

**INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA SUL-RIO-
GRANDENSE - CÂMPUS PASSO FUNDO
CURSO DE TECNOLOGIA EM SISTEMAS PARA INTERNET**

CAMILA DA COSTA MARQUES

**CASO DE USO: INTEGRAÇÃO DE FERRAMENTAS PARA
AUTOMAÇÃO DE TESTES FUNCIONAIS DE SOFTWARE**

**PASSO FUNDO
2018**

CAMILA DA COSTA MARQUES

CASO DE USO: INTEGRAÇÃO DE FERRAMENTAS PARA AUTOMAÇÃO DE TESTES FUNCIONAIS DE SOFTWARE

Monografia apresentada ao Curso de Tecnologia em Sistemas para Internet do Instituto Federal Sul-rio-grandense, Câmpus Passo Fundo, como requisito parcial para a obtenção do título de Tecnólogo em Sistemas para Internet.

Orientador (a): Me. Carmen Vera
Scorsatto Brezolin

CAMILA DA COSTA MARQUES

**CASO DE USO: INTEGRAÇÃO DE FERRAMENTAS PARA
AUTOMAÇÃO DE TESTES FUNCIONAIS DE SOFTWARE**

Trabalho de Conclusão de Curso aprovado em 10/12/2018 como requisito parcial para a obtenção do título de Tecnólogo em Sistemas para Internet.

Banca Examinadora:

Prof. Me.Carmen Vera Scorsatto Brezolin
Orientadora

Prof. Me Vanessa Lago Machado
Avaliadora

Prof. Me André Fernando Rollwagen
Avaliador

Prof. Me. Rafael Marisco Bertei
Coordenação do Curso

PASSO FUNDO
2018

RESUMO

O uso de ferramentas para testes funcionais automatizados de software, é baseado na execução automatizada de scripts que simulam a atividade do usuário em um sistema. Neste contexto se torna importante entender a implementação de testes funcionais automatizados em um projeto de software para assim elencar vantagens e desvantagens de sua aplicação. Dessa forma o presente trabalho tem por objetivo apresentar a arquitetura de automação dos testes de software, desenvolvendo para isso a criação e execução de cenários dos casos de testes mapeados, perante o Sistema de Inscrições em Eventos {Sys Eve} do IFSUL Câmpus Passo Fundo. Para codificar a automação dos passos dos cenários criados foi utilizada a linguagem Ruby e a metodologia BDD, com uso de Gherkins aplicados ao framework Cucumber e o framework Capybara. Por fim, são apresentados os processos desenvolvidos para a automação dos testes do sistema, os quais resultaram na capacidade de executar testes automatizados em menor tempo do que o contabilizado durante a execução destes mesmos testes de forma manual.

Palavras chave: Automação de testes funcionais, BDD, Cucumber, Capybara.

ABSTRACT

The use of functional tests software automation is based on automated execution of scripts to simulate the user activity in a system. Therefore is important to understand the implementation of automated tests software automation to validate de vantagens and disadvantage of its application. In this way, this paper aims presents the fundamentals and the modeling of the architecture of software automation and the development the definition and execution of defined tests cases scenarios of the System to Event Inscription {Sys Eve} of IFSUL Passo Fundo. To encode the automation of the scenarios was used Ruby programming language and BDD methodology with Gherkins applied to the Cucumber framework. In the last part, this paper presents the steps developed for the automation of the system tests which resulted in the feature to execute automated tests in a shorter time than the manual execution of these same tests.

Keywords: Automation of Functional Tests, BDD. Cucumber, Capybara.

LISTA DE FIGURAS

Figura 1 - Estrutura Modelo V.....	12
Figura 2 - Funcionamento do Cucumber.....	19
Figura 3 - Caso de teste escrito com Gherkin.....	20
Figura 4 - Comandos Capybara.....	21
Figura 5 - Comandos Selenium WebDriver.....	22
Figura 6 - Integração das ferramentas.....	23
Figura 7 - Habilitação do plugin para escrita de Gherkins no Visual Studio Code.....	27
Figura 8 - Criação da pasta raiz tests e da arquitetura padrão de um projeto Cucumber.....	28
Figura 9 - Arquivo gemfile.....	29
Figura 10 - Arquivo env.rb localizado dentro da pasta support.....	30
Figura 11 - Primeiro cenário: login_aluno.feature.....	31
Figura 12 - Primeira execução do cenário: login_aluno.feature.....	33
Figura 13 - Geração da sintaxe básica dos passos do cenário: login_aluno.rb.....	34
Figura 14 - Código de automação dos passos do cenário: login_aluno.feature.....	35
Figura 15 - Inspeção da página: Painel de Sistemas do IFSUL - Campus Passo Fundo.....	36
Figura 16 - execução do cenário: @login_aluno.....	37
Figura 17 - Contexto do arquivo: contexto_usuario_aluno.feature.....	38

SUMÁRIO

1 INTRODUÇÃO	8
2 REFERÊNCIAL TEÓRICO	9
2.1 TIPOS DE TESTES DE SOFTWARE	9
2.1.1 Testes de Caixa Preta	9
2.1.2 Testes de Caixa Branca	10
2.1.2.1 Testes Unitários	11
2.2 TESTES DE REGRESSÃO	11
2.3 OS PROCESSOS DO TESTE DE SOFTWARE	12
2.3.1 PLANEJAMENTO DOS TESTES	13
2.3.2 Os CASOS DE TESTES	14
2.3.3 EXECUÇÃO DOS TESTES MANUAIS	14
2.4 AUTOMAÇÃO DOS PROCESSO DE TESTES	15
2.4.1 BEHAVIOR DRIVEN DEVELOPMENT (BDD)	16
2.4.1.1 Redução do desperdício.....	17
2.4.1.2 Redução de custos	17
2.4.1.3 Releases mais rápidas.....	17
2.4.1.4 Documentação dinâmica	17
2.4.2 RUBY	18
2.4.3 CUCUMBER	18
2.4.3.1 Gherkin.....	19
2.4.3.2 Capybara	20
2.4.3.3 API Selenium WebDriver	21
3 METODOLOGIA	23
3.1 SISTEMA DE INSCRIÇÕES EM EVENTOS {SYS EVE}.....	24
3.2 REQUISITOS DO SISTEMA DE INSCRIÇÕES EM EVENTOS {SYS EVE}	24
3.3 REQUISITOS EQUIVALENTES AOS PERFIS DE USUÁRIOS.....	25
3.3.1 Perfil Todos os usuários - Permissões e funcionalidades padrões:	25
3.3.2 Perfil Usuários Externos - Visitante	25
3.3.3 Perfil - Usuários Internos (Alunos, Docentes e Técnicos Administrativos)	25
3.3.4 Perfil Docente - Usuário Interno - Com permissão de cadastro de eventos e atividades.....	26
4 RESULTADOS OBTIDOS	27
4.1 CRIAÇÃO E ARQUITETURA DO PROJETO	27
4.2 CRIAÇÃO DO CENÁRIO	31
4.3 GERAÇÃO DOS STEPS DEFINITIONS	32

4.4 EXECUÇÃO DE UM CENÁRIO.....	37
4.5 UTILIZAÇÃO DE CONTEXTO NOS CENÁRIOS CRIADOS	38
4.6 OUTROS CENÁRIOS E STEP DEFINITIONS	39
5 CONSIDERAÇÕES FINAIS.....	40
REFERÊNCIAS.....	42
APÊNDICES	44

1 INTRODUÇÃO

Com o avanço da tecnologia e da internet, o uso de softwares está presente nas mais diferentes áreas do conhecimento. Por conta disso a exigência na qualidade de softwares desenvolvidos torna-se cada vez maior, o que implica diretamente na menor aceitabilidade de defeitos de um sistema de software. Diante desse cenário, os testes de software se salientam como prática fundamental, na tentativa de reduzir tais defeitos.

Nesse contexto estão presentes os testes manuais e automatizados. Todavia, a prática de testes manuais pode se tornar demorada e custosa. Entretanto, tem se difundido a prática de automação de testes, os quais se baseiam na execução de *scripts* com as funcionalidades do sistema, simulando a utilização do usuário (RIOS e MOREIRA FILHO, 2013).

Ainda, segundo o autor a automação de testes proporciona que tais *scripts* possam ser executados diversas vezes, obtendo-se assim uma maior cobertura de testes sem inconsistências de informações, diferentemente da prática de testes manuais. Foi usado como linguagem de programação para automatizar os *scripts* dos casos de teste a linguagem de programação Ruby, sendo essa classificada, segundo Souza (2008), como uma linguagem de alto nível e de fácil aprendizagem.

Diante disso, o problema que impulsiona o presente trabalho é: Quais os benefícios da implementação de Testes Funcionais automatizados em um projeto? Para responder essa questão, foram analisados os processos e etapas da Automação de Testes Funcionais, apontando vantagens e desvantagens de seu uso no desenvolvimento de um software, contemplando assim: o estudo da arquitetura e modelagem dos casos de testes funcionais automatizados; A criação de Casos de Testes Funcionais automatizados, utilizando a Metodologia BDD; a execução dos Cenários automatizados utilizando o framework Cucumber; o levantamento do custo desempenho obtido ao se automatizar testes funcionais de software.

Visto isso, este trabalho justifica-se pela importância do processo de automação de testes funcionais de software, os quais, quando aplicados corretamente, podem trazer inúmeros benefícios para o desenvolvimento de um software otimizando seu tempo e qualidade.

2 REFERÊNCIAL TEÓRICO

Nos próximos capítulos segue-se a discussão sobre os temas e assuntos que compõem o conhecimento inicial e necessário para o trabalho.

2.1 TIPOS DE TESTES DE SOFTWARE

Atualmente a qualidade de um software tem se tornado um diferencial dentre as empresas, pois não existem softwares perfeitos, ou seja, sem nenhum *bug*. E entregar um produto final com a maior qualidade possível para o cliente é um grande desafio (RIOS e MOREIRA FILHO, 2013).

Segundo Sommerville (2011), o teste de software tem como objetivo mostrar se um programa faz o que é proposto a fazer, e descobrir os defeitos deste programa antes de seu uso. Para realização de tal teste utilizam-se dados fictícios, que posteriormente terão seus resultados verificados e averiguados a procura de erros, anomalias ou informações não funcionais do programa.

A essa validação costuma-se denominar testes de software, e se faz necessária, pois desenvolvedores cometem erros, alguns em grau maior ou menor. Tais testes têm como objetivo assegurar maior qualidade na entrega do software como um todo, e evitar principalmente o retrabalho futuro à medida que as fases de desenvolvimento evoluem. Ainda segundo Sommerville (2011), pois quanto melhores forem os testes realizados durante a fase de desenvolvimento, estima-se que menores serão os custos e a manutenção deste software.

Nesse sentido os testes de software se fundamentam como uma das principais práticas fundamentais na redução de manutenção e uma melhor qualidade do software.

2.1.1 Testes de Caixa Preta

Em decorrência da importância da execução de testes em softwares customizados, recomenda-se que se elabore um teste para atender cada um dos requisitos documentados do sistema. E para softwares genéricos é recomendado

que se tenha testes para todas as características do sistema, com as combinações que serão incorporadas à *releases* (versões) do produto. Esses testes são denominados Testes de Caixa Preta, pois visam verificar a funcionalidade e a adesão dos requisitos, sob uma visão externa ou do usuário, sem precisar de qualquer conhecimento do código e da lógica interna do componente testado (RIOS e MOREIRA FILHO, 2013).

Na validação destes requisitos se salienta a prática do teste de defeitos que objetiva detectar os comportamentos indesejáveis do sistema, tais como panes, processamentos incorretos, corrupção de dados, e interações indesejáveis. Segundo Sommerville(2011) ambos os testes (de defeitos e de validação) têm sua importância, e se diversificam verificando dois aspectos:

O primeiro objetivo leva a testes de validação, nos quais você espera que o sistema execute corretamente usando determinado conjunto de casos de teste que refletem o uso esperado do sistema. O segundo objetivo leva a testes de defeitos, nos quais os casos de teste são projetados para expor os defeitos (SOMMERVILLE, 2011, p.145).

A diferença dos dois testes se baseia no tipo da entrada de dados e no seu resultado esperado, pois, nos casos de testes de validação a entrada de dados é do tipo correto e adequado aos requisitos, refletindo o uso esperado do sistema. Já nos casos de testes de defeitos que têm como objetivo, detectar determinados defeitos, as entradas de dados são incorretas propositalmente, e o seu resultado esperado está atrelado ao tratamento dessas entradas de dados incorretas.

Como por exemplo o campo de um formulário que só aceita números, o caso de teste padrão será moldado para que sejam inseridos números, letras e caracteres especiais neste campo e o resultado esperado será de que o campo não aceite os dados inseridos que não sejam equivalentes a números, se o campo aceitar tais dados será encontrada uma falha no software.

2.1.2 Testes de Caixa Branca

Um software é composto de diversas partes, não somente da interface e é por isso que outros tipo de testes também são importantes, como os Testes de Caixa Branca (White Box) que visam avaliar partes do código, entendendo a lógica interna

do componente codificado, as configurações e demais elementos técnicos (RIOS e MOREIRA FILHO, 2013).

Dentre os testes de caixa branca se destacam a prática dos Testes Unitários descrito no item a seguir.

2.1.2.1 Testes Unitários

Os Testes Unitários se caracterizam como uma prática dos testes de caixa branca, com a intenção de testar a menor unidade do código. Tendo como base os componentes descritos no projeto em desenvolvimento, que terão todos os caminhos de seus módulos percorridos ao menos uma vez.

Visto que o teste unitário envolve os componentes e módulos de um programa, esse teste é realizado pela equipe de desenvolvedores, visando garantir que as condições limites sejam testadas e que as especificações e funcionalidades estejam sendo atendidas de forma correta (PRESSMAN, 2011).

2.2 TESTES DE REGRESSÃO

Desenvolver um software não é uma tarefa fácil, por vezes programadores se deparam com novos defeitos causados por conta da correção de outras falhas, ou até mesmo, uma mudança mal implementada que por consequência inseriu dados ou parâmetros errados, causando problemas em partes do software que já tinham sido testadas. Por essa razão é que a estratégia dos testes de regressão tem sido aplicada.

De acordo com Pressman (2011), tais testes exercem a reexecução de um subconjunto de testes que já foram executados anteriormente, procurando se certificar de que as novas alterações ou integrações das funcionalidades ou componentes do sistema, não tenham propagado efeitos colaterais indesejados.

