ДЕПАРТАМЕНТ ОБРАЗОВАНИЯ И НАУКИ ГОРОДА МОСКВЫ
ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ГОРОДА МОСКВЫ
«МОСКОВСКИЙ КОЛЛЕДЖ УПРАВЛЕНИЯ, ГОСТИНИЧНОГО БИЗНЕСА И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ «ЦАРИЦЫНО»

ДИПЛОМНЫЙ ПРОЕКТ

Тема: Разработка комплекса автотестов для автоматизация тестирования web-сервисов

Пояснительная записка
ДП.09.02.07.41.25.2022 01 ПЗ

Руководитель проекта: Трутнев А.Е Дипломник: Левина К.Е

Рецензент: _____

Допустить к защите на заседании
Государственной экзаменационной комиссии

Учебно-методический отдел отделения _____

«____» _____2023 г. _____ Т.Н. Михайлова

Москва, 2023

Согласовано
На заседании Кафедры информационных технологий
Протокол № _____
«____» _____2022 г.
Заведующий кафедрой
_____

Утверждаю
УМО ОУИТ

от «____» _____2023 г.

_____

ЗАДАНИЕ НА ДИПЛОМНОЕ ПРОЕКТИРОВАНИЕ

Студенту _Левиной Ксении Евгеньевне_____

группы _____ специальности _____

Дата выдачи задания: «____» _____ 2023 г.
Срок сдачи проекта: «____» _____ 2023 г.

Тема дипломного проекта: Разработка комплекса автотестов для автоматизация тестирования web-сервисов ____

ТЕХНИЧЕСКОЕ ЗАДАНИЕ

Исходные данные:
1.
2.
3.

Содержание пояснительной записки

Раздел пояснительной записки	Объем, %
Введение	3%
1. 1.1.	

1.2.	
2. 2.1. 2.2. 2.3.	
2.4.	
3.	
Заключение	
Список используемой литературы	
Приложение 1 Приложение 2	

Графическая часть:

Лист 1. _____ Лист 2. _____

Мультимедиа презентация.

Дипломник: _____ /_____/

Руководитель проекта: _____ / Трутнев А.Е_____/

Заведующий кафедрой: _____ /_____/

СОДЕРЖАНИЕ

1 ВВЕДЕНИЕ

2 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

2.1 Уровни автоматизации. Место тестирования пользовательского интерфейса в общем процессе автоматизированного тестирования.

2.2 Подходы к автоматизации. Паттерн Page Object.

2.3 Исследование техник и методологий тестирования приложений

2.3.1 Техники тестирования веб-приложений

2.3.1 Методологии тестирования web приложения

2.3.3 Проблема автоматизации тестирования

2.6 выбор языка программирования, фреймворка, среды разработки для тестирования

2.6.1 Выбор языка программирования

2.6.2 Выбор фрейм-ворка

2.6.3 Выбор среды программирования

3 ПРАКТИЧЕСКАЯ ЧАСТЬ

3.1 Разработка логической модели автоматизации тестирования

3.2 Выбор технологий и спецификации

3.3 Выбор тестируемого проекта

3.4 Основные классы проекта

3.4.1 Класс BasePage

3.4.2 класс BasketPage

3.4.3 LoginPage

4 ЗАКЛЮЧЕНИЕ

5 СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

ДЕПАРТАМЕНТ ОБРАЗОВАНИЯ ГОРОДА МОСКВЫ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ГОРОДА МОСКВЫ «МОСКОВСКИЙ КОЛЛЕДЖ УПРАВЛЕНИЯ, ГОСТИНИЧНОГО БИЗНЕСА И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ «ЦАРИЦЫНО»

ОТЗЫВ

на дипломный проект студента _____
(фамилия, имя, отчество)

группы _____ специальности _____

на тему: _____

_____

В отзыве необходимо отразить:

1) Наименование темы проекта;
2) Количество страниц пояснительной записки и количество листов демонстрационного материала;
3) Краткий перечень основных вопросов, изложенных в записке и их характеристику;
4) Актуальность темы и реальность дипломного проекта, возможность использования в промышленности (рекомендован для внедрения, внедрен, материалы проекта используются на производстве, изготовлен макет и т.д.);
5) Достоинства и недостатки проекта;
6) Самостоятельность при решении поставленной задачи;
7) Оценку качества пояснительной записки и демонстрационного материала;
8) Общее заключение о выполнении дипломного проекта с оценкой по пятибалльной системе.

Выполненная работа заслуживает оценки Отлично

Место работы и должность руководителя дипломного проекта:

_____

Фамилия, имя, отчество руководителя дипломного проекта:

_____

«____» _____ 2023 г.

