МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Кафедра вычислительной математики

Курсовая работа

КРОССБРАУЗЕРНЫЕ ИНСТРУМЕНТЫ АВТОМАТИЗАЦИИ ТЕСТИРОВАНИЯ WEB-ПРИЛОЖЕНИЙ

Жихаревич Александра Андреевна

Студентка 3 курса 13 группы,
специальность «Прикладная информатика»

Научный руководитель:
доцент П. А. Мандрик

Куратор-эксперт:
Е. А. Лысов

Минск, 2019

Задание курсовой работы:

1. Обзор основный техник тестирования web-приложений.
2. Разработка стратегии внедрения автоматизации тестирования на примере проекта web-приложения.
3. Разработка фреймворка для автоматизации web-приложения.
4. Подготовка отчета и презентации по курсовой работе.

СОДЕРЖАНИЕ

Введение

Автоматизированное тестирование web-приложений – часть процесса тестирования на этапе контроля качества в процессе разработки web-приложения. В связи с большим ростом программных продуктов и веб-приложений, внедрение автоматизации тестирования web-приложений является необходимым.

Плюсы автоматизации тестирования:

+ Экономия времени - автоматизация не требует вмешательства тестировщика, в это время он может переключиться на другие задачи;
+ Оперативность – автоматизированный скрипт не сверяется с инструкциями и документацией;
+ Повторное использование – сценарий тестирования может использоваться неоднократно;
+ Отсутствие «человеческого фактора» – тестовый сценарий не допустит оплошностей в результатах и не пропустит времени тестирования;
+ Автоматическая отчетность – результаты тестирования автоматически сохраняются и рассылаются причастным специалистам.

Минусы автоматизации тестирования:

- Затраты – хорошие инструменты автоматизированного тестирования, как и обучение автоматизированному тестированию ПО требует вложений;
- Однообразие – написанные тесты работают всегда одинаково, что не всегда плохо, но иногда позволяет пропустить дефект, который заметил бы живой человек;
- Затраты на поддержку и разработку – чем сложнее приложение и чем чаще оно обновляется, тем более затратна разработка и модификация автоматизированных тестов;
- Пропуск мелких недочетов – тесты пропускают небольшие ошибки, на проверку которых не запрограммированы.

На сегодняшний день существует большое количество фреймворков автоматизации тестирования. Для упрощения процесса автоматизации важно грамотно выбрать инструмент для тестирования.

Инструменты автоматизации тестирования

Инструменты автоматизации играют важную роль в тестировании. Примерами инструментов для тестирования непосредственно web-приложений могут послужить Selenium, Katalon Studio, UFT (Unified Function Testing), TestComplete, Watir. По различным параметрам эти инструменты схожи или отличаются (см. Приложение 1). Опираясь на критерии выбора инструмента, такие как, например, простота разработки и исполнение скриптов, соответствие платформы языку разработки теста, поддержка новейших функций операционных систем, полностью интегрированные в среду IDE инструменты и другие, важно выбрать подходящий инструмент или инструменты для автоматизации.

Selenium

Наиболее популярным является набор инструментов для автоматизации web-браузера Selenium. В нем используются лучшие методы, доступные для удаленного управления экземплярами браузера и эмуляции взаимодействия пользователя с браузером. Основным принципом проекта является поддержка одного интерфейса для всех браузерных технологий. Веб-браузеры - это невероятно сложные, высокотехнологичные приложения, которые выполняют свои операции совершенно по-разному, но при этом часто выглядят одинаково. Selenium скрывает эти различия, их детали и тонкости от человека, пишущего код. Это позволяет использовать написанный код для выполнения во всех поддерживаемых браузерах.

Как говорилось ранее, Selenium – набор инструментов. В рамках проекта Selenium разрабатываются следующие программные продукты с открытым кодом:

- Selenium WebDriver
- Selenium RC
- Selenium Grid
- Selenium IDE

Selenium RC

Selenium RC (Selenium Remote Control) – проект Selenium, который долгое время был основным проектом, до тех пор, пока не произошло слияние WebDriver и Selenium и не привело к появлению Selenium 2.0. Selenium RC называют Selenium 1.0.

Проект включает в себя следующие компоненты:

- Selenium RC Server получает команды Selenium тестовой программы, интерпретирует их и сообщает программе результаты выполнения этих тестов.

- Клиентские библиотеки, которые обеспечивают интерфейс между каждым языком программирования и Selenium RC Server

Клиентские библиотеки взаимодействуют с сервером, передавая каждую команду Selenium для выполнения (см. Приложение 2). Затем сервер передает команду Selenium в браузер, используя команды Selenium-Core JavaScript. Браузер, используя свой интерпретатор JavaScript, выполняет команду Selenium. При этом выполняется действие Selenese или проверка, указанная в тестовом скрипте.

Selenium Grid

Selenium Grid – это кластер, состоящий из нескольких Selenium-серверов. Логика Selenium Grid заключается в том, что один сервер является концентратором, отправляющий команды тестирования на другие сервера, называющиеся узлами, зарегистрированные в Grid. Одна из задач Selenium Grid заключается в том, чтобы «подбирать» подходящий узел, когда во время старта браузера указываются требования к нему – тип браузера, версия, операционная система, архитектура процессора и ряд других атрибутов.