Para selecionar o subconjunto de testes a ser regredido é feita uma análise dos requisitos do sistemas, levantando uma amostra dos testes, englobando funcionalidades que contemplem a maioria das funções do *software*, e demais testes

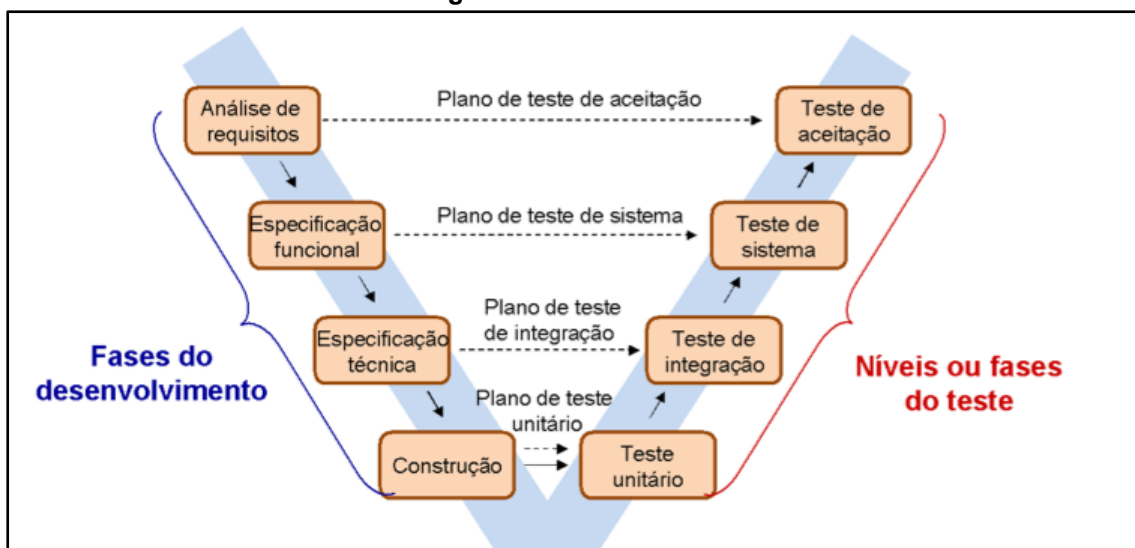
que tenham foco em funcionalidades que tendem a ser afetadas por essas alterações.

2.3 OS PROCESSOS DO TESTE DE SOFTWARE

Um teste para ser efetivamente bem conduzido não pode ser executado sem um planejamento anterior, este planejamento é estruturado seguindo o modelo de desenvolvimento de *software* aplicado ao projeto. Atualmente existem diversos modelos de desenvolvimento de software, sendo os mais conhecidos: Modelo Cascata de desenvolvimento sequencial, Modelo V, Modelo Espiral, Modelo RUP de desenvolvimento iterativo e incremental, e Modelos Ágeis como XP e *Scrum* (TI.exames, 2015).

Este trabalho abordará apenas o Modelo V de Verificação e Validação, visto que este modelo engloba diferentes tipos de testes alinhados com o desenvolvimento, conforme pode ser visto na Figura 1. De acordo com Pressman (2011) o conceito de verificação, se dá ao conjunto de tarefas que irão garantir que o software implemente uma função específica de forma correta. Já o princípio da Validação se relaciona ao conjunto das tarefas que asseguram que o software foi criado de acordo com os requisitos do cliente.

Figura 1 - Estrutura Modelo V



Fonte: TI.exames, 2015.

As fases de desenvolvimento representadas na Figura 1, correspondem a etapa de Verificação do *Software*, nelas se concentram a análise de requisitos, que

servirá de base para a construção da especificação funcional, conduzindo a especificação técnica para a implementação e construção do código do Software. Alinhados a cada uma das fases do desenvolvimento se observam os níveis e fases de teste.

Com a análise de requisitos finalizada, tendo-se uma especificação funcional e técnica sobre o *software* que será testado, iniciam-se os testes unitários, testando a menor parte do código, seus métodos e componentes. Assim que os testes unitários forem concluídos, inicializam-se os testes de integração a fim de encontrar defeitos nas interfaces e interações de componentes e sistemas integrados. Após os teste de integração são executados os testes de sistema, visando validar as regras de negócio de forma detalhada, garantindo o comportamento do sistema como um todo e que seus requisitos foram atendidos (TI.EXAME, 2015).

Normalmente os testes de aceite são executados após a execução dos testes de sistema, no entanto tais testes podem e devem ser realizados iterativamente a cada etapa verificada. Esses testes de aceite possuem como objetivo estabelecer a confiança no sistema produzido por meio da verificação antecipada da evolução do sistema em relação às expectativas do cliente.

2.3.1 Planejamento dos Testes

O processo de planejamento dos testes se inicia, de acordo com (RIOS e MOREIRA, 2013), por meio da análise e leitura dos requerimentos e requisitos do sistema. Requisitos estes necessários para definição da estratégia de testes e para estruturar o plano de teste, visando a criação dos casos de testes que serão executados pelos *scripts* ou procedimento de testes.

Englobando todo esse contexto surgem as diversas estratégias de testes, defendidas por Pressman (2011), como formas de fornecer um roteiro que descreve os passos a serem executados como parte do teste, definindo o momento em que tais passos devem ser executados, e quantos recursos serão necessários para realizá-los. De acordo com o autor, de modo geral um bom gerenciamento de teste deve incorporar as etapas de: planejamento de testes; criação dos casos de teste; execução dos casos de testes e a coleta e avaliação dos dados resultantes.

2.3.2 Os Casos de Testes

Após ser definida qual a estratégia de testes a ser seguida, bem como elaborado o plano de teste, pode-se então focar na análise e escrita dos casos de testes que irão cobrir os requisitos definidos pelo sistema e sua prioridade de execução.

O caso de teste é descrito como o documento que servirá de guia para o testador realizar a execução dos testes. Nele são detalhados os itens e campos de teste: as pré-condições, para a realização do teste; as pós-condições; o objetivo do teste; as entradas a serem testadas, contendo a documentação necessária para consulta na execução destes testes (RIOS e MOREIRA, 2013).

Descreve-se com riqueza de detalhes os resultados esperados, após terem sido realizadas as ações correspondentes a cada passo. No documento também é inserida a data de criação do caso de teste e observações referentes às alterações no mesmo, se realizada.

2.3.3 Execução dos Testes Manuais

Assim que finalizada a etapa de análise e criação dos casos de testes, estes são liberados para sua execução. Esta etapa é responsável pela validação do software através de uma série de testes que irão comprovar a conformidade do sistema em relação aos seus requisitos. A execução manual dos testes é feita seguindo como base os casos de testes levantados e documentados, em que, para, cada ação realizada em seu referido passo, se deve verificar seu resultado esperado e evidenciá-lo.

Tendo concluído tal tarefa, segundo Pressman (2011, p. 417) podem ser constatadas duas condições "(1) a característica de função ou desempenho está de acordo com a especificação e é aceita ou (2) descobre-se um desvio da especificação e é criada uma lista de deficiências". Estas deficiências mencionadas pelo autor, devem ser documentadas com todas as evidências dos passos executados para se chegar ao problema relatado. A equipe de desenvolvimento fica

responsável pela leitura do documento encaminhado, realizando análise e posterior correção do problema.

2.4 AUTOMAÇÃO DOS PROCESSO DE TESTES

A medida em que a complexidade das aplicações web, e sistemas evoluem, os seus testes aumentam gradativamente, assim como o número de componentes a serem testados e consequentemente o seu grau de dificuldade. Tais dificuldades podem ocasionar, como observa Rios e Moreira Filho (2013), no aumento do tempo e recursos requeridos para os testes, em uma qualidade inferior do sistema produzido e na crescente pressão sobre a entrega do projeto.

Diante desse contexto, visto a necessidade de se diminuir o tempo e recursos gastos ao realizarem tais testes, surgem novas ferramentas com o objetivo de otimizar o processo de testes e talvez reduzir ou eliminar estas dificuldades. No entanto o projeto de automação de testes requer alguns cuidados, sendo estruturado por três principais pilares, sendo eles: Ferramenta, Metodologia e Infraestrutura (RIOS e MOREIRA, 2013).

É de fundamental importância perante a estrutura do projeto, que a escolha da ferramenta, seja compatível com a tecnologia usada e que possa ser integrada a metodologia já aplicada ou que será adotada. Além de ter-se definido a ferramenta e a metodologia empregada, também é preciso analisar a infraestrutura necessária para a realização destes processos tendo em vista a disponibilidade de máquinas e recursos que tal projeto irá requerer.

O projeto de automação de testes não é um processo simples e demanda um custo relativamente alto, necessita-se ter a consciência de que nem todos os testes devem ser automatizados, como Oliveira (2018) justifica que não é viável automatizar o teste de uma funcionalidade que sofre alterações frequentemente. Pois o esforço requerido para manutenção desses *scripts* tende a ser muito maior do que a execução manual, observando também que o nível de sua qualidade não se equivale à dos testes manuais.

Porém os testes de regressão tornam-se grandes candidatos a automação de testes, dado que serão estes executados em menor tempo e terão um retorno e *feedback* muito mais rápido. A cobertura das funcionalidades críticas da aplicação, também se torna uma candidata a automação, por conta de se tratar de uma parte fundamental do sistema de alta visibilidade que será utilizada diversas vezes.

2.4.1 Behavior Driven Development (BDD)

A maior parte dos problemas dos sistemas e aplicações desenvolvidas ocorre por conta da relação complicada entre desenvolvedores e *stakeholders* (interessados no negócio), por meio de requisitos mal interpretados ou especificados, que não atendem as expectativas e as prioridades levantadas pelo cliente.

Ao perceber isso, Dan North criou em 2003 a metodologia de Desenvolvimento Orientada ao Comportamento, mais conhecida como BDD (BARAÚNA, 2013). O BDD tem seu foco, na criação do código por meio das interações entre usuários e desenvolvedores, a partir do processo denominado *outside-in-development* (desenvolvimento de fora para dentro), em que são mapeados exemplos de histórias de usuários que validem o comportamento do software.

A especificação de requisitos se apresenta como uma das etapas primordiais para o desenvolvimento de um *software*. No entanto por vezes essas especificações podem se tornar confusas, e ricas de detalhes que outros integrantes do time, não sendo de desenvolvimento, venham a não compreender. Como solução para este problema tende-se a utilizar como uma das formas de especificação do software as histórias dos usuários (*users stories*), que descrevem de maneira detalhada o comportamento e critério de aceite de cada funcionalidade, deixando claro para quem está sendo criada a funcionalidade e qual o seu propósito (SOARES, 2011).

Para que essa comunicação entre interessados no negócio e desenvolvedores funcione, adota-se o uso de uma linguagem ubíqua, ficando assim compreensível por todos, tanto programadores, analistas, quanto utilizadores do negócio (SOARES 2011, apud ANDERLE, 2015). Tal prática permite que a equipe de desenvolvimento tenha seu foco voltado ao porquê da criação deste código, e não aos seus detalhes técnicos, permitindo também uma comunicação bem mais eficiente entre a equipe de testes e desenvolvimento.

Os benefícios da metodologia BDD são muitos, dentre eles (Anderle, 2015) destaca a redução de desperdício, redução de custos, releases mais rápidas e documentação dinâmica .

2.4.1.1 Redução do desperdício

A redução do desperdício torna-se uma das vantagens ao utilizar-se da metodologia BDD, ao passo que o foco do desenvolvimento está todo orientado ao comportamento, fornecendo valor ao negócio, sem desperdiçar esforço e tempo, com alguma implementação ou aplicação que não venha a ser útil para os objetivos do negócio (ANDERLE, 2015).

2.4.1.2 Redução de custos

A redução de custos se apresenta como, uma consequência direta da redução de tempo de desenvolvimento, pois tendo-se exatamente quais funcionalidades devem ser criadas e como será o comportamento delas, torna-se automaticamente mais fácil desenvolvê-las, reduzindo tempo e aumentando a qualidade do código que reduzirá os erros, e portanto os custos de manutenção deste software (ANDERLE, 2015).

2.4.1.3 Releases mais rápidas

Os testes automatizados aceleram os ciclos das releases notoriamente. Pois os testes de regressões e aceite que antes eram executados manualmente levando uma boa parte de tempo e recursos. Com a aplicação do BDD alinhado ao processo de automação, podem ser executados de forma automatizada diminuindo tempo para aproveitar o mesmo recurso de outra forma mais produtiva e eficiente (ANDERLE, 2015).

2.4.1.4 Documentação dinâmica

Com o aumento das demandas e modificações em um projeto torna-se necessário que o fluxo de informações seja contínuo e rápido. Esta vantagem pode ser obtida ao utilizar *frameworks* que implementam o BDD, pois esta documentação

será criada dinamicamente, podendo até gerar relatórios em formatos HTML para execução de futuras consultas (ANDERLE, 2015).

2.4.2 Ruby

A linguagem *Ruby* foi criada em 1995 pelo japonês Yukihiro Matz Matsumoto, com o propósito de ser uma linguagem limpa e direta, sendo ela totalmente orientada a objetos, tornando-se assim bem mais simples de se aprender e trabalhar (SOUZA, 2008).

