade: | Baixa |
| R.F 02 | |
| Nome: | Acessos ao sistema |
| Descrição: | O acesso ao sistema deverá ser somente com autenticação de usuários que estejam devidamente cadastrados em uma base de dados. |
| Prioridade: | Alta |
| R.F 03 | |
| Nome: | Acessos de alunos bloqueados ao sistema |
| Descrição: | Se o acesso de um aluno estiver bloqueado este não deverá conseguir autenticar-se no sistema. |
| Prioridade: | Alta |
| R.F 04 | |
| Nome: | Acessos de alunos simultâneos limitados |
| Descrição: | Enquanto o aluno estiver autenticado no sistema, este não irá conseguir se autenticar em outro dispositivo como os mesmos dados de acesso. |
| Prioridade: | Média |
| R.F 05 | |
| Nome: | Acessos ao conteúdo do sistema limitado por níveis de usuários |
| Descrição: | O acesso ao conteúdo e as ações sobre o sistema deve ser classificado e limitado de acordo com o nível de usuário, sendo eles: coordenador, regente, professor e aluno. |
| Prioridade: | Média |
| R.F 06 | |
| Nome: | Acessos ao sistema com o nível de coordenador |
| Descrição: | Este deverá ter gerenciamento por completo sobre todos os R.F do sistema. |
| Prioridade: | Média |
| R.F 07 | |
| Nome: | Acessos ao sistema com o nível de regente |
| Descrição: | Apenas poderá gerenciar as turmas bem como os alunos vinculados a estas as quais lhe foi delegado a tal função, e também terá acesso de gerenciamento as avaliações que cadastrou bem como o acesso total sobre o R.F 23. |
| Prioridade: | Média |
| R.F 08 | |
| Nome: | Acessos ao sistema com o nível de professor |
| Descrição: | Ficará restringido somente a gerenciar as avaliações que cadastrou bem como a função de bloqueio e desbloqueio de acessos de alunos e acesso total ao R.F. 23. |
| Prioridade: | Média |
| R.F 09 | |
| Nome: | Acessos ao sistema com o nível de aluno |
| Descrição: | Somente irá visualizar e responder as avaliações que estão disponíveis bem como visualizar e exportar em um arquivo externo as avaliações que os professores já corrigiram. |
| Prioridade: | Alta |
| R.F 10 | |
| Nome: | Bloquear envio de avaliações |
| Descrição: | O sistema não deverá permitir que um aluno responda e envie uma mesma avaliação mais de uma vez. |
| Prioridade: | Alta |
| R.F 11 | |
| Nome: | Iniciar uma avaliação |
| Descrição: | O aluno só irá ser destinado a tela para iniciar uma avaliação se a tecla F11 for pressionada, onde tal ação em qualquer navegador faz com que o mesmo fique em tela cheia. |
| Prioridade: | Alta |
| R.F 12 | |
| Nome: | Abandonar uma avaliação antes do término |
| Descrição: | Caso o aluno pressione uma tecla indevida antes de concluir e enviar a avaliação, seus dados de acesso deveram ser bloqueados. |
| Prioridade: | Alta |
| R.F 13 | |
| Nome: | Cadastro duplicado de alunos |
| Descrição: | O sistema não deverá permitir o cadastro deum aluno com as mesmas informações no banco de dados. |
| Prioridade: | Média |
| R.F 14 | |
| Nome: | Cadastro duplicado de turmas |
| Descrição: | O sistema não deverá permitir o cadastro de uma turma caso ela já exista no banco de dados. |
| Prioridade: | Média |
| R.F 15 | |
| Nome: | Vincular turma com o regente |
| Descrição: | Cada regente pode ter mais de uma turma vinculada a seu usuário, sendo assim o mesmo conseguirá gerenciar mais turmas. |
| Prioridade: | Baixa |
| R.F 16 | |
| Nome: | Cadastro duplicado de disciplinas |
| Descrição: | Não deverá ser possível cadastrar uma disciplina caso já exista. |
| Prioridade: | Media |
| R.F 17 | |
| Nome: | Vincular disciplinas com turmas |
| Descrição: | Uma mesma disciplina poderá ser vinculada a mais de uma turma. |
| Prioridade: | Baixa |
| R.F 18 | |
| Nome: | Vincular disciplinas com turmas |
| Descrição: | Uma mesma disciplina poderá ser vinculada a mais de uma turma. |
| Prioridade: | Baixa |
| R.F 19 | |
| Nome: | Cadastro de avaliação desabilitada |
| Descrição: | Ao concluir o cadastro de uma avaliação a mesma deverá estar desabilitada, podendo o usuário habilitar e desabilitar a qualquer momento, pois se a mesma estiver habilitada qualquer aluno a qualquer momento, poderá entrar e responder desde que possua acesso ao sistema. |
| Prioridade: | Media |
| R.F 20 | |
| Nome: | Cálculo das questões |
| Descrição: | Ao selecionar uma avaliação para correção o sistema deverá apresentar o valor de cada questão de acordo com a quantidade até que se tenha peso máximo de 10 pontos totais, ex.: Avaliação com 5 questões, valor de cada questão 2 pontos. |
| Prioridade: | Alta |
| R.F 21 | |
| Nome: | Correção de avaliação |
| Descrição: | No momento da correção deverá aparecer para seleção as opções de certo, errado ou valor onde caso seja selecionado esta opção, deverá ser preenchido pelo usuário um valor que será atribuído no somatório final. |
| Prioridade: | Alta |
| R.F 22 | |
| Nome: | Somatório das questões |
| Descrição: | Ao concluir a correção o sistema irá calcular de forma automática a nota final e atribuída a avaliação. |
| Prioridade: | Alta |
| R.F 23 | |
| Nome: | Verificação de alunos bloqueados |
| Descrição: | Deverá ter uma tela de verificação de alunos bloqueados, esta deverá atualizar a cada 5 segundos de forma automática e caso identifique que um aluno saiu da tela de avaliação, o usuário deste será bloqueado e o sistema deverá emitir um sinal sonoro em formato de alarme. |
| Prioridade: | Alta |

(FONTE: PRÓPRIO AUTOR. 2018)

* 1. Requisitos não funcionais

O Quadro 4 apresenta os requisitos não funcionais que serão primordiais na usabilidade, ergonomia e disponibilidade do sistema, definidos da mesma maneira que os requisitos funcionais, estes também serão classificados por prioridade de alta, média e baixa, eles foram definidos seguindo padrões de UX que deveram garantir aos usuários boas experiências de uso.

1. Requisitos não funcionais

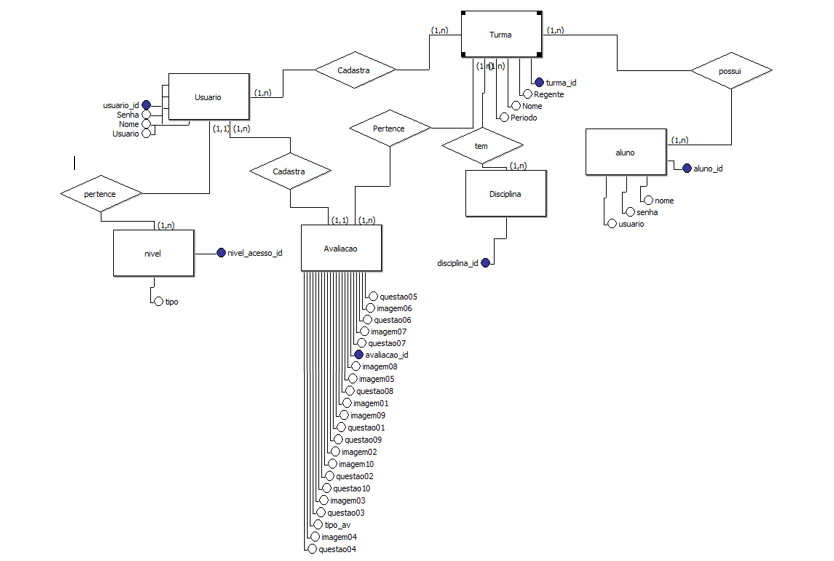
|  |  |
| --- | --- |
| R.N.F Requisitos Não Funcionais | |
| R.N.F 01 | |
| Nome: | Ergonomia |
| Descrição: | O sistema deve manter um padrão de ícones seguidos por um título que o define. |
| Prioridade: | Baixa |
| R.N.F 02 | |
| Nome: | Ergonomia |
| Descrição: | As páginas devem seguir a mesma apresentação de disposição dos elementos, para que o usuário tem uma boa experiência na navegação. |
| Prioridade: | Média |
| R.N.F 03 | |
| Nome: | Ergonomia |
| Descrição: | O mesmo padrão de cores deve ser seguido em todo o sistema, para que o usuário associe as informações de forma mais rápida. |
| Prioridade: | Média |
| R.N.F 04 | |
| Nome: | Ergonomia |
| Descrição: | O tamanho da fonte deve ser relativamente grande se comparado com outros sistemas, para que o usuário não precise se aproximar do monitor para uma leitura rápida das informações. |
| Prioridade: | Baixa |
| R.N.F 05 | |
| Nome: | Desempenho |
| Descrição: | O sistema não deverá ficar muitos segundo processando uma requisição ao ponto de um usuário abandoná-lo tal operação. |
| Prioridade: | Alta |
| R.N.F 06 | |
| Nome: | Usabilidade |
| Descrição: | Caso o usuário tente fazer uma ação indevida o sistema deverá alertar e instruir o mesmo. |
| Prioridade: | Baixa |
| R.N.F 07 | |
| Nome: | Portabilidade |
| Descrição: | O acesso ao sistema deverá ser possível de ser feito somente por computadores desktop ou portáteis na modalidade de netbook, notebook, ou ultrabooks, por questões de segurança. |
| Prioridade: | Media |
| R.N.F 08 | |
| Nome: | Portabilidade |
| Descrição: | O sistema foi desenvolvimento para rodar em praticamente todos os navegadores, exceto Microsoft Internet Explorer e Edge que podem apresentar problemas de incompatibilidade, sendo assim se o usuário tentar acessar por um deste dois navegadores o sistema irá restringir seu acesso instruindo o mesmo utilizar outros navegadores. |
| Prioridade: | Media |
| R.N.F 09 | |
| Nome: | Segurança |
| Descrição: | Todos os dados de acesso de um usuário são criptografados no momento do cadastro, assegurando que o sistema não seja burlado por ataques externos como por exemplo de SQL injection. |
| Prioridade: | Alta |
| R.N.F 10 | |
| Nome: | Segurança |
| Descrição: | Todos os dias o servidor deverá passa por um backup automático para garantir a disponibilidade e segurança dos arquivos dos usuários. |
| Prioridade: | Alta |

(FONTE: PRÓPRIO AUTOR, 2018)

* 1. Modelagem conceitual

A modelagem conceitual é a descrição da informação que o sistema irá gerenciar, sendo um artefato do domínio do problema, sendo assim a Figura 1 abaixo apresenta o modelo conceitual que faz parte de um dos processos de modelagem dos dados.

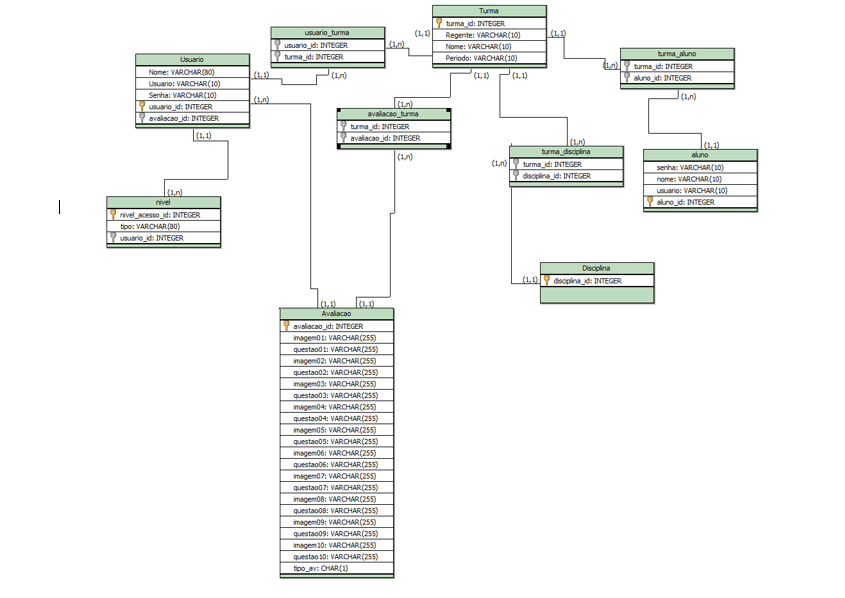
1. Modelagem conceitual



* 1. Modelagem lógica

O modelo lógico leva em conta algumas limitações e implementa recursos como adequação de padrão e nomenclatura, define as chaves primárias e estrangeiras, normalização, integridade referencial, entre outras. Para o modelo lógico deve ser criado levando em conta os exemplos de modelagem de dados criados no modelo conceitual, a Figura 2 abaixo apresenta o modelo lógico do sistema.

1. Modelagem lógica



* 1. Conclusão

A modelagem dos dados é um fator de suma importância, sendo que as informações aqui contidas foram cruciais para que todos os módulos fossem desenvolvidos corretamente, inclusive na fase de implementação do sistema, seguindo o planejamento de funcionalidades descritas neste capítulo, onde se expõe os requisitos principais sobre a concepção de funcionamento, seguido das modelagens para a construção do banco de dados, que faz o gerenciamento e o armazenamento das informações que se faz presente no momento da utilização do sistema.

