# **ВВЕДЕНИЕ**

На данный момент времени сложно представить себе современного человека, который в течение дня не посещает хотя бы единожды тот или иной веб-сайт. Интернет-технологии настолько тесно взаимосвязаны с ежедневным бытом каждого из нас (общение с друзьями и знакомыми в мессенджерах, поиск необходимой информации, заказ продуктов питания из интернет-гипермаркетов, покупка гаджетов и техники) и нашими привычками (чтение интересующих нас новостей, просмотр видеороликов, прослушивание музыки), что никто не усомнится в важности качества веб-сайтов и других интернет-сервисов.

В условиях жесткой конкуренции и постоянной борьбы за аудиторию на рынке веб-технологий, качество **–** критический фактор, влияющий на посещаемость веб-сайта и итоговую прибыль, которую получит бизнес благодаря популярности своего сервиса.

Огромную роль в качестве программного обеспечения занимает тестирование. Оно помогает обеспечить корректную и стабильную работу веб-сайта в различных браузерах и на различных конфигурациях компьютерного “железа”, сделать его удобным для пользователя и оптимизировать производительность.

Таким образом, вопрос создания удобного и эффективного средства для тестирования веб-сайтов и приложений является безусловно важным.

Задачей дипломного проектирования является разработка библиотеки, которая позволит автоматизировать тестирование UI (User Interface) и API (Application Programming Interface) веб-сайтов.

Для решения поставленной **задачи** необходим следующее:

1. Исследовать современные подходы и методы автоматизированного тестирования ПО;
2. выявить особенности веб-сервисов, влияющие на процесс их тестирования;
3. рассмотреть и проанализировать современные системы для тестирования UI и API веб-сервисов, провести их сравнительную характеристику, выявить достоинства и недостатки;
4. освоить технологию создания фреймворков для тестирования веб-сервисов;
5. выявить необходимость разработки универсальной системы для автоматизации тестирования веб-сайтов;
6. разработать фреймворк для автоматизации процесса тестирования UI и API веб-сервисов;
7. протестировать веб-сервис с использованием разработанного фреймворка.

# **АНАЛИЗ ИСХОДНЫХ ДАННЫХ И ПОСТАНОВКА ЗАДАЧ НА ДИПЛОМНОЕ ПРОЕКТИРОВАНИЕ**

## **1.1 Анализ исходных данных к дипломному проекту**

Общая цель веб-тестирования — найти проблемы, также известные как ошибки или баги, проблемы или дефекты, которые могут отрицательно повлиять на работу веб-сайта или приложения. Веб-тестирование также можно использовать, чтобы добавить новую функциональность, обнаружить определенные области или аспекты веб-сайта, которые можно улучшить и, таким образом, получить лучшие результаты, будь то больше запросов, больше продаж, больше повторных посетителей или более счастливые пользователи.

Процедура тестирования чрезвычайно тщательна. Веб-тестировщики проверяют всю веб-систему от начала до конца, прежде чем выпускать веб-сайт на рынок.

По способам тестирование можно подразделить на:

1. Автоматическое тестирование

Для автоматической проверки веб-сайта тестировщику понадобятся специальные программы. Программа настраивается под нахождение и проверку на соответствия определенных параметров сайта. Такой тип тестирования в наибольшей степени подходит для объемных проектов, нагруженных сложной функциональностью.

1. Ручное тестирование

Ручной тип тестирования программного обеспечения выполняется тестировщиком вручную без использования каких-либо автоматизированных инструментов. Ручное тестирование помогает находить критические ошибки в программном приложении, проверяет реальный клиентский опыт, навигацию и многое другое.

Автоматизация помогает выстроить баланс и проверять вручную то, что требует человеческого внимания (как правило, до 25% кейсов); автоматизировать остальные кейсы.

При этом автоматизация в долгосрочной перспективе снижает как расходы на тестирование, так и риски, связанные с человеческим фактором.

Кроме того, при необходимости можно ускорить релизы. Например, если в спринте разработки нужно проверить около 400 кейсов, то их ручная проверка займет до двух недель, а автотесты можно провести ночью и проанализировать за 4 часа.

Бизнес, благодаря автоматизации, получает возможность в любое время убедиться в том, что ключевые функции системы работают правильно и проверить, нет ли ошибок (а если они есть, то в чем заключаются).

## **1.2 Обзор существующих программных средств по теме дипломного проекта**

При выборе фреймворка стоит обращать внимание на его популярность на GitHub, поддержку сообщества, количество открытых issues и совместимость с технологиями, которые используются у вас в системе, а также на особенности каждого инструмента, применяемого для автоматизации.

Например, Protractor хорошо подходит для проектов, написанных на Angular, потому что в нем есть встроенные функции для специфических загрузок Angular-элементов.

Ниже подробнее рассмотрим самые популярные JS-фреймворки, для автоматизации тестирования.

**Protractor**

Плюсы:

* Protractor — это единственный фреймворк, который из коробки поддерживает кастомные определения AngularJS-элементов. Если у вас Angular, используйте Protractor.
* Удобная поддержка TypeScript и разных фреймворков для unit testing (Jasmine, Mocha, Cucumber и пр.).

Минусы:

* Нет поддержки мобильного тестирования.
* Написан как обертка над WebDriverJS. Если будут проблемы с WebDriverJS, автоматически они будут и в ProtractorJS.

**Nightwatch.js**

Плюсы:

* Похож на WebdriverIO и тоже является кастомной имплементацией W3C WebDriver API.
* Легко добавить новую функцию.
* Не нужно выбирать между Jasmine и Mocha.