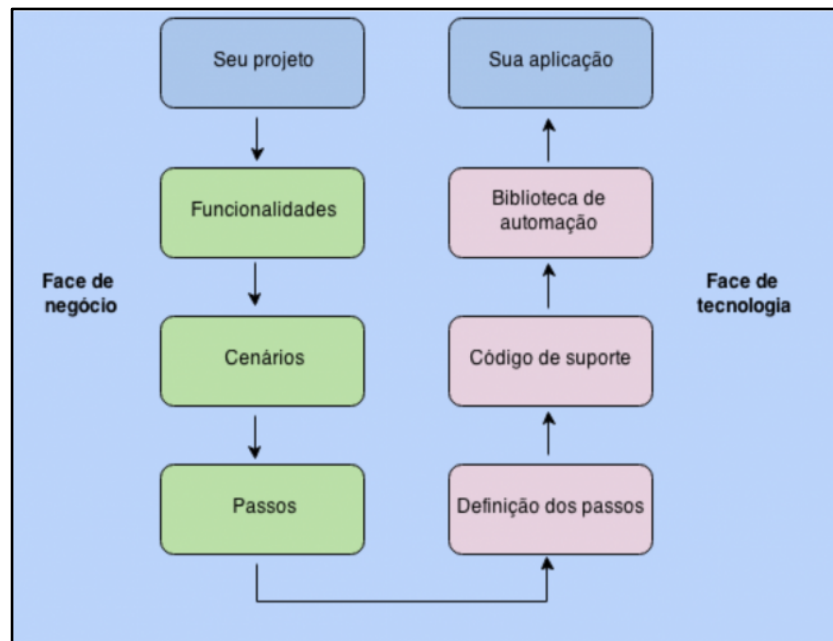
Segundo (CORBUCCI e ANICHE, 2014) uma característica importante da linguagem que a diferencia das demais é o fato do Ruby ser uma linguagem dinamicamente tipada, em que variáveis podem ser atribuídas dinamicamente, não necessitando especificar o seu tipo na sua criação. Tal condição resulta na vantagem de reduzir-se a verbosidade, tornando-se mais fácil definir as variáveis e os comportamentos dos objetos. Outra vantagem adquirida utilizando *Ruby* é o fato da língua funcionar em multi-plataformas, sendo esta compatível com diversos sistemas operacionais.

2.4.3 Cucumber

O *Cucumber* é uma ferramenta orientada pela metodologia BDD, criada por Aslak Hellesoy em 2008 com o propósito de especificar o valor do negócio em uma linguagem natural (MENDES, 2012). Essas especificações são elaboradas através de exemplos para que qualquer integrante da equipe consiga-as compreender. Após coletados os exemplos, são criadas as descrições funcionais construídas através de *Gherkins* para realizar a automação de testes de aceite e regressão.

O funcionamento do *Cucumber* acontece por linha de comando, sendo criado um projeto com a estrutura como ilustrada na Figura 2.

Figura 2 - Funcionamento do Cucumber



Fonte: THOUGHTWORKS, 2015.

O arquivo com a extensão ".feature" irá conter os cenários descritos através de *Gherkins* e é nos cenários que irão conter passos que dirão ao cucumber o que fazer.

2.4.3.1 Gherkin

O *Gherkin* é uma linguagem natural, pois apresenta texto livre com algumas palavras reservadas em que o cucumber interpreta para realizar a execução dos testes (BERÇAM, 2017). O autor destaca as palavras:

- **Feature/Funcionalidade**; algum texto descritivo conciso do que é desejado
As a/ A Fim de realizar um valor de negócio
In order to/ Como ator explícito do sistema
I want to/ Eu quero ganhar algum resultado benéfico que promova a meta.
- **Background/Contexto**: são definidos os requisitos para determinada funcionalidade
- **Scenario/ Cenário**: uma determinada situação de negócio
- **Given/ Dado** : representando uma determinada pré-condição
- **When/Quando**: uma ação é feita pelo ator

- Then/ Então: um resultado testável é alcançado
- And/But - E/Mas : usados para complementar alguma descrição anterior

O *Gherkin* deve descrever de forma simples e compreensível a todos os integrantes do time, qual o objetivo de determinada funcionalidade, suas pré-condições, ações e resultados esperados equivalentes, conforme apresentados na Figura 3.

Figura 3 - Caso de teste escrito com Gherkin

```
#language: pt
Funcionalidade: Criar uma conta no Facebook
  Eu quero criar uma conta no facebook
  A fim de entrar na rede social.
Cenário: criando conta no Facebook
  Dado o site do Facebook
  Quando apresentar o campo Criar Nova Conta
  E Preencher os campos e acionar o comando Criar
  Então o Facebook irá exibir a tela inicial
```

Fonte: BERÇAM, 2017.

Após a criação da *feature* e descrição do cenário através dos *Gherkins*, observa-se que, tem-se definido os passos para a validação dos cenários, porém estes escritos em linguagem natural. Cada passo detalhado no *gherkin* corresponde a uma ação de usuário que será automatizada com o uso do *framework* Capybara que irá integrar aos comandos descritos no Gherkin as ações a serem executadas. Estas ações podem ser localizadas dentro da estrutura do projeto no arquivo “.step_definitions/passos”, que detalha os passos a serem executados descritos no gherkin.

2.4.3.2 Capybara

O Capybara é um framework de automação de testes em aplicações web, *open-source* escrito em *Ruby*. Que de acordo com Lara (2017), para realizar automação destas ações, o Capybara utiliza *Drivers* e *frameworks* que controlam navegadores, tornando possível simular ações equivalentes às que os usuários reais executariam na aplicação. Por meio dos comandos apresentados na Figura 4.

Figura 4 - Comandos Capybara

Comando	Descrição	Exemplo
visit	Acessar uma URL	visit "https://medium.com/jaguaribetech"
expect(page).to have_content	Verifica se a página possui o texto esperado	expect(page).to have_content (‘Cadastro realizado com sucesso’)
fill_in	Adiciona um texto a um elemento ou campo	fill_in "login", with: "samyralara"
has_content?	Verifica se um texto está sendo apresentado em um elemento	page.has_content?("Samyra Lara")
visible?	Verifica se um elemento está visível	find(".status").visible?
click_button	Clica em um determinado botão	click_button 'login'

Fonte: LARA, 2017.

Os comandos apresentados na Figura 4, ilustram os principais comandos para automatizar as ações manuais que os usuários reais, executariam no sistema.

2.4.3.3 API Selenium WebDriver

Para realizar a automação das ações do navegador é possível a utilização da API do Selenium *WebDriver*, que através de comandos específicos realiza a navegação de um sistema ou página. Coletando dados e executando tarefas, como *submits* de formulários, seleções em menus *drop down*, digitação em campos de texto, varredura de dados em elementos, HTML, etc.

A partir destes comandos é possível complementar os passo descritos no arquivo “.step_definitions/passos” descrevendo as ações de navegação dentro de uma página, equivalentes as ações que os usuários reais executariam na aplicação. Utilizando como exemplo o *Gherkin* criado na seção 4.3.1 para criar uma conta no Facebook, a definição e detalhamento dos passos para que a criação desta conta seja efetuada, ocorre através do uso dos comandos do Selenium WebDriver conforme é apresentado na Figura 5.

Figura 5 - Comandos Selenium WebDriver

```

4  #Declarando a variável @Driver atribuindo o Webdriver do Chrome.
5  @driver = Selenium::WebDriver.for :chrome
6  #Pedindo para ir para o endereço do Facebook
7  @driver.get "http://www.facebook.com"
8  sleep 3
9
10 end
11
12 Quando("apresentar o campo Criar Nova Conta") do
13   @driver.find_element(:xpath, "//div[@id='content']/div/div/div/div/div[2]/div/div/span").displayed?
14 end
15
16 Quando("Preencher os campos e acionar o comando Criar") do
17   @driver.find_element(:id, "u_0_1").click
18   @driver.find_element(:id, "u_0_1").clear
19   @driver.find_element(:id, "u_0_1").send_keys "rafael"
20   sleep 1
21   @driver.find_element(:id, "u_0_n").click
22   @driver.find_element(:id, "u_0_n").clear
23   @driver.find_element(:id, "u_0_n").send_keys "bercam"
24   sleep 1
25   @driver.find_element(:id, "u_0_q").click
26   @driver.find_element(:id, "u_0_q").clear
27   @driver.find_element(:id, "u_0_q").send_keys "emailteste@gmail.com"
28   sleep 1
29   @driver.find_element(:id, "u_0_t").click
30   @driver.find_element(:id, "u_0_t").clear
31   @driver.find_element(:id, "u_0_t").send_keys "emailteste@gmail.com"
32   sleep 1
33   @driver.find_element(:id, "u_0_x").click
34   @driver.find_element(:id, "u_0_x").clear
35   @driver.find_element(:id, "u_0_x").send_keys "w3e4r5t6y7"
36   sleep 1
37   @driver.find_element(:id, "day").click
38   Selenium::WebDriver::Support::Select.new(@driver.find_element(:id, "day")).select_by(:text, "8")
39   @driver.find_element(:id, "day").click
40   sleep 1
41   @driver.find_element(:id, "month").click
42   Selenium::WebDriver::Support::Select.new(@driver.find_element(:id, "month")).select_by(:text, "Maio")
43   @driver.find_element(:id, "month").click
44   sleep 1
45   @driver.find_element(:id, "year").click
46   Selenium::WebDriver::Support::Select.new(@driver.find_element(:id, "year")).select_by(:text, "1984")
47   @driver.find_element(:id, "year").click
48   sleep 1
49   @driver.find_element(:xpath, "//span[@id='u_0_11']/span[2]/label").click
50   sleep 1

```

Fonte: BERÇAM, 2017.

Segundo Berçam (2010), o comando `@driver.get "http://www.facebook.com"` da linha 7, demonstrado na Figura 5. executa uma requisição através do método GET passando o site do facebook como endereço a ser navegado. E os demais comandos executam ações como clicar (`.click`), limpar um campo (`.clear`), preencher um campo, enviar um valor (`send_keys`), com determinado valor passado por parâmetro. A interligação dos valores enviados e ações automatizadas com os elementos da página se dá através de atributos únicos e identificadores como o id.

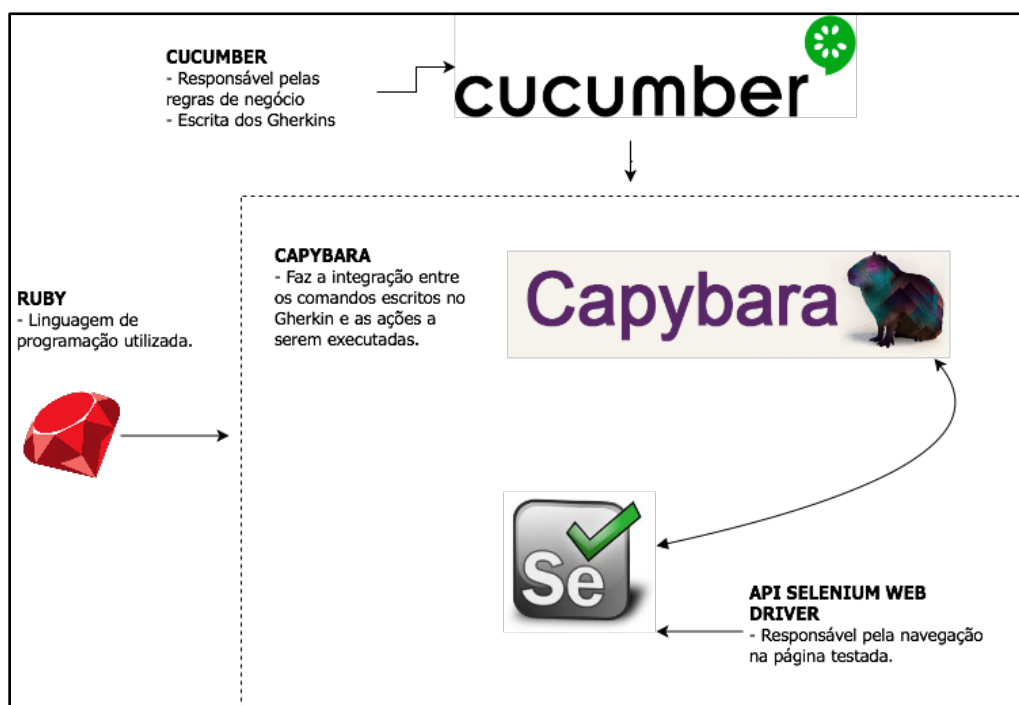
3 METODOLOGIA

Para fins de validação deste estudo foram realizados testes automatizados no Sistema de: Inscrições em Eventos {Sys Eve} do Instituto Federal Sul-Rio-Grandense Campus - Passo Fundo. Tais testes validaram as permissões de determinados usuários perante o sistema, e simularam algumas ações que estes executariam na aplicação, como por exemplo: Efetuar login; Navegação entre páginas; Visualização de Eventos ativos; Inscrição em um Evento; Controle sobre as Inscrições dos usuários, e Obtenção dos certificados gerados, dentre outras funcionalidades.

Mesmo na automação de testes, segue-se uma hierarquia quanto a estrutura dos processos de automação, tendo-se na primeira etapa a utilização da metodologia BDD, para levantar as histórias de usuários e elencar os critérios de aceite das funcionalidades validadas, que irão originar os *scripts* criados em *Gherkins*, descrevendo as pré-condições, ações e os resultados esperados de cada cenário a ser executado.

A interligação das regras de negócio, histórias de usuários e critérios de aceite com o código, foi realizada através do *framework* Cucumber conforme representado na Figura 6.

Figura 6 - Integração das Ferramentas



Fonte: Do Autor, 2018.

Utilizando como linguagem e sintaxe o *Gherkin* para realizar a especificação dos testes. Em que a automação das ações do usuário no sistema web, se dá por meio do *framework* Capybara, responsável por integrar os *Gherkins* com as ações a serem executadas, através da utilização da API Selenium Web Driver que controlará a navegação nas páginas testadas dos sistema.

3.1 SISTEMA DE INSCRIÇÕES EM EVENTOS {Sys Eve}

O Sistema de Inscrições em Eventos {Sys Eve}, identificado como um sistema Web, foi desenvolvido com o propósito de auxiliar nas Inscrições dos eventos e suas atividades, sendo também utilizado para otimizar a geração de certificados dos participantes, em relação aos eventos sediados pelo Instituto Federal Sul-Rio-Grandense - Campus Passo Fundo.

3.2 REQUISITOS DO SISTEMA DE INSCRIÇÕES EM EVENTOS {Sys Eve}

Os eventos podem ser classificados em dois tipos: Interno - disponível para alunos e servidores, e externo - disponível para o público em geral. Os requisitos do sistema compõem:

- O cadastro do período de inscrição para cada evento.
- O registro de presenças nas atividades por código de inscrição ou nome.
- A geração dos certificados para os participantes com presença registrada.
- O cadastro de atividades relacionadas ao evento ocorrido, ex: palestras e oficinas ofertadas.
- A geração de relatórios de inscritos por atividade, em relação às presenças computadas de acordo com seus crachás, código de barra, e presença por dia e turno.
- A listagem de eventos ativos aos usuários.
- A permissão do cadastro dos usuários nos eventos e atividades.