1. IMPLEMENTAÇÃO DO SISTEMA
   1. Funcionalidades

Conforme levantado em reuniões com o coordenador pedagógico e professor do ensino médio, Everton Luiz Patrício, o qual atual na unidade do SENAI de Lages – SC, descreve as principais funcionalidades do sistema, sendo que as mesmas foram estruturadas no capítulo anterior como histórias de usuário, requisitos funcionais e não funcionais.

* + 1. Apresentação do sistema

O sistema conta com 4 níveis de acesso, sendo: coordenador, regente, professor e aluno. Será apresentado as telas do sistema classificando-as pelas principais funcionalidades de cada nível juntamente com imagens dos fragmentos de códigos. Logo de início a Figura 3 abaixo mostra o formulário de acesso que é utilizado por todos os níveis de usuários, sendo que ao tentar autenticar-se no sistema, o mesmo fará uma verificação na base de dados, para identificar se as informações inseridas estão corretas, após esta validação, o usuário será destinado a tela inicial do sistema.

1. Formulário de acesso



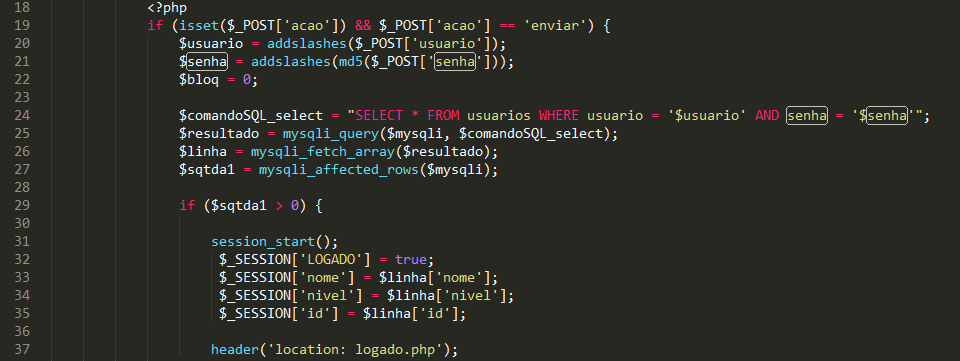
Se os dados inseridos pelo usuário sobre o formulário de acesso estejam incorretos, este irá se deparar com um aviso de que seus dados de acesso possuem informações invalidas, conforme Figura 4 abaixo.

1. Aviso de dados inválidos



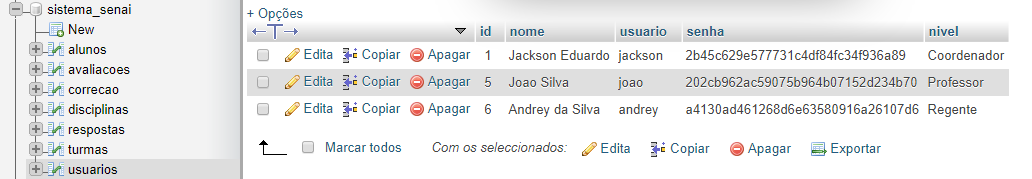
A FIGURA 5 - mostra um fragmento do código que faz a verificação dos dados de acesso de um usuário, onde é utilizado 2 funções de segurança, uma delas é a *addslashes* a qual garante uma maior segurança contra ataques de injeções de SQL, e um formato de criptografia do tipo MD5 onde o objetivo é criptografar a senha digitada pelo usuário e checar com a já existente no campo senha da tabela usuários no banco de dados, também é possível identificar a declaração de *sessions,* que são utilizadas para garantir que uma seção foi iniciada após a verificação e validação de informações existentes no banco de dados, posteriormente estas *sessions* são enviadas para a página de destino, que no exemplo está declarada como: logado.php.

1. Código fonte da validação de dados para acesso



A Tabela de usuários representada pela Figura 6, exibi os registros com o campo senha criptografado no formato MD5, sendo que este padrão de criptografia necessita de 32 caracteres livres no campo o qual será aplicado.

1. Registros da tabela usuários



A imagem da Figura 7 abaixo apresenta o código fonte da página logado.php, onde há uma verificação de *session,* cujo o objetivo é garantir que o acesso as informações da página por um usuário só irá acontecer caso exista uma sessão recebida de uma página remetente, caso isto não aconteça, automaticamente o usuário será destinado ao formulário de acesso ao sistema para fazer a autenticação.

1. Verificação de session

C:\Users\Jackson\Documents\21-07-2017\telas\session.PNG

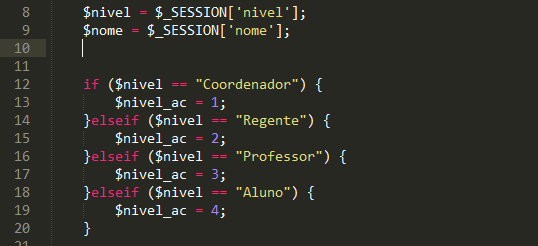
Após os dados verificados e checados no banco de dados, o sistema irá identificar qual o nível de acesso que se autenticou para apresentar a tela inicial, no exemplo da Figura 8 é possível observar que a autenticação foi feita por um usuário com nível de coordenador.

1. Apresentação inicial após um acesso



O código fonte na Figura 9 demonstra como foi implementado parte da verificação dos diferentes níveis de acesso, os quais já foram citados no item 4.1.1 deste trabalho, neste fragmento de código é apenas verificado o nível do usuário, e descritas condições, atribuindo um número de 1 a 4 para cada um tipo de usuário.

1. Verificação de acesso do nível de usuário

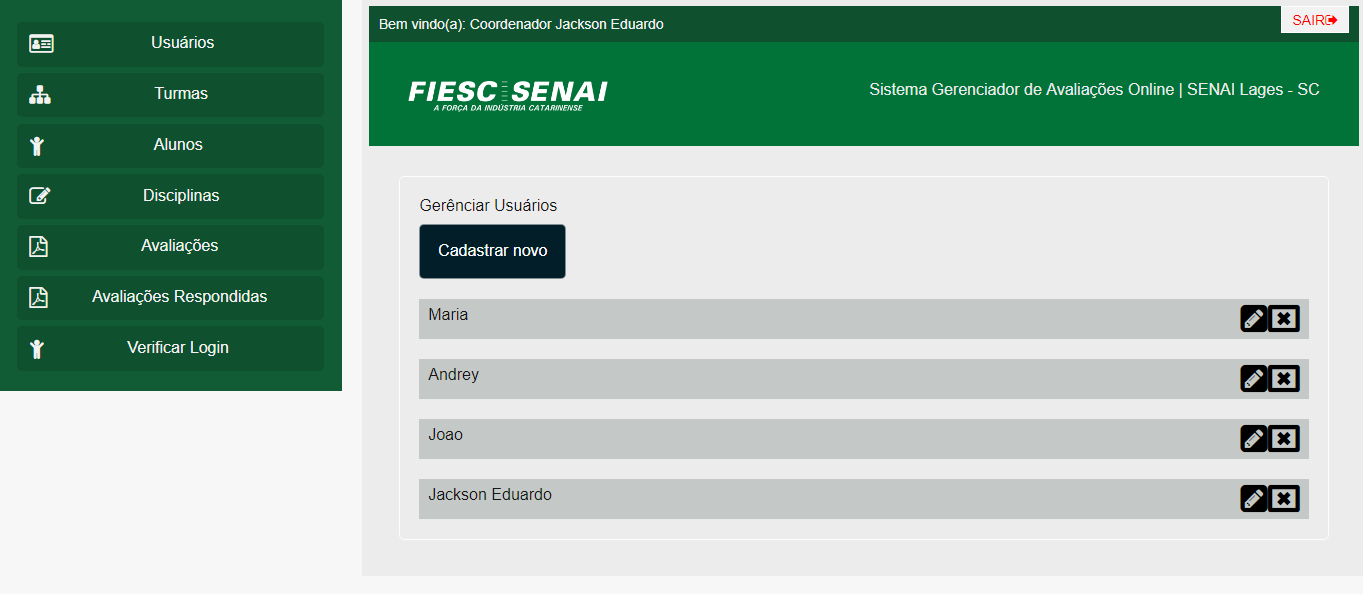


* + - 1. Nível coordenador

O acesso do tipo coordenador tem total gerenciamento das informações sobre o sistema, neste tópico serão apresentadas todas as telas iniciais que se fazem disponíveis neste nível de acesso, bem como as principais funcionalidades.

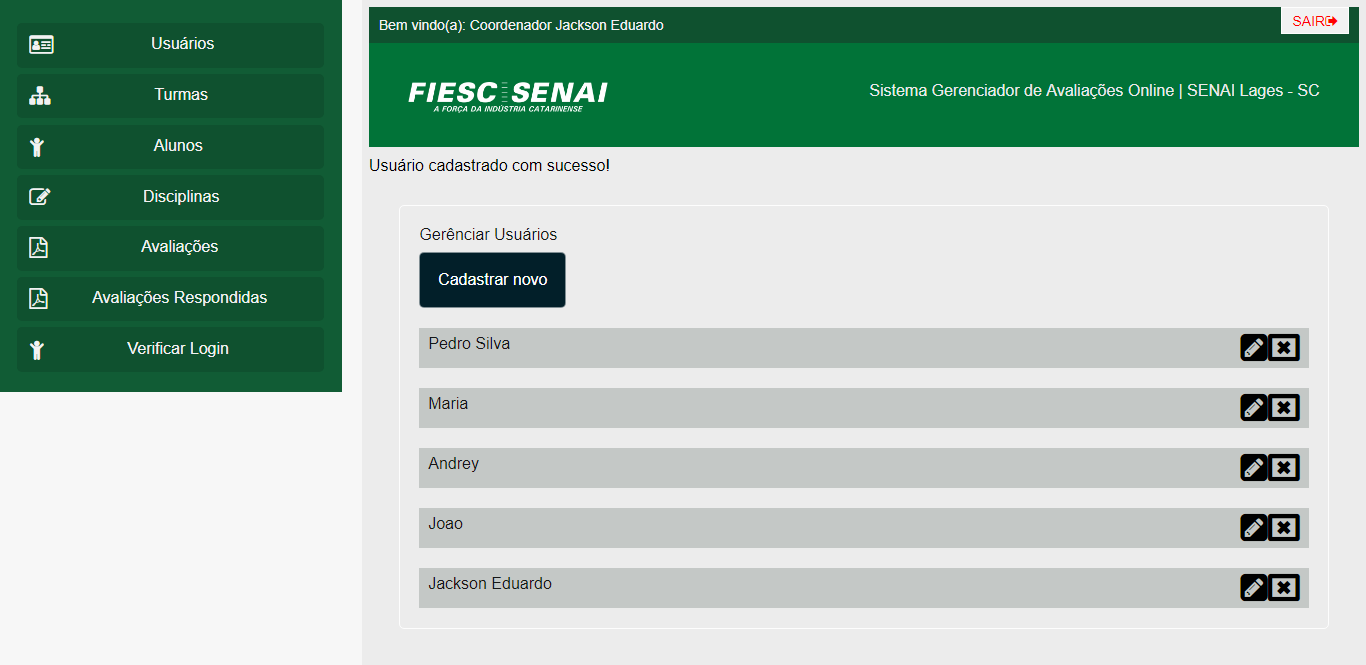
A imagem representada pela Figura 10 demonstra a tela de gerenciamento de usuários, onde é possível cadastrar um novo, editar e excluir.

1. Tela gerenciamento de usuários



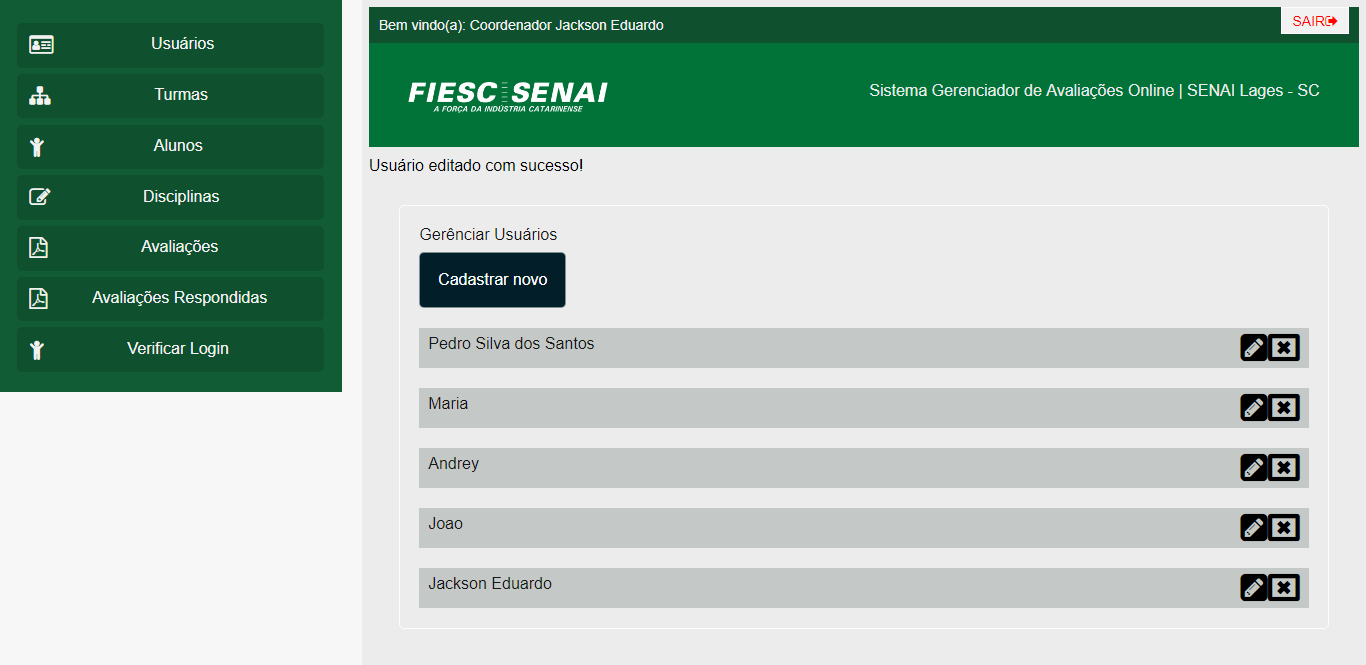
A imagem abaixo mostra um aviso ao usuário após este fazer uma ação sobre o sistema, sendo que no exemplo da Figura 11 este acabou de realizar o cadastro de um usuário sobre o sistema.