Минусы:

* Нет поддержки Mocha и мобильного тестирования.
* Меньше поддержки, чем у WebdriverIO и Protractor.

**Cypress**

Плюсы:

* Архитектура работы с браузером была написана полностью c нуля. Cypress не запускает удаленные команды через сеть, а работает напрямую с программой.
* Быстрая и удобная настройка. Необходимо запустить только одну команду для установки всех пакетов, и нет необходимости ставить все отдельно.
* Вся документация в одном месте, не нужно отдельно искать, как правильно делать проверки или как использовать Chai.
* Необычный интерфейс запуска с описанием, какие команды выполняются, что позволяет быстрее понимать, что происходит на странице в определенный момент.

Минусы:

* Cypress не использует Selenium для end-to-end-тестирования. Что порождает много проблем в работе. Работа с браузером не всегда стабильна, и на сегодняшний день на GitHub висят открытыми более чем 900 проблем.
* Нет поддержки мобильного тестирования, и, судя по комментариям создателей нативной поддержки, никогда и не будет.
* Поддержка ограниченного количества браузеров: Canary, Chrome, Chromium, Electron.
* Платная параллелизация тестов. Если необходимо запускать тесты в несколько потоков, то придется платить.

**WebdriverIO**

Плюсы:

* WebdriverIO — это кастомная имплементация W3C WebDriver API. Не нужно привязываться к имплементации WebDriverJS.
* Поддержка синхронного кода. Можно забыть об асинхронности JavaScript.
* Удобная базовая настройка с помощью встроенного интерфейса для командной строки wdio.
* Поддерживает почти все тестовые фреймворки BDD и TDD.
* Поддерживает удобную библиотеку ‘webdrivercss’ для сравнения CSS-стилей элементов на странице.

Минусы:

* Большое различие между последними версиями (WebdriverIO 4 и 5).
* Хуже кастомизирован для автоматизации AngularJS, чем Protractor.
* Синхронность WebdriverIO обеспечивает технология Fibers, иногда ее еще называют coroutines. У этой технологии много плюсов и преимуществ, но существуют подводные камни, например запуск в асинхронном коде сделает весь код асинхронным.

**Selenium**

Selenium - настоящий старожил среди всех инструментов для функционального тестирования. Его история началась в 2004 году, и сейчас он по праву считается самым популярным инструментом своей отрасли. Несмотря на повсеместное применение Selenium такими гигантами как Google, у него все еще остается ряд недостатков. Пожалуй, главным из них является необходимость владения продвинутыми навыками программирования и написания скриптов для полноценной работы с продуктом, что в свою очередь часто затрудняет освоение инструмента начинающими тестировщиками. Зато Selenium распространяется абсолютно бесплатно, поддерживает огромное количество языков для написания скриптов, а также является кроссплатформенным инструментом, и за это его невозможно не любить.

Плюсы:

* Бесплатный
* Поддержка всех основных ЯП для сценариев
* Кроссплатформенность

Минусы:

* Тестирование ограничивается веб приложениями
* Нужны продвинутые навыки программирования
* Ограниченность функционала в сравнении с платными аналогами

## **1.3  Обоснование и описание выбора языка программирования, средств разработки, используемых технологий и сторонних библиотек**

Приступая к автоматизации, каждая команда принимает решение на основе своих знаний или руководствуясь потребностями заказчика. Сейчас появляется все больше возможностей выбирать язык автоматизации исходя из стека технологий конкретного проекта.

Разница между языками все больше стирается и становится незаметней. Поэтому, просматривая современные тренды в автоматизации, можно заметить, что все больше проектов используют **JavaScript** для разных уровней и видов тестирования. Как пример можно привести статистику Github за 2018 год по популярности языков программирования (Рисунок 1.1).

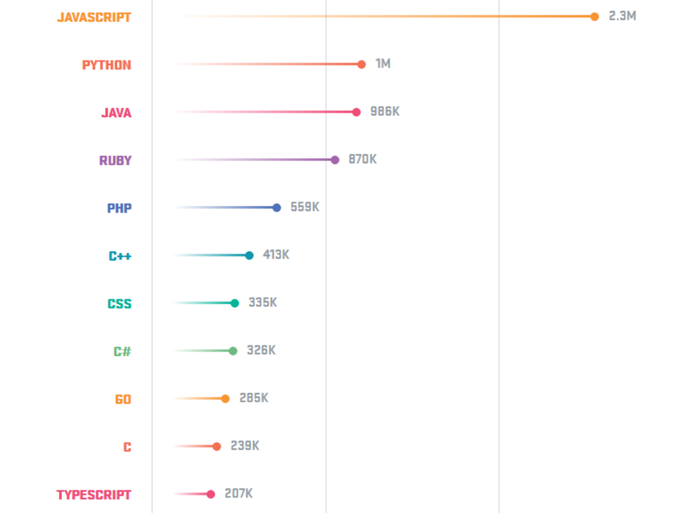


Рисунок 1.1 – Статистику GitHub по популярности языков программирования

Первое, на что нужно обратить внимание при выборе языка для автоматизации, это стек технологий, которые используются в проекте. Так на текущий момент времени практически все актуальные веб-сайты используют JavaScript для так называемого “оживления” веб-страниц, как минимум это весьма существенный плюс в пользу того, чтобы использовать этот же язык программирования при написании автотестов.