Selenium Grid позволяет запускать тесты параллельно на нескольких машинах и централизованно управлять различными версиями браузера и конфигурациями браузера (вместо каждого отдельного теста).

Grid условно можно поделить на 2 компоненты (см. Приложение 3):

- HUB (концентратор)
- NODES (узлы)

Задачи и функции концентратора:

- Посредник и менеджер
- Принимать запросы на запуск тестов
- Принимать инструкции от клиента и выполнять их удаленно на узлах
- Управлять потоками

Задачи и функции узлов:

- Место расположения браузеров
- Регистрируются в концентраторе и передают свои возможности
- Получают запросы от концентратора и выполняют их

Selenium IDE

Selenium IDE – расширение к браузеру Firefox. Проект помогает разрабатывать тестовые сценарии веб-страниц. Он записывает определенный сценарий поведения пользователя на сайте, а потом воспроизводит записанные действия в автоматическом режиме. Selenium IDE также обладает более 500-ми встроенными командами, которые позволяют

зафиксировать практически любые действия на сайте и воспроизвести их вновь или осуществить проверку какого-то элемента.

Selenium WebDriver

Selenium WebDriver (Selenium 2.0) - инструмент для автоматизации реального браузера, как локально, так и удалённо, наиболее близко имитирующий действия пользователя. WebDriver – последнее пополнение Selenium, являющееся основным вектором развития проекта.

До WebDriver, или Selenium 2.0, основным проектом был Selenium RC, или Selenium 1.0, однако WebDriver полностью затмил предыдущую версию (Selenium 1.0 больше не поддерживается). Такой успех объясняется тем, что WebDriver использует абсолютно иной способ взаимодействия с браузерами. Он напрямую вызывает команды браузера, используя для каждого конкретного браузера родной для него API (Application Programming Interface). В сравнении с Selenium RC WebDriver не внедряет в браузер JavaScript для управления браузером, а использует способ взаимодействия наиболее близкий к действиям реального пользователя.

Подробнее о Selenium WebDriver

Для работы с WebDriver необходимы следующие компоненты:

1. Браузер. Так как Selenium WebDriver это инструмент для автоматизации браузера, тот браузер является необходимым компонентом для работы с WebDriver.
2. Драйвер браузера. WebDriver предоставляет возможности для работы с многими браузерами, поэтому для работы с определенным браузером необходимо указать драйвер конкретного браузера. У каждого браузера существует свой собственный драйвер, так как у каждого браузера свои отличительные команды управления и реализации их отличны. Драйвер сам по себе является web-сервисом, открывающим браузер, отправляющий ему команды, и закрывающий его по окончании работы.
3. Тест. Тестом является код, написанный на определенном языке, поддерживаемым проектом Selenium, предоставляющий команды для драйвера браузера.

Основными понятиями Selenium WebDriver являются WebDriver, WebElement, By.

WebDriver

WebDriver является самой главной сущностью, ответственной за управлением браузера. Простым языком, WebDriver – драйвер, предоставляющийся для управления определенным браузером.

Создание WebDriver для управления браузера Chrome:
WebDriver driver = new ChromeDriver();

Класс WebDriver предоставляет ряд методов, для управления браузером:

- get (String url) – переход по указанной ссылке в текущем окне браузера;
- getCurrentUrl () – получение URL текущей страницы браузера;
- getTitle () – получение заголовка текущей страницы браузера;
- findElement (By by) – поиск первого элемента по заданному локатору;
- findElements (By by) – поиск всех элементов по заданному локатору;
- close () – закрытие текущего окна браузера, если окон несколько; завершение работы браузера, если текущее окно единственное;
- quit () – завершение работы браузера, закрытие всех окон браузера;
- и другие.

WebElement

WebElement представляет абстракцию над web-элементом. Класс WebElement предоставляет методы для работы непосредственно с элементом web-страницы.

Получение веб-элемента страницы браузера происходит через метод драйвера findElement() или findElements():
WebElement element = driver.findElement(By.xpath(“//div/a”));

После получения веб-элемента, element является объектом класса WebElement, предоставляющий работу непосредственно с элементами веб-страницы. Выделяют различные типы веб-элементов такие так поля для редактирования, ссылки, кнопки, изображения / изображения-ссылки / изображения-кнопки, текстовые поля, чек-бокс, радиокнопки, выпадающие списки. В зависимости от типа полученного элементы, доступны методы для работы с этим элементом.

Методы, предоставляемые классом WebElement:

- click () – нажатие на элемент;
- getAttribute (String attribute_name) – получение значения атрибута веб-элемента по имени;
- getText () – получение текста элемента;
- sendKeys (String key) – имитирует ввод данных пользователем;
- и другие.

Бывает, что методов, предоставляемых классом WebElement не хватает, однако Selenium WebDriver предоставляет возможность более сложного взаимодействия с элементами за счет использования методов класса Actions. Помимо использования какого-то метода, класс позволяет составлять цепочки методов.

Методы класса Actions:

- click () / click (WebElement element) – нажатие по текущему расположению курсора / на элемент
- clickAndHold () / clickAndHold (WebElement element) – нажатие и удержание по текущему расположению курсора / на элемент
- contextClick () / contextClick (WebElement element) – нажатие правой кнопкой мыши по текущему расположению курсора / на элемент
- doubleClick () / doubleClick (WebElement element) – двойное нажатие по текущему расположению курсора / на элемент
- dragAndDrop (WebElement source, WebElement target) – нажатие и удержание на элемент и перемещение его на заданное смещение, после чего отпускание мыши.
- keyUp (CharSequence key) / keyDown (CharSequence key) – отпускание / нажатие указанной клавиши на клавиатуре.
- и другие.