1. Mensagem de sucesso ao cadastrar um novo usuário



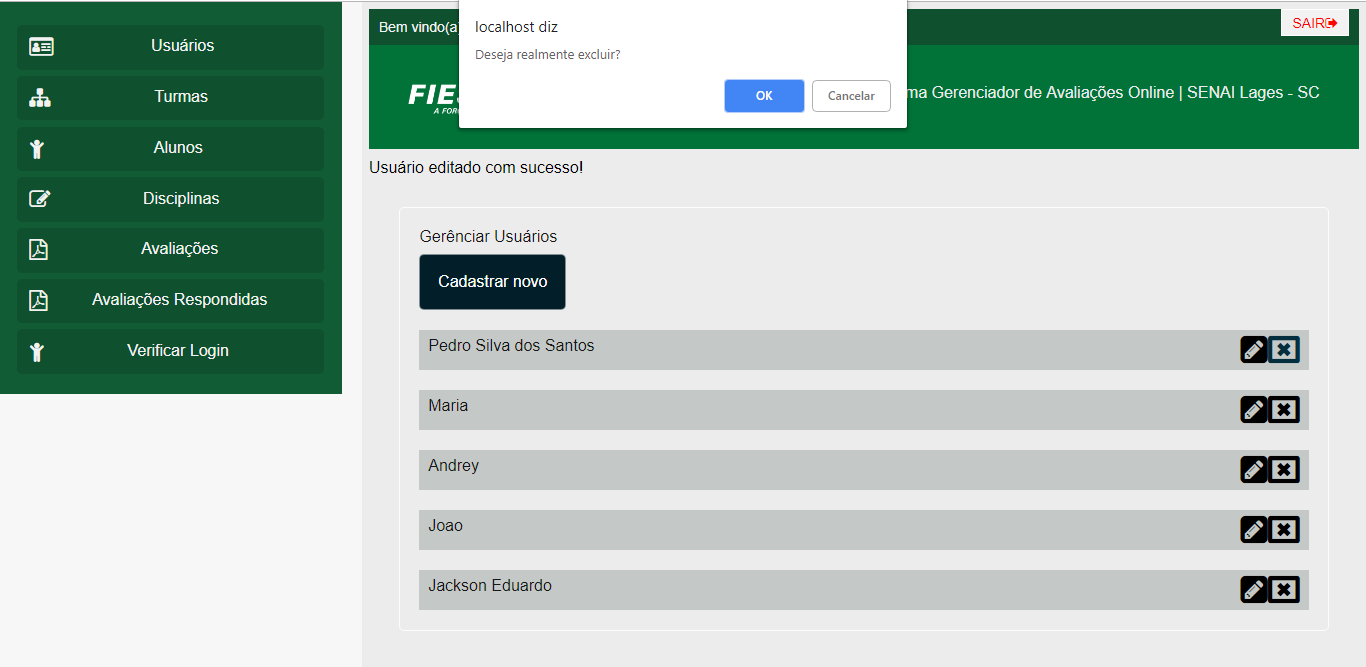
Após fazer uma edição sobre os dados de um usuário, a Figura 12 demonstra a mensagem de confirmação sobre esta ação executada no sistema.

1. Mensagem de sucesso ao editar um novo usuário



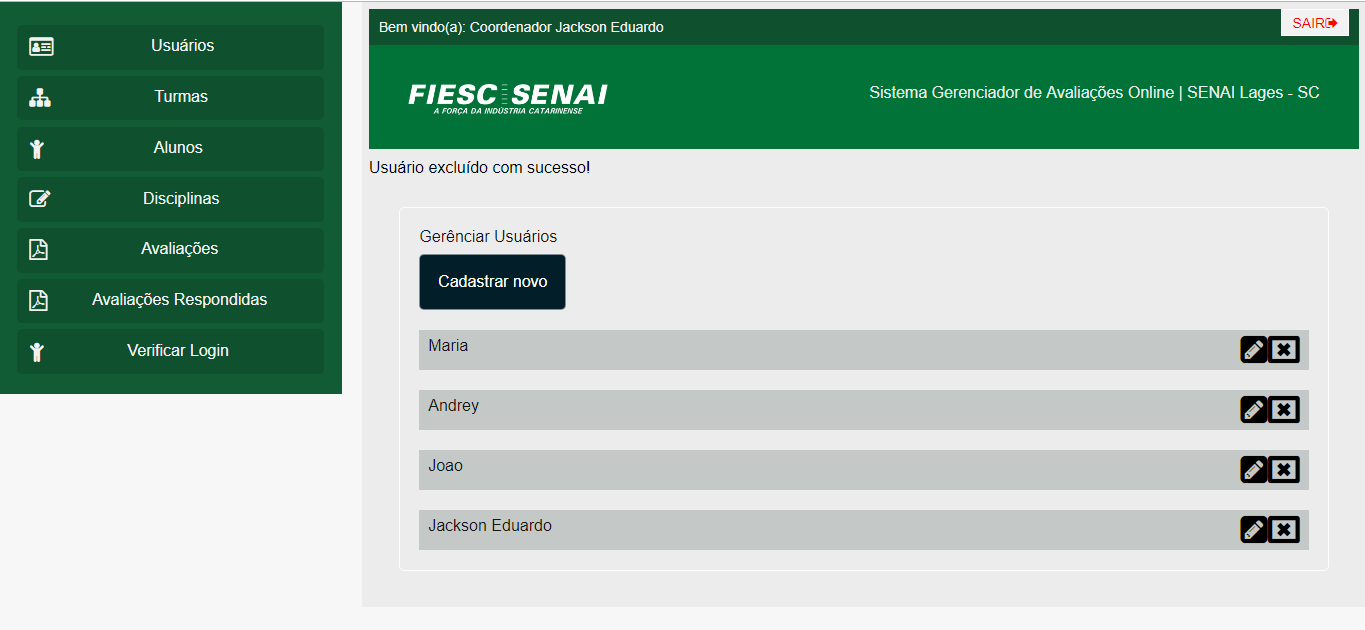
Se em um eventual descuido o usuário clique sobre o ícone de exclusão, o sistema irá abrir uma caixa de diálogo com uma confirmação sobre esta ação do usuário ao sistema, a Figura 13 demonstra um exemplo de quando esta ação é exercida.

1. Alerta de confirmação de exclusão de um usuário



Na circunstância do usuário confirmar a exclusão, este será redirecionado para a tela de gerenciamento de usuários, representada pela Figura 14 abaixo, sendo apresentado através de uma mensagem a confirmação de que tal ação foi executada com sucesso.

1. Mensagem de sucesso ao confirmar a exclusão um usuário



Ao clicar na opção turma no menu lateral do sistema, conforme Figura 15, o usuário irá se deparar com as turmas já cadastradas sobre o sistema e as opções de cadastramento edição e exclusão.

1. Tela gerenciamento de turmas



Navegando até a opção alunos no menu lateral do sistema, conforme Figura 16, o usuário irá se deparar com as turmas que já estão cadastradas na base de dados do sistema, ele deverá clicar sobre uma determinada turma para então visualizar todos os alunos que estão vinculados a esta, podendo cadastrar um novo aluno, editar e excluir.

1. Tela gerenciamento de alunos



Ao clicar na opção disciplinas no menu lateral de navegação, é possível visualizar as informações já cadastradas, conforme mostra a Figura 17, sendo possível cadastrar uma nova disciplina, editar e excluir as já existentes.

1. Tela gerenciamento de disciplinas



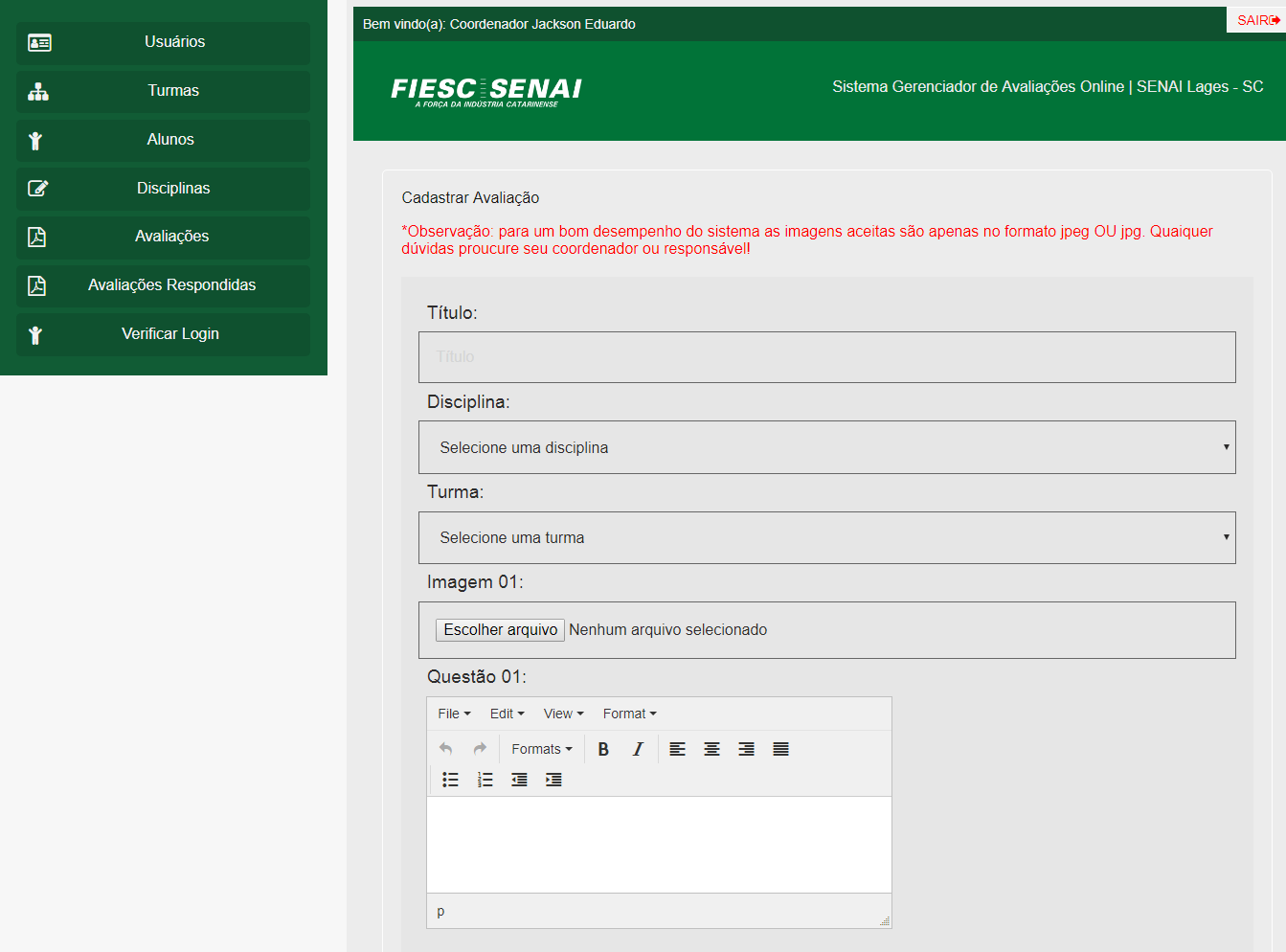
O usuário cadastrado como coordenador, poderá ter acesso a todas as avaliações cadastradas pelos demais níveis de usuários, a imagem representada pela Figura 18 mostra todas as avalições cadastradas, onde é possível visualizá-las em formato PDF, editar, excluir, habilitar e desabilitar a visualização para que os alunos não tenham acesso após ter respondido, ou antes mesmo da data de aplicação, possibilitando que o professor mantenha o espaço como um repositório de avaliações.