Помимо вышеозначенного момента, можно выделить следующие преимущества использования JavaScript-фреймворков для автоматизации тестирования:

* Скорость написания тестов значительно выше, чем на Java или C#.
* Ниже порог входа для старта проекта.
* Большое количество готовых решений очень разных проблем, которые возникают.

По вышеозначенным причинам для реализации библиотеки данного дипломного проекта был выбран язык программирования JavaScript

В качестве среды разработки был выбран **Visual Studio Code** - это сервис, который позиционируется как «легкий» редактор кода для кроссплатформенной разработки веб- и облачных приложений.

VS Code позволяет разрабатывать как консольные приложения, так и приложения с графическим интерфейсом, в том числе с поддержкой технологии Windows Forms, а также веб-сайты, веб-приложения, веб-службы как в родном, так и в управляемом кодах для всех платформ.

В редакторе присутствуют встроенный отладчик, инструменты для работы с Git и средства рефакторинга, навигации по коду, автодополнения типовых конструкций и контекстной подсказки.

Продукт поддерживает разработку для платформ ASP.NET и Node.js, и считается легковесным решение, которое позволяет обойтись без полной интегрированной среды разработки.

Большим плюсом редактора является поддержка большого количества языков, таких как C++, C#, Python, PHP, JavaScript и других.

Можно выделить следующие преимущества данного редактора кода:

* множество настроек (как всей программы, так и интерфейса);
* расширяемая библиотека дополнений и готовых решений;
* мультифункциональность (редактор поддерживает почти все языки, используемые для создания приложений);
* простота и гибкость.

В качестве инструмента для общения с серверной частью веб-сайтов была выбрана библиотека **Axios** **-** JavaScript-библиотека, основанная на обещаниях, для выполнения HTTP-запросов.

Некоторые преимущества Axios:

* Трансформаторы: позволяют выполнять преобразования данных перед выполнением запроса или после получения ответа.
* Перехватчики: позволяют полностью изменить запрос или ответ (в том числе и заголовки). также выполнять асинхронные операции до того, как будет сделан запрос или до того, как Promise выполнит
* Встроенная защита XSRF
* Поддержка всеми актуальными браузерами (Рисунок 1.2)

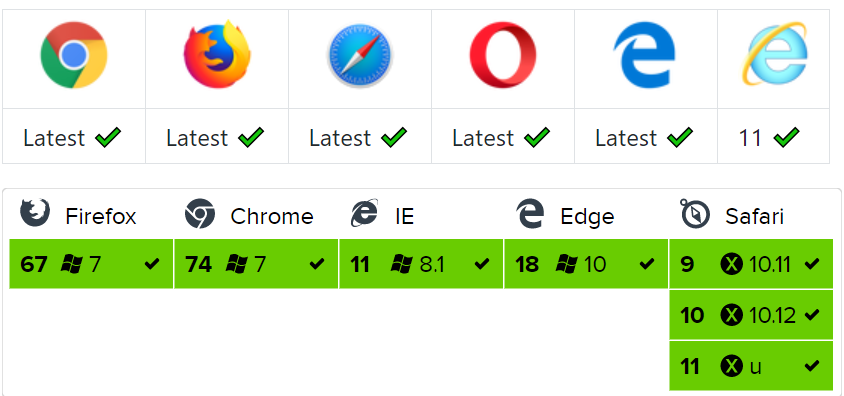


Рисунок 1.2 – Поддержка Axios популярными браузерами

Выбор так называемой assertion библиотеки пал на весьма популярную **Chai**. Можно даже сказать, что большинство разработчиков и тестировщиков используют именно Chai для написания тестовых утверждений.

В Chai сужествует три типа assertions:

* should (должен)
* expect (ожидать)
* assert (утверждать)

Данная библиотека достаточно гибкая и предоставляет большой выбор тестовых утверждений, который способен покрыть фактически любые запросы при написании автоматизированных тестов.

**Mocha** — это JS-фреймворк, облегчающий тестирование асинхронного кода в Node.JS модуле или браузерном приложении. Тесты в Mocha имеют улучшенное качество трассировки исключений и могут прогоняться сериями.

Используя Mocha, можно получить необходимый минимум и ничего лишнего. Это один из популярнейших фреймворков на Node.JS с достаточно активным сообществом и большим количеством документации. Но при всех достоинствах у него есть и недостаток — это отсутствие assert-ов. Однако данный недостаток можно легко назвать и преимуществом, поскольку мы можем подключить фактически любую из имеющихся библиотек утверждений (выбор был сделан в пользу Chai).

## **Постановка задач по разработке программного средства**

Разработка любого программного средства начинается с постановки задач, которые должны быть решены в ходе работы, для того чтобы получить конечный продукт. В ходе дипломного проектирования необходимо разработать библиотеку “kbv-testdriver” на базе Selenium Webdriver для автоматизации тестирования UI и API веб-сайтов. Для этого необходимо решить следующие задачи:

1. Проанализировать исходные данные.
2. Произвести анализ рынка библиотек для автоматизированного тестирования веб-сайтов.
3. Выбрать язык программирования, средства разработки и сторонние библиотеки. Привести описание выбранного языка программирования, средств разработки и сторонних библиотек. Обосновать выбор языка программирования, средств разработки и сторонних библиотек.
4. Спроектировать структуру библиотеки.
5. Формализировать предметную область программного средства.
6. Рассмотреть интеграцию разработанной библиотеки с тестируемым веб-сервисом.
7. Описать и реализовать используемые в библиотеке алгоритмы.
8. Протестировать программное средство.
9. Произвести оценку временных показателей программного средства.
10. Произвести оценку ресурсных показателей программного средства.
11. Ввести в эксплуатацию и обосновать минимальные технические требования к оборудованию.
12. Разработать руководство по эксплуатации программным средством.
13. Произвести технико-экономическое обоснование разработки библиотеки “kbv-testdriver” на базе Selenium Webdriver для автоматизации тестирования UI и API веб-сайтов.

