

**INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
DO RIO GRANDE DO SUL
CAMPUS RESTINGA
ANÁLISE E DESENVOLVIMENTO DE SISTEMAS**

**PYBOT: *FRAMEWORK* DE AUTOMAÇÃO EM
BROWSER COM SELENIUM E PYTHON**

FELIPE DOS SANTOS VIEGAS

PORTO ALEGRE

2017

FELIPE DOS SANTOS VIEGAS

**PYBOT: *FRAMEWORK* DE AUTOMAÇÃO EM
BROWSER COM SELENIUM E PYTHON**

Trabalho de Conclusão de Curso apresentada como requisito parcial para obtenção do grau de Tecnólogo em Análise e Desenvolvimento de Sistemas da Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul - IFRS, Campus Restinga.

Orientador: Prof. Me. Roben Castagna Lunardi

Porto Alegre
2017

FELIPE DOS SANTOS VIEGAS

**PYBOT: *FRAMEWORK* DE AUTOMAÇÃO EM
BROWSER COM SELENIUM E PYTHON**

Trabalho de Conclusão de Curso apresentado como
requisito parcial para obtenção do grau de Tecnólogo
em Análise e Desenvolvimento de Sistemas do
Instituto Federal de Educação, Ciência e Tecnologia
do Rio Grande do Sul - IFRS, Campus Restinga.

Data de Aprovação: 31/01/2018

Banca Examinadora

Prof. Me. Roben Castagna Lunardi - IFRS - Campus Restinga
Orientador

Prof. Eliana Beatriz Pereira- IFRS- Campus Restinga
Membro da Banca

Prof. Rodrigo Lange- IFRS- Campus Restinga
Membro da Banca

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO RIO GRANDE DO SUL

Reitor: Prof. Osvaldo Casares Pinto

Pró-Reitora de Ensino: Profa. Clarice Monteiro Escott

Diretor do Campus Restinga: Prof. Gleison Samuel do Nascimento

Coordenador do Curso de Ciência da Computação: Prof. Rafael Pereira Esteves

Bibliotecária-Chefe do Campus Restinga: Paula Porto Pedone

Dedico esse trabalho a todos aqueles que me ajudaram do fim ao começo!

RESUMO

Em muitas empresas existe uma carência quando o assunto é automação de testes ou processos web em navegadores. A necessidade de testes de regressão, testes de funcionalidades e ou automação de processos cresce junto com o sistema, porém a pratica dessas atividades só ganha força quando aparece alguma necessidade ou problema.

Para resolver este problema, propõe-se neste trabalho, um *framework* com o objetivo de trazer aos usuários uma ferramenta de fácil uso e com recursos úteis para o desenvolvimento dessas tarefas, contando com a facilidade e versatilidade da linguagem *Python* e a integração de navegadores com o *framework Selenium*.

O conjuntos de ferramentas que o *framework* proposto apresenta são: gerenciamento automático dos controladores de navegadores(*drivers*), módulo de relatórios e *logs* para controle de atividades executadas, padronização de criação de elementos de tela utilizando o padrão PageObject e a identificação de alteração de *layout*.

Palavras-chave: Automação, Selenium, Testes.

ABSTRACT

Currently, many companies lack a solution for test automation or web process management integrated to the browsers. The need for regression testing, functionality testing, and / or process automation grows along with the system, but the practice of these activities is only emerge when a need or problem appears.

In order to solve this problem, we propose a framework aims to present to users an easy-to-use tool with useful resources for the development of these tasks, with the simplicity and versatility of the *Python* language and the integration with browsers with *selenium framework*.

The sets of tools that the proposed framework presents are: automatic management of the drivers of navigators (*drivers*); module containing reports and logs to control activities performed; standardization of screen elements development using the standard *PageObject* and identification of layout change.

Palavras-chave: Automation, Selenium, Tests.

LISTA DE FIGURAS

Figura 1 – Diagrama de Componentes	18
Figura 2 – Exemplo - Uso da classe Manager	19
Figura 3 – Exemplo - Uso do Configuration	19
Figura 4 – Estrutura pybot.ini	20
Figura 5 – Exemplo - Uso PageObject com PageElement e PageElements	20
Figura 6 – Lista de Seletores	21
Figura 7 – Exemplo - Uso do Logger	22
Figura 8 – Exemplo - Uso do CsvHandler	23
Figura 9 – Exemplo de uso com Selenium	26
Figura 10 – Exemplo de uso com Pybot	27
Figura 11 – Profissões	29
Figura 12 – Facilidade de instalação	29
Figura 13 – Facilidade de Uso	30
Figura 14 – Atende necessidades básicas	30
Figura 15 – Atende necessidades avançadas	31
Figura 16 – Uso padrão Selenium	35
Figura 17 – Uso padrão Selenium com módulo	36
Figura 18 – Uso padrão Pybot	37

LISTA DE TABELAS

Tabela 1 – Comparativo dos Frameworks	16
---	----

LISTA DE ABREVIATURAS E SIGLAS

API	Interface de Programação de Aplicação (do Inglês Application Programming Interface - API)
BDD	Desenvolvimento Orientado a Comportamento (do Inglês Desenvolvimento Guiado por Comportamento - BDD)
DDT	Desenvolvimento Orientado a Dados (do Inglês Data-driven testing - DDT)
DOM	Modelo de Documento Objeto (do inglês Document Object Model - DOM)
TDD	Desenvolvimento orientado a testes (do ingles Test Driven Development - TDD)

SUMÁRIO

1	INTRODUÇÃO	12
2	TRABALHOS RELACIONADOS	14
2.1	Selenium Webdriver	14
2.2	Robotframework	15
2.3	Comparativo	15
3	ARQUITETURA DO PYBOT	18
3.1	Core	18
3.1.1	Manager	18
3.1.2	Configuration	19
3.2	Component	20
3.2.1	PageObject	20
3.2.2	WebElement	21
3.2.3	PageElement e PageElements	21
3.3	Report	21
3.3.1	Logger	22
3.3.2	CsvHandler	22
3.4	Driloader	23
4	TECNOLOGIAS EMPREGADAS	24
4.1	Python	24
4.2	Selenium WebDriver	24
4.3	Git e GitHub	24
5	DESENVOLVIMENTO PROPOSTO	25
6	AVALIAÇÃO DA SOLUÇÃO	28
6.1	Profissão do Usuário	28
6.2	Facilidade de instalação	29
6.3	Facilidade de Usabilidade	29
6.4	Atende as necessidades básicas	30
6.5	Atende as necessidades avançadas	30
7	CONCLUSÃO	32
	REFERÊNCIAS	33
	APÊNDICE A – PESQUISA DE SATISFAÇÃO DO PYBOT	34
	APÊNDICE B – DIAGRAMAS DE CENÁRIOS	35

1 INTRODUÇÃO

O ciclo de vida de software tem diversas etapas, das quais podem ser elencadas: Análise de requisitos, Concepção do Projeto, Desenvolvimento, Implantação e por fim Manutenção. Usualmente, nas etapas de Desenvolvimento e Manutenção ocorre a maior parte da criação ou a codificação do *software*, e na concepção de um projeto a necessidade da criação de um processo de testes que cresça junto do sistema não costuma ter a relevância necessária.