3.3 REQUISITOS EQUIVALENTES AOS PERFIS DE USUÁRIOS

O Sistema de Inscrições em Eventos possui quatro perfis de usuários, sendo eles: Alunos, Docentes, Técnicos Administrativos e Visitantes, sendo esses usuários Internos (Alunos, Docentes e Técnicos Administrativos) e/ou Externos (Visitante). Estes perfis possuem determinadas permissões e acessos a funcionalidades restritas ou específicas de cada perfil.

3.3.1 Perfil Todos os usuários - Permissões e funcionalidades padrões:

Diante dos requisitos do sistema, todos os usuários possuem permissão para:

- Visualizar os eventos ativos;
- Realizar *download* do seu número de inscrição com código de barras;
- Visualizar as suas inscrições;
- Cancelar e realizar novas inscrições em atividades - desde que respeitando o período de inscrições.
- Imprimir os certificados dos eventos passados em que participou.

3.3.2 Perfil Usuários Externos - Visitante

Sendo o usuário do perfil Visitante este poderá visualizar os eventos ativos e abertos ao público externo no sistema, se inscrever nas atividades destes eventos, e realizar download dos certificados dos eventos, os quais já tenha participado.

3.3.3 Perfil - Usuários Internos (Alunos, Docentes e Técnicos Administrativos)

Diante de usuários com perfis Internos, podendo estes se tratarem de Alunos, Docentes, Técnicos Administrativos, e demais usuários que possuam vínculos com a Instituição, tais usuários poderão, visualizar os eventos ativos, realizar inscrições nas atividades dos eventos e baixar os certificados dos eventos os quais já tenham participado.

3.3.4 Perfil Docente - Usuário Interno - Com permissão de cadastro de eventos e atividades.

Possuindo usuário do perfil Docente, este poderá realizar o cadastro de um novo evento e atividade, podendo também listar e obter relatórios das inscrições dos usuários, filtrando-as por: evento, categoria, atividade, e nome do inscrito. Assim como Registrar as presenças dos usuários, de acordo com o evento, atividade e a forma de identificação pessoal do participante, sendo pelo seu cpf, ou nome.

Além das demais permissões comuns a todos os usuários, como: Visualizar os eventos ativos, realizar inscrições nas atividades dos eventos e efetuar download dos certificados dos eventos ocorridos os quais já tenha participado.

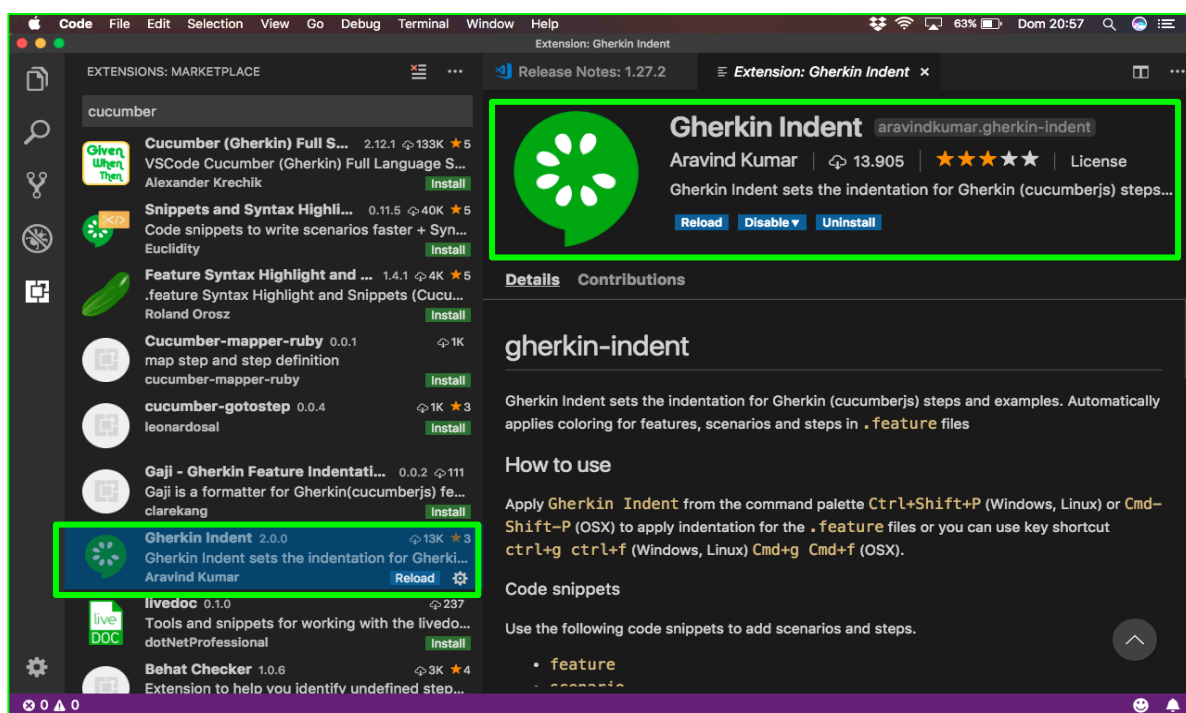
4 RESULTADOS OBTIDOS

Esta seção apresenta o desenvolvimento do trabalho, com as etapas necessárias para sua construção. Além disso estão sendo apresentados os resultados obtidos a partir das etapas de criação e a execução dos casos de testes desenvolvidos.

4.1 CRIAÇÃO E ARQUITETURA DO PROJETO

Para a realização deste trabalho foi utilizada a IDE Visual Studio Code, sendo necessário primeiramente realizar a instalação do *Ruby* na versão 2.5 e habilitar alguns plugins que facilitam o desenvolvimento dos testes, como: ***vscode-icons*** (plugin que reconhece a extensão dos arquivos e adapta-a ao ícone correspondente, tendo-se assim uma melhor visualização da estrutura do projeto), ***Ruby*** (plugin para ler e debugar arquivos em Ruby), ***Gherkin Indent*** (plugin alinhado ao framework Cucumber - responsável pela interpretação dos gherkins descritos nos cenários, conforme Figura 7.

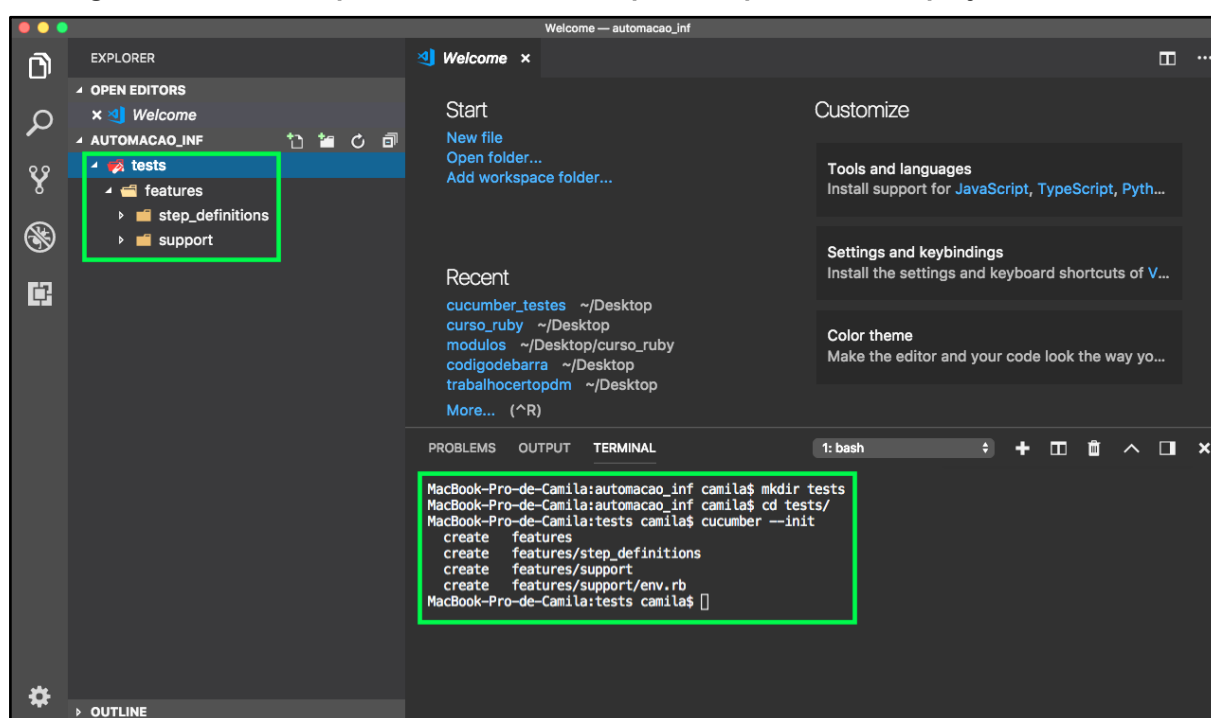
Figura 7 - Habilitação do plugin para escrita de Gherkins no Visual Studio Code.



Fonte: Do Autor, 2018.

Após ter-se habilitado estes plugins, foi necessário instalar via terminal as gems¹ cucumber - para escrita e leitura dos cenários, e rspec² - para validação dos passos esperados contidos em cada cenário. Logo após a instalação das duas principais gems, foi criada uma pasta contendo o nome do projeto na área de trabalho do computador, e dentro desta pasta executado o comando : mkdir tests, que criou a pasta raiz de testes do projeto. A partir da criação da pasta tests, foi executado o comando: cucumber --init, criando assim uma arquitetura padrão pronta do cucumber, conforme pode ser visto na Figura 8.

Figura 8 - Criação da pasta raiz tests e da arquitetura padrão de um projeto Cucumber.



Fonte: Do Autor, 2018.

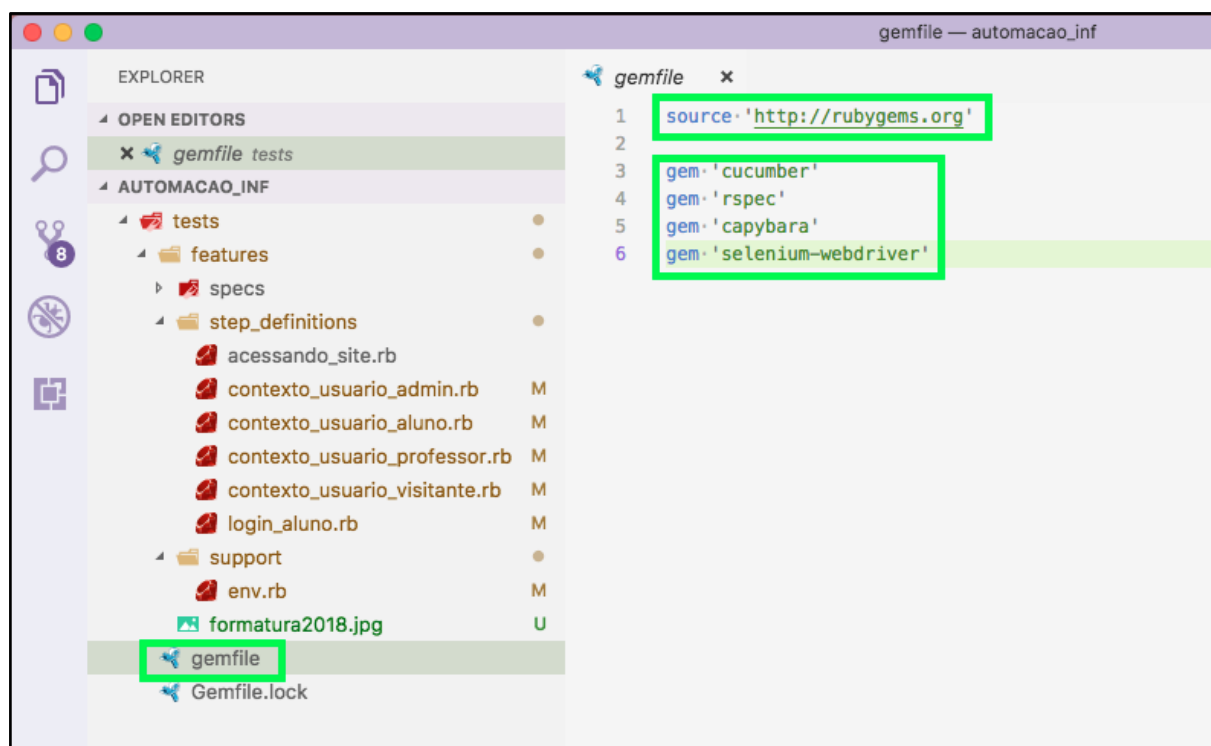
A arquitetura padrão gerada é composta pela estrutura principal de três pastas, sendo elas: “features”, “step_definitions” e “support”, onde a pasta “features” contém os contextos e cenários dos casos de testes, descritos através de Gherkins. Já a pasta “step_definitions” contém o código necessário para realizar a automação, das ações dos usuários, de acordo com os cenários elaborados. E a pasta “support” contendo as configurações utilizadas para realizar a automação do projeto.

¹ gems é um sistema de pacotes Ruby que facilita a criação, compartilhamento e instalação de bibliotecas (pode-se dizer que é um sistema de distribuição de pacotes similar, por exemplo, ao apt-get).

² o rspec é composto de várias bibliotecas, que são projetadas para trabalhar juntas ou podem ser usadas independentemente com outras ferramentas de teste, como o Cucumber ou o Minitest.

Além desta estrutura, também foi necessário instalar a gem bundler, por meio do comando: `gem install bundler`, para gerenciar o download dos pacotes das gems requeridas no projeto, assim como também foi necessário criar manualmente dentro da pasta raiz tests o arquivo gemfile, como demonstrado na Figura 9.

Figura 9 - Arquivo gemfile.