Подпись: _____

ДЕПАРТАМЕНТ ОБРАЗОВАНИЯ ГОРОДА МОСКВЫ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ГОРОДА МОСКВЫ «МОСКОВСКИЙ КОЛЛЕДЖ УПРАВЛЕНИЯ, ГОСТИНИЧНОГО БИЗНЕСА И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ «ЦАРИЦЫНО»

РЕЦЕНЗИЯ

на дипломный проект студента _____
(фамилия, имя, отчество)

группы _____ специальности _____

на тему: _____

_____

Рецензия должна включать:

1) Заключение о степени соответствия выполненного проекта техническому заданию;
2) Характеристику выполнения каждого раздела проекта;
3) Вывод об использовании последних достижений науки и техники, передовых методов работы;
4) Вывод о глубине экономических обоснований, принятых в проекте решений;
5) Перечень положительных сторон проекта и его основных недостатков (при наличии), оценку реального значения проекта;
6) Оценку качества оформления дипломного проекта;
7) Заключение о возможности использования работы студента на производстве, её практическом значении; 8) Отзыв о проекте в целом с оценкой по пятибалльной системе.

Выполненная работа заслуживает _____ оценки.

Место работы и должность рецензента:

_____

Фамилия, имя, отчество рецензента:

_____

«____» _____ 2022 г.

Подпись: _____

1 ВВЕДЕНИЕ

2 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

2.1 Уровни автоматизации. Место тестирования пользовательского интерфейса в общем процессе автоматизированного тестирования.

2.2 Подходы к автоматизации. Паттерн Page Object.

2.3 Исследование техник и методологий тестирования приложений

2.3.1 Техники тестирования веб-приложений

2.3.2 Методологии тестирования web приложения

2.3.3 Проблема автоматизации тестирования

2.4 Выбор языка программирования, фреймворка, среды разработки для тестирования

2.4.1 Выбор языка программирования

2.4.2 Выбор фрейм-ворка

2.4.3 Выбор среды программирования

3 ПРАКТИЧЕСКАЯ ЧАСТЬ

3.1 Разработка логической модели автоматизации тестирования

3.2 Выбор технологий и спецификации

3.3 Выбор тестируемого проекта

3.4 Основные классы проекта

3.4.1 Класс BasePage

3.4.2 класс BasketPage

3.4.3 LoginPage

4 ЗАКЛЮЧЕНИЕ

5 СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1 ВВЕДЕНИЕ

В настоящее время web-сервисы занимают важное место в современных информационных технологиях. Они позволяют обеспечить высокоскоростной и гибкий обмен данными между различными системами и устройствами. Однако, создание, развитие и поддержка web-сервисов требуют большого количества времени и ресурсов. Один из способов оптимизировать этот процесс - автоматизировать тестирование web-сервисов. Целью данной дипломной работы является создание шаблона для разработки комплекса автотестов для автоматизации тестирования web-сервисов на примере которого можно тестировать различные web-сервисы. Данный комплекс позволит оптимизировать процесс тестирования и обеспечит быстрое и качественное тестирование web-сервисов.

В рамках данной работы были поставлены следующие задачи:

1. изучить теоретический материал по созданию и тестированию web-сервисов;
2. выбрать инструменты для автоматизации тестирования web-сервисов;
3. разработать автотесты для тестирования различных функциональных возможностей web-сервисов;
4. провести тестирование созданного комплекса автотестов на примере выбранного web-сервиса.

Объектом исследования являются web-сервисы, а предметом исследования - процесс их тестирования и автоматизации этого процесса.

В работе используется метод эксперимента с использованием существующего web-сервиса и инструментов для автоматизированного тестирования.

Мы живем в эпоху информационных технологий, и web-сервисы занимают важное место в современных IT-решениях. Это обусловливает актуальность задачи автоматизации тестирования web-сервисов, которая постоянно набирает

обороты. Эффективное тестирование и обеспечение качества являются ключевыми задачами в создании и поддержании в работоспособности web-сервисов.

Теоретическая ценность данной работы заключается в исследовании технологий и инструментов для автоматизации тестирования web-сервисов. Работа направлена на анализ и оценку существующих методов и инструментов тестирования web-сервисов. Полученные знания и опыт могут быть использованы в качестве руководства или рекомендаций при создании и поддержании web-сервисов.

Для достижения данной цели были опробованы современные инструменты автоматизации тестирования web-сервисов. Разработанный комплекс автотестов может быть использован для тестирования различных web-сервисов, в том числе для оценки функциональности и производительности. Результат работы может быть использован в качестве справочного материала и руководства для технических специалистов и разработчиков web-сервисов.