Atualmente, existem diversas ferramentas que possibilitam a criação de testes automatizados como por exemplo o *Selenium Webdriver* (SELENIUM, 2017) e o *Robotframework* (ROBOTFRAMEWORK, 2017), e esses testes automatizados podem ser desde os mais simples, como a verificação de campos dentro da página ou um determinado título, até testes mais complicados, como um teste de regressão, onde todas as funcionalidades e requisitos do software são testadas novamente para garantir que uma atualização do software não impacte em outras partes. Porém, para fazer uso dessas ferramentas existentes é necessário avaliar previamente diferentes questões, como compatibilidade com sistema operacional, linguagem suportada, funcionalidades oferecidas e padrões a serem utilizados, e a curva de aprendizagem para começar a utilizá-las pode ser demorada e custosa.

Este trabalho de conclusão de curso tem como objetivo criar um *framework* para auxiliar nas tarefas codificação e criação de processos de testes automatização para sistemas e ou sites em navegadores, contando com uma forma de utilização fácil e simples, e trazendo para si algumas das preocupações básicas que os desenvolvedores enfrentam ao utilizar outras *frameworks* de testes. Levando o nome de PyBot, união das palavras *Python*, linguagem utilizada para criação do projeto, e *Bot*, que em inglês quer dizer robô, essa ferramenta propõe disponibilizar para os usuários uma série de ferramentas para auxiliar a criação e implantação desses processos, contando com uma estrutura de criação dos *scripts* de testes. Para isso, esse *framework* contará com uma instalação simples com o mínimo de configurações e permitirá a criação de testes no padrão *PageObject* e *PageElement*, geração de registros de *logs* para controle de tarefas e passos executados e o gerenciamento automático de *drivers* de navegadores com a ferramenta *Driloader*.

Com isso a codificação de *scripts* de testes automatizados pode se tornar mais simples, pois parte das complexidades de se iniciar com esses testes automatizado será feita inteiramente pelo *framework*, encarregando o usuário em apenas na codificação das páginas do seu site no modelo simples e intuitivo do *Pybot* e na criação dos métodos que serão encarregados de executar

os determinados comandos daquela página.

O restante deste documento é organizado da seguinte forma: No Capítulo 2 serão apresentados os trabalhos relacionado, contando com 2 exemplos de ferramentas existentes e um comparativo dessas ferramentas e o *framework* proposto; No Capítulo 3 irá apresentar os módulos presentes no *framework* proposto com seus determinados propósitos e funcionalidades; No Capítulo 4 irá apresentar as tecnologias utilizadas para o desenvolvimento e controle do código fonte do *framework*; No Capítulo 5 irá apresentar um proposta de utilização do *framework*, fazendo uso de alguns conceitos apresentados; No Capítulo 6 irá apresentar uma análise de dados coletados a partir de um questionário com usuários sobre a utilização do *framework*; E por fim no Capítulo 7 será apresentado a conclusão deste trabalho junto de possíveis trabalhos futuros.

2 TRABALHOS RELACIONADOS

Conforme Max dos Santos (SANTOS, 2016) a importância da criação de processos automatizados cresce junto da importância que os *softwares* vão tendo na sociedade, e as empresas tem uma certa dificuldade quando tentam de adotar ou implantar tais tipos de processos, devido a falta de profissionais com esse conhecimento ou pelas técnicas presentes hoje.

A solução para automação de testes e processos em navegadores com maior adoção pela comunidade de desenvolvimento de software é o *Selenium* (SELENIUM, 2017). O *Selenium* trata-se de uma ferramenta composto por diversos projetos tais como *Selenium Grid*, *Selenium IDE*, *Selenium Remote Control* e o *Selenium WebDriver*, cada um com suas determinadas funcionalidades. Dentre os projetos citados, o *Selenium WebDriver* é o que mais se assemelha ao projeto proposto pois trata-se de um *framework* para integração com navegador, este por sua vez é utilizado como uma das dependências do Pybot (SEÇÃO 4.2).

Ainda, outra ferramenta também relacionada ao projeto proposto é o *Robotframework*, esta por sua vez é uma solução bem mais sofisticada, contendo uma quantidade abrangente de módulos e usos, porém junto traz uma complexidade maior para seu uso.

Similar a estes sistemas, a solução proposta neste Trabalho de Conclusão de Curso busca seguir direcionando a integração do *Selenium* com o *Python*. Porém, pretende-se com o Pybot o desenvolvimento ferramenta simples e de fácil uso, portanto.

Para entender melhor o problema, apresenta-se uma análise das funcionalidades e usos das ferramentas citadas anteriormente junto de um comparativo de alguns pontos fortes e fracos de cada uma delas.

2.1 SELENIUM WEBDRIVER

O *Selenium Webdriver* (WEBDRIVER, 2017) é de um *framework* onde disponibiliza-se para usuário uma *API*, Interface de Programação de Aplicação (do Inglês Application Programming Interface - API), para integração com o navegador escolhido. Com essa *API* é possível enviar diversos tipos de comandos para o navegador, tais como, verificação e iteração qualquer tipo de elemento presente no DOM (Modelo de Documento Objeto, do inglês *Document Object Model*), geração de *prints* de tela e manipulação do próprio navegador, como por exemplo maximizar a janela, redimensionar ou abrir uma nova janela. Todos os comando executados no navegador são comandos nativos de cada sistema operacional, sendo necessário possuir o *driver* específico para cada navegador e sistema operacional para que funcione.

Ainda, possui suporte as seguintes linguagens de programação: *Java*, *Csharp*, *Python*, *Ruby*, *Php*, *Perl* e *Javascript*. Na maioria dos casos, o suporte e funcionalidade são os mesmos para todas linguagens, porém o maior suporte é dado para a linguagem *Java*.

2.2 ROBOTFRAMEWORK

O *Robotframework* (ROBOTFRAMEWORK, 2017) é um dos *frameworks* mais abrangentes disponíveis para teste de software para linguagem *Python*. Esta solução consiste de uma vasta variedade de módulos distintos, contando com 11 módulos próprio do *framework* e mais 30 de projetos parceiros, possíveis de serem habilitados. Desta forma, possível de optar por incluir determinado módulo ou não no projeto, tendo, também, suporte a *Java* na maioria de seus módulos.

Sua arquitetura foi feita para executar testes utilizando-se de ATDD (*Acceptance Test Driven Development* ou Desenvolvimento Orientado a Testes de Aceitação) que trata-se de uma abordagem ou prática para a criação de requisitos colaborativamente entre o cliente e a equipe e fazendo uso da técnica de desenvolvimento Ágil BDD (*Behavior Driven Development* ou Desenvolvimento Guiado por Comportamento em português) onde são criados cenários para cada tipo de comportamento da aplicação levando em conta 3 estados, **Dado que**, **Quando** e **Então**, como no exemplo a seguir. Como por exemplo um caso de teste onde temos que verificar uma lista com objetos dentro não podem estar vazias, a escrita desse teste ficaria da seguinte maneira: **Dado** que uma nova lista é criada; **Quando** eu adiciono um objeto a ela; **Então** a lista não deve estar vazia. Dessa maneira para o *Robotframework* cara um desses estados é um passo a ser executado, e o mesmo foi codificado para se adequar a esse caso de teste O *Robotframework* conta também com os seguinte recursos:

- geração de relatórios de execução, tanto de sucesso como em falhas;
- interface por linha de comando;
- resposta de execução por XML.