Таким образом, для получения готового продукта необходимо решить 13 задач.

# **2 ПРОЕКТИРОВАНИЕ, РАЗРАБОТКА И ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА**

## **2.1. Проектирование архитектуры библиотеки**

Для реализации фреймворка будет использоваться объектно-ориентированная архитектура, так как она обладает следующими полезными свойствами:

1. можно изменять реализацию того или иного объекта или добавлять новые объекты без воздействия на остальную часть системы;
2. имеющиеся объекты могут быть повторно использованы при создании новых объектов;
3. фреймворк состоит из независимых компонент, что облегчает процесс разработки и модификации системы.

Для реализации библиотеки “kbv-testdriver” будет использоваться Selenium Webdriver, который позволяет взаимодействовать с браузером и элементами веб-страниц – это популярный инструмент для управления реальным браузером, который можно использовать как для автоматизации тестирования веб приложений, так и для выполнения других рутинных задач, связанных с работой в вебе. Разрабатываемая библиотека будет являться наследником Selenium Webdriver и вбирать в себя все основные и наиболее часто используемые средства для написания и проведения автоматизированного тестирования пользовательского интерфейса приложений. В дополнение к этому в “kbv-testdriver” будет реализована работа с Representational State Transfer Application Programming Interface (REST API). По умолчанию “из коробки”, Selenium Webdriver лишен такой возможности. Также будет добавлена возможность загрузки изображений на веб-сервис и последующий анализ их изменений в процессе обработки веб-сервисом.

Создание набора тестов с использованием WebDriver потребует от нас понимания и эффективного использования ряда различных компонентов. Как и все в программном обеспечении, разные люди используют разные термины для обозначения одной и той же идеи. Ниже приведем описание терминов, которые будут использоваться в дальнейшем.

API: интерфейс прикладного программирования. Это набор «команд», которые вы используете для управления WebDriver.

Библиотека: модуль кода, который содержит API и код, необходимый для их реализации. Библиотеки зависят от привязки к каждому языку, например файлы .jar для Java, файлы .dll для .NET и т. Д.

Драйвер: отвечает за управление самим браузером. Большинство драйверов создаются самими производителями браузеров. Драйверы, как правило, представляют собой исполняемые модули, которые запускаются в системе с самим браузером, а не в системе, в которой выполняется набор тестов. (Хотя это может быть одна и та же система.) Некоторые люди называют драйверы прокси-серверами.

Framework: дополнительная библиотека, используемая для поддержки пакетов WebDriver. Эти фреймворки могут быть тестовыми фреймворками, такими как JUnit или NUnit. Они также могут быть фреймворками, поддерживающими функции естественного языка, такие как Cucumber или Robotium. Фреймворки также могут быть написаны и использованы для таких вещей, как манипулирование или настройка тестируемой системы, создание данных, тестовые оракулы и т.д.

Как минимум, WebDriver общается с браузером через драйвер. Связь является двухсторонней: WebDriver передает команды браузеру через драйвер и получает информацию обратно по тому же маршруту. Данный вариант представлен на рисунке 2.1.

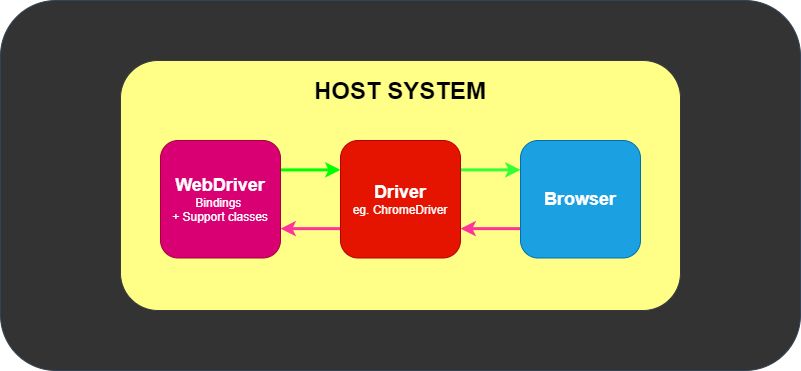


Рисунок 2.1 – Простейший вариант взаимодействия Selenium Webdriver с браузером (прямое общение)

Драйвер специфичен для браузера, например ChromeDriver для Google Chrome / Chromium, GeckoDriver для Mozilla Firefox и т. Д. Драйвер работает в той же системе, что и браузер. Это может быть или не быть той же системой, в которой выполняются сами тесты.

Этот простой пример выше - прямое общение. Связь с браузером также может быть удаленной через Selenium Server или RemoteWebDriver. RemoteWebDriver работает в той же системе, что и драйвер и браузер. Данный вариант представлен на рисунке 2.2.