1. Tela gerenciamento de avaliações



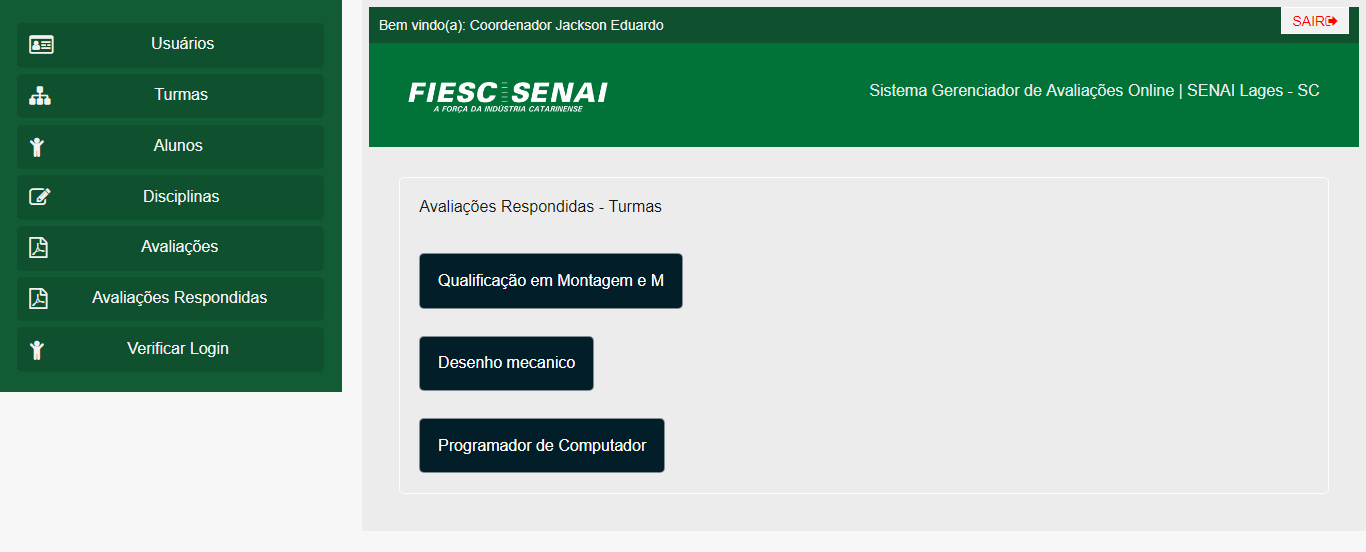
Quando clicado na opção “Cadastrar nova” na tela de avaliações, o usuário será direcionado para o cadastro de avaliações, a imagem representada pela Figura 19 abaixo, apresenta uma prévia do formulário de cadastramento, sendo que o professor deverá vincular a nova avaliação a uma Disciplina e Turma, onde é possível fazer um upload de uma imagem para cada questão, porem fica a critério do professor quando este julgar necessário tal ação. O número de questões cadastradas em uma única avaliação pode variar de uma a vinte, ficando também sob o domínio do professor esta decisão, no momento do cadastro das questões é possível descrever questões objetivas e subjetivas, não alterando em nada o modo de como o aluno irá responder.

1. Previa tela de cadastro de uma nova avaliação



A opção Avaliações Respondidas que se faz presente no menu do sistema, quando selecionada, direcionará o usuário para selecionar uma turma, conforme demonstra a Figura 20 abaixo.

1. Tela gerenciamento de avaliações respondidas - turmas



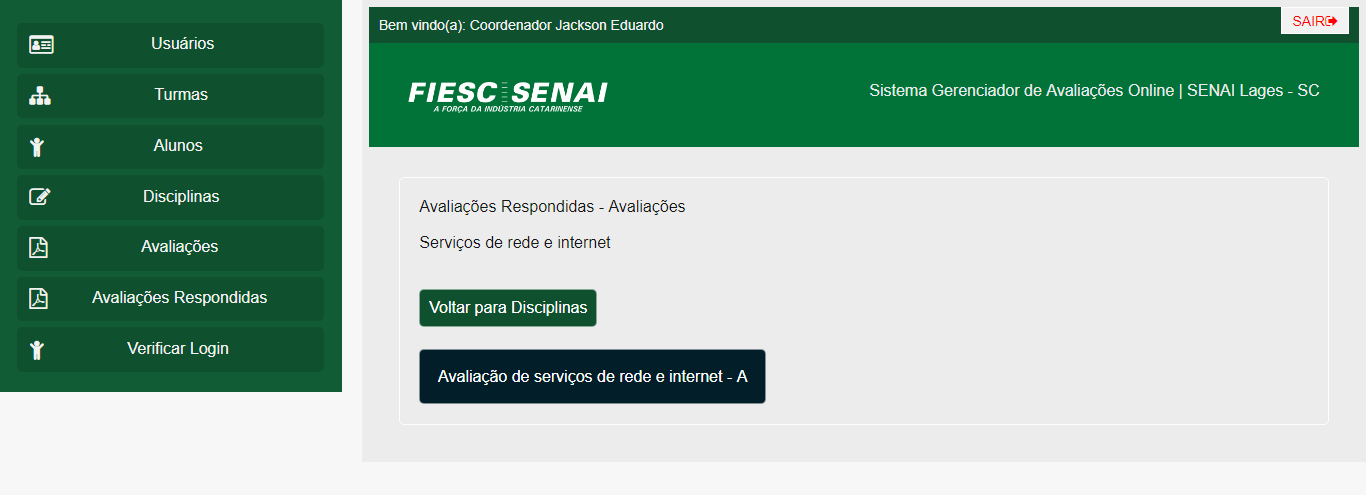
Todas as disciplinas vinculadas a turma escolhida pelo usuário, iram aparecer quando a tela for carregada, conforme Figura 21 abaixo, sendo possível também clicar sobre a opção “Voltar para Turmas”.

1. Tela gerenciamento de avaliações respondidas - disciplinas



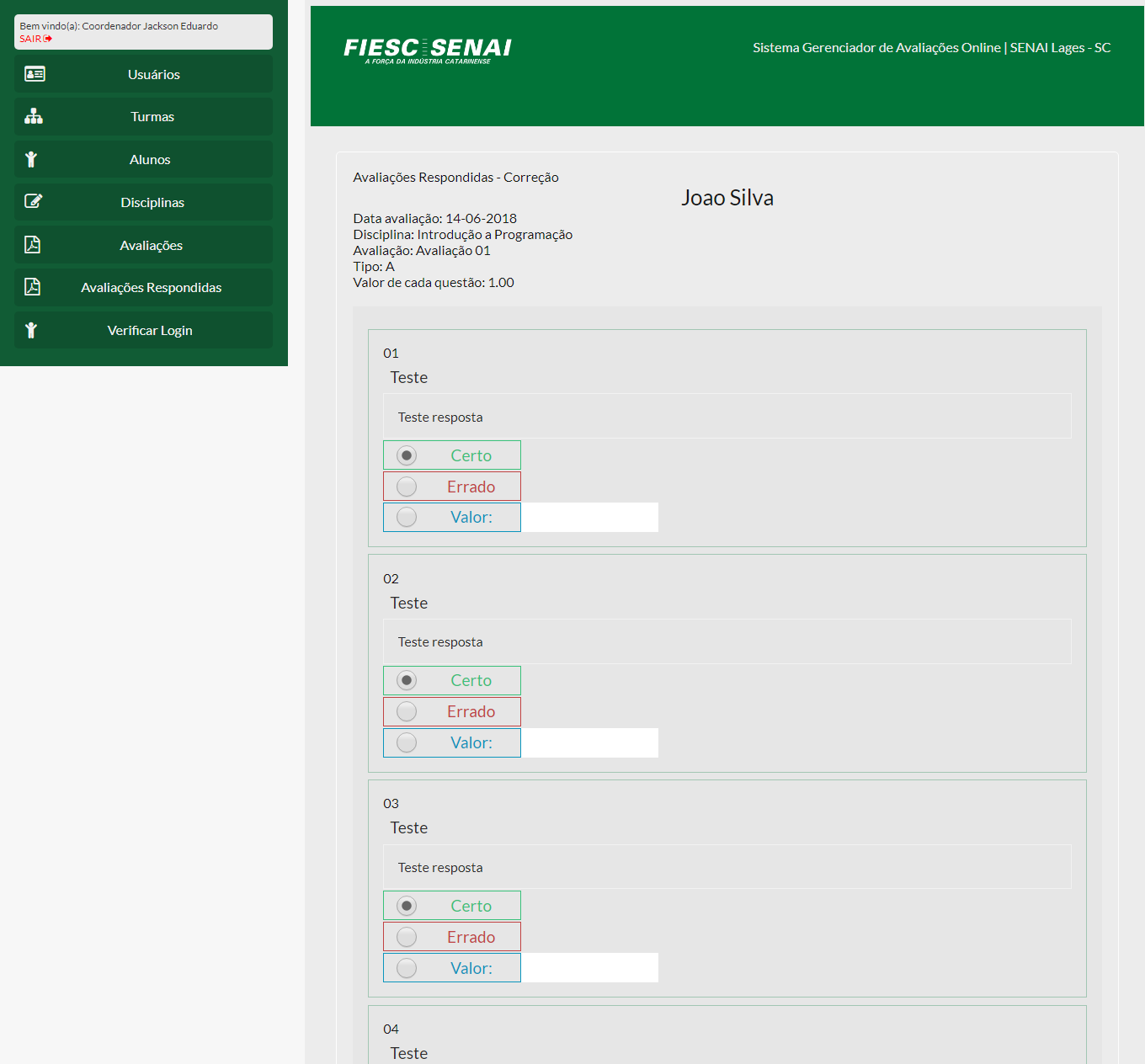
Por final irá aparecer todas as avaliações que foram vinculadas a turma e a disciplina, o exemplo da imagem abaixo representada pela Figura 22, mostra uma avaliação de exemplo.

1. Tela gerenciamento de avaliações respondidas - avaliações



Ao selecionar uma avaliação para correção, será possível identificar informações de quando foi respondida, a qual disciplina esta pertence, nome da avaliação, tipo e valor de cada questão, conforme demonstra a Figura 23, sendo que o valor das questões o sistema irá calcular de acordo com a quantidade cadastradas, levando em consideração o peso total com valor 10, o cálculo é feito da seguinte maneira: supondo uma avaliação com 5 questões, este irá sugerir que cada uma valerá 2 pontos, porém há também um campo onde é possível atribuir um valor específico para cada questão, quando o professor for corrigir, este poderá então marcar a opção de Certo, Errado ou digitar um valor especifico.

1. Previa tela de correção de uma avaliação



Na Figura 24 é possível identificar que após feita a correção da avaliação a nota será calculada e apresentada automaticamente de acordo com as questões que o aluno acertou, podendo ainda ser feito uma edição na correção.

1. Tela gerenciamento de avaliações respondidas - alunos



Após a correção, será possível visualizar a avaliação no formato de arquivo PDF, para que facilite o download caso necessário e um impressão para simples conferência em uma reunião de pais por exemplo, basta o usuário clicar sobre o nome do aluno, onde a Figura 25 demonstra uma avaliação sendo apresentada no formato PDF.

1. Previa avaliação corrigida em PDF



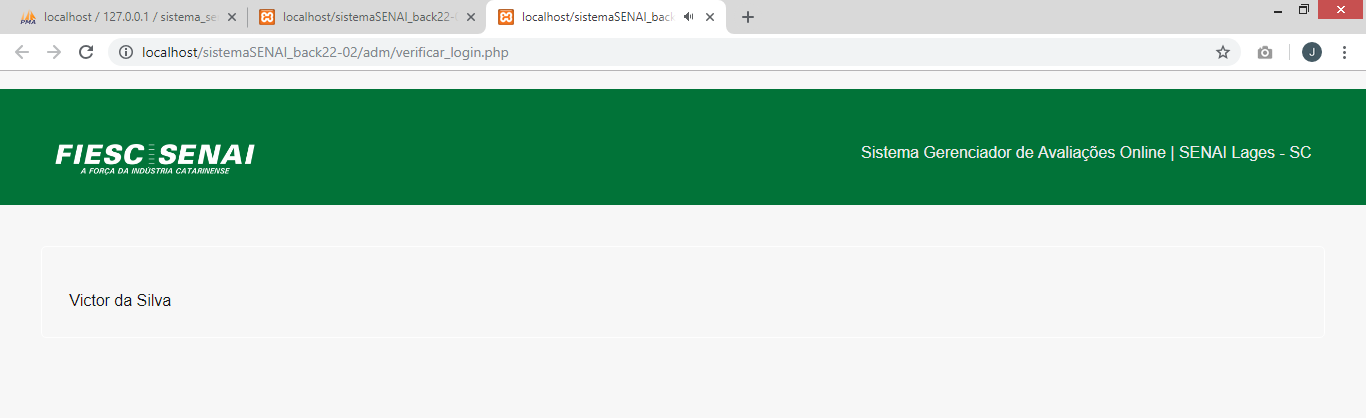
A figura 26 demonstra um fragmento de código utilizado pela classe FPDF, utilizada para gerar a avaliação no formato de arquivo PDF.

1. Fragmento de código classe FPDF



A Figura 27 representa o layout da tela verificação de “login”, esta página irá se abrir em uma nova guia do navegador quando selecionada no menu lateral do sistema, isto porque a mesma irá ser atualizada automaticamente a cada 10 segundos, filtrando por alunos que tenham tomado ações indevidas sobre o sistema no momento que estiverem respondendo uma avaliação, esta verificação irá auxiliar o professor a identificar o momento exato em que o aluno teve tal atitude, pois a página irá emitir um sinal sonoro, tal funcionalidade estará disponível sem nenhuma diferenciação para acesso nos perfis de regente e professor.