Для построения цепочки действий используется метод build (). Для выполнения построенной цепочки действий используется метод perform ().

By. Локаторы

Класс By представляет абстракцию над локатором web-элемента и содержащит статические методы для идентификации этого элемента. WebElements - ни что иное, как объекты, находящиеся на веб странице. А для того, чтобы осуществлять какие-то действия над этими объектами (web-элементами) необходимо их точным образом определить (найти). Как говорилось ранее, поиск элемента осуществляется драйвером при помощи методов findElement (By by) / findElements (By by), куда передается объект класса By, содержащий методы для идентификации элементов.

Находить web-элементы можно различными способами: по имени класса, по ID, по пути к этому элементу и др. В зависимости от способа нахождения элемента, выделяют разные типы локаторов и соответственно разные методы, предоставляемые классом By для нахождения элемента по локатору. Локатором называют строку, уникально идентифицирующую элемент html-страницы.

Типы локаторов:

- Class name (имя класса)
- ID
- Link text (текст ссылки)
- Name (имя)
- Partial link text (частичный текст ссылки)
- Tag name (имя тэга)
- XPath (путь, по которому находится элемент)

Соответственно методы класса By для нахождения элемента по определенному локатору:

- By.className (String className) – по имени класса
- By.id (String id) – по ID
- By.linkText (String linkText) – по тексту ссылки
- By.name (String name) – по имени
- By.partialLinkText (String partialLinkText) – по частичному тексту ссылки
- By.tagName (Sting tagName) – по имени тэга
- By.xpath (String xpath) – по пути
- By.cssSelector (String selector) – по css селектору

Ожидания

Скрипт (код) выполняется намного быстрее реакции приложения на команды, поэтому часто в скриптах необходимо дожидаться определенного состояния приложения для дальнейшего с ним взаимодействия. Например, нужно открыть страницу и нажать на кнопку. Бывает, что при загрузке страницы, элементы отображаются не все сразу, а тест уже пытается нажать на элемент, который ещё не отобразился. Результатом будет ошибка, что

элемент не может быть найден. Именно для этого необходимо воспользоваться ожиданиями.

Типы ожиданий:

- Неявные ожидания
- Явные ожидания
- Свободные ожидания

Неявные ожидания, или Implicit Waits, обычно устанавливаются после создания драйвера и существуют до тех пор, пока существует драйвер. Неявные ожидания указывают драйверу на то, что нужно выполнять поиск элемента в течении указанного времени до тех пор, пока элемент не будет найдет. Если по истечении времени элемент не найдет, будет выброшено исключение ElementNotFoundException, сигнализирующее о том, что элемент не найден.

Неявные ожидания объявляются следующим образом:

driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS),

где driver – экземпляр WebDriver, 10 – число, указывающее, сколько нужно искать элемент, TimeUnit.SECONDS – единицы измерения времени.

В данном примере драйвер будет выполнять поиск каждого элемента в течении 10 секунд до тех пор, пока элемент не будет найден. После чего выдаст исключение о том, что элемент не найден.

К неявным ожиданиям можно так же отнести ожидание загрузки страницы (pageLoadTimeout (10, TimeUnit.SECONDS)), ожидающее в течении указанного времени загрузки страницы, и ожидания асинхронного сценария для завершения выполнения (setScriptTimeout (10, TimeUnit.SECONDS)).

Явные ожидания, или Explicit Waits, это код, которым определяется, какое необходимое условие должно произойти для того, чтобы дальнейший код исполнился. Явное ожидание срабатывает один раз в указанном месте. Под “условием” понимается не только появление элемента на web-странице, но и исчезновение, видимость, кликабельность элемента, видимость заголовка web-страницы и так далее. Условие вызывается с определенной частотой, пока не истечет время ожидания. Это означает, что до тех пор, пока условие возвращает ложное значение, оно будет продолжать пытаться и ждать.

Явные ожидания используют классы WebDriverWait и ExpectedConditions. WebDriverWait позволяет создать объект ожидания и связать его с драйвером, а также указать время ожидания выполнения условия. ExpectedConditions позволяет непосредственно указать событие, наступление которого мы ждем.

Использование явного ожидания:

```
WebDriverWait wait = new WebDriverWait (driver, 5);
wait.until(ExpectedConditions.presenceOfAllElementsLocatedBy(By locator));
```

К явным ожиданиям также можно отнести ожидание такого вида - Trhred.sleep(1000), однако такое ожидание является наихудшим способом использования явных ожиданий, так как такой способ не гарантирует, например, появления элемента, или элемент появляется быстрее указанного времени, в этом случае, время автоматизации увеличивается.

Главным отличием явного ожидания от неявного является то, что для явного ожидания конкретно указывается элемент, для которого будет выполняться ожидание, в неявном – ожидание выполняется для всех элементов. Неявное ожидание используется поиска для элемента, явные ожидания позволяют задать условия ожидания. Решать, что является лучше, приходится отталкиваясь от конкретной ситуации. Например, минусом явных ожиданий может быть то, что их нужно прописывать каждый раз для отдельного элемента.