2.3 COMPARATIVO

Foram levantados alguns dos pontos mais relevantes para a execução dos *scripts* de testes de cada um dos *framework* e feito uma tabela comparativa entre eles. Todos os *frameworks* apresentados no comparativo a seguir na Tabela 1 utilizam em sua base o próprio *Selenium*

Webdriver, portanto todas as funcionalidades padrões de integração com os navegadores descritas na SEÇÃO 2.1 não serão tratadas nesse comparativo.

Tabela 1 – Comparativo dos Frameworks

	Selenium	Robotframework	Trabalho Proposto
Integração com navegador	Sim	Sim	Sim
Gerenciamento de drivers dos navegadores	Não	Não	Sim
Linguagens Suportadas	Java, C#, Python, Ruby, Php, Perl e Javascript	Python e Java	Python
Supote ao conceito de Page Object	Não	Sim	Sim
Suporte à Técnicas de Desenvolvimento	Nenhuma	ATDD e BDD	Nenhuma
Verificação dinâmica dos elementos	Não	Sim	Sim
Arquivo de Configuração	Não	Sim	Sim
Modular	Não	Sim	Não
Código Aberto	Sim	Sim	Sim

Como podemos ver na Tabela 1, o diferencial é nas funcionalidades específicas de cada um dos *frameworks*, onde podemos perceber uma separação de níveis. Onde o *Robotframework* uma ferramenta super completa com módulos independentes e específicos para cada necessidade, dando assim uma versatilidade maior para usuário e o *Selenium Webdriver* sendo mais mais básico, onde não possui muito além de prover a integração com o navegador e suporte a diversas linguagens de programação.

O projeto proposto pretende trazer melhorias em relação ao *Selenium Webdriver* padrão mas não pretende ser uma ferramenta tão abrangente quanto o *Robotframework*, sendo um meio termo entre essas ferramentas. Para isso o projeto propõe dar uma maior facilidade para

os usuários no início das tarefas, gerenciando automaticamente os *driver* dos navegadores, utilizando-se de algumas técnicas para agilizar e organizar o desenvolvimento dos *scripts* com o padrão *Page Object* e a pesquisa dinâmica dos elementos nas paginas, geração de relatórios de execução e contendo um arquivo de configurações centralizado para as execuções dos *scripts*.

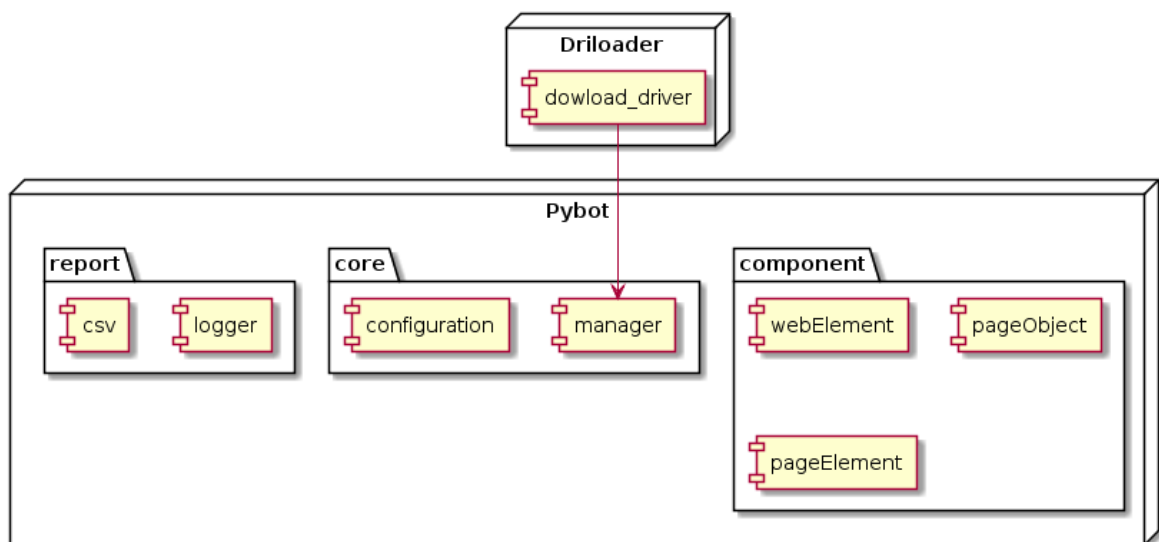
3 ARQUITETURA DO PYBOT

O *framework* proposto foi formado por 4 módulos básicos, cada um com suas devidas utilidades e funções. A separação de cada módulos foi dada com base em suas características e funcionalidades, onde temos um modulo responsável pela gerenciamento e funcionamento do *framework*, outro pela abstração das paginas dos sistema *web* a serem mapeados, controle e criação de relatórios e logs de execuções e por fim um responsável pelo download e gerenciamento dos *drivers* de cada navegador utilizados pelos *scripts* escritos pelos usuários.

Na Figura 1 é apresentado um diagrama dos módulos junto das suas classes principais que compõe o *framework* e nas próximas seções serão descritas as suas funcionalidades e usos.

Figura 1 – Diagrama de Componentes

Packages - Component Diagram



3.1 CORE

Este é o módulo principal do framework, nele contém as funcionalidades básicas para a operação do *framework* com o Selenium *Webdriver*, execução e o gerenciamento dos parâmetros de execuções de cada *script*.

3.1.1 Manager

A classe *Manager* é a classe principal do framework, ela serve para abstrair o uso do Selenium *Webdriver* criando uma camada de métodos próprios fazendo com que os *scripts* criados com o *Pybot* não sejam impactados por possíveis atualização na API do *Selenium*. Dentro

dele é feito o uso do *Driloader*, descrito na subseção 3.4 mais a frente, que verifica a necessidade do download do *driver* para poder executar o *Selenium Webdriver* e assim abrir o navegador escolhido pelo usuário.

Como exemplo na Figura 2 temos o exemplo de alguns dos comando disponíveis do Manager, sendo o da linha 4 para o início do processo e execução do *driver* do navegador, na linha 7 para navegar para uma URL e na linha 12 para finalizar o processo.

Figura 2 – Exemplo - Uso da classe Manager

```
1  from pybot import Manager
2  from Pages.ads import adsPage
3
4  manager = Manager()
5  page = adsPage(manager)
6
7  manager.go('https://ads.restinga.ifrs.edu.br')
8
9  page.goToDisciplinas()
10
11
12 manager.end()
13
```

3.1.2 Configuration

Responsável por gerar e gerenciar as configurações básicas para o *framework* e disponibilizá-las no arquivo *pybot.ini* disponível na mesma pasta que o *script*, caso não exista este arquivo é criado automaticamente para cada script. Nele é possível adicionar qualquer configurações ou parâmetros customizado do usuário para qualquer necessidade da execução do *script*.

Como exemplo, pode ser observado na linha 8 da código apresentado na Figura 3 o comando base para buscar uma determinada configuração do arquivo.

Figura 3 – Exemplo - Uso do Configuration

```
1  from pybot import Manager, getConfig
2  from Pages.ads import adsPage
3
4
5  manager = Manager()
6  page = adsPage(manager)
7
8  retingaUrl = getConfig('Restinga', 'url')
9
10 manager.go(retingaUrl)
11 page.goToDisciplinas()
12
13 manager.end()
14
```

O padrão de escrita das configurações pode ser visto na Figura 4, onde é necessário ter uma Seção e as variáveis desejadas. E para a utilização delas segue o mesmo padrão: `configuration.getConfig('Seção', 'variável')`.