Одним из важных результатов работы является практическая ценность. Были разработаны автотесты для проверки различных функциональных возможностей web-сервисов. Полученный комплекс автотестов может быть использован в качестве шаблона тестирования web-сервисов различных типов и сложности, что позволит значительно сократить время и затраты на тестирование и улучшить качество и производительность web-сервисов.

Актуальность данной работы обусловлена современными тенденциями в сфере информационных технологий, в частности, в области web-сервисов. Это быстро развивающееся направление, которое требует высокой производительности и стабильности работы. Одним из ключевых аспектов обеспечения высокой производительности и надежности работы web-сервисов является их тестирование.

Автоматизация тестирования web-сервисов становится все более актуальной задачей, так как это позволяет оптимизировать процесс тестирования и повысить качество работы web-сервисов. Кроме того, автоматизация тестирования позволяет решать задачи тестирования в общих случаях, когда тестовые данные необходимо проверять путем автоматического выполнения большого количества проверок.

Кроме того, автоматизация тестирования web-сервисов может значительно увеличить эффективность процесса разработки. Повышение качества и производительности с помощью автоматизации тестирования может ускорить процесс создания web-сервисов, а также повысить надежность и стабильность работы в долгосрочной перспективе.

Наконец, автоматизация тестирования web-сервисов обеспечивает повышение качества и производительности, что важно для конечных пользователей. Решение проблем, связанных с низким качеством и производительностью web-сервисов, поможет улучшить пользовательский опыт и повысить удовлетворенность клиентов.

В заключение можно сказать, что данная работа имеет как теоретическую, так и практическую ценность. Полученные результаты и знания могут быть использованы для улучшения качества и производительности web-сервисов, а также для развития технологий и методов тестирования в целом.

Объектом исследования работы являются web-сервисы. Предметом исследования тестирование и автоматизация тестирования web-сервисов. А его целью разработка комплекса автотестов для автоматизации тестирования web-сервисов.

Данная работа решает следующие задачи исследования:

1.Изучение теоретического материала по созданию и тестированию web-сервисов.

2.Выбор инструментов для автоматизации тестирования web-сервисов.

3.Разработка автотестов для тестирования различных функциональных возможностей web-сервисов.

4.Проведение тестирования созданного комплекса автотестов на различных web-сервисах.

Методы исследования:

1.Анализ литературы по созданию и тестированию web-сервисов

2.Сравнительный анализ инструментов для автоматизации тестирования web-сервисов

3.Разработка автотестов с использованием выбранного инструмента

4.Тестирование комплекса автотестов на web-сервисах

Исследование будет проводиться в несколько этапов:

1.Изучение теоретического материала по созданию и тестированию web-сервисов

2.Сравнительный анализ инструментов для автоматизации тестирования web-сервисов

3.Разработка автотестов для различных функциональных возможностей web-сервисов с использованием выбранного инструмента

4.Тестирование комплекса автотестов на различных web-сервисах

5.Анализ и обработка результатов тестирования, проверка их соответствия заявленным критериям качества работы web-сервисов

В рамках исследования будут использоваться следующие инструменты:

1.Язык программирования для написания автотестов

2.Инструменты автоматизации тестирования web-сервисов

3.Web-сервисы для проведения тестирования

Ожидаемыми результатами исследования являются:

1.Разработка комплекса автотестов для автоматизации тестирования web-сервисов

2.Описание выбранного инструмента для автоматизации тестирования web-сервисов

3.Анализ и обработка результатов тестирования

Структура работы:

-Введение

-Теоретическая часть

-Заключение

-Заключение

В теоретической части работы будет описан существующий опыт в области тестирования web-сервисов и сравнительный анализ существующих инструментов для автоматизации тестирования web-сервисов. В этой же части будет приведено описание выбранного инструмента для автоматизации тестирования.

В практической части работы будет представлено описание комплекса автотестов для тестирования различных функциональных возможностей web-сервисов. Описание будет включать описание структуры и архитектуры комплекса, а также примеры написания автотестов.

В заключении будут сделаны выводы по результатам исследования и описаны дальнейшие возможности для развития данного направления.

2 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

В настоящее время автоматизация тестирования программного обеспечения является неотъемлемой частью разработки. Использование специализированных инструментов и языков программирования может значительно повысить эффективность и скорость тестирования. Цель данной работы - исследовать возможности автоматизации тестирования на языке Python и разработать систему автоматизированного тестирования для тестирования программного обеспечения. В работе будет рассмотрен процесс разработки системы, ее функциональность и эффективность в сравнении с традиционными методами тестирования. Результаты работы могут быть полезны для разработчиков и тестировщиков, которые занимаются созданием качественного программного обеспечения.