Свободные ожидания, или FluentWait, ожидание, которое в отличии от явного и неявного, принимает два параметра: время ожидания и частоту опроса. Как и в явных ожиданиях, в свободных ожиданиях можно указывать условие, выполнение которого ожидается. Драйвер будет проверять выполнение условия через промежуток времени, равный частоте опроса в течении времени, указанного для ожидания. По истечении времени, при невыполнении условия, также, как и в явном, и в неявном ожиданиях, будет сгенерировано исключение о том, что условие не выполнено.

Еще одним преимуществом свободного ожидания можно указать возможность игнорировать различные виды исключения и указывать сообщения, которые будут выводиться по истечении времени.

Архитектура тестового фреймворка

Отличие автоматизации тестирования от разработки заключается в целях написания кода, однако и автоматизация, и разработка занимается тем, что пишет код. Наравне с тем, как важно писать грамотный код при разработке приложения, важно писать грамотный код при разработке тестового фреймворка для этого приложения. Отсутствие архитектуры при проектировании тестового фреймворка приводит к многочисленным переписываниям кода, что в свою очередь требует серьезных временных затрат.

Большую роль в архитектуре тестового фреймворка играет Page Object паттерн (шаблон проектирования). Данный шаблон позволяет отделить работы с web-страницами и их элементами от теста. Основная идея – инкапсулировать логику поведения страницы в классе страницы. Таким образом, тесты будут работать не с низкоуровневым кодом, а с высокоуровневой абстракцией. Простыми словами, каждая web-страница приложения описывается в отдельном классе. В нем определяются элементы, используемые на этой странице и методы для работы с этими элементами.

Плюсы использования паттерна Page Object:

+ Отделение описания страниц от теста
+ Объединение всех действий по работе с web-страницей в одном месте

Однако использование данного шаблона не всегда удобно. Приложение может содержать кастомные элементы, с которыми WebDriver работать не умеет. Кастомными элементами называются элементы, созданными разработчиками самостоятельно, которые не имеют стандартных свойств и методов. Для взаимодействия с такими элементами, необходим описать их и методы для взаимодействия с ними. Для работы с такими элементами удобнее воспользоваться паттером Page Element, суть которого заключается в разделении web-страницы на более мелкие части и описании этих частей по отдельности.

При тестировании масштабного проекта приходится взаимодействовать с большим количеством web-элементов, соответственно с большим количеством локаторов для поиска этих элементов. При изменении локатора какого-то элемента в приложении, поиск этого локатора в тестовом фреймворке пусть и не займет много времени, но определенно будет неудобным. Поэтому еще одной частью архитектуры тестового фреймворка является вынесение всех локаторов в отдельный файл, например, properties файл. Таким образом, при необходимости изменить локатор, потребуется изменить его лишь в одном файле.

Во время проектирования тестового фреймворка происходит настройка окружения для тестирования, указание ссылок страниц, типов браузеров для тестирования и другое. Для организации грамотного кода необходимо вынести все данные для настройки в отдельный файл, что опять же позволит

быстро изменить данные при необходимости, избавит от использования так называемых «магических значений» и упростит код.

Ещё одной рекомендаций при построении архитектуры тестового фреймворка является вынесение часто повторяющихся действий в отдельные классы. Отдельным классом можно вынести, например, работу с ожиданиями или чтение данных из properties файлов. Также рекомендуется выносить тестовые данные, используемые при тестировании, в отдельные файлы.

Есть еще множество рекомендаций по упрощению и развитию тестового фреймворка. В конечном итоге, архитектуру тестового фреймворка можно представить следующим образом (см Приложение4).

Стратегия внедрения автоматизации тестирования на примере проекта web-приложения

В курсовой работе в качестве тестируемого web-приложения используется веб-сайт Onliner (Онлайнер), найти который можно по ссылке https://www.onliner.by/ .

Проведение Unit теста (модульного теста) направлено на то, чтобы проверить работу отдельных модулей web-приложения по сценарию, имитирующему действия пользователя.

Unit тест проходит следующих сценарий:

Номер шага	Шаг	Ожидаемый результат
1	Зайти на страницу https://www.onliner.by/	Главная страница onliner открылась
2	Выполнить процесс авторизации	Вход успешно выполнен. Пользователь авторизован
3	Получить список популярных тем (см. Приложение 5)	
4	Перейти на страницу случайной темы из списка популярных	
5	Получить название открытой страницы (см. Приложение 6)	Проверить, что после перехода по ссылки, открывается корректная тема
6	Перейти на главную страницу сайта	
7	Выполнить выход из системы	Проверить, что пользователь успешно вышел из системы

Для тестирования данного приложения используются следующие инструменты:

1. Selenium WebDriver
2. UnitTest Framework: TestNG

Тестовые данные, понадобившиеся для тестирования:

1. Логин пользователя
2. Пароль пользователя

Окружение:

Операционная система: Windows 8.1, 64-разрядная ОС, процессор x64

Браузеры:

1. Google Chrome, версия 81.0.4044.138 (Официальная версия), (64 бит)
2. Firefox, версия 72.0.2, (64 бит)

Фреймворк для автоматизации web-приложения

Взаимодействие фреймворка с браузером происходит при помощи инструмента Selenium WebDriver. Фреймворк позволяет тестировать приложение в двух браузерах: Google Chrome и Firefox. В зависимости от типа браузера, устанавливается драйвер, соответствующий данному типу браузера. Тип браузера хранится в файле prop.properties. Чтение данных из этого файла вынесено в отдельный класс PropertiesHandler.