Figura 4 – Estrutura pybot.ini

```

1
2  [Seção]
3      variavel1 = 'valor'
4      variavel2 = 1
5      variavel3 = True
6

```

Para o funcionamento correto dos *scripts* o *Pybot* necessita de duas configurações básicas dentro desse arquivo, que são o navegador utilizado na execução e o endereço no computador do *driver* desse navegador. Ambas configurações caso não existam são criadas automaticamente.

3.2 COMPONENT

Módulo criado para seguir os padrões de *PageObject* e *PageElement*, contendo as classes de *PageObject* e *PageElement* e a classe responsável pela abstração para o *WebElement* do *Selenium WebDriver*.

Pode-se observar na Figura 5 o uso das classes *PageObject*, *PageElement* e *PageElements* nas linhas 3, 4 e 5 respectivamente.

Figura 5 – Exemplo - Uso PageObject com PageElement e PageElements

```

1  from pybot import Manager, PageElement, PageElements, PageObject
2
3  class localPage(PageObject):
4      input1... = PageElement(id="b")
5      input2    = PageElements(name="a")
6      dropdown  = PageElement(tag_name='select')
7

```

3.2.1 PageObject

Classe de abstração das páginas web. Serve para poder juntar diversos *PageElement* descritos pelo usuário em uma classe para melhor legibilidade e componentização das páginas mapeadas e disponibilizar para os *PageElement* acesso ao navegador.

3.2.2 WebElement

Serve para abstrair o uso da classe *WebElement* do próprio *Selenium WebDriver*. Contendo uma classe para cada tipo de campo dos *html*, ele dispõe de algumas funcionalidades básica, como a atribuição de uma valor para um elemento do tipo *input text* irá escrever valor dentro do campo, *select* irá selecionar a opção cujo texto seja igual ao valor informado, *radio* irá selecionar o a opção que tenha o *value* do valor informado e para o tipo *checkbox* irá marcar ou desmarcar as opções se o valor for verdadeiro(*True*) ou falso(*False*)

3.2.3 PageElement e PageElements

Essas classes servem para controlar os elementos mapeados das telas. Sempre quando serão acessadas a classe faz novamente a pesquisa do elemento em tela, prevenindo assim uma das exceções mais comum do Selenium WebDriver que é a *StaleElementReferenceException*, que é quando o elemento em questão não existe mais no DOM ou a referência que tinha não é mais a mesma. Conta com uma lista de seletores que facilitam para o usuário buscar os elementos e deixam o código mais legível, os mesmos podem ser observados na Figura 6 dentro de cada parênteses das definições dos *PageElement*. Usa-se a classe *PageElements* quando quiser pegar mais de um elemento com o mesmo seletor.

Figura 6 – Lista de Seletores

```

1 class TestPage(PageObject):
2     e01 = PageElement(css='#rso > div:nth-child(1) > div > div > div > h3 > a')
3     e02 = PageElement(id='id123')
4     e03 = PageElement(name='name1')
5     e04 = PageElement(xpath='//*[@id="palavras"]')
6     e05 = PageElement(link_text='Texto')
7     e06 = PageElement(partial_link_text='IFRS-Re')
8     e07 = PageElement(tag_name='input')
9     e08 = PageElement(class_name='class01')

```

3.3 REPORT

Estes módulo é destinado para geração de *logs* das execuções internas do *framework*, a criação e controle de *logs* de execuções de texto e no terminal definidos pelos usuário e a criação de planilhas analíticas de dados extraídos das páginas.

3.3.1 Logger

Classe de geração dos *Logs* de execução do *framework*, onde cada execução, ação, sucessos e falhas podem ser configurados para que no decorrer do processo sejam mostrados no terminal de execução e no final salvos em um arquivo de texto.

Os *logs* podem ser configurados com níveis que podem ser *CRITICAL*, *ERROR*, *WARNING*, *INFO* e *DEBUG*, e no arquivo de configurações é possível definir qual o nível desejado para que os *logs* apareçam. Numa escala de 1 a 5, *CRITICAL* tem valor 5 e *DEBUG* valor 1, dessa maneira se o nível do *log* for definido como *DEBUG* apenas ele aparecerá e se for definido como *CRITICAL* todos apareceram.

Na Figura 7 podemos verificar 2 tipos de escritas do *log*, onde na linha 11 temos a escrita de um *log* e na linha 21 é o tratamento de erros de execução, onde este para a execução do código ao ser chamado.

Figura 7 – Exemplo - Uso do Logger

```

1  from pybot import PageObject, PageElement, write_log, handle_error
2
3
4  class restingaPage(PageObject):
5
6      textPesq = PageElement(id='palavras')
7      buttonPesq = PageElement(css='.ok')
8      linksPesq = PageElement(link_text='Superior')
9
10     def preencherPesquisa(self, texto):
11         write_log('Escrevendo {} na pesquisa'.format(texto))
12         self.textPesq = texto
13
14     def clicarPesquisa(self):
15         self.buttonPesq.click()
16
17     def clicarLink(self):
18         try:
19             self.linksPesq.click()
20         except Exception as e:
21             handle_error('Não deu pra clicar', e)
22

```

3.3.2 CsvHandler

Utilizado para geração de planilhas com dados extraídos das páginas para análise posterior dos usuários. Os dados tem que ser extraídos manualmente pelo usuário, e uma vez extraídos são salvos automaticamente em um arquivo com a extensão *csv*.

Como exemplo, podemos observar na Figura 8 um exemplo de utilização do *CsvHandler*, onde, na linha 5 será criado uma planilha contendo uma coluna Disciplina e na linha 21 será inserido registros na planilha abaixo da coluna.

Figura 8 – Exemplo - Uso do CsvHandler

```

1  from pybot import PageObject, PageElement, PageElements
2  from pybot.report import csvHandler
3  from pybot.report.logger import write_log
4
5  csvHandler.setup(header=['Disciplina'])
6
7  class adsPage(PageObject):
8      disciplinasLink = PageElement(link_text='Disciplinas')
9      semestres = PageElements(xpath='//*[@id="post-89"]/div/table/tbody/tr/td[1]')
10     dis = PageElements(xpath='//*[@id="post-89"]/div/table/tbody/tr/td[2]/a')
11
12     def goToDisciplinas(self):
13         self.disciplinasLink.click()
14
15     def get_semestres(self):
16         self.num = len(self.semestres)
17         write_log('Numero de semestres: {}'.format(self.num))
18
19     def get_semestre_disciplina(self, num):
20         for e in self.dis:
21             csvHandler.write({'Disciplina': e.element.text})
22

```

3.4 DRILOADER

Driloader é o responsável pelo download dos *drivers* de cada navegador, suportando *download* dos *drivers* do *Internet Explorer*, *Firefox* e *Chrome*, sendo possível para o usuário selecionar uma versão específica, a ultima versão ou detectar automaticamente qual a versão adequada para o navegador instalado do usuário. Como para utilização do *Selenium Webdriver* é necessário um *driver* específico de cada navegador foi tomada a decisão da criação desse projeto que inicialmente era um módulo do *framework* mas pela autonomia e praticidade que ele proporciona aos usuários do Selenium Webdriver foi feita a separação dele do *Pybot*. Seu uso é feito na classe Manager(SUBSEÇÃO 3.1.1) automaticamente quando nenhuma *driver* é encontrado no computador do usuário.