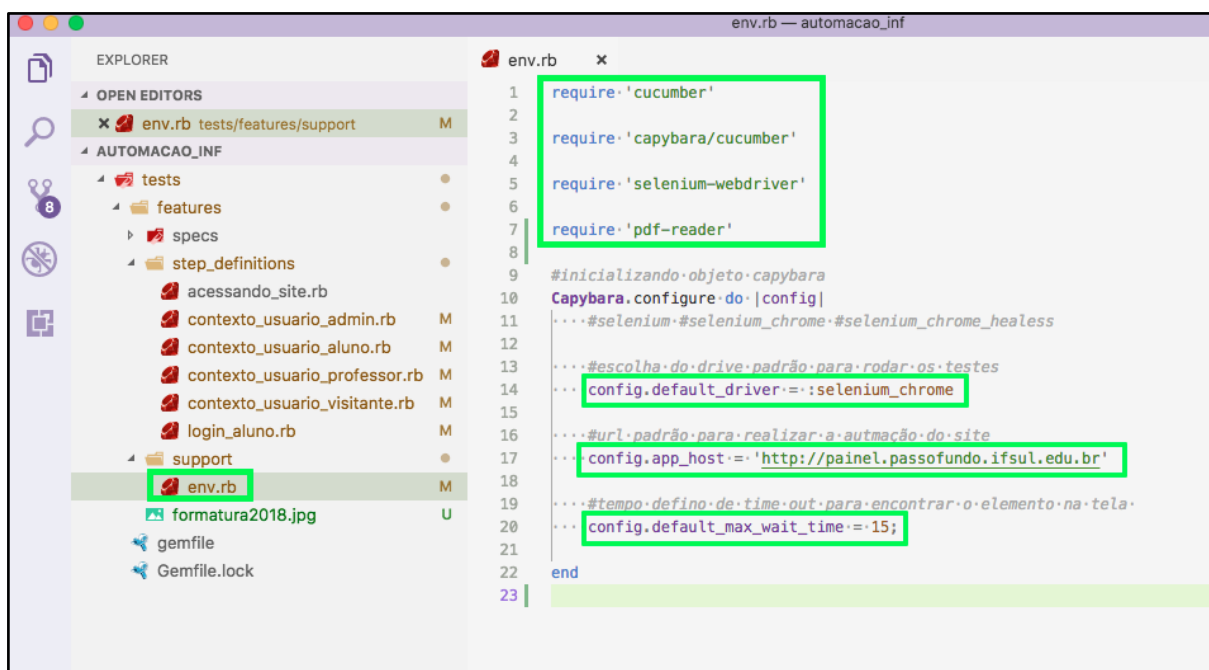


Fonte: Do Autor, 2018.

Contendo neste arquivo, as gems que serão utilizadas no projeto, e o repositório para que elas sejam encontradas, e baixados seus pacotes, onde o comando *source* representa o repositório atual, onde as gems (cucumber, rspec, capybara, selenium-webdriver) foram baixadas.

A partir da arquitetura gerada ao se expandir o arquivo support pode-se perceber a existência do arquivo `env.rb`, que foi utilizado como arquivo de configuração do projeto, de acordo com a Figura 10.

Figura 10 - Arquivo env.rb localizado dentro da pasta support.



Fonte: Do Autor, 2018.

Neste arquivo foram especificadas as gems exigidas na configuração do projeto, mediante o comando *require*, em seguida foram definidas algumas configurações do *framework* capybara, como por exemplo a escolha do *selenium_chrome* como driver padrão para rodar os testes, através do comando: “*config.default_driver = :selenium_chrome*”, a definição da url padrão para realizar a automação dos testes, por meio do comando: “*config.app_host = 'https://painel.passofundo.ifsul.edu.br'* ”, identificando o painel de sistemas do IFSUL - Campus - Passo Fundo como host padrão para a automação dos testes. Foi feita a definição do tempo máximo de espera por um objeto em determinada página, pelo comando: “*config.default_max_wait_time = 15*”, que definiu o tempo máximo de 15 segundos.

Após ter-se realizado estas configurações pode-se baixar a última versão do *chromedriver* (versão 2.43) , identificado como um servidor autônomo encarregado da implementação do protocolo de comunicação do *WebDriver* para o *Chromium*, sendo este caracterizado como uma ferramenta livre para testes automatizados em aplicações da Web englobando diversos navegadores.

Com a arquitetura base do projeto criada e concluída, deu-se início a criação dos cenários dos casos de testes a partir dos requisitos levantados na seção 8.

4.2 CRIAÇÃO DO CENÁRIO

Buscando uma melhor organização do projeto, a fim de manter todos os cenários dos casos de testes reunidos em uma só pasta criou-se dentro da pasta “features” a pasta “specs”. Para realizar a geração do primeiro cenário necessitou-se criar o arquivo “login_aluno.feature” dentro da pasta “specs”, sendo este identificado pela sua extensão “.feature” que caracteriza o arquivo como um cenário do framework Cucumber.

Primeiramente neste arquivo foi especificada a língua: Português para que os Gherkins descritos fossem interpretados, através do comando “language: pt” na primeira linha do arquivo, posteriormente foi definida uma tag específica para a execução deste cenário com o nome: “@login_aluno”, em seguida iniciou-se a descrição dos Gherkins, definindo como funcionalidade a ser validada neste cenário, a Realização do Login, conforme pode ser visualizado na Figura 11.

Figura 11 - Primeiro cenário: login_aluno.feature



Fonte: Do Autor, 2018.

A palavra reservada "Dado" da sintaxe do *Gherkin* representa uma pré-condição opcional, antes das ações a serem executadas dentro do cenário, tendo o objetivo de validar ou efetuar alguma ação. Como exemplo, no cenário atual, é realizada a validação do usuário, estar na página de login, antes de ser executada a próxima ação deste cenário, descrita por meio da palavra reservada "Quando",

exemplificando que o campo usuário deve ser preenchido com o usuário aluno. As palavras reservadas "E" e "Mas" na sintaxe do *Gherkin* em cenários do cucumber funcionam como complementações a ações anteriores em um cenário, sendo a palavra "E" comumente usada para adicionar mais uma ação a um cenário, enquanto que a palavra "Mas" adiciona uma contraposição a uma ação anterior já descrita.

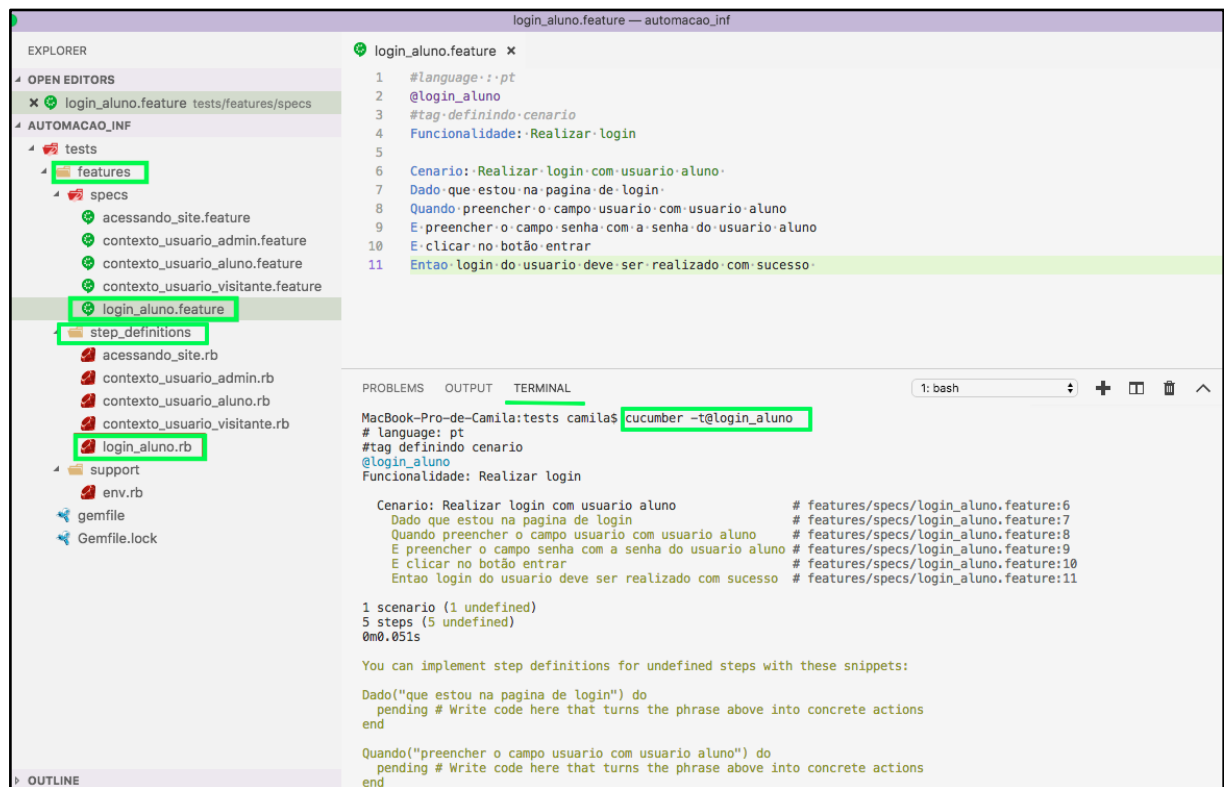
Dando continuidade a leitura de um cenário através de Gherkins, a palavra reservada "Então" representa um resultado esperado que deve ser validado. Como no cenário da Figura 11, após obter-se a pré condição de se estar na página de login, tendo-se efetuado as ações, equivalentes aos passos de preencher os campos do formulário com os devidos valores adequados e clicar no botão entrar, deve-se validar que o login do usuário escolhido deve ser realizado com sucesso.

4.3 GERAÇÃO DOS STEPS DEFINITIONS

A automação dos passos descritos através de Gherkins em um cenário, acontece por meio dos comandos descritos em um arquivo, localizado dentro da pasta "step_definitions", necessitando que este arquivo tenha o mesmo nome do cenário já criado anteriormente, diferenciando-se apenas na sua extensão final ".rb". Assim, identificando este arquivo, como um arquivo a ser interpretado em comandos ruby, e não mais em feature, equivalentes a cenários do cucumber.

Para gerar a sintaxe dos passos de um cenário, de acordo com os gherkins descritos, após ter-se criado um arquivo ruby dentro da pasta "step_definitions", com o mesmo nome do cenário anteriormente criado. É necessário rodar dentro do projeto via terminal o comando: cucumber, podendo este comando possuir uma tag atrelada a ele ou não, conforme pode ser visto na Figura 12.

Figura 12 - Primeira execução do cenário: login_aluno.feature



Fonte: Do Autor, 2018.

O comando `cucumber -t@login_aluno` executou o cenário localizado no arquivo: "login_aluno.feature", compilando os passos deste cenário, através do arquivo: "login_aluno.rb". Com esta execução foi possível identificar a existência do cenário mencionado, pois o arquivo "login_aluno.feature" continha a descrição do comportamento esperado para este cenário. Porém, não foi encontrado o código com a descrição dos passos atrelados a este cenário.

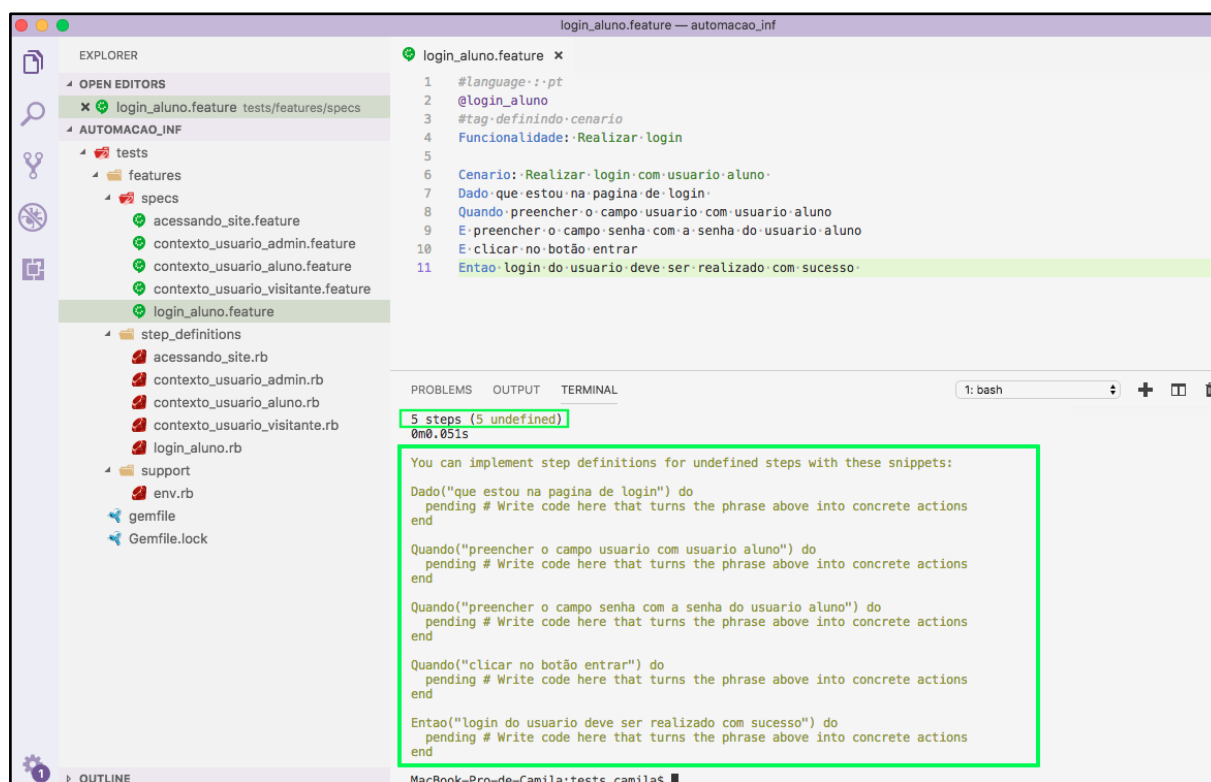
Esta ocorrência foi prevista propositalmente, pois o *framework* Cucumber segue a metodologia do BDD. Em que primeiro deve ser descrito o comportamento esperado do teste a ser validado, observando-se que este teste, ao ser executado irá falhar, pois o código que especifica este comportamento, ainda não foi descrito.

Sendo assim ao rodar o comando `cucumber`, testando o cenário login_aluno, foi validada a descrição do comportamento do cenário "login_aluno.feature", e a pendência de 5 passos indefinidos neste cenário, localizados no arquivo "login_aluno.rb". Esta pendência ocorreu por conta do arquivo "login_aluno.rb", não

conter nenhum código para automatizar as ações contempladas no cenário. Tendo o mesmo somente ter sido criado, permanecendo em branco.

Como sugestão para resolver esta pendência o próprio framework cucumber, gera a sintaxe básica correspondente aos Gherkins gerados neste cenário para iniciar a descrição dos códigos que irão validar estas ações, como pode ser observado pela Figura 13.