Придерживаясь рекомендаций по организации архитектуры фреймворка, используется паттерн проектирования Page Object. Каждой странице, тестируемой проектом, соответствует свой класс. В классе инициализируется объекты WebElement, используемые на web-странице.

Поиск WebElement происходит через локаторы. Все локаторы, используемые в проекте, располагаются в одном файле – locators.properties. В проекте используется поиск элементов по трем типам локаторов: XPath, Class name, CSS. Для организации работы с локаторами был создан отдельный класс, выполняющий чтение значения локатора по имени и определяющий тип используемого локатора.

В проекте используются 2 вида ожидания: явные и неявные. Для работы с явными ожиданиями был создан класс WaitHandler, содержащий методы для управления ожиданиями. Работа с ожиданиями в проекте включет три метода:

1. ожидать, пока элемент не пропадет со страницы
2. ожидать, пока элемент не появится на странице
3. ожидать, пока элемент не станет кликабельным

Необходимые тестовые данные хранятся в файле prop.properties. Чтение этих данных также организовано через класс PropertiesHandler.

Используя тестовый фреймворк TestNG, тестовый класс организовала следующим образом:

- @BeforMethod функция setUp (), выполняющая получение тестовых данных и настройку браузера. Данная функция, за счет аннотирования BeforeMethod, будет выполнять соответствующую настройку перед каждым тестовым методом.
- @Test функция onlinerTest (). Непосредственно тестовый метод, выполняющий автоматизацию тестового сценария.
- @AfterMethod функция quit (). Выполняет закрытие экземпляра драйвера, а соответственно и закрытие браузера.

Тестовый метод выполняет следующее:

1. Методом драйвера get () переходит по указанной ссылке на главную страницу сайта.
2. Проверяет, что открылась именно главная страница сайта за счет поиска главного логотипа сайта (см. Приложение 7). Если главная

страница не открылась, тест не пройден, выдается сообщение об ошибке.

3. Нажимает на кнопку входа для авторизации (см. Приложение 11).
4. Проверяет, что открылась страница авторизации путем поиска заголовка страницы авторизации (см. Приложение 8). Если страница авторизации не открылась, выдается сообщение об ошибке.
5. Выполняет авторизацию: вводятся данные пользователя (логин, пароль), наживается кнопка «Войти».
6. Ожидает закрытия страницы авторизации.
7. Проверяет, что авторизация выполнена успешна за счет поиска изображения пользователя (см. Приложение 9). Если данный элемент не найден, подразумевается, что авторизация не выполнена, тест не пройден, выдается сообщение об ошибке.
8. Получает номер случайной темы из списка популярных тем.
9. Переходит по ссылке на страницу указанной темы.
10. Проверяет, что открытая страница соответствует заданной теме путем получения заголовка (см. Приложение 6) страницы и сравнения ее с заранее сохраненным названием темы. Если тексты не совпадают, подразумевается, что открылась не та страница, тест не пройден, выдается сообщение об ошибке.
11. Переходит на главную страницу путем нажатия на главный логотип сайта (см. Приложение 7).
12. Проверяет, открылась ли главная страница сайта за счет поиска главного логотипа сайта (см. Приложение 7). Если главная страница не открылась, тест не пройден, выдается сообщение об ошибке.
13. Выполняется выходи из системы: нажатие на изображение пользователя (см. Приложение 9), после чего нажатие на ссылку “Выйти” (см. Приложение 10).
14. Проверяет, вышел ли пользователь из системы, за счет поиска кнопки “Вход” на странице (см. Приложение 11). Если элемент не найден, тест не пройден, выдается сообщение об ошибке.

Исходный код тестового фреймворка можно найти по ссылке: https://github.com/zzhiharevich/lab-2-zzhiharevich

Полную документацию тестового фреймворка можно найти по ссылке:

https://test-framework-documentation.herokuapp.com/

Заключение

В ходе проделанной курсовой работы были изучены такие инструменты для автоматизации тестирования как Selenium WebDriver и TestNG.

На основе изученного материала был разработан тестовый фреймворк для автоматизации веб-приложения.

В тестовом фреймворке наглядно продемонстрирована работа изученных инструментов, а именно:

- Взаимодействие с браузером при помощи использования WebDriver
- Тестирование приложения в нескольких браузерах
- Взаимодействие в веб-элементами используя методы класса WebElement
- Поиск элемента при помощи объектов класса By, использующего локаторы
- Использование разных типов локаторов (XPtah, Class name, CSS)
- Продемонстрирована работа с явными и неявными ожиданиями
- Организация тестового класса при помощи фреймворка TestNG
- Использование аннотаций @BeforeMethod и @AfterMethod для определения функций, автоматически вызываемых перед и после тестового метода соответственно

Тестовый фреймворк разработан с использованием шаблона проектирования Page Object.