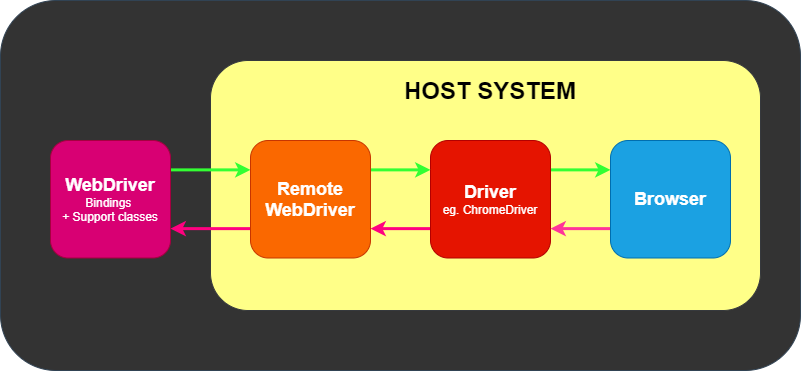


Рисунок 2.2 – Вариант взаимодействия Selenium Webdriver с браузером с использованием RemoteWebDriver

Удаленное общение также может происходить с использованием Selenium Server или Selenium Grid, которые, в свою очередь, взаимодействуют с драйвером в хост-системе. Данный вариант представлен на рисунке 2.3.

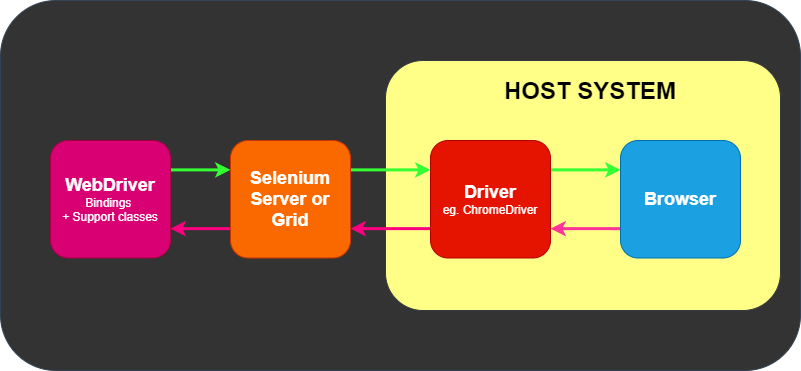


Рисунок 2.3 – Вариант взаимодействия Selenium Webdriver с браузером с использованием Selenium Server или Selenium Grid

Однако при рассмотрении каждого из представленных вариантов взаимодействия Selenium Webdriver с браузером важно понимать то, что у WebDriver одна и только одна задача: общаться с браузером любым из вышеперечисленных методов. WebDriver ничего не знает о тестировании: он не знает, как сравнивать, утверждать, прошел или не прошел тот или иной тест, и, конечно же, ничего не знает об отчетах или грамматике Given / When / Then.

Здесь на помощь приходят различные фреймворки. Как минимум нам понадобится тестовая среда, которая соответствует языковым привязкам, например NUnit для .NET, JUnit для Java, RSpec для Ruby и как уже было описано ранее – Mocha для JavaScript.

Платформа тестирования отвечает за запуск и выполнение вашего WebDriver и связанных шагов в ваших тестах. Таким образом, мы можем это представить следующим образом (Рисунок 2.4).

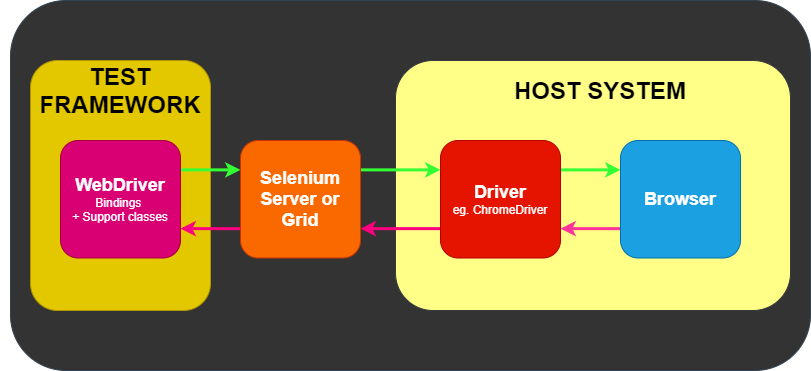


Рисунок 2.4 – Вариант взаимодействия Selenium Webdriver с браузером при условии использования тестового фреймворка

Selenium Webdriver состоит из набора определенных методов, каждый из которых обладает целевым предназначением. Только слаженная работа этих методов предоставляет разработчикам возможность создавать очень широкий набор функциональных тестов пользовательского интерфейса.

Далее приведем список основных методов, предусмотренных в Selenium Webdriver, необходимых для коммуникации с браузером, и краткое описание их предназначения. Разумеется, синтаксис для каждого из поддерживаемых языков программирования будет свой, мы же сосредоточимся на выбранном нами для реализации библиотеки JavaScript.

Первое, что вам нужно сделать после запуска браузера, - это открыть веб-сайт. Это можно сделать одной строкой:

await driver.get('https://selenium.dev')

Мы можем прочитать текущий URL-адрес из адресной строки браузера, используя:

await driver.getCurrentUrl();

Для нажатие кнопки возврата (Back) в браузере:

await driver.navigate().back();

Для нажатия кнопки перехода (Forward) в браузере:

await driver.navigate().forward();

Для обновления текущей страницы:

await driver.navigate().refresh();

Мы можем также прочитать заголовок текущей страницы из браузера:

await driver.getTitle();

Когда мы закончили наш сеанс браузера, мы должны вызвать quit:

await driver.quit();

Метод quit() выполнит следующие действия:

* Закроет все окна и вкладки, связанные с этим сеансом WebDriver;
* Закроет процесс браузера;
* Закроет процесс фонового драйвера;
* Уведомит Selenium Grid о том, что браузер больше не используется, чтобы его можно было использовать в другом сеансе (если мы используем Selenium Grid);

Если же не вызвать quit(), на нашем компьютере будут запущены дополнительные фоновые процессы и порты, что может вызвать проблемы позже.