Figura 13 - Geração da sintaxe básica dos passos do cenário : login_aluno.rb

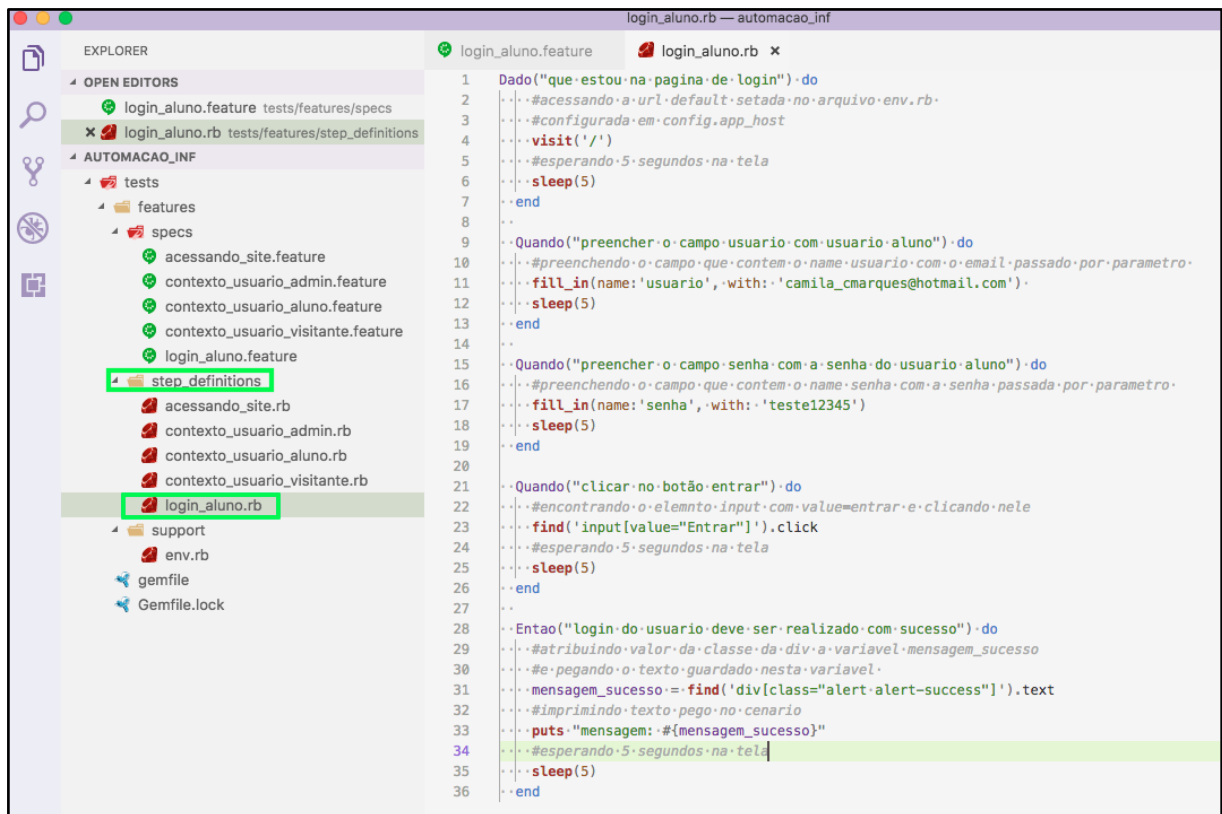


Fonte: Do Autor, 2018.

Para gerar os passos do cenário "login_aluno.rb", foi utilizada a sintaxe gerada, através do comando "cucumber-t@login_aluno", sendo esta copiada e colada para dentro do "login_aluno.rb", que até então estava vazio.

Em seguida foi apagada a descrição pendente de cada ação, equivalente a um passo. E iniciou-se a escrita dos códigos que automatizam as ações descritas e validadas dentro dos *gherkins* do cenário, por meio dos comandos do *framework* Capybara de automação de testes em aplicações web, como pode ser visualizado na Figura 14.

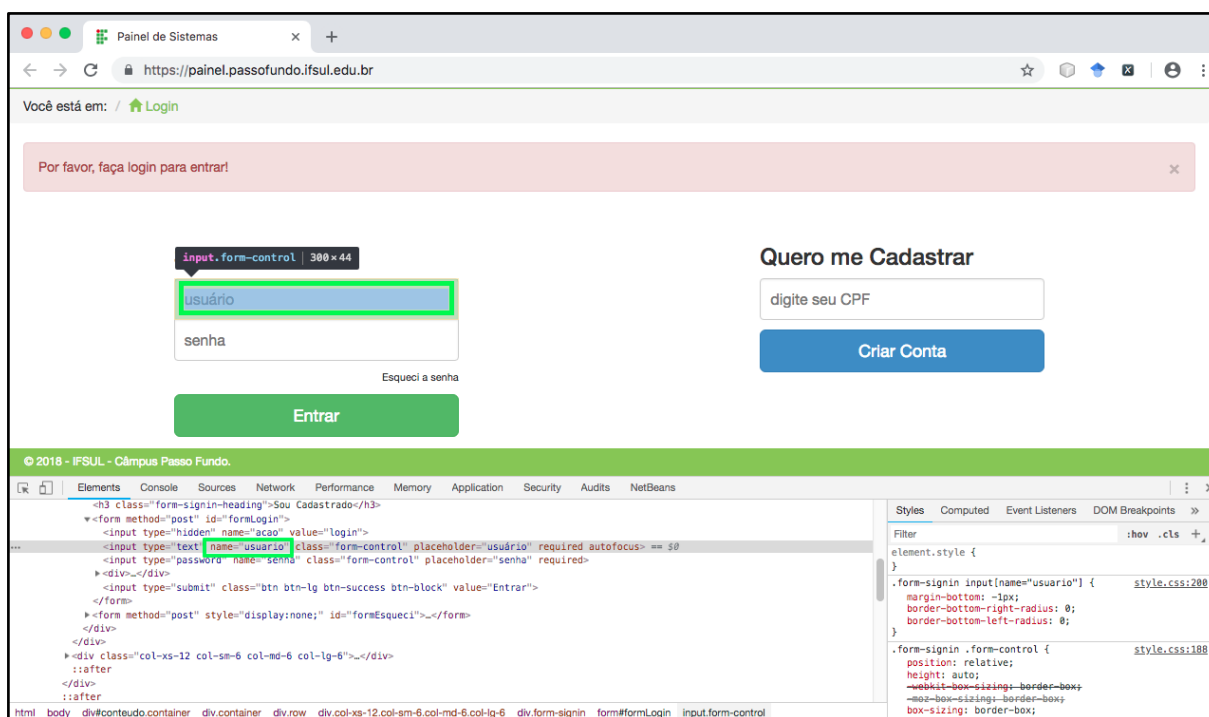
Figura 14 - Código de automação dos passos do cenário: login_aluno.feature



Fonte: Do Autor, 2018.

Para validar a pré-condição "Dado" descrita, foi utilizado o comando *visit*, que irá acessar a url passada por parâmetro, que no caso do cenário atual, não possui nenhuma outra segmentação de rota, portanto a url visitada será a do painel de sistemas do IFSUL - Campus - Passo Fundo que foi setada nas configurações "config.app_host" do arquivo env.rb já descrito. Abaixo do comando *visit*, visualiza-se o comando *sleep* que é responsável por aguardar cinco segundos na tela, antes de prosseguir para o próximo passo. A verificação dos elementos correspondentes às ações que foram automatizadas, ocorreu por meio da inspeção da página do Painel de Sistemas, como pode ser visualizado na Figura 15.

Figura 15 - Inspeção da página: Paine de Sistemas do IFSUL - Campus Passo Fundo.



Fonte: Do Autor, 2018.

A partir da inspeção da página do painel, foi possível identificar que o campo usuário possuía o atributo `'name=usuário'` dentro de sua `tag input`, possibilitando assim, a utilização do comando `fill_in`, equivalente a ação de preenchimento de formulários, que recebem dentre seus parâmetros, algum atributo para identificar um elemento, como por exemplo `name:'usuario'`, e o valor com que este campo deverá ser preenchido, por meio de seu outro parâmetro `with`.

O mesmo comando `fill_in` também foi utilizado para validar o preenchimento do campo senha, com a determinada senha do usuário aluno. Já o comando `"find(input[value='Entrar']).click"` na linha 23 da Figura 14, buscou encontrar o elemento que continha o html passado por parâmetro, e após validar este html, foi efetuado a ação do clique sobre este, no objetivo de validar a ação de clique no botão "Entrar" do login.

Objetivando-se validar o resultado esperado "Então", foi criada uma variável `ruby` com o nome `"mensagem_sucesso"`, fazendo com que esta variável recebesse um valor do texto capturado ao encontrar o elemento passado por parâmetro. Como ilustrado na linha 31 da Figura 14, sendo este elemento correspondente a uma alerta de sucesso, contendo o usuário logado. Por fim para exibir a validação da captura do texto desse alerta, foi utilizado o comando `ruby "puts"` na linha 33 da Figura 14,

que exibe na tela a palavra mensagem concatenada com a captura do texto exibido no alerta do usuário logado.

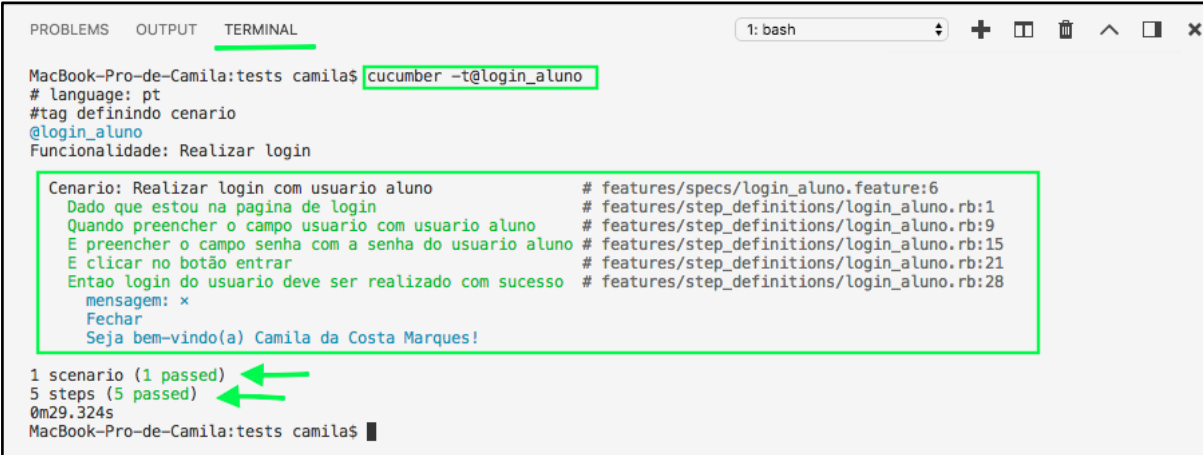
4.4 EXECUÇÃO DE UM CENÁRIO

A execução de um cenário pelo *framework* Cucumber, se dá pela compilação de dois arquivos, sendo que estes devem estar completos. Primeiramente o Cucumber analisa a descrição do comportamento do cenário por meio dos Gherkins elaborados, dentro do arquivo do cenário contendo a extensão ".feature", localizado dentro da pasta "features".

Em seguida o framework busca os passos equivalentes as ações que serão automatizadas neste cenário, requisitando então o código contido dentro do arquivo deste cenário com extensão ".rb" localizado dentro da pasta "step_definitions".

A partir do comando cucumber, é verificado a compatibilidade destes dois arquivos internamente, onde cada Gherkin, deve corresponder a uma ação automatizada, via código descrito. Como pode ser visualizado perante a Figura 16.

Figura 16 - execução do cenário: @login_aluno



```
MacBook-Pro-de-Camila:tests camila$ cucumber -t@login_aluno
# language: pt
# tag definindo cenário
@login_aluno
Funcionalidade: Realizar login

  Cenário: Realizar login com usuario aluno # features/specs/login_aluno.feature:6
    Dado que estou na pagina de login # features/step_definitions/login_aluno.rb:1
    Quando preencher o campo usuario com usuario aluno # features/step_definitions/login_aluno.rb:9
    E preencher o campo senha com a senha do usuario aluno # features/step_definitions/login_aluno.rb:15
    E clicar no botão entrar # features/step_definitions/login_aluno.rb:21
    Entao login do usuario deve ser realizado com sucesso # features/step_definitions/login_aluno.rb:28
      mensagem: x
      Fechar
      Seja bem-vindo(a) Camila da Costa Marques!

1 cenário (1 passed)
5 steps (5 passed)
0m29.324s
MacBook-Pro-de-Camila:tests camila$
```

Fonte: Do Autor, 2018.

O comando "cucumber -t@login_aluno" descrito na Figura 16, foi utilizado para execução do cenário que continha a tag "@login_aluno". Observando-se que, ao executar este cenário, todos os cinco passos englobados nesta feature, passaram, resultando assim na passagem do cenário ao todo. Além do resultado final de "passed" ou "failed" equivalente a quando o teste é passado, executado com sucesso, ou falhado, quando não foi possível executar algum passo determinado do

cenário. Também é possível visualizar na cor azul as validações ou mensagens requeridas no código a serem exibidas na tela.

4.5 UTILIZAÇÃO DE CONTEXTO NOS CENÁRIOS CRIADOS

Os casos de testes foram criados a partir dos requisitos funcionais levantados, em relação às ações que os usuários executariam perante o Sistema de Inscrição de Eventos do Instituto Federal Sul-Rio-Grandense - Campus - Passo Fundo.

Onde observou-se que o cenário de Login, seria executado anteriormente por diversos Cenários, várias vezes. Funcionando como uma espécie de pré-cenário, antes da execução de cada cenário descrito e contido no arquivo feature indicado, conforme ilustrado pela Figura 17.

Figura 17 - Contexto do arquivo: contexto_usuario_aluno.feature



Fonte: Do Autor, 2018.

A palavra reservada Contexto, é usada para identificar passos e etapas que são comuns a todos os cenários contidos no mesmo arquivo feature, no qual se encontra o contexto descrito. Como por exemplo antes da execução do primeiro cenário: Validar acesso ao Sistema de Eventos com usuário Aluno, deverá ser

rodado o contexto: Usuário aluno, que irá executar todos os passos para efetuar o login do usuário aluno, e somente depois da execução deste contexto, irá então começar a executar de fato o primeiro cenário, realizando as ações de clicar no ícone do sistema de eventos e ser redirecionado para respectiva página.