4 TECNOLOGIAS EMPREGADAS

Para o desenvolvimento do *framework* foi utilizado apenas como linguagem para desenvolvimento o *Python* na versão 3.6 e para a manipulação e integração com o navegador a biblioteca em *Python* do *Selenium Webdriver* também na versão 3.6. Por ser um projeto que visa ser o mais simples e leve suas dependências são mínimas, fazendo uso apenas dos módulos padrões do *Python* e do *Selenium Webdriver* para o desenvolvimento desta ferramenta.

4.1 PYTHON

Foi escolhido o *Python* (PYTHON, 2017) por que trata-se de uma linguagem de programação fácil de aprender e poderosa. Possuindo uma estruturas dados de alto nível e uma abordagem simples, mas eficaz, para a programação orientada a objetos. Contendo uma Sintaxe elegante e tipagem dinâmica, juntamente com uma interpretação natural, tornam a linguagem ideal para criação dos *scripts* do Pybot.

4.2 SELENIUM WEBDRIVER

Selenium Webdriver (WEBDRIVER, 2017) é um *framework* utilizado para se comunicar e enviar comandos para os navegadores em conjunto com um controlador específico de cada navegador. Em comparação com seu antecessor, *Selenium RC*, o *Selenium Webdriver* não precisa de um *server* para enviar os comandos para o navegador, sendo preciso apenas a utilização desse controlador. Utilizando comando nativos do sistema operacional ao invés de comando *javascript*, usados pelo *Selenium RC*, deixam o *Selenium Webdriver* uma excelente ferramenta para integração com diversos navegadores.

4.3 GIT E GITHUB

Para o controle de versões e alterações do código fonte do *framework* e *scripts* de exemplo foi utilizado a ferramenta *Git* (GIT, 2017) em conjunto com os servidores do *Github* (GITHUB, 2017) para hospedagem e gerenciamento. Com eles foi possível fazer alterações dos códigos fontes em qualquer computador e gerenciar os erros e melhorias do *framework*.

5 DESENVOLVIMENTO PROPOSTO

O *framework* proposto é baseado no conceito de *PageObject*, onde todas as páginas web são tratadas como objetos. Ainda, cada componente que seja necessário para automação, um *input*, *span*, etc, é um atributo desse objeto.

Para melhor exemplificar o uso do conceito de *PageObject* será utilizado como exemplo um simples *login* para 2 usuários, onde será utilizada uma página que possui dois campos de texto, campo de usuário e outro de senha, e um botão para submeter o *login*.

Primeiramente se utilizou de um exemplo básico de como o *Selenium* propõe o desenvolvimento desse *script* de *login*. No começo é iniciado o *driver* do navegador, navegar para a *URL*, depois são seguidos 3 passos para cada um dos campos de texto, procurar por ele, limpar o seu conteúdo (porque não se sabe se ele possui algum texto pré cadastrado) e enviar os caracteres necessário para cada campo e por final, procurar e clicar no botão para submeter a ação do formulário. Não é um método muito viável, pois caso tenhamos 2 *logins* esse *script* deverá ser duplicado para atender ambas necessidades.

Num segundo exemplo poderíamos separar o *script* de *login* e criar um módulo separado para ele. Desse jeito os *scripts* podem fazer uso do mesmo código e caso uma terceira pessoa precise dele não seria um problema. Porém temos todo o mapeamento dessa página está preso num módulo e caso seja necessário a criação de outro módulo que use essa mesma página ainda assim teremos que duplicar mais código.

Chegando num terceiro exemplo onde agora fazemos uso do *framework Pybot*, onde utilizando-se do módulo *Component* (SEÇÃO 3.2) podemos separar todos os componentes da tela em atributos da nossa página e criar um método onde precisando de dois parâmetros ele faz o processo de *login*, e ainda assim, caso necessário pode-se utilizar os campos mapeados para fazer algum outro método sem impactar o *login*. E caso alguma referencia dos campos mapeados mude, será necessário alterar apenas um local e nenhum *script* será impactado.

Para exemplificar esses cenários mostrados foram feitos alguns diagramas que estão presentes no Apêndice B, onde: Na Figura 16 exemplifica o primeiro cenário mostrado, onde temos dois usuários com *scripts* de *login* idênticos, porém cada com a sua implementação; Na Figura 17 exemplifica o segundo cenário mostrado, onde agora temos um *script* de *Login* separado dos *scripts* de cada usuário, porem ainda temos toda o mapeamento da pagina preso num único script; Na Figura 18 exemplifica o terceiro cenário mostrado, onde com o uso do *Pybot* a página de *login* é tratada como um objeto e dentro dela temos o método *login* para ser

usado nos *scripts* de cada usuário.

Tendo isso em mente é possível entender uma das vantagens do uso do *PageObject* para criação dos *scripts* que o *Pybot* faz uso, e com isso analisar como fica a implementação desse *script* dos exemplo fazendo uso apenas do *Selenium*, primeiro cenário mostrado, e outro usando o *Pybot*, terceiro cenário mostrado.

A Figura 9 mostra o uso do *Selenium* onde temos toda a execução e os comandos do *script* em apenas um arquivo. Seus comando são simples e bem verbosos, tornando assim o *script* de fácil entendimento das ações que serão executadas. No início do código é feita a importação do *framework*, seguido da iniciação do navegador *Firefox*(caso o *driver* esteja nas variáveis de ambiente da máquina da execução), seguido da pesquisa, limpeza e preenchimento dos valores para os campos de texto, seguido da pesquisa e *click* no botão e por fim o encerramento do *script*. Ele resolve o cenário dos usuários descrito anteriormente mas não dá assim a flexibilidade da criação de outros *scripts* com o aproveitamento de código.

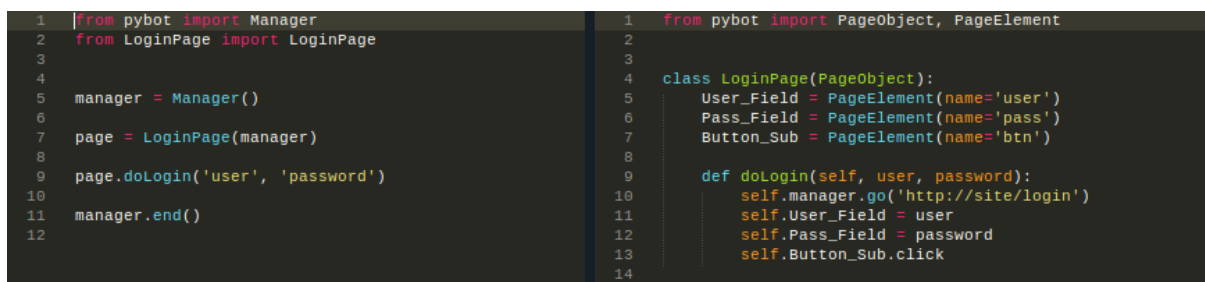
Figura 9 – Exemplo de uso com Selenium

```
1 | from selenium import webdriver
2
3 | driver = webdriver.Firefox()
4 | driver.get('http://site/login')
5
6 | User_Field = driver.find_element_by_name('user')
7 | User_Field.clear()
8 | User_Field.send_keys('user')
9
10 | Pass_Field = driver.find_element_by_name('pass')
11 | Pass_Field.clear()
12 | Pass_Field.send_keys('password')
13
14 | Button_Sub = driver.find_element_by_name('btn')
15 | Button_Sub.click()
16
17 | driver.close()
18
```

Agora na Figura 10 temos um código fazendo uso do *Pybot*. Com ele foi feita a separação do código da página de *login* em outro arquivo, e com o uso da classe *PageObject* para a criação da nossa página e dentro dela utilizado a classe de *PageElement* para o mapeamento dos elementos da página, dessa maneira o código fica mais organizado e agora temos um contexto sobre onde cada elemento definido pertence. Junto da classe de *Login* foi feita a definição de um método de login que faz o uso dos *PageElements* para a atribuição dos valores nos campos de texto seguido do *click* do botão. Utilizando o *Pybot* o código responsável pelo login fica

totalmente independente dos *scripts*, e os mesmo se tornam muito mais simples e diretos para mostrar o seu propósito. Com isso o script ficou muito menor e simples sendo necessário apenas fazer a importação do próprio *Pybot* para dar inicio ao *script* e da página de login para utilizar-se do método.