Также одна из основ при работе с Selenium Webdriver – это поиск (locating) элементов на веб-страницах. WebDriver предлагает ряд встроенных типов селекторов, в том числе поиск элемента по его атрибуту ID. Пример:

const example = driver.findElement(By.id(example'));

Как видно из примера, поиск элементов в WebDriver выполняется на WebDriver-объекте экземпляра. findElement(By) метод возвращает другой фундаментальный тип объекта (WebElement).

WebDriver представляет браузер, в то время как WebElement представляет конкретный узел DOM (элемент управления, например, ссылку или поле ввода и так далее).

Если у нас уже есть ссылка на «найденный» веб-элемент, мы можем сузить область поиска, используя тот же вызов для этого экземпляра объекта:

const example = driver.findElement(By.id(example'));

const partofexample = example.findElement(By.id(partofexample));

Часто возникают ситуации, когда нам необходимо найти сразу все элементы, удовлетворяющие тому или иному условия. Для этого в Selenium Webdriver существует метод driver.findElements(). Этот метод возвращает коллекцию веб-элементов. Если найден только один элемент, он все равно вернет коллекцию (из одного элемента). Если ни один элемент не соответствует локатору, будет возвращен пустой список; данный нюанс в работе метода зачастую используется при формировании тестовых сценариев, так как метод не выдаст undefined ни при каких условиях, а следовательно наш тест не упадет.

В примерах выше мы уже рассмотрели один из возможных способов поиска (по id элемента). Однако таких способов значительно больше и каждый из них удобно использовать в тех или иных обстоятельствах. Рассмотрим данные способы в Таблице 2.1:

|  |  |
| --- | --- |
| **Локатор** | **Описание** |
| class name | Находит элементы, имя класса которых содержит значение поиска (составные имена классов не допускаются) |
| css selector | Находит элементы, соответствующие селектору CSS |
| id | Находит элементы, атрибут ID которых соответствует поисковому значению |
| name | Находит элементы, атрибут NAME которых соответствует поисковому значению |
| link text | Находит элементы привязки, видимый текст которых соответствует поисковому значению |
| partial link text | Находит элементы привязки, видимый текст которых содержит значение поиска. Если совпадают несколько элементов, будет выбран только первый. |
| tag name | Находит элементы, имя тега которых соответствует поисковому значению |
| xpath | Находит элементы, соответствующие выражению XPath |

Таблица 2.1 – Виды локаторов в Selenium Webdriver

В общем, если HTML-идентификаторы доступны, уникальны и постоянно предсказуемы, они являются предпочтительным методом поиска элемента на странице. Они, как правило, работают очень быстро и не требуют большой обработки, связанной со сложным обходом DOM.

Если уникальные идентификаторы недоступны, хорошо написанный селектор CSS является предпочтительным методом поиска элемента. XPath работает так же, как селекторы CSS, но синтаксис сложен и часто бывает трудно отладить. Хотя селекторы XPath очень гибкие, они, как правило, не тестируются на производительность поставщиками браузеров и, как правило, работают довольно медленно. Однако их принято считать самыми надежными.

Стратегии выбора, основанные на linkText и partialLinkText, имеют недостатки в том, что они работают только с элементами ссылки. Кроме того, они вызывают селекторы XPath внутри WebDriver.

Имя тега может быть опасным способом поиска элементов. На странице часто присутствует несколько элементов одного и того же тега. Это в основном полезно при вызове метода findElements (By), который возвращает коллекцию элементов.

Рекомендуется, чтобы локаторы были как можно более компактными и удобочитаемыми. Просить WebDriver обойти структуру DOM - дорогостоящая операция, и чем больше вы сможете сузить область поиска, тем лучше.

Также имеет смысл рассмотреть ожидания в Selenium Webdriver по тому или иному условию. В целом можно сказать, что WebDriver имеет блокирующий API. Поскольку это внепроцессная библиотека, которая указывает браузеру, что делать, и поскольку веб-платформа имеет асинхронную природу, WebDriver не отслеживает активное состояние DOM в реальном времени. Это связано с некоторыми проблемами, которые мы обсудим далее.

По опыту, большинство периодических проблем, возникающих при использовании Selenium и WebDriver, связаны с *условиями*, которые возникают между браузером и инструкциями пользователя. Примером может быть то, что пользователь дает браузеру команду перейти на страницу, а затем получает ошибку отсутствия такого элемента при попытке найти элемент.

Проблема здесь в том, что стратегия загрузки страницы по умолчанию, используемая в WebDriver, ожидает document.readyState изменения "complete" перед возвратом из вызова для перехода. Если элемент добавляется после завершения загрузки документа, этот сценарий в WebDriver может работать с перебоями. Он может быть прерывистым, потому что нельзя дать никаких гарантий относительно элементов или событий, которые запускаются асинхронно без явного ожидания или блокировки этих событий.

Чтобы преодолеть проблему состояния «гонки» между браузером и вашим скриптом WebDriver, большинство клиентов Selenium поставляются с пакетом ожидания. При использовании ожидания мы используем то, что обычно называют явным ожиданием.