1. Verificação de alunos bloqueados



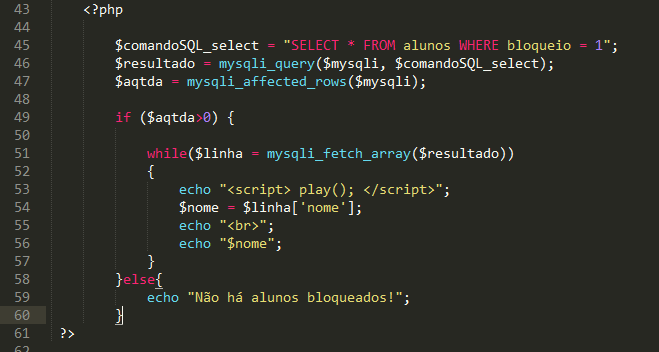
Parte do código demonstrado na Figura 28, foi implementado na linguagem JavaScript, onde apresenta uma função de atualização da página a cada 10000 milissegundos que equivalem a 10 segundos, seguido da reprodução de um áudio a cada nova atualização, caso encontre um aluno com os dados de acesso bloqueado.

1. Código atualização página de alunos bloqueados



A Figura 29 apresenta um comando SQL que busca todos os registros da tabela alunos com a condição de que o campo bloqueio deve estar com o valor 1, caso a condição seja satisfeita, a página irá apresentar na tela os nomes dos respectivos alunos e chamar a função play do JavaScript para executar o disparo do sinal sonoro.

1. Código verificação de alunos bloqueados



* + - 1. Nível regente

O acesso com nível do tipo regente tem um gerenciamento reduzido das informações e dados sobre o sistema se comparado com o nível coordenador, os usuários com este nível terão como responsabilidade principal o cadastro de novos alunos sobre a turma o qual foi designado para tal tarefa, neste tópico será apresentado todas as telas iniciais que se fazem disponíveis neste nível de acesso, bem como as principais funcionalidades.

A Figura 30 abaixo mostra a tela do primeiro item que faz parte do menu de navegação do sistema, listando as turmas que o regente foi encarregado por um coordenador para cadastrar os alunos que iram pertencer a tais, ao clicar no ícone representado por um sinal de + (mais), o professor irá se deparar com um formulário para cadastro de novo aluno.

1. Cadastro de novos alunos



A imagem abaixo ilustrada pela Figura 31, apresenta as turmas que faz parte de sua regência, para que este possa fazer as edições quando julgar necessário.

1. Gerenciamento de alunos por turma



A tela do sistema ilustrada pela Figura 32 mostra os alunos de uma respectiva turma, sendo que o professor poderá a qualquer momento que julgar necessário habilitar e desabilitar o acesso ao sistema de um determinado aluno bem como editar e excluir seus dados de acesso.

1. Gerenciamento de alunos por turma



Diferente do nível de coordenador, o regente só poderá gerenciar as avaliações que ele cadastrou, a FIGURA 33 - abaixo apresenta o layout da tela de listagem de avaliações, o exemplo demonstra apenas uma avaliação cadastrada para a turma escolhida, porem é possível cadastrar mais avaliações, habilitar/desabilitar, editar e excluir.

1. Gerenciamento de avaliações



Ao selecionar o item Avaliações Respondidas no sistema, o professor irá se deparar com as turmas cadastradas, os demais procedimentos para a correção da avaliação são idênticos aos já apresentados anteriormente, que são representados pelas Figuras 21 à 25.

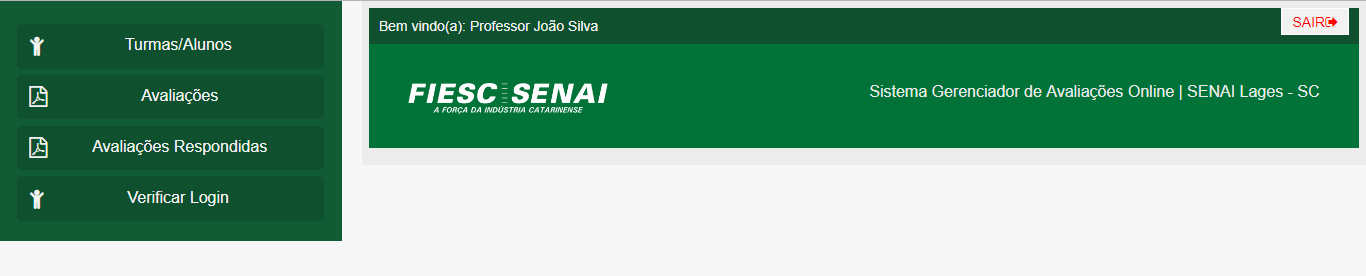
1. Gerenciamento de avaliações respondidas



* + - 1. Nível professor

Professor tem um gerenciamento reduzido das informações e dados sobre o sistema se comparado com os níveis de coordenador e regente, pois somente irá gerenciar: bloqueio de alunos, cadastrar e gerenciar avaliações.

1. Acesso nível professor

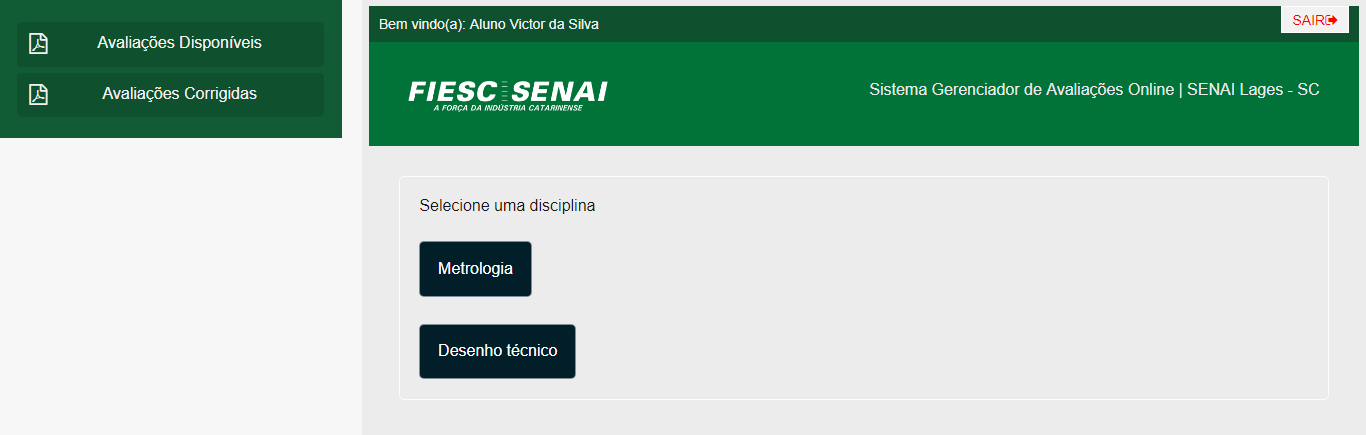


* + - 1. Nível aluno

Cada aluno receberá seus dados de acesso ao sistema, sendo que ficara sob responsabilidade de um coordenador ou regente de turma fazer a entrega destes dados após cadastrados. Ao acessar, o aluno se depara com duas opções, são elas: avaliações disponíveis e avaliações corrigidas, esta seção irá apresentar as principais telas e fragmentos de códigos que estão disponíveis aos níveis de acesso do tipo aluno.

A Figura 36 abaixo, mostra a tela de avaliações disponíveis, onde o aluno deverá selecionar uma disciplina.

1. Avaliações disponíveis



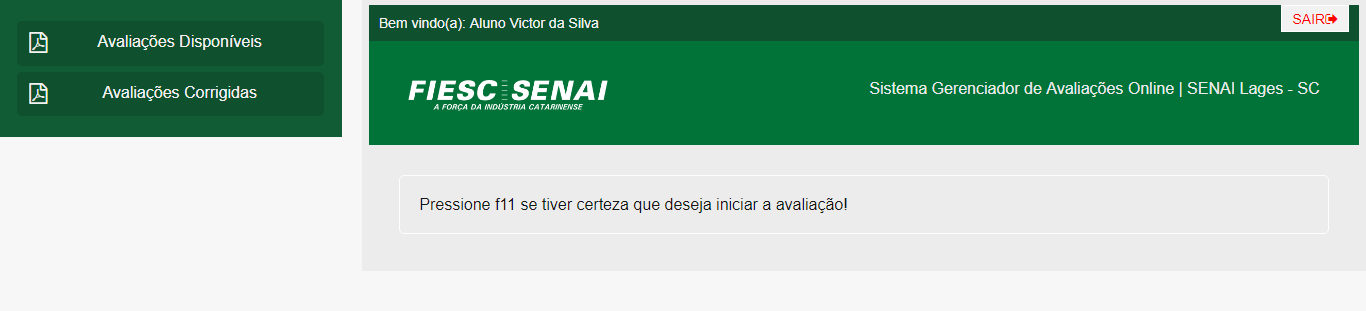
Conforme demonstrado pela Figura 37, após selecionar uma disciplina especifica, o aluno irá se deparar com a avaliação para ser respondida, lembrando que o professor tem total autoridade em habilitar e desabilitar a avaliação, para que assim o aluno não tenha acesso após ter respondido, ou antes mesmo de conduzir a turma para o laboratório de informática e iniciar os procedimentos de aplicação da prova.

1. Nível aluno – avaliações disponíveis, seleção de avaliação



O próximo passo, logo que o aluno seleciona a avaliação para responder, será confirmar o início da mesma, para isso irá aparecer uma mensagem informando que é necessário pressionar a tecla F11 como mostra a Figura 38, o navegador irá ficar no modo tela cheia, restringindo assim o discente de qualquer outra operação que não seja a de responder a sua avaliação.

1. Confirmação início de avaliação



O código fonte da Figura 39 abaixo verifica o momento exato em que o aluno pressiona a tecla f11, fazendo com que o navegador fique no modo tela cheia, o evento em JavaScript identifica a ação e direciona o usuário para a página que está definida na função *location*.

1. Código fonte verificação de tecla pressionada



Ao início e durante o procedimento em que o aluno estará respondendo a avaliação, ele visualizara uma mensagem de atenção, notificando que algumas teclas não poderão ser pressionadas, de acordo como mostra o exemplo da Figura 40.

1. Tela com avaliação já iniciada

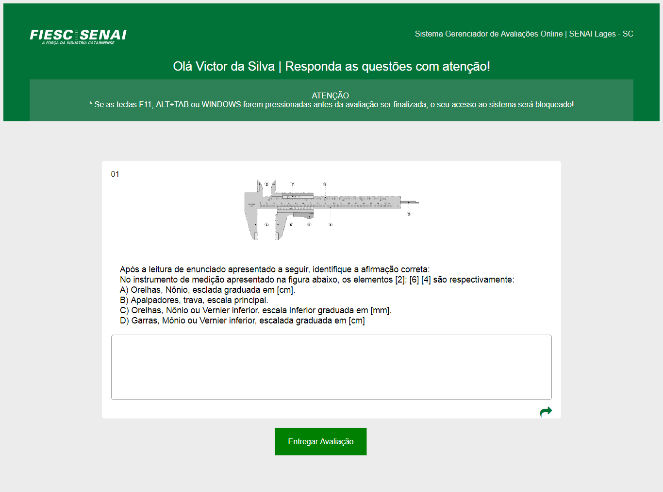
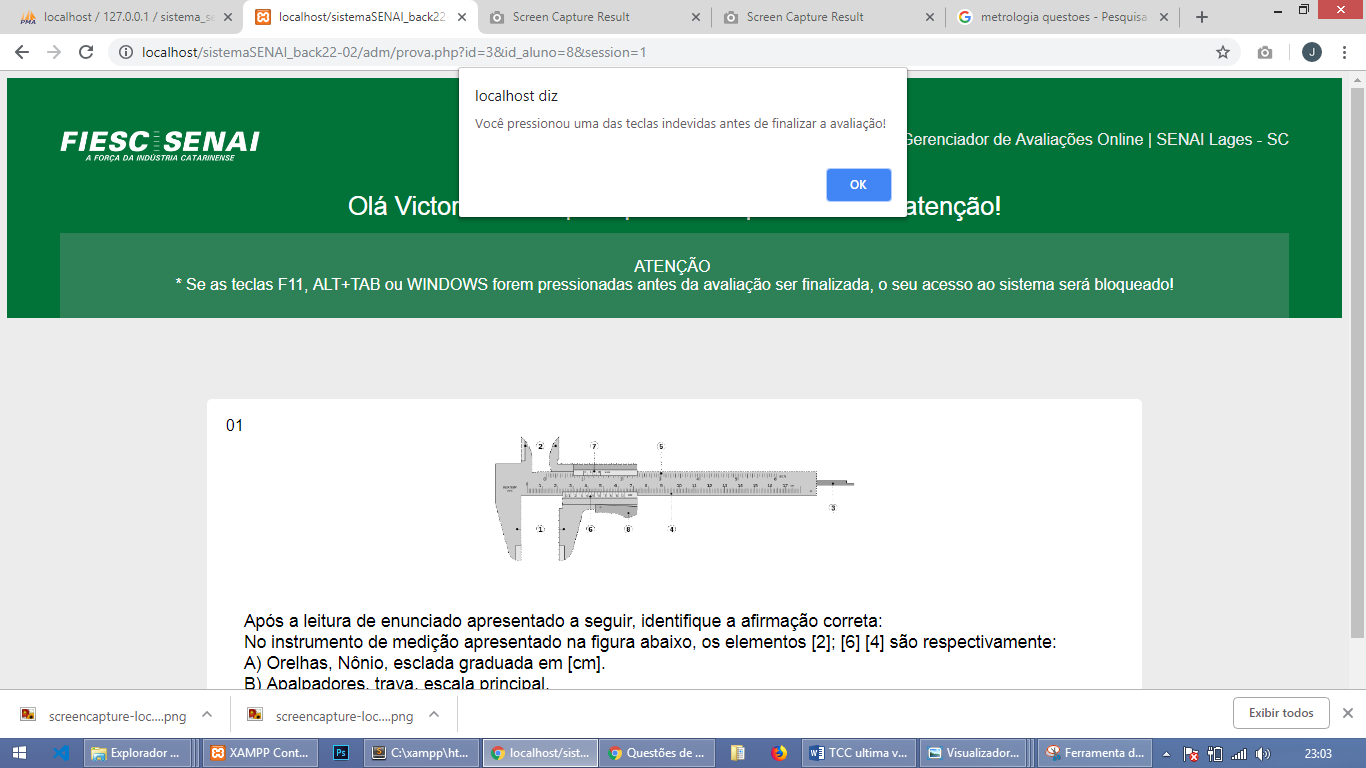


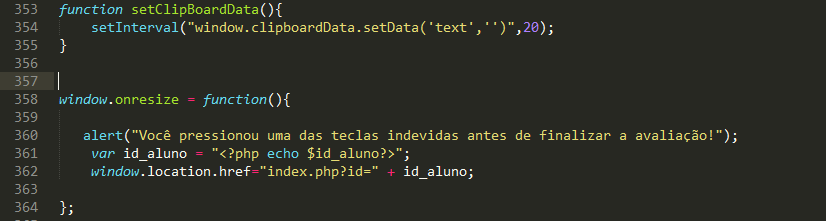
Figura 41 abaixo demostra o alerta de mensagem de tecla indevida pressionada, neste caso o aluno será redirecionado a tela de autenticação do sistema.