Список использованной литературы

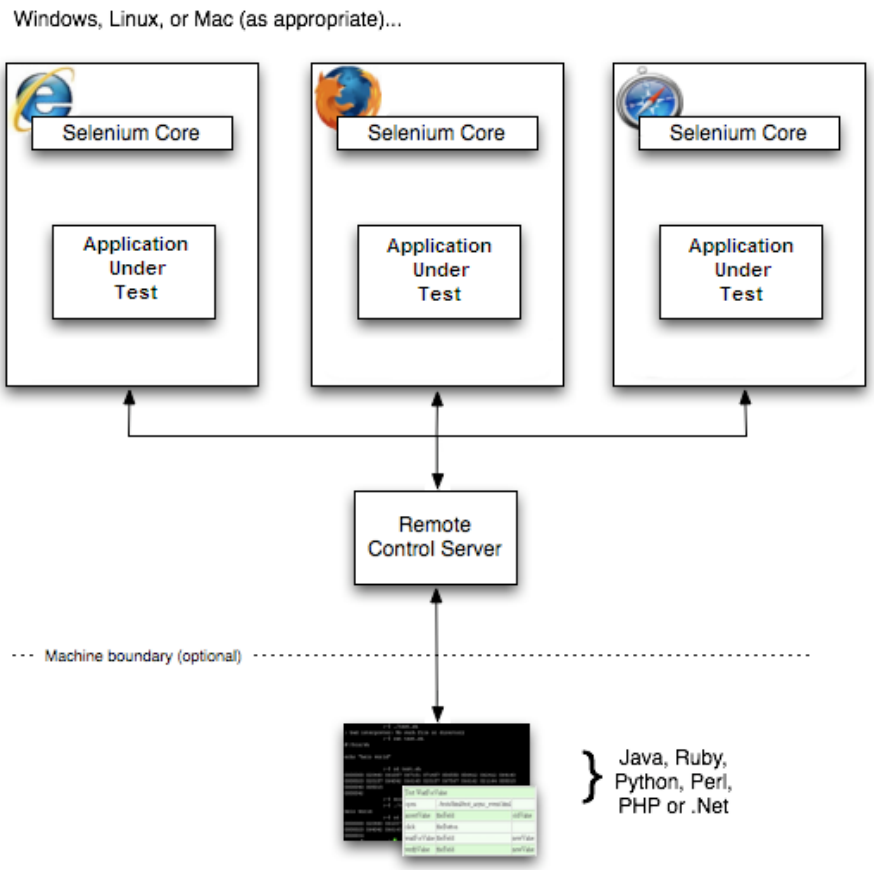
1. https://www.software-testing.ru/library/testing/testing-automation/1787--web-
2. https://habr.com/ru/post/152653/
3. https://www.selenium.dev/documentation/en/
4. https://comaqa.gitbook.io/selenium-webdriver-lectures/
5. https://www.selenium.dev/selenium/docs/api/java/overview-summary.html
6. https://sqadays.com/ru/talk/7636
7. https://testerslittlehelper.wordpress.com/2016/03/25/waits-in-selenium/
8. Презентация «Unit-тестирование», прочтенная на лекции
9. Презентация «Локаторы», прочтенная на лекции