Figura 10 – Exemplo de uso com Pybot



```
1 from pybot import Manager
2 from LoginPage import LoginPage
3
4
5 manager = Manager()
6
7 page = LoginPage(manager)
8
9 page.doLogin('user', 'password')
10
11 manager.end()
12
```

```
1 from pybot import PageObject, PageElement
2
3
4 class LoginPage(PageObject):
5     User_Field = PageElement(name='user')
6     Pass_Field = PageElement(name='pass')
7     Button_Sub = PageElement(name='btn')
8
9     def doLogin(self, user, password):
10         self.manager.go('http://site/login')
11         self.User_Field = user
12         self.Pass_Field = password
13         self.Button_Sub.click
14
```

6 AVALIAÇÃO DA SOLUÇÃO

Visando validar se o *framework* proposto atende as necessidades dos usuários foi feito uma pesquisa de uso do *framework* com algumas pessoas da área da Programação. Foram executados alguns casos testes e, através da plataforma *Google Forms*, foram aplicados 5 perguntas sobre a experiência dos usuários com o *framework* em relação ao *Selenium Webdriver*. Num total de 7 usuários foram selecionados para criar e executar esses casos de testes e automatiza-los utilizando o *Pybot*, todos os participantes tinham conhecimentos entre básicos e intermediário em *Selenium Webdriver* e os cenários foram executados em computadores com *linux* com o *Python* instalado.

Os casos de testes foram baseados na aplicação web desenvolvida pela empresa em que trabalham e os mesmos eram: criação de *script* de *login*, criação de *script* de cadastro e a criação de *script* para extração de dados. Para melhor validação do *scripts* foi pedido para que nos testes fossem utilizados mais de um tipo de seletor para mapeamentos dos campos, iteração com *iframes* e múltiplas janelas, geração dos logs de execução e execuções em mais de um browser.

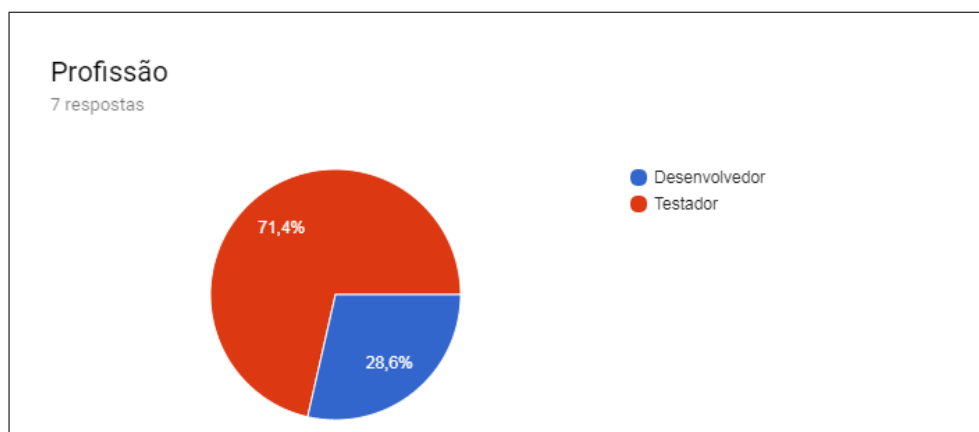
Com o desenvolvimento e finalização dos *scripts* foram levantados os seguintes pontos sobre a utilização do framework: Profissão do Usuário, Facilidade de instalação, Facilidade de Uso, Atende necessidades básicas, Atende necessidades avançadas.

A seguir, serão mostrados e analisado os dados coletados em cada um dos pontos do questionário, este que pode ser encontrado no apêndices deste trabalho.

6.1 PROFISSÃO DO USUÁRIO

Como trata-se de um *framework* de automação de testes foi solicitado um numero maior de profissionais da área de *Testes de Software*. Como exemplo, pode ser observado na Figura 11, mais de 70% dos usuários são Testadores de Software, enquanto menos de 30% são Desenvolvedores.

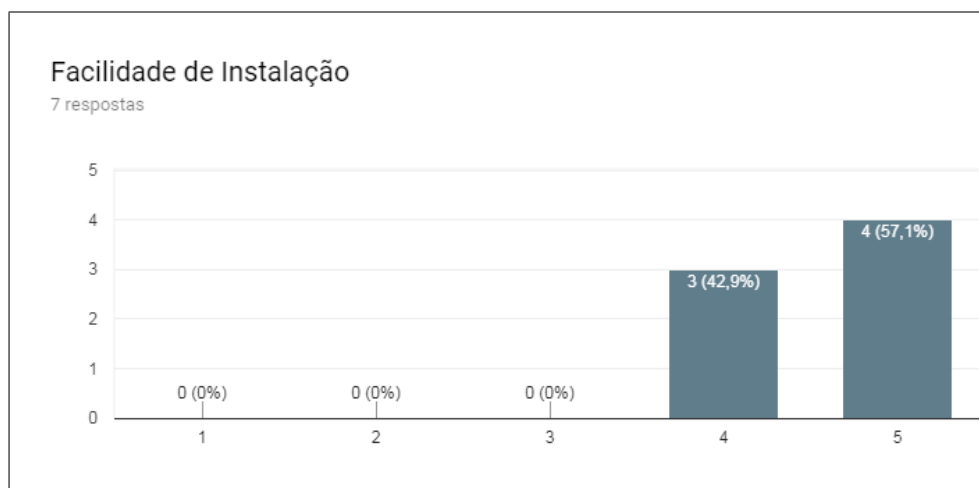
Figura 11 – Profissões



6.2 FACILIDADE DE INSTALAÇÃO

Dos usuários na grande maioria consideraram fácil a instalação, como pode ser observado na Figura 12. O único ponto a melhorar foi que o projeto hoje está disponível apenas pelo repositório do *Github* e não no *PIP*, repositório padrão do *Python*.