1. Alerta de tecla indevida pressionada



O código fonte apresentado na Figura 42 abaixo, identifica qualquer tipo de alteração na resolução da página, sendo que se o aluno pressionar a combinação das teclas Alt+Tab por exemplo, a função já irá identificar tal ação.

1. Código fonte alteração na resolução da tela



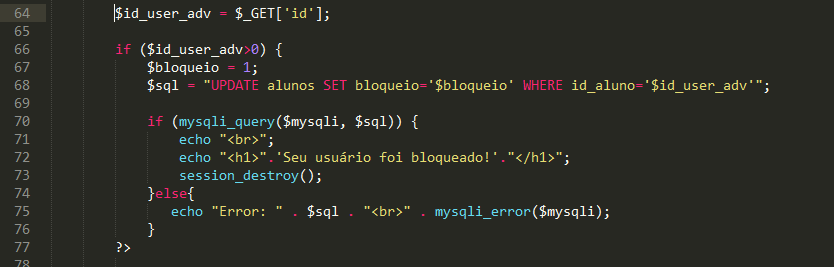
Os dados de acesso do aluno ficaram bloqueados e este será redirecionado a tela do formulário de autenticação do sistema conforme demonstra Figura 43, sendo que no mesmo momento que esta situação ocorrer, o sistema notificará o professor com um sinal sonoro, sinalizando assim qual o aluno que tomou esta decisão, ficando de responsabilidade do professor em decidir em liberar ou não o aluno para autenticar-se ao sistema novamente e dar continuidade a avaliação.

1. Aviso de usuário bloqueado



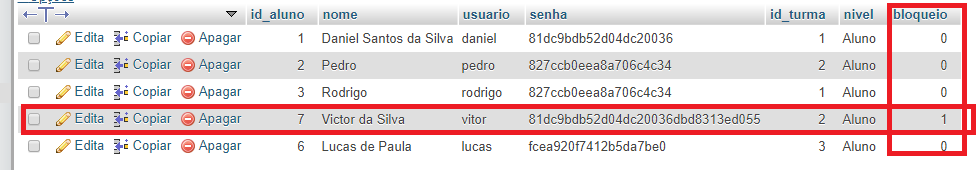
A Figura 44 abaixo demonstra o código fonte identificando o id de um aluno e fazendo uma atualização via SQL no campo bloqueio da tabela alunos.

1. Código fonte SQL UPDATE tabela alunos



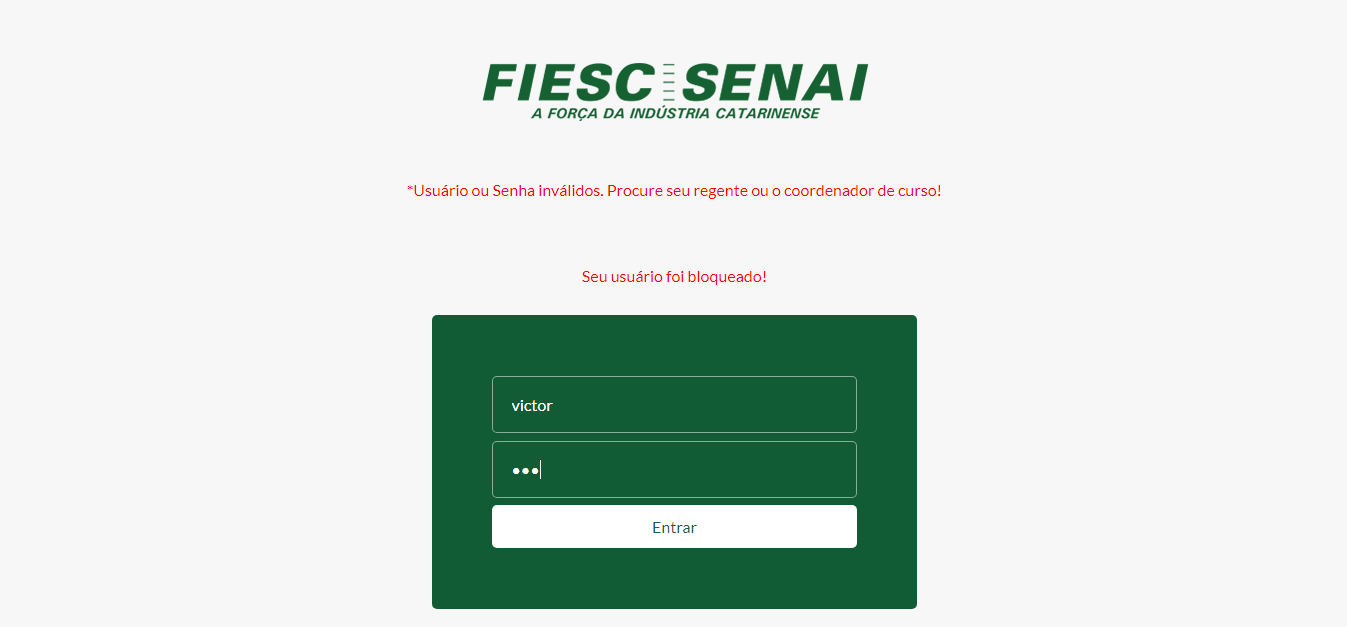
Já na figura 45 logo abaixo, é possível visualizar o resultado da ação do código fonte demostrado pela Figura 44 acima, sendo exibido os dados que já estão cadastrados na tabela alunos, com ênfase no campo bloqueio, onde o exemplo mostra o aluno Victor da Silva com valor 1, isto significa que este não poderá autenticar-se ao sistema novamente até que um professor ou responsável faça o desbloqueio de seu acesso, sendo que se realizada esta ação, o campo bloqueio do respectivo aluno irá atualizar e ficar com o valor 0, permitindo assim o acesso do aluno ao sistema novamente.

1. Dados da tabela alunos



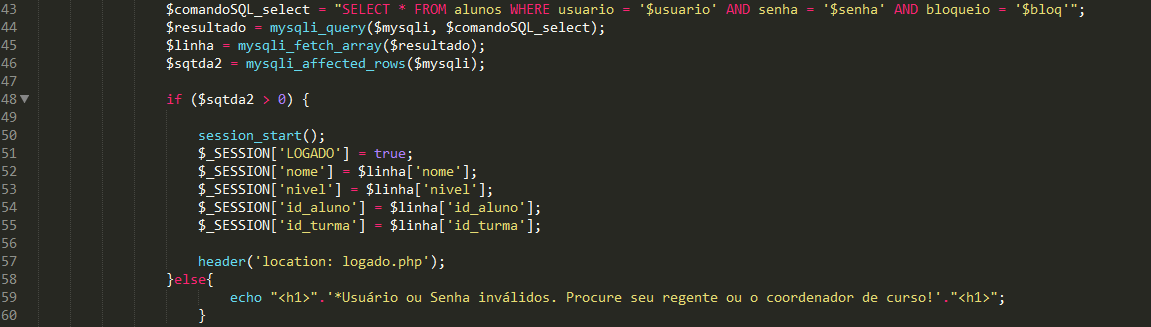
Em sequência a tela demonstrada pela figura 46 abaixo representa o aluno tentando autenticar-se no sistema novamente, porem acaba sendo restrito e alertado de que os seus dados são inválidos, isso porque no momento em que ele pressionou uma das teclas indevidas automaticamente foi enviado o valor 1 para o campo bloqueio na tabela alunos, anulando seus dados de acesso.

1. Tentativa de acesso usuário bloqueado



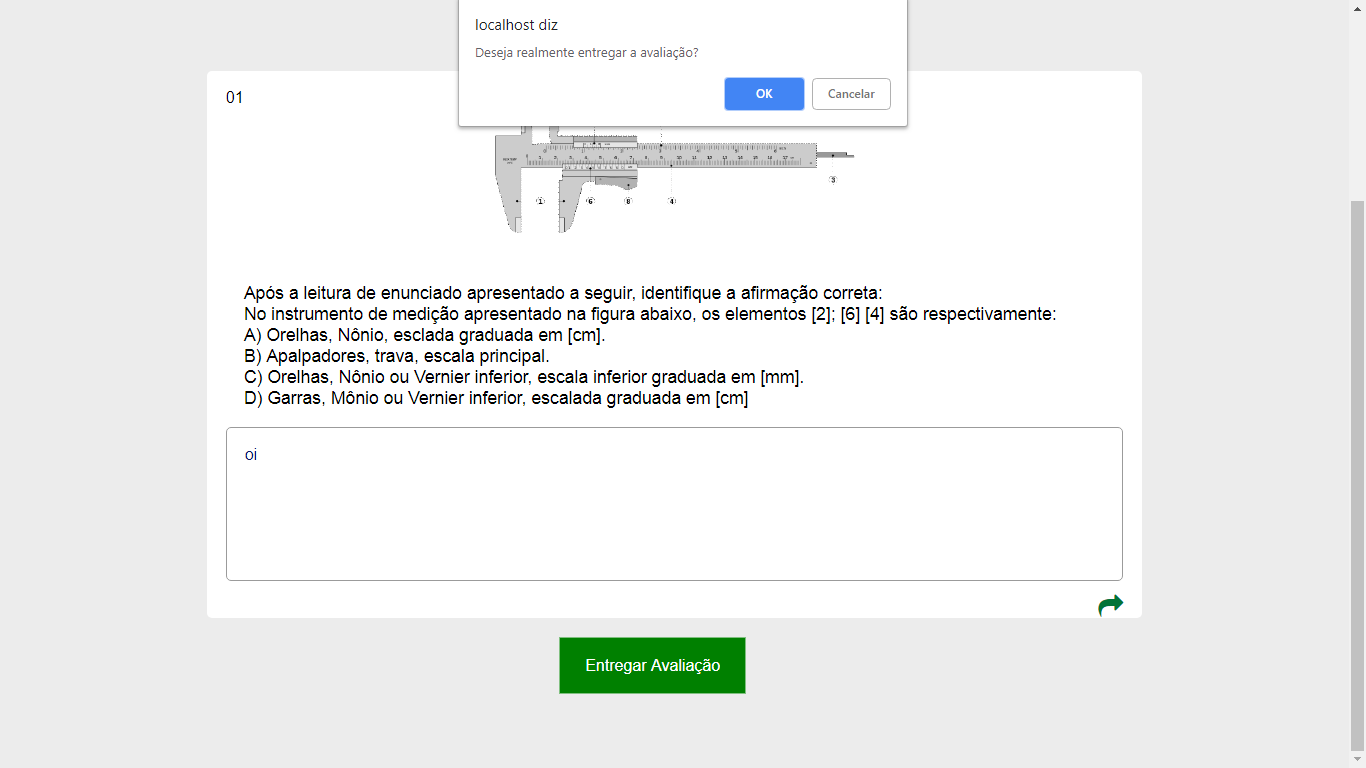
O fragmento de código abaixo representado pela Figura 47, mostra um comando SQL com a verificação do campo usuário, senha e bloqueio na tabela alunos, sendo a verificação satisfeita com sucesso o aluno poderá ter acesso a página inicial do sistema novamente, caso contrário irá se deparar com o seguinte aviso: \*Usuário ou Senha inválidos. Procure seu regente ou o coordenador de curso!

1. Código fonte SQL SELECT tabela alunos



A Figura 48 mostra um exemplo de uma ação de clique no botão Entregar Avaliação, em seguida o sistema apresenta uma caixa de diálogo solicitando a confirmação exercida pelo usuário, se a opção escolhida for cancelar, a caixa apenas irá se fechar, fazendo com que o usuário permanecerá sobre a tela de realização da avaliação.

1. Alerta de confirmação de entrega avaliação



Caso o usuário clique na opção OK na caixa de diálogo que se faz presente na imagem demonstrada na Figura 48 acima, o sistema irá encerrar a seção e irá direcionar o usuário para uma tela, conforme Figura 49 abaixo, alertando que a avaliação foi enviada com sucesso, disponibilizando um botão com a opção para que o aluno possa voltar a tela inicial, onde se faz presente o formulário de autenticação ao sistema novamente.