Они позволяют нашему коду останавливать выполнение программы или замораживать поток до тех пор, пока условие, которое вы ему передаете, не разрешится. Условие вызывается с определенной частотой, пока не истечет время ожидания. Это означает, что пока условие возвращает ложное значение, оно будет пытаться и ждать.

Поскольку явные ожидания позволяют дождаться возникновения условия, они хорошо подходят для синхронизации состояния между браузером и его DOM, а также вашим скриптом WebDriver.

Мы передаем условие как ссылку на функцию, что ожидание будет выполняться повторно, пока его возвращаемое значение не станет истинным. «Правдивое» возвращаемое значение - это все, что оценивается как логическое истинное на данном языке, например строка, число, логическое значение, объект (включая WebElement) или заполненная (непустая) последовательность или список. Это означает, что пустой список считается ложным. Когда условие является истинным и блокирующее ожидание прерывается, возвращаемое значение из условия становится возвращаемым значением ожидания.

Условие ожидания можно настроить в соответствии с нашими потребностями. Иногда нет необходимости ждать полного тайм-аута по умолчанию, так как штраф за невыполнение успешного условия может быть дорогостоящим.

Ожидание позволяет нам передать аргумент для отмены тайм-аута:

await driver.wait(until.elementLocated(By.id('foo')), 30000)

Условия, доступные в разных языковых привязках, различаются, но это неполный список некоторых из них:

* предупреждение присутствует;
* элемент существует;
* элемент виден;
* заголовок содержит;
* название;
* устаревший элемент;
* видимый текст.

## **2.2. Формализация предметной области программного средства**

Разработанная библиотека “kbv-testdriver” на базе Selenium Webdriver для автоматизации тестирования UI и API веб-сайтов будет иметь структуру, как указано на рисунке 2.5.

Как видно из рисунка, директория «framework» включает в себя следующие папки:

* baseElement;
* basePage;
* browser;
* utils.

Непосредственно в данных папках содержаться классы, из которых состоит библиотека “kbv-testdriver”.

Для лучшего понимания выбранной структуры библиотеки необходимо разобрать такое понятие как Page object models или Page object pattern.

Объект страницы - это шаблон проектирования, который стал популярным в автоматизации тестирования для улучшения обслуживания тестов и уменьшения дублирования кода. Объект страницы - это объектно-ориентированный класс, который служит интерфейсом для страницы вашего автоматизированного теста. Затем тесты используют методы этого класса объекта страницы всякий раз, когда им нужно взаимодействовать с пользовательским интерфейсом этой страницы. Преимущество состоит в том, что при изменении пользовательского интерфейса для страницы не нужно менять сами тесты, нужно изменять только код внутри объекта страницы. Впоследствии все изменения для поддержки этого нового пользовательского интерфейса расположены в одном месте.

Шаблон дизайна объекта страницы дает следующие преимущества:

* Существует четкое разделение между тестовым кодом и кодом страницы, таким как локаторы (или их использование, если вы используете карту пользовательского интерфейса) и макет.
* Существует единый репозиторий для сервисов или операций, предлагаемых страницей, вместо того, чтобы эти сервисы были разбросаны по тестам.