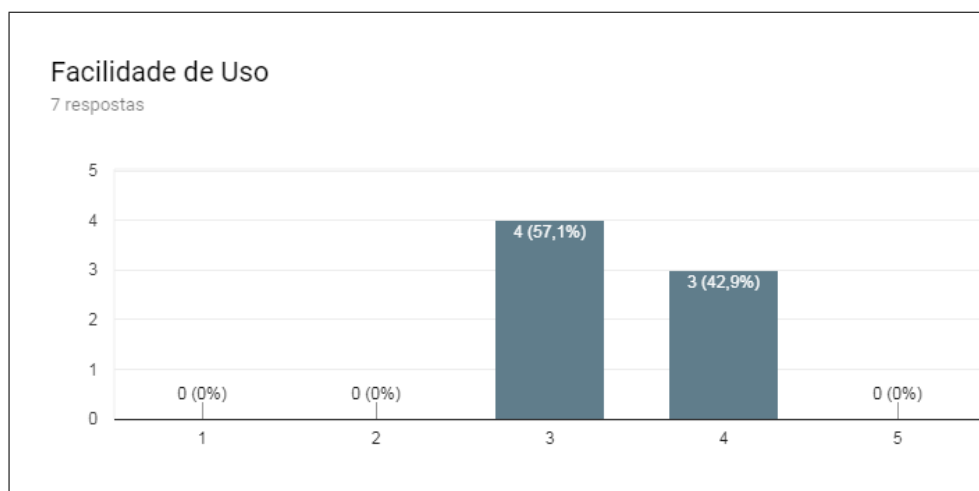
Figura 12 – Facilidade de instalação



6.3 FACILIDADE DE USABILIDADE

Em relação a facilidade de uso a grande maioria não teve muitas dificuldades, onde mais de 50% considerou a usabilidade mediana e o restante de fácil uso, como podemos observar na Figura 13. Onde o maior problema levantado a falta de documentação para verificar todos os comandos de cada classe.

Figura 13 – Facilidade de Uso

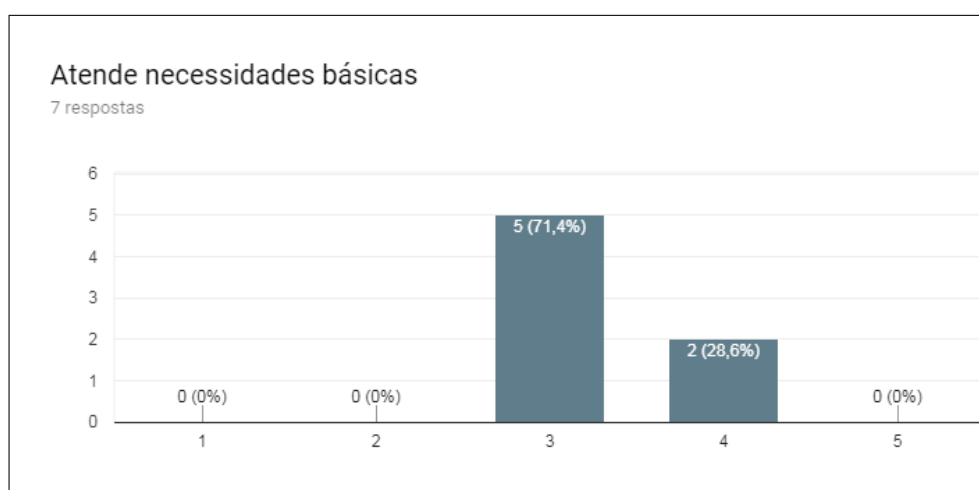


6.4 ATENDE AS NECESSIDADES BÁSICAS

As funcionalidades disponíveis do *framework* foram, para a maioria, suficientes para atender as necessidades básicas de criação de um processo de testes. Como para a Facilidade de Uso, a documentação também foi um ponto negativo levantado.

Como pode ser visto na Figura 14, nenhum usuário classificou com nota 5 que representa que o *framework* não atende completamente todas as necessidades básicas dos usuários.

Figura 14 – Atende necessidades básicas



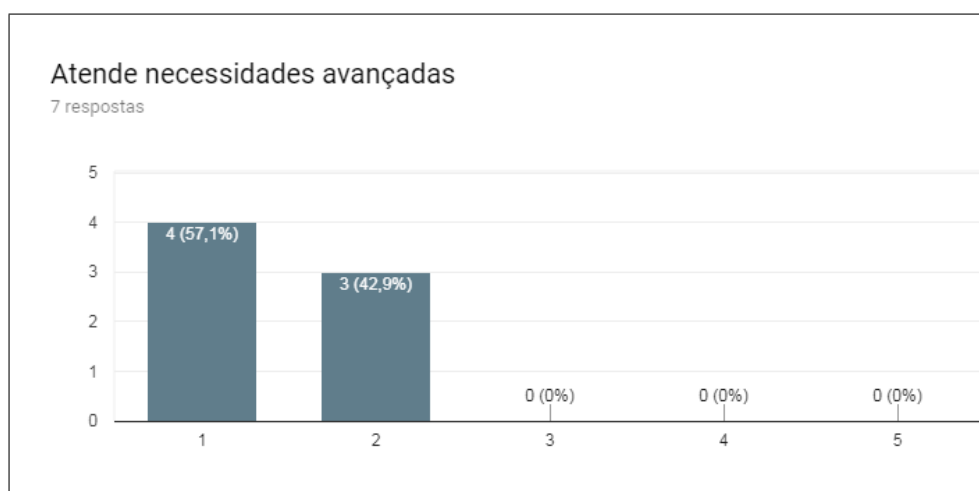
6.5 ATENDE AS NECESSIDADES AVANÇADAS

O *framework* não atingiu as necessidades avançadas. Os usuários tiveram diversos problemas para utilizar múltiplas janelas e elementos renderizados em processos assíncronos do

Javascripts, sendo necessário fazer uso do *Selenium* padrão contido no *framework* para realizar tais tarefas.

Como pode ser visto na Figura 15, todos os usuários consideraram que o *Pybot* não atendeu necessidades avançadas.

Figura 15 – Atende necessidades avançadas



7 CONCLUSÃO

Este trabalho apresentou um *framework* para automatização de testes e processos em navegadores utilizando *Python* e o *framework Selenium Webdriver* como base, sendo assim possível a criação de *scripts* de testes automatizado em qualquer tipo de ambiente sem a necessidade de configurações e instalações complexas, contando com ferramentas que facilitam e agilizam o processo de criação desses *scripts* com padrões de *Page Object*.

Com base nos resultados obtidos pelo questionário aplicado aos usuários no Capítulo 6 pode-se observar que o *framework* tem potencial para auxiliar os desenvolvedores a criar processos de testes com uma certa facilidade em relação aos *frameworks* existentes atualmente, conforme o objetivo do projeto.

Por fim, com base no desenvolvimento e andamento do projeto e juntos das análises de dados foi notado algumas melhorias e trabalhos futuros faltaram para tornar o *Pybot* numa ferramenta mais completa, como a criação de uma documentação completa dos módulos, controle de processos assíncronos dos *Javascripts*, gerenciamento de múltiplas janelas e abas e a hospedagem do *framework* no repositório padrão do *Python*.

REFERÊNCIAS

GIT. **Git**. 2017. Disponível em: <<https://www.git-scm.com>>.

GITHUB. **Github**. 2017. Disponível em: <<http://www.github.com>>.

PYTHON. **The Python Tutorial**. 2017. Disponível em: <<https://docs.python.org/3.6/tutorial/>>.

ROBOTFRAMEWORK. **robotframework**. 2017. Disponível em: <<http://robotframework.org>>.

SANTOS, M. d. O. dos. Um estudo sobre a influência das técnicas de testes automatizados no desenvolvimento de software. Universidade Federal do Amazonas, 2016.

SELENIUM. **Selenium**. 2017. Disponível em: <<http://www.seleniumhq.org/>>.

WEBDRIVER, S. **Selenium Webdriver**. 2017. Disponível em: <http://www.seleniumhq.org/docs/03_webdriver.jsp>.