1. Aviso de avaliação enviada com sucesso



Ao selecionar o item Avaliações Corrigidas o aluno irá se deparar com as respectivas disciplinas da turma a qual ele faz parte, conforme Figura 50.

1. Tela de avaliações corrigidas - disciplinas



Após uma disciplina selecionada, deve-se escolher uma das avaliações que fazem parte da respectiva disciplina, fato este sendo demonstrado pela Figura 51.

1. Tela de avaliações corrigidas



A tela da Figura 52 abaixo identifica se existe resposta do aluno sobre a avaliação selecionada, o exemplo demonstra que o aluno ainda não respondeu tal avaliação.

1. Aviso de avaliação sem resposta



Em seguida a Figura 53 abaixo identifica se um professor efetuou a correção da avaliação selecionada, o exemplo demonstra que este fato ainda não ocorreu.

1. Aviso de avaliação sem correção



Já a tela da Figura 54, representa que avaliação selecionada já foi corrigida, apresentando o nome do aluno seguido de sua nota, sendo que o mesmo pode clicar em seu nome para abrir a avaliação no formato PDF com a devida correção, após esta ação o arquivo avaliação será aberto em uma nova guia do navegador, garantindo assim a navegação ainda sobre o sistema, sem ter a necessidade de fechar a visualização de sua avaliação.

1. Avaliação corrigida



* 1. Conclusão

Dado o exposto das principais telas do sistema num todo, é imprescindível deixar de citar o fato do quanto é trabalhoso conciliar todos os requisitos de funcionalidades sobre o desenvolvimento de um sistema, o qual irá gerenciar informações sobre o conhecimento de alunos inseridos na ferramenta após as abordagens dos professores em sala de aula, sendo que durante o capitulo foi possível observar as diferenciações de acesso do conteúdo no sistema, contidas em cada nível de usuário.

Após intendido o aspecto de funcionamento da ferramenta apresentada neste capítulo, conclui-se que os aspectos da tecnologia podem garantir a evolução, como por exemplo, a tarefa da aplicação e gerenciamento das avaliações, que são instrumentos de mediação de um conhecimento gerido por um professor.

Ressalta-se ainda que o desenvolvimento de um sistema de nível como o apresentado neste capitulo só flui quando as dificuldades são encaradas como parte do processo, e sanadas com o a evolução sobre a absorção de novos conhecimentos, pois a complexidade sobre algo só é realmente entendida quando se é preciso fazer parte dela, e para isso a persistência e determinação precisam ser encaradas de forma precisa para que algo realmente funcione e seja concluído.

* 1. Validação

Neste capítulo será apresentado a metodologia utilizada para validação das principais funcionalidades do sistema, bem como o resultado da pesquisa aplicada aos participantes envolvidos.

A validação ocorreu nos dias 3 e 4 de dezembro de 2018 na instituição de ensino Serviço Nacional de Aprendizagem Industrial – SENAI em Lages SC, com uma turma de aprendizagem industrial em programador de computador, o cenário de testes realizou-se em um laboratório de informática com a presença de alunos e professores da referida turma, dentre os envolvidos, 14 discentes e 3 docentes.

* + 1. Teste do sistema

O propósito do cenário de testes fundamentou-se em validar o funcionamento de cadastro, edição e exclusão de: Usuários, Turmas, Alunos, Disciplinas, Avaliações, Correção de Avaliações e Verificação de Acessos Indevidos, sendo que esta etapa de testes foi realizada pelos docentes, já os discentes deveriam acessar o sistema e inicializar o processo para descrever as respostas de uma avaliação, realizando também ações indevidas sobre o sistema.

* + - 1. Perfil coordenador

Foi escolhido o perfil de coordenador para exercer boa parte dos testes, pois este perfil pode ter acesso a todas as funcionalidades do sistema, diante deste fato foi submetido que um professor se autenticasse e realizasse todos os cadastros, edições e exclusões possíveis, sempre checando os dados para garantir a assiduidade das informações.

* + - 1. Perfil aluno

Após cada aluno autenticar-se ao sistema, foi solicitado para que selecionassem uma avaliação que se encontrasse disponível, orientando-os para que fizessem os devidos testes de inserção de respostas, utilização de teclas indevidas para checagem de cancelamento de acesso do seu usuário, finalizando com a visualização de uma avaliação no formato PDF, após corrigida por um professor.

* + - 1. Formulário para pesquisa de avaliação

Finalizado a etapa de testes de funcionamento e experiência sobre o sistema, alunos e professores respondiam um formulário de avaliação. As questões abordadas neste formulário, visavam obter dados cruciais para quantificar as possíveis dificuldades que os professores e alunos encontram ao trabalhar com métodos de gerenciamento de avaliações de forma manual.

Foi estruturado dois formulários para uma melhor apresentação das respostas, sendo um para os discentes e outro para os docentes.

* + - * 1. Discente

As questões abaixo foram utilizadas no formulário de avaliação do discente, estas apresentadas com a pergunta seguida de suas respectivas alternativas, podendo o aluno apenas escolher uma alternativa, sendo elas:

1. Você utilizaria um sistema de avaliações online presencial, reduzindo assim impressões e garantindo um ambiente mais sustentável?

( ) SIM

( ) NÃO

1. No seu ponto de vista, qual a melhor maneira de se responder uma avaliação teórica?

( ) De forma manual

( ) De forma informatizada

1. Quanto tempo em média seu professor demora para postar a nota de uma avaliação?

( ) De 1 à 3 dias

( ) De 4 à 5 dias

( ) Mais de 5 dias

( ) De uma a duas semanas

( ) Duas semas ou mais

1. Você acredita que, a informatização do processo de aplicação e gerenciamento de avaliações, possa trazer benefícios em relação ao método utilizado atualmente?

( ) SIM

( ) NÃO

1. Em uma escala de 0 a 10, qual seu nível de satisfação ao utilizar um sistema de avaliações online?
   * + - 1. Docente

As perguntas abaixo foram estruturadas com uma pergunta cada, seguida por suas alternativas, entretanto o professor ficou restrito da escolha de somente uma alternativa por questão, sendo as elencadas:

1. Você possui um repositório de avaliações em um único lugar específico?

( ) SIM

( ) NÃO

( ) Não sei responder

1. Quanto tempo em média você leva para corrigir uma prova com questões objetivas?

( ) Menos de um minuto

( ) De 1 à 2 minutos

( ) De 3 à 4 minutos

( ) Mais de quatro minutos

1. Quanto tempo em média você leva para corrigir uma prova com questões subjetivas?

( ) Menos de um minuto

( ) De 1 minuto a 2

( ) De 2 minutos a 3

( ) De 3 minutos a 4

( ) Mais de quatro minutos

1. Quanto tempo em média você leva para corrigir uma prova com questões objetivas e subjetivas?

( ) Menos de um minuto

( ) De 1 minuto a 2

( ) De 2 minutos a 3

( ) De 3 minutos a 4

( ) Mais de quatro minutos

1. Qual a média de alunos numa turma que você aplica provas teóricas frequentemente?

Menos de 15 alunos

De 15 a 20 alunos

De 20 a 25 alunos

De 25 a 30 alunos

Até 40 alunos

Mais de 40 alunos

1. Ao concluir as correções das avaliações, no mesmo instante você repassa as notas para um ambiente online onde os alunos possam visualizar?

SIM

NÃO

As vezes

1. Você gostaria de possuir um repositório para manter suas avaliações online, podendo ter acesso a elas de qualquer lugar e a qualquer momento?

( ) SIM

( ) NÃO

( ) Não sei responder

1. Em média quantas avaliações você aplica em uma única turma no trimestre?

( ) De 1 à 2 avaliações

( ) De 2 à 3 avaliações

( ) De 3 à 4 avaliações

( ) Mais de 4 avaliações

1. Você utilizaria o sistema de avaliações online presencial, reduzindo assim impressões e garantindo um ambiente mais sustentável?

( ) SIM

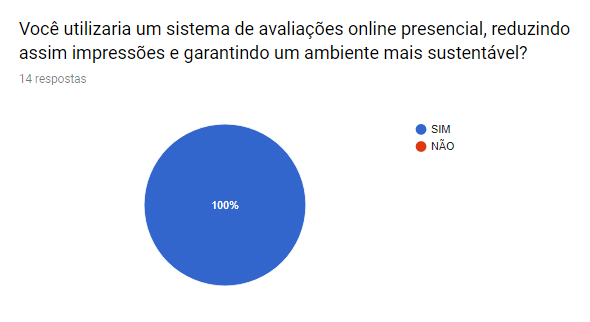
( ) NÃO

1. Em uma escala de 0 a 10, qual seu nível de satisfação ao utilizar um sistema de avaliações online?
   * 1. Resultado da pesquisa de avaliação

Por meio de 2 formulários, sendo um respondido por 14 discentes e outro por 3 docentes, os devidos resultados serão apresentados através de gráficos.

* + - 1. Resultado discentes

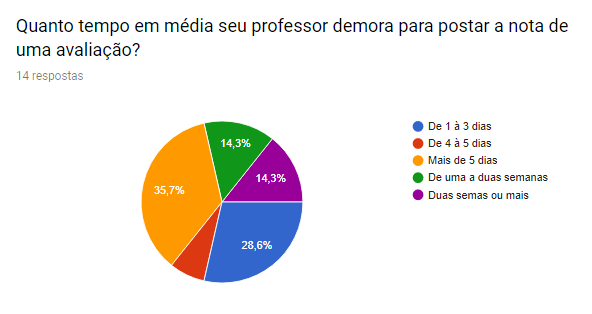
1. Primeira pergunta – Avaliação discente



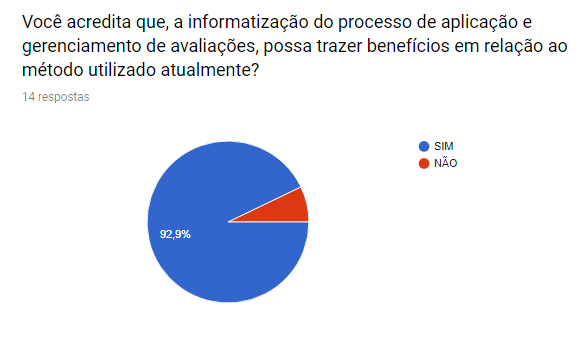
1. Segunda pergunta – Avaliação discente



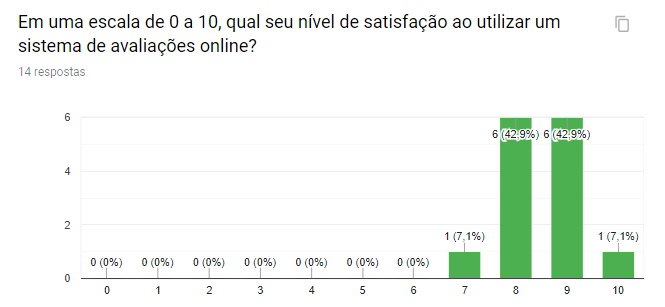
1. Terceira pergunta – Avaliação discente



1. Quarta pergunta – Avaliação discente

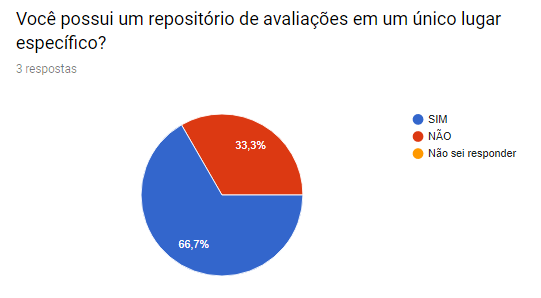


1. Quinta pergunta – Avaliação discente



* + - 1. Resultado docentes

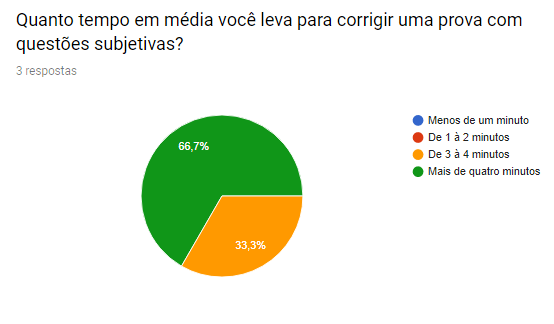
1. Primeira pergunta – Avaliação docente



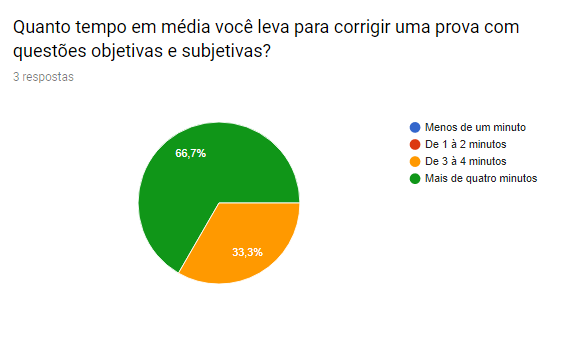
1. Segunda pergunta – Avaliação docente



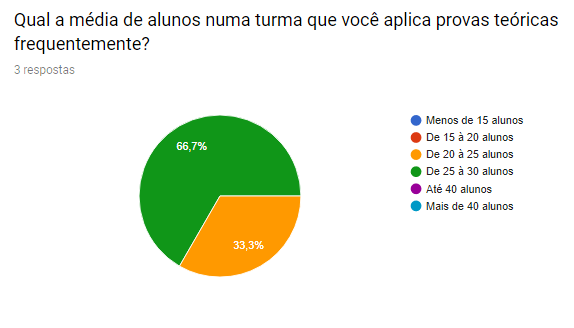
1. Terceira pergunta – Avaliação docente