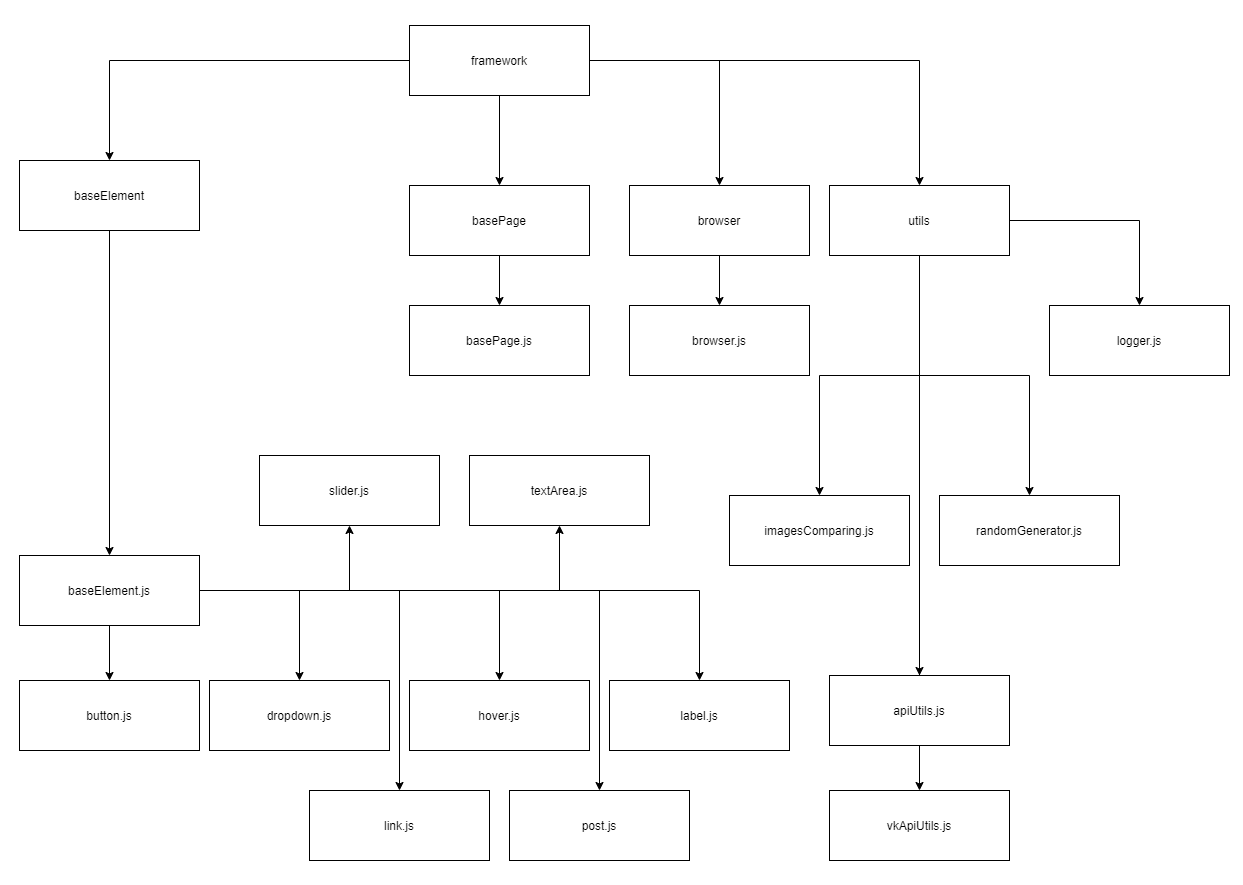


Рисунок 2.5 – Структурное изображение библиотеки «kbv-testdriver»

В обоих случаях это позволяет вносить любые изменения, необходимые из-за изменений пользовательского интерфейса, в одном месте.

Существует большая гибкость в том, как могут быть спроектированы объекты страницы, но есть несколько основных правил для достижения желаемой ремонтопригодности вашего тестового кода.

Сами объекты страницы никогда не должны делать проверки или утверждения. Это часть вашего теста и всегда должна быть в коде теста, а не в объекте страницы. Объект страницы будет содержать представление страницы и услуги, которые страница предоставляет с помощью методов, но никакой код, связанный с тем, что тестируется, не должен находиться в объекте страницы.

Существует одна, единственная проверка, которая может и должна быть в объекте страницы, а именно проверка того, что страница и, возможно, критические элементы на странице были загружены правильно. Эта проверка должна выполняться при создании экземпляра объекта страницы.

Объект страницы не обязательно должен представлять все части самой страницы. Те же принципы, что и для объектов страницы, можно использовать для создания «Объектов компонентов страницы», которые представляют дискретные фрагменты страницы и могут быть включены в объекты страницы. Эти объекты-компоненты могут предоставлять ссылки на элементы внутри этих дискретных блоков и методы для использования предоставляемых ими функциональных возможностей. Вы даже можете вкладывать объекты компонентов в другие объекты компонентов для более сложных страниц. Если страница в автоматизированных тестах имеет несколько компонентов или общих компонентов, используемых на всем сайте (например, панель навигации), это может улучшить ремонтопригодность и уменьшить дублирование кода.

Есть и другие шаблоны проектирования, которые также можно использовать при тестировании. Некоторые используют фабрику страниц для создания экземпляров своих объектов страницы.

Теперь мы лучше понимаем с какой целью библиотека «kbv-testdriver» имеет представленную на рисунке 2.5 структуру.

Дадим краткие пояснения относительно того, какие функции возложены на классы, расположенные в данной структуре:

1. baseElement – в данной директории представлены классы, которые ответственны за работу с различными элементами на веб-страницах (кнопки, выпадающие списки, ховер-элементы, надписи, ссылки, публикации (посты), слайдеры, области для ввода текста). Все классы перечисленных элементов наследуются от родительского класса BaseElement.
2. basePage - в данной директории содержится одноименный класс BasePage, который ответственен за работу со веб-страницами. Здесь важно отметить, что у него также есть классы-наследники, однако она располагаются в другой директории, а именно в project, что будет более подробно рассмотрено позже.
3. browser - в данной директории содержится одноименный класс Browser, который отвечает за работу Selenium Webdriver непосредственно с нашим браузером (открытие/закрытие браузера, переход на веб-страницы, увеличение открытого окна и так далее).
4. utils – как логично следует из названия, здесь располагаются все необходимые утилиты, предназначенные для работы с API, логировщик событий, генераторы случайных числе и так далее.

Для более глубокого понимая того какие действия выполняет тот или иной класс, подробно рассмотрим методы, имеющиеся в вышеописанных классах и приведем описание этих методов.

Класс BaseElement содержит следующие методы:

|  |  |
| --- | --- |
| **Метод класса** | **Описание метода** |
| findElement | Поиск элемента по локатору |
| click | Нажатие курсором мыши на элемент |
| getText | Получение текста элемента |
| getAttribute | Получение одного из атрибутов элемента |
| isElementDisplayed | Проверка отображения элемента |

Таблица 2.2 – Описание методов класса BaseElement

Как уже было упомянуто ранее, класс BaseElement имеет восемь классов-наследников. Многие из них не содержат в себе специфических методов, а только наследуют базовые из родительского класса BaseElement. Тем не менее, наличие данных классов обусловлено использованием в нашей библиотеке паттерна дизайна Page object pattern. Ниже будут рассмотрены только те классы-наследники, которые ы

## **2.3. Интеграция разработанной библиотеки с тестируемым веб-сервисом**

## **2.4. Описание и реализация используемых в библиотеке алгоритмов**

## **2.5 Тестирование программного средства**