APÊNDICE A – Pesquisa de satisfação do Pybot

Profissão: () Desenvolvedor () Testador

Facilidade de Instalação: 1 ○ 2 ○ 3 ○ 4 ○ 5 ○

Facilidade de Uso: 1 ○ 2 ○ 3 ○ 4 ○ 5 ○

Atende necessidades básicas: 1 ○ 2 ○ 3 ○ 4 ○ 5 ○

Atende necessidades avançadas 1 ○ 2 ○ 3 ○ 4 ○ 5 ○

Comentarios: _____

APÊNDICE B – Diagramas de Cenários

Figura 16 – Uso padrão Selenium

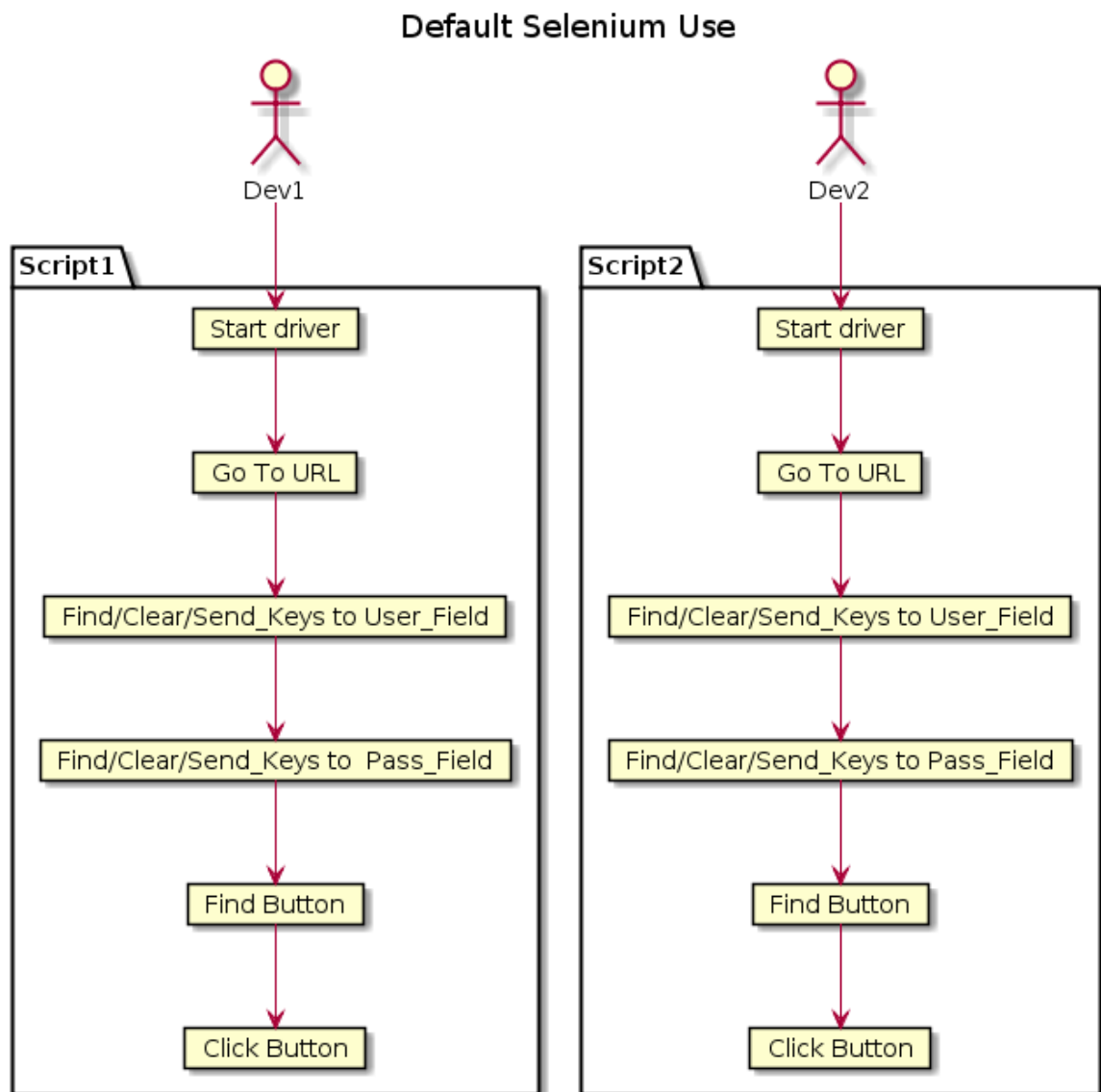


Figura 17 – Uso padrão Selenium com módulo
With Login Module

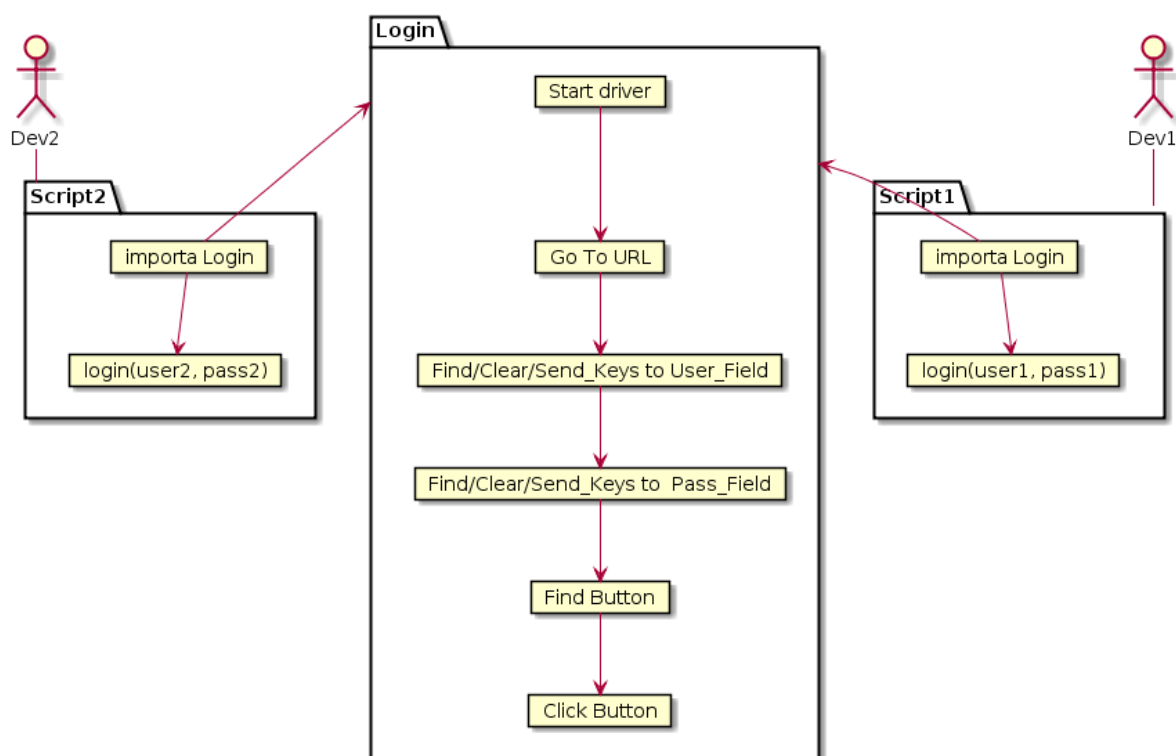


Figura 18 – Uso padrão Pybot

With Pybot Modules