Приложения

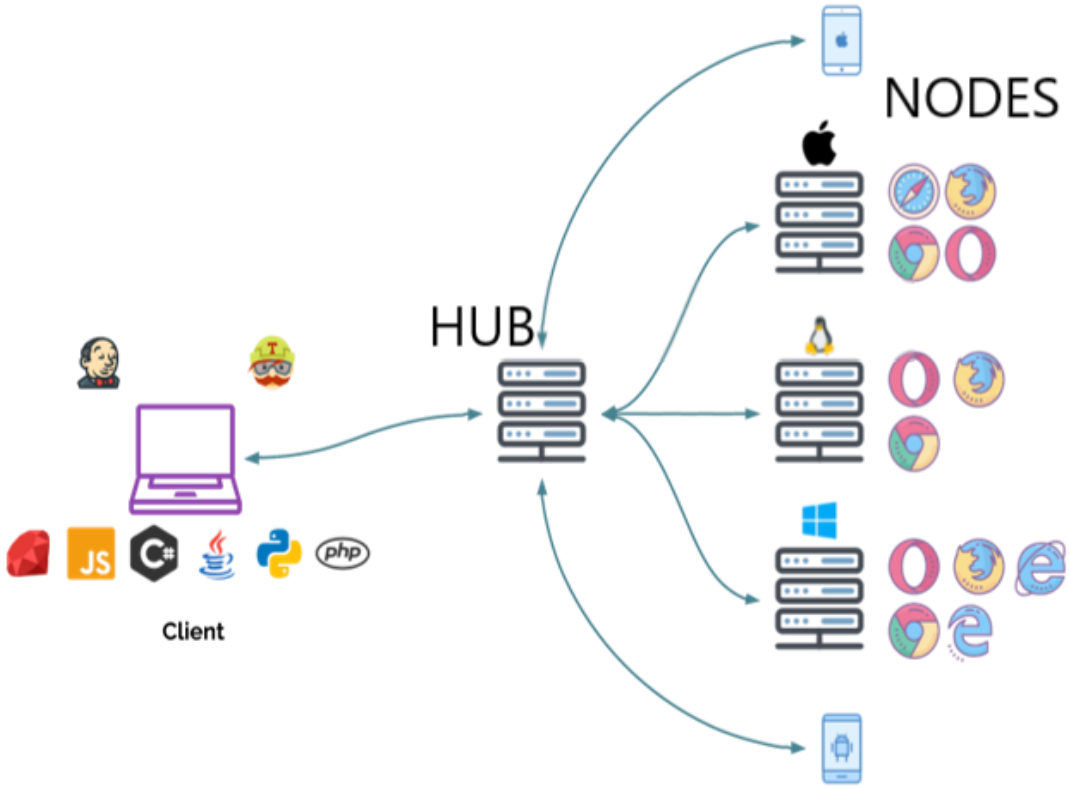
Приложение 1

Product	Selenium	Katalon Studio	Unified Functional Testing	TestComplete	watir
Available since	2004	2015	1998	1999	2008
Application Under Test	Web apps	Web (UI & API), Mobile apps	Web (UI &API), Mobile, Desktop, Packaged apps	Web (UI &API), Mobile, Desktop apps	Web apps
Pricing	Free	Free	$$$$	$$	Free
Supported Platforms	Windows Linux OS X	Windows Linux OS X	Windows	Windows	Windows Linux OS X
Scripting languages	Java, C#, Perl, Python, JavaScript, Ruby, PHP	Java/Groovy	VBScript	JavaScript, Python, VBScript, JScript, Delphi, C++ and C#	Ruby
Programming skills	Advanced skills needed to integrate various tools	Not required. Recommended for advanced test scripts	Not required. Recommended for advanced test scripts	Not required. Recommended for advanced test scripts	Advanced skills needed to integrate various tools
Ease of Installation and Use	Require advanced skills to install and use	Easy to setup and use	Complex in installation. Need training to properly use the tool	Easy to setup. Need training to properly use the tool	Advanced skills needed to integrate various tools