Todos os outros Cenários deste arquivo, como por exemplo: Validar exibição de eventos ativos com usuário aluno e Validar download do certificado de um evento de um usuário aluno, funcionaram da mesma forma, primeiramente sendo executado o contexto, descrito no arquivo *feature*, antes de cada um dos cenários, e posteriormente os passos específicos de cada cenário.

4.6 OUTROS CENÁRIOS E STEP DEFINITIONS

Ao todo foram criados 4 contextos para efetuar a realização do login, nos 4 perfis de usuários do Sistema, sendo eles: administrador, aluno, professor e visitante. Foi também realizada a criação de 8 cenários, juntamente com seus *step_definitions*, nomeados: 1 - Acessar a url com sucesso; 2 - Validar as inscrições do usuário com usuário Admin; 3 - Validar acesso ao Sistema de Eventos com usuário aluno; 4 - Validar exibição de eventos ativos com usuário aluno; 5 - Validar download do certificado de um evento de um usuário aluno; 6 - Validar cadastro de um novo evento; 7 - Realizar inscrição em um evento com usuário Visitante; 8 - Realizar login com usuário aluno. Que contemplaram as principais funcionalidades e ações dos requisitos levantados.

A descrição destes outros cenários e "*steps_definitions*" não foi descrita porque estas seguem a mesma lógica, do contexto "*contexto_usuario_aluno*" especificado no item 9.5, e foram dispostas na seção de anexos deste trabalho.

5 CONSIDERAÇÕES FINAIS

O presente trabalho teve como objetivo o desenvolvimento da automação dos testes funcionais do Sistema de Inscrições em Eventos do Instituto Federal Sul-Rio-Grandense do Campus Passo Fundo. Com intuito de apresentar o estudo de caso da integração de ferramentas utilizadas para realizar a automação dos Casos de Testes criados, a partir dos requisitos funcionais levantados.

Para realizar o desenvolvimento deste trabalho foi escolhida a linguagem de programação Ruby, juntamente com a utilização do framework Capybara, responsável pela automação de testes em aplicações web, e o framework Cucumber encarregado de gerar a arquitetura do projeto e gerenciar a execução dos Casos de Testes.

Para atingir o objetivo foi necessário a criação e execução de oito Cenários dos casos de testes mapeados, elaborados a partir dos requisitos levantados do sistema. Primeiramente foi necessário estudar a modelagem da automação dos casos de testes funcionais, e em seguida o funcionamento da integração das ferramentas e dos processos de automação de testes.

Com a realização deste trabalho, foi possível identificar que a utilização da Metodologia BDD e do framework Cucumber, interligados com a automação dos Cenários criados por meio do framework Capybara, apresentou a capacidade de executar seus testes em menor tempo do que o estimado durante a execução destes mesmos testes de forma manual.

E quanto ao planejamento dos casos de testes criados, o BDD mostrou apresentar a partir do uso de Gherkins e os critérios de aceite de um Cenário, uma forma mais clara nas especificações dos requisitos funcionais do negócio, potencializando um maior entendimento e colaboração entre todos os integrantes de um projeto. A escolha do framework Capybara para realizar a codificação dos passos dos cenários criados foi de fácil aprendizado por conta, deste framework ser *open-source*, dispor de sua inteira documentação e exemplos de uso na internet.

O custo benefício de se utilizar a automação de casos de testes funcionais em um projeto se verificou a partir dos pontos relatados anteriormente, com a ressalva de que é preciso ter-se a consciência de que um automatizador de testes, muitas vezes poderá custar mais caro do que um testador manual, pois este profissional

necessita obter o conhecimento de programação e páginas web além de conhecer os processos de teste de software.

Como trabalhos futuros é sugerida a continuação deste estudo, utilizando a gem SitePrism juntamente com o framework Capybara para realizar um melhor mapeamento dos elementos contidos nas páginas web, através da modelagem centralizada dos elementos de cada página, que serão automatizados.

REFERÊNCIAS

ANDERLE, Angelita. **Introdução de BDD (Behaviour Driven Development) como Melhoria de Processo no Desenvolvimento Ágil de Software**. 44f. Trabalho de Conclusão de Curso (Especialização) - Pós-Graduação Lato Sensu Qualidade de Software, Universidade do Vale do Rio dos Sinos - UNISINOS, 2015.

BARAÚNA, Hugo. **Cucumber RSpec**: Construa aplicações Ruby e Rails com testes e especificações. [S.L.]: Casa do Código, 2013. 438 p.

BERÇAM, Rafael. Automação de testes: Cucumber + Selenium em ruby (Introdução e Conceitos). **Medium**, [SL], dez./jun. 2017. Disponível em: <<https://medium.com/@rafaTelberam/automação-de-testes-cucumber-selenium-em-ruby-introdução-e-conceitos-2bfa28793980>>. Acesso em: 11 jun. 2018.

CORBUCCI, Hugo; ANICHE, Mauricio. **Test-driven Development**: Teste e Design no Mundo Real com Ruby. 1 ed. São Paulo: Casa do Código, 2014. 191 p.

DEVMEDIA. **Desenvolvimento orientado por comportamento (bdd)**. Disponível em: <<https://www.devmedia.com.br/desenvolvimento-orientado-por-comportamento-bdd/21127>>. Acesso em: 01 jun. 2018.

LARA, Samyra. Testes automatizados com Capybara. **Medium**, [S.L], mar./jul. 2017. Disponível em: <<https://medium.com/jaguaribetech/testes-automatizados-com-capybara-579e9688e3ab>>. Acesso em: 29 mai. 2018.

MENDES, Eduardo. Desenvolvimento Baseado em Testes: BDD + Cucumber. **INSLIDESHARE**, [S.L], out./jul. 2012. Disponível em: <<https://pt.slideshare.net/edumendes/bdd-com-cucumber-14910137>>. Acesso em: 02 jun. 2018.

OLIVEIRA, Thaís. **Automação de testes**: o que é, quando e por que automatizar. Disponível em: <<https://medium.com/venturus/quais-as-razões-para-automação-de-testes-c177cbd9397>>. Acesso em: 05 jun. 2018.

PRESSMAN, Roger S. **Engenharia de Software**: Uma abordagem Profissional. 7 ed. Porto Alegre: AMGH, 2011. 771 p.

RIOS, Emerson; MOREIRA FILHO, Trayahú. **Teste de Software**. 3. ed. Rio de Janeiro, RJ: Alta Books, 2013. 296 p.

SOARES, Ismael. **Desenvolvimento orientado por comportamento (BDD)**. 2011. Disponível em: <<https://www.devmedia.com.br/desenvolvimento-orientado-por-comportamento-bdd/21127>>. Acesso em: 05 jun. 2018.

SOMMERVILLE, Ian. **Engenharia de Software**. 9 ed. São Paulo: Prentice Hall, 2011. 529 p.

DE SOUZA, Thiago Cruz. Conhecendo a Linguagem Ruby. **Medium**, [S.L], jun./jul. 2008. Disponível em: <<https://www.devmedia.com.br/conhecendo-a-linguagem-ruby/8226>>. Acesso em: 10 jun. 2018.

TI.EXAMES. **Módulo 2 - modelos de desenvolvimento de software**. Disponível em: <<http://www.tiexames.com.br/novosite2015/index.php>>. Acesso em: 28 mai. 2018.

THOUGHTWORKS. Disponível em: <<https://www.thoughtworks.com/pt/insights/blog/3-essential-basics-setting-automation-suite-web-apps>>. Acesso em: 10 jun. 2018.

APÊNDICES

APÊNDICE A - arquivos do cenário: acessando site

Descrição do cenário do arquivo *acessando_site.feature*

```
#language : pt
@acessando_site
Funcionalidade: acessando site do inf

Cenário: acessar a url com sucesso
Quando acessar a url
Entao eu verifico que estou na pagina certa
```

Código dos steps_definitions do arquivo: *acessando_site.rb*

```
Quando("acessar a url") do
  #visitar a url passada por parametro
  visit('http://painel.passofundo.ifsul.edu.br')
  #esperando 20 segundos na tela
  sleep(20)
end

Entao("eu verifico que estou na pagina certa") do
  #verificando se a pagina atual contem o caminho passado por parametro
  expect(page).to have_current_path('/')
end
```

APÊNDICE B - arquivos do cenário contexto_usuario_admin

Descrição do arquivo: *contexto_usuario_admin.feature*

```
#language:pt
@contexto_usuario_admin
Funcionalidade: Usuario Administrador (ADMIN)
#definicao do cenario ou passo que deve rodar antes de cada cenario descrito neste arquivo
#uma especie de pré-cenario
Contexto: Usuario Admin
Dado que estou na pagina de login
Quando preencher o campo usuario com usuario Admin
E preencher o campo senha com a senha do usuario Admin
E clicar no botão entrar
Entao login do usuario deve ser realizado com sucesso
```

@validar_inscricao_admin

Cenário: Validar as inscrições do usuário

Dado que acessei o Sistema de Inscrições em Eventos

E selecionei um evento

Quando acessar o menu principal

E clicar em Minhas Inscrições

Então devo visualizar as inscrições que realizei no evento

Código dos steps_definitions do arquivo: contexto_usuario_admin.rb

```
Quando("preencher o campo usuario com usuario Admin") do
  #preenchendo o campo que contem o nome usuario com o identificador passado por parametro
  fill_in(name:'usuario', with: 'taeteste' )
  sleep(5)
end

Quando("preencher o campo senha com a senha do usuario Admin") do
  #preenchendo o campo que contem o nome senha com a senha passada por parametro
  fill_in(name:'senha', with: 'camila')
  sleep(5)
end

#cenario_validar_inscricao_admin
Dado("que acessei o Sistema de Inscrições em Eventos") do
  #encontrando o elemento com id=sysEve e clicando nele
  find_by_id("sysEve").click
end

Dado("selecionei um evento") do
  #encontrando elemento que contenha o href descrito e efetuando evento hover sobre ele
  find("[href='./?eve=24']").hover
  sleep(8)
  find("[href='./?eve=24']").click
  sleep(3)
end

Quando("acessar o menu principal") do
  click_on 'Menu'
  sleep(8)
end

Quando("clicar em Minhas Inscrições") do
  find(:css, "[href='/syseve/inscricoes/?acao=minhas']").hover
  sleep(5)
  find(:css, "[href='/syseve/inscricoes/?acao=minhas']").click
  sleep(15)
end
```

```

end
Entao("devo visualizar as inscrições que realizei no evento") do
  #percorrendo toda a pagina procurando a classe panel body e para cada div
  page.all(:css, '[class="panel-body"]').each do |div|

    #pegando o conteudo de texto dessa div e armazenando na variavel conteudo
    conteudo_div = div.text
    #percorrendo cada linha do conteudo inteiro armazenado
    conteudo_div.each_line do |line|

      #se a linha encontrada inclui a palavra Segunda
      if line.include? "Segunda"
        puts "Dia: #{line}"
      end

      #se a linha encontrada inclui a palavra Terça
      if line.include? "Terça"
        puts "Dia: #{line}"
      end

      #se a linha encontrada inclui a palavra Quarta
      if line.include? "Quarta"
        puts "Dia: #{line}"
      end

      #se a linha encontrada inclui a palavra Atividade
      if line.include? "Atividade"
        puts "#{line}"
      end

      #se a linha encontrada inclui a palavra Ministrante(s)
      if line.include? "Ministrante(s):"
        puts "#{line}"
      end
    end
  end
end
end
end

```

APÊNDICE C - arquivos do cenário contexto_usuario_aluno

Descrição do arquivo: contexto_usuario_aluno.feature

```
#language: pt
```

@contexto_usuario_aluno*Funcionalidade: Usuario Aluno**#definicao do cenario ou passo que deve rodar antes de cada cenario descrito neste arquivo**#uma especie de pré-cenario**Contexto: Usuario aluno**Dado que estou na pagina de login**Quando preencher o campo usuario com usuario aluno**E preencher o campo senha com a senha do usuario aluno**E clicar no botão entrar**Entao login do usuario deve ser realizado com sucesso***@acesso_sistema_aluno***Cenario: Validar acesso ao Sistema de Eventos com usuario Aluno**Quando clicar no icone do sistema de eventos**Entao devo ser redirecionado para pagina do sistema***@eventos_ativos***Cenario: Validar exibição de eventos ativos com usuario aluno**Dado que cliquei no icone do Sys Eve**Quando for direcionado para a pagina do sistema de eventos - Sys Eve**Então devo visualizar os eventos ativos***@download_certificado_aluno***Cenario: Validar Download do certificado de um evento de um usuário Aluno**Dado que acessei o Sistema de Incrições em Eventos**Quando acessar o Menu principal selecionando a opção Meus Certificados**E for redirecionado para pagina dos certificados disponíveis**E clicar em Gerar Certificado**Entao download do certificado escolhido deverá ser realizado***Código dos steps_definitions do arquivo: contexto_usuario_aluno.rb***#exigindo a utilizacao da gem pdf-reader nesse arquivo**require 'pdf-reader'**#cenario_acesso_sistema_aluno**Quando("clicar no icone do sistema de eventos") do**#encontrando o elemento com id=sysEve e clicando nele**find_by_id("sysEve").click**#esperando 5 segundos na tela**sleep(5)**end**Entao("devo ser redirecionado para pagina do sistema") do*


```

#encontrando o elemento com id=nome-sistema e clicando nele
#declarando a variavel titulo_sistema para receber o texto desse elemento
titulo_sistema = find_by_id('nome-sistema').click
titulo_sistema = find_by_id('nome-sistema').text
#imprimindo a mensagem de sistema exibido concatenando a variavel a mensagem
puts "Sistema exibido: #{titulo_sistema}"
#verificando se o texto armazenado na variavel titulo_sistema é igual com titulo descrito
expect(titulo_sistema).to eq! 'Sys{Eve} (Sistema de Inscrições em Eventos)'
sleep(5)
end

#cenario_eventos_ativos
Dado("que cliquei no icone do Sys Eve") do
  #encontrando elemento por seu id=sysEve e clicando nele
  find_by_id('sysEve').click
end

Quando("for direcionado para a pagina do sistema de eventos - Sys Eve") do
  sleep(3)
  #verificando se a pagina atual contem o caminho passado por parametro
  expect(page).to have_current_path('/syseve/eventos/')
  #declarando a variavel titulo e atribuindo a ela o valor pego
  # pelo h3 sendo este igual ao texto descrito
  titulo = first(:css,'div [class="destaques"] h3').text
  sleep(5)
  puts "#{titulo}"
end

Então("devo visualizar os eventos ativos") do
  #encontrando elemento que contenha o href descrito e efetuando evento hover sobre ele
  evento1 = find("[href='./?eve=27']").hover
  sleep(8)
  titulo_evento1 = find("[href='./?eve=27'] h3").text
  puts "Título evento 1 : #{titulo_evento1}"

  #evento2 = find("[href='./?eve=26']").hover
  #sleep(8)
  #titulo_evento3 = find("[href='./?eve=26']h3").text
  #puts "Título evento 2 : : #{titulo_evento3}"

  sleep(3)
  evento2 = find("[href='./?eve=24']").hover
  sleep(8)
  titulo_evento2 = find("[href='./?eve=24'] h3").text
  puts "Título evento 2 : #{titulo_evento2}"