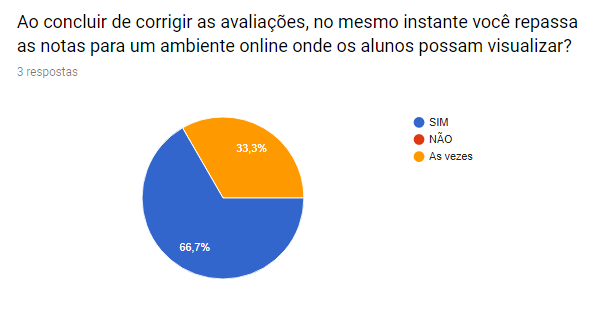
1. Quarta pergunta – Avaliação docente



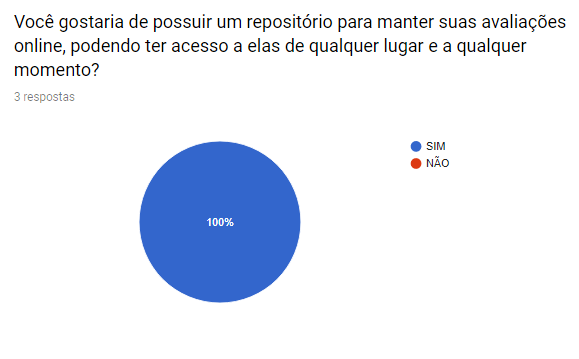
1. Quinta pergunta – Avaliação docente



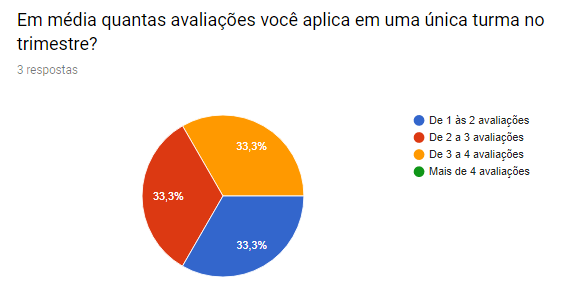
1. Sexta pergunta – Avaliação docente



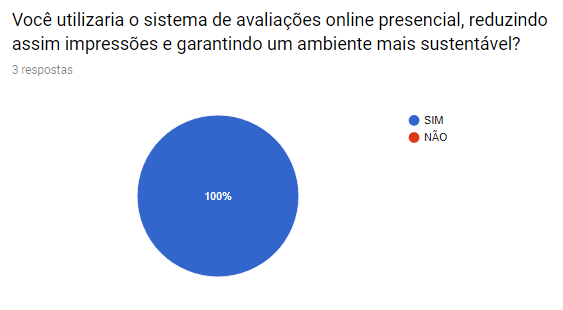
1. Sétima pergunta – Avaliação docente



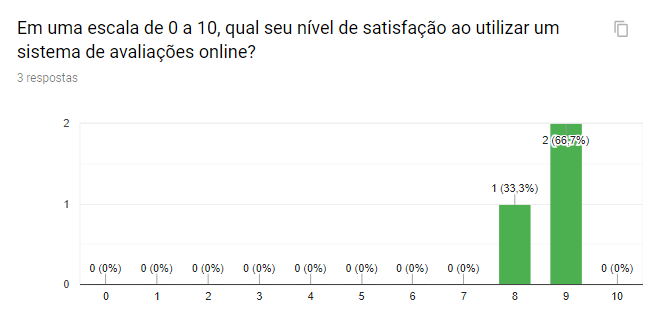
1. Oitava pergunta – Avaliação docente



1. Nona pergunta – Avaliação docente



1. Décima pergunta – Avaliação docente



1. Considerações finais

O desenvolvimento do sistema encontra-se atualmente na fase de testes e melhorias, pois algumas funcionalidades precisam de ajustes, sendo que após algumas validações com professores do ensino médio da instituição SENAI em Lages SC, ao utilizar e verificar a situação atual do sistema, citaram alguns pontos positivos, sendo: a facilidade na correção das avaliações, o princípio de utilizar o sistema como repositório, o cunho ambiental na consequência de redução do consumo de papel para aplicação das avaliações, e a agilidade em propor um feedback ao aluno em relação as notas.

É importante ressaltar que este é um protótipo de sistema, e em conjunto com a instituição de ensino já citada, este ganhará novos recursos para que seja liberado para uso dos docentes em conjunto com os discentes, alguns recursos para melhoria já foram pontuados e poderão serem utilizados em um trabalho futuro, como por exemplo:

* Uso de *sessions,* recurso este que faz parte da linguagem de programação PHP, sendo que após o usuário fechar a aba do navegador o acesso ao sistema novamente só acontecerá caso seja feita uma nova autenticação.
* Utilização de recursos para desenvolvimento em Ajax e AngularJS, que fazem requisições em *client-side,* para garantir o armazenamento dos dados, em casos de queda de energia ou reinicialização do computador, o aluno não irá perder o que já havia respondido em sua avaliação, podendo continuar de onde parou.
* A possibilidade do professor aplicar avaliações qualitativas, garantindo assim uma melhor mediação dos conhecimentos digeridos pelo discente.
* Correção da avaliação, possibilitar um campo de texto para que o professor faça observações para o aluno.
* Propor um método de correção automática das avaliações, com uso de um gabarito cadastrado no sistema, agilizando ainda mais o processo.

Estas e outras melhorias serão implementadas para garantir uma melhor usabilidade e segurança sobre o sistema, lembrando que esta versão ainda é considerada um protótipo, porem com funcionalidades que possibilitam o uso para fins de testes em salas de aula. A implantação acontecerá no SENAI de Lages SC, com o objetivo de expansão para outras unidades em âmbito estadual e até mesmo nacional.

Barry Referências Bibliográficas

PRESSMAN, Roger S. Engenharia de Software. Uma abordagem Profissional. 7. ed. Cidade: AMGH Ed., 2011.

PRESSMAN, Roger S. Engenharia de Software. São Paulo: Makron Books, 1995.1056p.

BARTIÉ, Alexandre. Garantia de Qualidade de Software: adquirindo maturidade organizacional. Rio de Janeiro: Editora Campus, 2002. PRESSMAN, Roger S. Engenharia de Software; tradução Rosângela Delloso Penteado. 6ª ed. São Paulo: McGraw-Hill, 2006. In: PINTO JÚNIOR, José Gonçalves. O uso da Metodologia XP no Desenvolvimento de Software e os Impactos da Gestão de Riscos. Trabalho de Conclusão de Curso, Centro Universitário da Fundação de Ensino Octávio Bastos, São João de Boa Vista, SP: 2009. Disponível em: http://camilolopes.files.wordpress.com/2009/12/monografiajose-goncalves-p-junior.pdf>. Acesso em: 22 fev. 2019.

MYERS, Glenford J. The Art of Software Testing. 1979. In: BARTIÉ, Alexandre. Garantia de Qualidade de Software: adquirindo maturidade organizacional. Rio de Janeiro: Editora Campus, 2002.

INTHURN, Cândida. Qualidade & Teste de Software. Florianópolis: Visual Books, 2001. In: ZIMMERMANN, Ana Paula. Ferramenta para Automação de Testes de Caixa Preta. Trabalho de Conclusão de Curso, Universidade do Vale do Itajaí, Itajaí, SC: julho de 2006. Disponível em: <http://siaibib01.univali.br/pdf/Ana%20Paula%20Zimmermann.pdf>. Acesso em: 13 fev. 2012.

TOMELIN, Marcio. Testes de Software a partir da Ferramenta Visual Test. Trabalho de Conclusão de Curso, Universidade Regional de Blumenau, Blumenau, SC: junho de 2001. Disponível em: <http://campeche.inf.furb.br/tccs/2001-I/20011marciotomelinvf.pdf>. Acesso em: 20 jan. 2012.

BOEHM, BARRY W. and PHILIP N. PAPACIO. Understanding and Controlling Software Costs, IEEE Transactions on Software Engineering, v. 14, no. 10, October 1988, pp. 1462-1477.

STERLING, B. The Hacker Crackdown: Law and Disorder on the Electronic Frontier. Bantam Books, 1992.

MEIRELLES, Paulo Roberto Miranda. Teste Integrado de Software e Hardware: Reusando Casos de Teste de Software em Teste de Microprocessadores. Dissertação (mestrado), Universidade Federal do Rio Grande do Sul, Porto Alegre, RS: junho de 2008. Disponível em: <http://www.lume.ufrgs.br/bitstream/handle/10183/25520/000751158.pdf?sequence=1>. Acesso em: 22 fev. 2019.

BARTIÉ, Alexandre. Garantia de Qualidade de Software: adquirindo maturidade organizacional. Rio de Janeiro: Editora Campus, 2002.

FURLAN, Fabiane Pifer. Visualização de Informação como Apoio ao Planejamento do Teste de Software. Dissertação de Mestrado, Universidade Metodista de Piracicaba, Piracicaba, SP: 2009. Disponível em:<https://www.unimep.br/phpg/bibdig/pdfs/2006/PIICEDIGLYDU.pdf>. Acesso em: 22 fev. 2019.

BERNARDO, Paulo Cheque; KON, Fabio. A Importância dos Testes Automatizados. Artigo publicado na Engenharia de Software Magazine, 2008. Disponível em: <http://www.ime.usp.br/~kon/papers/EngSoftMagazine- IntroducaoTestes.pdf>. Acesso em: 16 fev. 2012.

PFLEEGER, Shari L. Engenharia de Software: Teoria e Prática. 2. ed. São Paulo: Pearson Prentice Hall Brasil, 2004.

SOMMERVILLE, Ian. Engenharia de Software. Tradução André Maurício de Andrade Ribeiro. 6ª edição. São Paulo: Pearson Addison Wesley, 2003. 592 p.

FEWSTER, M.; GRAHAM, D. **Software Test Automation**. Reading, MA: Addison-Wesley Professional, 1999.

SOARES, Michel dos Santos. Comparação entre Metodologias Ágeis e Tradicionais para o Desenvolvimento de Software. Artigo, Universidade Presidente Antônio Carlos (Unipac), Conselheiro Lafaiete, MG: [2004?]. Disponível em: <http://www.dcc.ufla.br/infocomp/artigos/v3.2/art02.pdf>. Acesso em: 30 abr. 2019.

BISSI, Wilson. SCRUM - Metodologia de Desenvolvimento Ágil. Artigo, Centro Universitário de Maringá (CESUMAR), Maringá, PR: 2007. Disponível em: <http://revista.grupointegrado.br/revista/index.php/campodigital/article/view/312/146> Acesso em: 30 abri 2019.

SANCHEZ, Ivan. Scrum em 2 minutos. 7 de fevereiro de 2007. Disponível em: <http://dojofloripa.wordpress.com/2007/02/07/scrum-em-2-minutos/>. Acesso em: 30 abr. 2019.

MATSUDO, Douglas Hideki. Sprints - Planejamento, execução e controle. 10 de agosto de 2009. Disponível em: <http://www.matsudo.com.br/scrum/11sprintsplanejamento-execucao-e-controle>. Acesso em: 30 abr. 2019.

AMANDA. **O papel tem futuro**: polato. 2013. Disponível em: <https://epoca.globo.com/ideias/noticia/2013/12/o-papelb-tem-futurob.html>. Acesso em: 11 jun. 2018.

HAMZE. Amelia. **Avaliação escolar**. 20-?. Disponível em: <https://educador.brasilescola.uol.com.br/trabalho-docente/avaliacao-escolar.htm>. Acesso em: 12 fevereiro 2018.

KRUG, Dircema Franceschetto. Arno. **Avaliação: Por que? O que? Como?** 1. Ed. Curitiba: J.M. Editora e Livraria Jurídica, 2015.

LUCKESI, C.C. **Avaliação da aprendizagem escolar.** 14. ed. São Paulo: Cortez, 2002.

MALDONADO, Leonardo. **Entendendo o DOM (Document Object Model).** 2018. Disponível em: <https://tableless.com.br/entendendo-o-dom-document-object-model/>. Acesso em: 12 maio 2018.

NEXAAS. 5 **vantagens de centralizar o acesso à informação em sua empresa.** Disponível em: < http://nexaas.com/blog/vantagens-centralizar-acesso-a-informacao-em-sua-empresa/>. Acesso em: 5 maio 2018.

PAPELADA. **PAPEL: QUAL O CUSTO AMBIENTAL DE UMA FOLHA A4?.** Disponível em: http://blog.papelada.com.br/consumo-consciente/consumo-concsiente-papel/>. Acesso em: 5 maio 2018.

PISA, Pedro. **O que é e como usar o MySQL?** 2012. Disponível em: <http://www.techtudo.com.br/artigos/noticia/2012/04/o-que-e-e-como-usar-o-mysql.html>. Acesso em: 25 abr. 2012.

PROCENGE (Sc). **Veja como a tecnologia pode ajudar a gestão educacional.** 2017. Disponível em: <http://blog.procenge.com.br/veja-como-a-tecnologia-pode-ajudar-a-gestao-educacional/>. Acesso em: 22 abr. 2018.

ROWLEY, J**. Informática para bibliotecas.** Brasília: Briquet de Lemos, 1994 FIGUEIREDO, Nice Menezes de. Serviços de referência e informação. São Paulo: Polis: APB, 1992.

SOARES, Walace. **PHP 5 Conceitos, Programação e Integração com Banco de Dados** 5. Ed. São Paulo: Editora Érica Ltda, 2009.