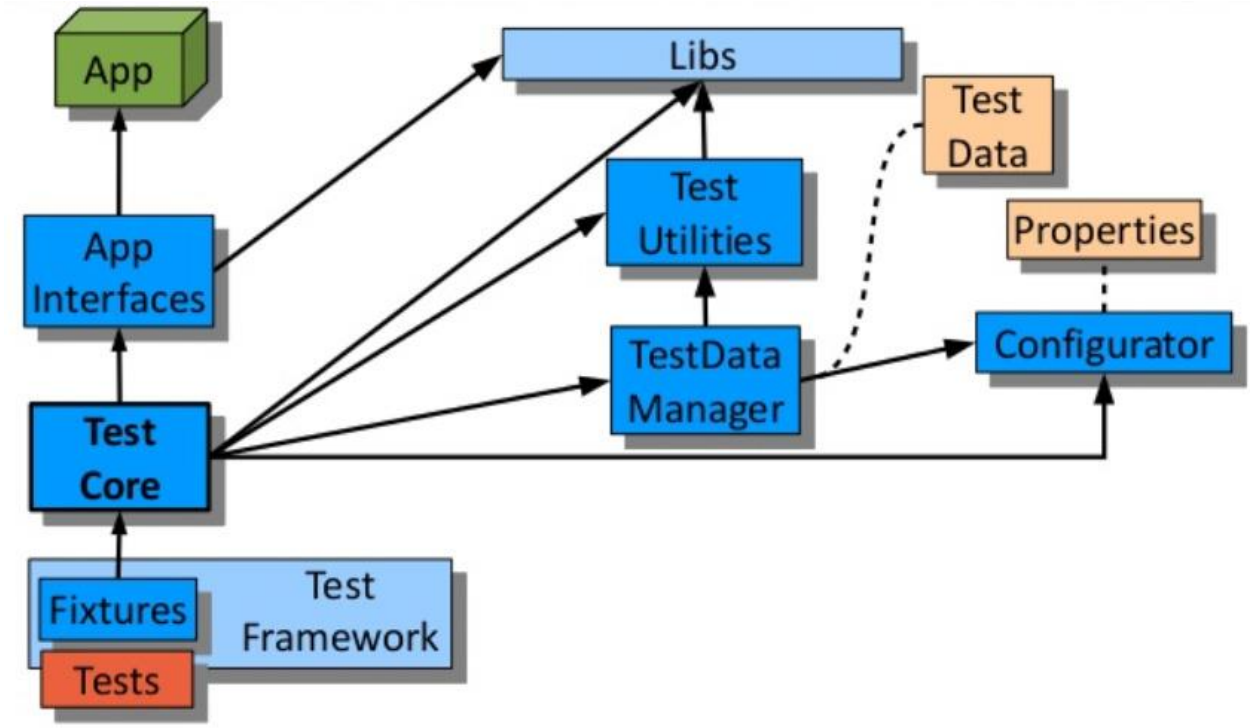
Приложение 2



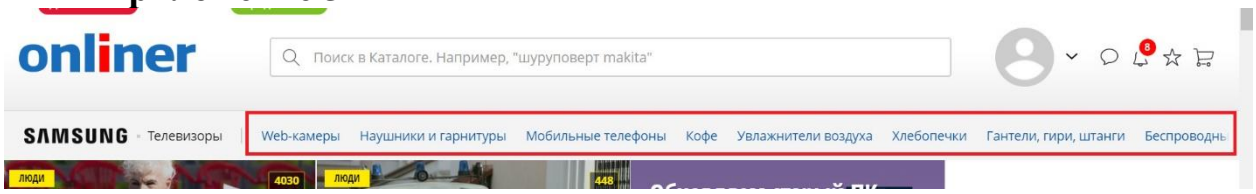
Приложение 3



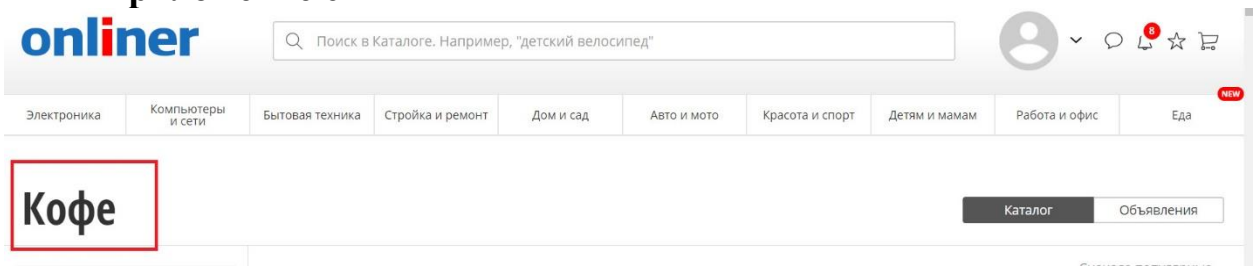
Приложение 4



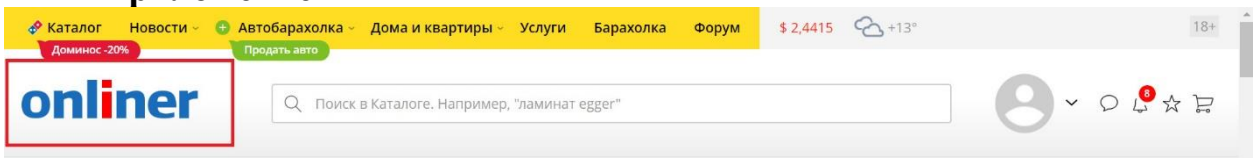
Приложение 5



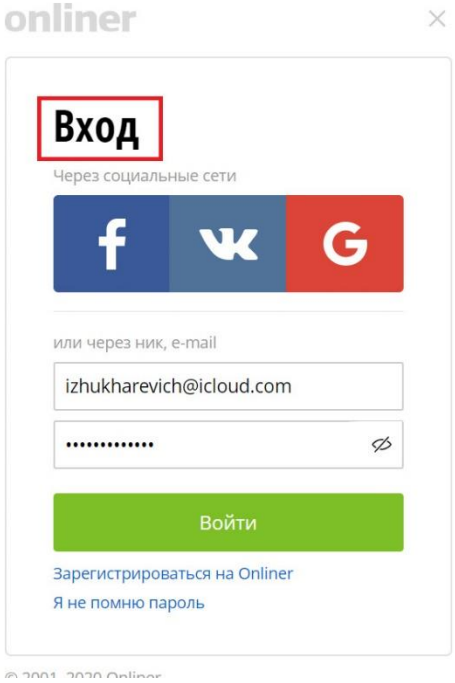
Приложение 6



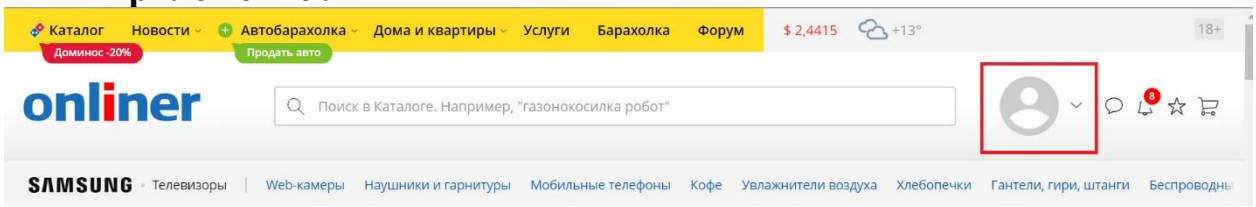
Приложение 7



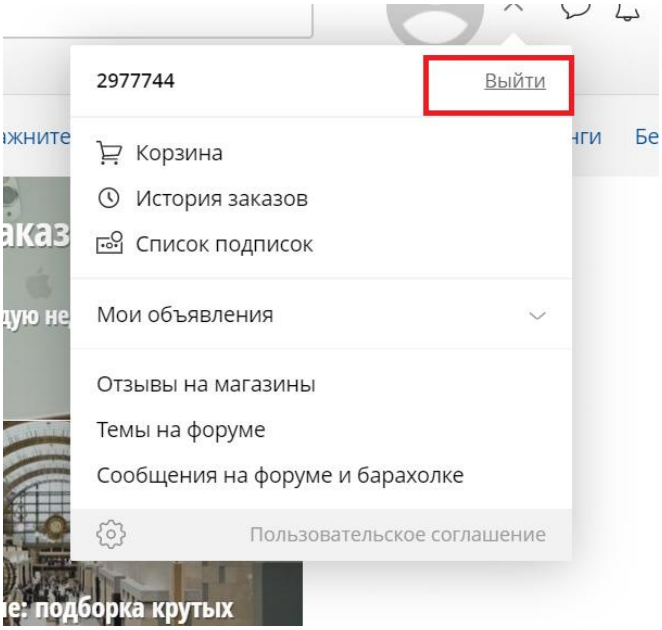
Приложение 8



Приложение 9



Приложение 10



Приложение 11