```

```

end

#cenario download_certificado_aluno
Quando("acessar o Menu principal selecionando a opção Meus Certificados") do
  #clikando sobre o menu principal
  click_on 'Menu'
  #encontrando elemento que contenha o href descrito e efetuando evento hover sobre ele
  find(:css, '[href="/syseve/inscricoes/?acao=certificados"]').hover
  sleep(3)
  click_on 'Meus Certificados'
end

Quando("for redirecionado para pagina dos certificados disponíveis") do
  #verificando que a pagina atual contenha o caminho passado por parametro
  expect(page).to have_current_path('/syseve/inscricoes/?acao=certificados')
  sleep(3)
end

Quando("clicar em Gerar Certificado") do
  #encontrando elemento que contenha o href descrito e efetuando evento de clique sobre ele
  #baixando o certificado do evento de id=9
  find(:css, 'a[href="/?acao=gerar_certificado&eve=9"]').click
  sleep(5)
end

Entao("download do certificado escolhido deverá ser realizado") do
  #declarando a variavel eventos_div_container para receber todos os elementos contidos dentro da
  div.container
  #e percorrendo cada linha dessa div
  eventos_div_container = page.all("div.container ").each do |div_linha|
    #declarando a variavel div_eventos para receber o texto de cada linha da div container
    div_eventos = div_linha.text
    sleep(2)
    #percorrendo cada linha da variavel div_eventos que armazena o texto das linhas da div container.
    div_eventos.each_line do |line|
      #verificando se a linha inclui a palavra "Sustentabilidade"
      if line.include? "Sustentabilidade"
        puts "Baixou o PDF do Evento: #{line}"
      end
    end
  end
  sleep(8)
end

#declarando a variavel downloads para receber o caminho passado

```

```

downloads = "/Users/camila/Downloads"
#se o diretorio atual for diferente do caminho declarado na variavel downloads
if Dir.pwd != downloads
  #CHDIR(change working directory ) muda o diretorio atual para o contido na variavel downloads
  Dir.chdir( downloads )
end
#imprimindo o diretorio
puts "#{downloads}"
#declarando a variavel arquivospdf para receber tudo que contiver
#no diretorio do caminho passad por parametro
arquivospdf = Dir["/Users/camila/Downloads/documento_*.pdf"]
#declarando a variavel arquivo_baixado para receber a primeira posicao de todo
#o diretorio contido na variavel arquivospdf
arquivo_baixado = arquivospdf[0]
puts "#{arquivo_baixado}"
#comando utilizado pela gem pdf-reader que
#abre o arquivo baixado o lendo-o
PDF::Reader.open("#{arquivo_baixado}") do |reader|
  #executando um for para ler cada pagina desse arquivo
  reader.pages.each do |page|
    #imprimindo na tela cada texto por pagina
    puts page.text
  end
end
end
end

```

APÊNDICE D - arquivos do cenário contexto_usuario_professor

Descrição do arquivo: contexto_usuario_professor.feature

```

#language: pt
@contexto_usuario_professor
Funcionalidade: Usuario Professor
#definicao do cenario ou passo que deve rodar antes de cada cenario descrito neste arquivo
#uma especie de pré-cenario
Contexto: Usuario Professor
Dado que estou na pagina de login
Quando preencher o campo usuario com usuario Professor
E preencher o campo senha com a senha do usuario Professor
E clicar no botão entrar
Entao login do usuario deve ser realizado com sucesso

@cadastro_evento_professor
Cenario: Validar cadastro de um novo evento

```

Dado que selecionei o cadastro de um novo Evento

Quando preencher o campo Nome do Evento

E preencher o campo Data Início do evento

E preencher o campo Data Fim do evento

E fazer upload da imagem do evento

E preencher a Data de Início das Incrições

E preencher a Data Fim das Incrições

E selecionar um Responsável pela Assinatura do comprovante de participação

E preencher a Data da Disponibilização do comprovante de participação

E preencher a Descrição da Assinatura do comprovante de participação

Quando clicar em Gravar

Entao o Evento deverá ser cadastrado com sucesso

Código dos steps_definitions do arquivo: contexto_usuario_professor.rb

```
Quando("preencher o campo usuario com usuario Professor") do
  #preenchendo o campo usuario com o usuario passado por parametro
  fill_in(name:'usuario', with: 'carmen.scorsatto@passofundo.ifsul.edu.br')
  sleep(2)
end

Quando("preencher o campo senha com a senha do usuario Professor") do
  #preenchendo o campo senha com a senha passada por parametro
  fill_in(name:'senha', with: 'xxxxx')
  sleep(2)
end

#cenario_cadastro_evento_professor
Dado("que selecionei o cadastro de um novo Evento") do
  #clitando no icone do sistema de eventos
  find_by_id('sysEve').click
  sleep(4)
  #visitando a url passada por parametro a partir da default configura no arquivo env.rb
  visit('/syseve/eventos/?acao=incluir')
  sleep(2)
end

Quando("preencher o campo Nome do Evento") do
  #preenchendo o campo Nome do Evento com o nome do evento que será cadastrado
  fill_in(name:'nome', with:'Formatura TSPI- 2018/2')
  sleep(2)
end

Quando("preencher o campo Data Início do evento") do
  #preenchendo o campo Data Início com a data do evento que será cadastrado
  fill_in(name:'data_inicio', with:'/21/10/2018/')
  sleep(5)
end
```

```

end
Quando("preencher o campo Data Fim do evento") do
  #preenchendo o campo Data Fim com a data do evento que será cadastrado
  fill_in(name:'data_fim' , with:'/23/02/2019/')
  sleep(2)
end
Quando("fazer upload da imagem do evento") do
  #declarando a variavel imagem para receber o diretorio atual concatenando com o caminho passado por
  parametro
  @imagem = File.join(Dir.pwd,'features/formatura2018.jpg')
  #preenchendo o campo imagem (identificado pelo type=file e o name=img ) com a imagem contida variavel
  imagem
  attach_file('img', @imagem)
  sleep(5)
end
Quando("preencher a Data de Início das Inscrições") do
  #preenchendo o campo Data Início com a data do evento que será cadastrado
  fill_in(name:'data_inicio_inscricao' , with:'/21/10/2018/')
  sleep(3)
end
Quando("preencher a Data Fim das Inscrições") do
  #preenchendo o campo Data Fim com a data do evento que será cadastrado
  fill_in(name:'data_fim_inscricao' , with:'/23/02/2019/')
  sleep(3)
end
Quando("selecionar um Responsável pela Assinatura do comprovante de participação") do
  #selecionando a opcao passada por parametro do select com name = assinatura_id
  select 'Juliana Favretto' , from: 'assinatura_id'
  sleep(5)
end
Quando("preencher a Data da Disponibilização do comprovante de participação") do
  #preenchendo o campo Data da Disponibilização com a data do evento que será cadastrado
  fill_in(name:'data_comprovante' , with:'/01/12/2018/')
  sleep(5)
end
Quando("preencher a Descrição da Assinatura do comprovante de participação") do
  sleep(4)
  find(:css, 'input[name="assinatura_descricao"]').click
  sleep(2)
  #preenchendo o campo Descrição da Assinatura com o valor passado por parametro
  fill_in(name:'assinatura_descricao' , with:'Formatura TSPI')
  sleep(3)
  #preenchendo o campo Formador (para cursos de extensão) com o valor passado por parametro

```

```

fill_in(name:'formador' , with:'formador teste')
sleep(5)
end
Quando("clicar em Gravar") do
  sleep(2)
  #clicando no botao com nome = Gravar
  click_on 'Gravar'
  sleep(8)
end
Entao("o Evento deverá ser cadastrado com sucesso") do
  sleep(5)
  #pegando o texto contido dentro da div identificada e passada por parametro
  # e atribuido-o a variavel mensagem_evento
  mensagem_evento = find(:css, 'div [class="alert alert-success"]').text
  sleep(5)
  puts "#{mensagem_evento}"
  #preenchendo o campo Pesquisar com identificado por sua classe com o valor passado por parametro
  fill_in(class:'form-control' , with:'Formatura TSPI- 2018/2')
  sleep(3)
  #verificando se o texto passado por parametro existe na página atual
  have_text? ('Formatura TSPI- 2018/2')
  sleep(3)
  puts "Evento cadastrado com sucesso"
  sleep(8)
end

```

APÊNDICE E - arquivos do cenário contexto_usuario_visitante

Descrição do arquivo: contexto_usuario_visitante.feature

```

#language: pt
@contexto_usuario_visitante

Funcionalidade: Usuario visitante
#definicao do cenario ou passo que deve rodar antes de cada cenario descrito neste arquivo
#uma especie de pré-cenário
Contexto: Usuario visitante
Dado que estou na pagina de login
Quando preencher o campo usuario com usuario visitante
E preencher o campo senha com a senha do usuario visitante
E clicar no botão entrar
Entao login do usuario deve ser realizado com sucesso

@inscricao_evento_visitante

```

Cenário: Realizar inscrição em um evento com usuario Visitante

Dado que escolhi um determinado evento

Quando for redirecionado para a página de inscrições abertas

E escolher uma atividade para me cadastrar

E realizar inscrição nesta atividade

Entao meu codigo de participante no evento deve ser exibido

Código dos steps_definitions do arquivo: contexto_usuario_visitante.rb

```
Quando("preencher o campo usuario com usuario visitante") do
  #preenchendo o campo que contem o name usuario com o email passado por parametro
  fill_in(name:'usuario', with: 'vteste')
  sleep(5)
end

Quando("preencher o campo senha com a senha do usuario visitante") do
  #preenchendo o campo que contem o name senha com a senha passada por parametro
  fill_in(name:'senha', with: 'camila')
  sleep(5)
end

#@inscricao_evento_visitante
Dado("que escolhi um determinado evento") do
  #encontrando o elemento com id=sysEve e clicando nele
  find_by_id('sysEve').click
  #esperando 3 segundos na tela
  sleep(3)
  #encontrando o elemento que contenha o href passad por parametro e clicando nele
  find("[href='./?eve=24']").click
  sleep(10)
end

Quando("for redirecionado para a página de inscrições abertas") do
  #comparando se a pagina esperada contem o caminho atual passado por parametro
  expect(page).to have_current_path('/syseve/inscricoes/')
  #declarando uma variavel nome_evento
  # e guardando nela o texto do primeiro h2
  nome_evento = first(:css, 'h2').text
  # imprimindo na tela o nome do evento guardado
  puts "Evento: #{nome_evento}"
end

Quando("escolher uma atividade para me cadastrar") do
```

```

#encontrando o elemento pelo atributo data-target
find("[data-target="#myModal249"]').click
sleep(5)
nome_atividade = find(:css, '[id="myModalLabel249"]').text
puts "Atividade inscrita: #{nome_atividade}"
sleep(15)
#clcando no botão com nome Fechar
click_button 'Fechar'
sleep(3)
end

Quando("realizar inscrição nesta atividade") do
#encontrando o elemento que contenha o input passado por parametro e efetuando click
find("input[value="249"]').click
sleep(3)
find("input[value="Realizar Inscrição"]').click
sleep(5)
end

Entao("meu codigo de participante no evento deve ser exibido") do
click_on 'Minhas Inscrições'
sleep(10)
#capturando o texto de dentro do atributo code e guardando-o na variavel codigo
codigo = find('code').text
#imprimindo o texto guardado na variavel codigo
puts "Número Código: #{codigo}"
end

```

APÊNDICE F - arquivos do cenário login_aluno

Descrição do arquivo: login_aluno.feature

```

#language:pt
@contexto_usuario_admin
Funcionalidade: Usuario Administrador (ADMIN)
#definicao do cenario ou passo que deve rodar antes de cada cenario descrito neste arquivo
#uma especie de pré-cenario
Contexto: Usuario Admin
Dado que estou na pagina de login
Quando preencher o campo usuario com usuario Admin
E preencher o campo senha com a senha do usuario Admin
E clicar no botão entrar
Entao login do usuario deve ser realizado com sucesso

```


@validar_inscricao_admin

Cenário: Validar as inscrições do usuário

Dado que acessei o Sistema de Inscrições em Eventos

E selecionei um evento

Quando acessar o menu principal

E clicar em Minhas Inscrições

Então devo visualizar as inscrições que realizei no evento

Código dos steps_definitions do arquivo: login_aluno.rb

```
Dado("que estou na pagina de login") do
  #acessando a url default setada no arquivo env.rb
  #configurada em config.app_host
  visit('/')
  #esperando 5 segundos na tela
  sleep(5)
end

Quando("preencher o campo usuario com usuario aluno") do
  #preenchendo o campo que contem o name usuario com o email passado por parametro
  fill_in(name:'usuario', with: 'camila_cmarques@hotmail.com')
  sleep(5)
end

Quando("preencher o campo senha com a senha do usuario aluno") do
  #preenchendo o campo que contem o name senha com a senha passada por parametro
  fill_in(name:'senha', with: 'xxxxx')
  sleep(5)
end

Quando("clique no botão entrar") do
  #encontrando o elemento input com value=entrar e clicando nele
  find("input[value='Entrar']").click
  #esperando 5 segundos na tela
  sleep(5)
end

Entao("login do usuario deve ser realizado com sucesso") do
  #atribuindo valor da classe da div a variavel mensagem_sucesso
  #e pegando o texto guardado nesta variavel
  mensagem_sucesso = find('div[class="alert alert-success"]').text
  #imprimindo texto pego no cenario
  puts "mensagem: #{mensagem_sucesso}"
  #esperando 5 segundos na tela
  sleep(5)
end
```