Основным предметом исследования является тестирование web-сервисов. Тестирование web-сервисов - это процесс проверки работоспособности веб-приложения, которое предоставляет данные и функциональность через интернет с использованием API (Application Programming Interface). Также будет рассмотрено использование библиотеки Requests для автоматизации тестирования API. В работе будут рассмотрены основные протоколы, такие как REST и SOAP, а также приведены примеры кода для тестирования API с использованием библиотеки Requests.

Основные этапы тестирования веб-сервисов включают в себя:

1. **Unit тестирование API-методов** API-методы являются основой web-сервисов, их обычно тестируют отдельно от остальной функциональности приложения. Используя unittest или pytest, вы можете написать unit-тесты для каждого API-метода, которые проверят корректность обработки входных параметров, корректность возвращаемых значений и обрабатывают ошибки.

2. **Интеграционное тестирование** Для полного тестирования web-сервиса, необходимо провести интеграционное тестирование, которое проверит работу приложения в условиях реального использования. Например, можно создать тестовые сценарии, которые эмулируют действия пользователя, и проверить правильность обработки данных, валидацию входных параметров и корректность возвращаемых значений.
3. **Тестирование производительности** Тестирование производительности необходимо для проверки, как быстро web-сервис работает при большом количестве запросов. Для этого можно использовать инструменты для тестирования нагрузки, такие как Locust, Apache JMeter или Gatling. Они позволяют эмулировать большое количество пользователей, которые делают запросы к API, и позволяют измерять время ответа.
4. **Тестирование безопасности** Тестирование безопасности очень важно для веб-сервисов. Необходимо проверить, что никакая чувствительная информация не передается в открытом виде, и что пользователи не имеют доступа к данным, которые им не положены. Можно использовать инструменты для тестирования безопасности, такие как OWASP ZAP, для автоматического сканирования web-сервиса на наличие уязвимостей.
5. **Тестирование совместимости** Тестирование совместимости необходимо для проверки, как web-сервис работает на различных платформах и устройствах. Необходимо проверить, что web-сервис работает корректно в разных браузерах и на разных операционных системах.

В целом, тестирование веб-сервисов требует комплексного подхода и использования различных инструментов и методов тестирования. Важно также учитывать особенности конкретного web-сервиса и его функциональности при выборе инструментов и методов тестирования.

Результаты данной работы помогут разработчикам и тестировщикам создавать эффективные и надежные системы автоматизированного тестирования на языке Python, что позволит существенно ускорить процесс разработки и повысить качество программного обеспечения.

Тестирование - это важный этап разработки программного обеспечения, который помогает обеспечить его качество. Качество программного обеспечения охватывает различные свойства, которые делают его пригодным для удовлетворения потребностей пользователей. Кроме того, программное обеспечение должно соответствовать стандартам и ожиданиям, чтобы считаться качественным. Тестирование помогает выявлять ошибки в программе. Ошибки - это недочеты в разработке программного обеспечения, которые могут привести к несоответствию ожидаемых и фактических результатов. Автоматизированное тестирование пользовательского интерфейса - это эффективный способ тестирования программного обеспечения, который позволяет быстро и эффективно выявлять ошибки и улучшать качество продукта.

Некоторые из основных преимуществ автоматизированного тестирования пользовательского интерфейса:

- Эффективность: автоматизированные тесты могут выполняться гораздо быстрее, чем ручное тестирование, что позволяет ускорить процесс разработки и повысить его эффективность.
- Повторяемость: автоматические тесты гораздо более надежны и повторяемы, чем ручные, что позволяет быстро выявлять и исправлять ошибки на разных этапах разработки.
- Обширность: автоматизированные тесты могут проверять множество тестовых сценариев, что позволяет выявлять даже скрытые ошибки или проблемы взаимодействия интерфейса с другими компонентами программы.

- Масштабируемость: автоматизированные тесты могут легко масштабироваться для проверки больших объёмов данных или различных версий приложения.
- Экономическая эффективность: автоматизация тестирования позволяет существенно сэкономить затраты на тестирование в долгосрочной перспективе, особенно в случаях, когда необходимо тестировать продукт на регулярной основе или при большом объёме разработки.

На практике, тестирование программного обеспечения может производиться путем выполнения определенного набора действий в приложении, получения результатов и сверки их с данными, определенными как эталонные. Этот процесс может выполняться вручную тестировщиками или автоматически при помощи инструментов для автоматизированного тестирования. Этот процесс называется автоматизированным тестированием программного обеспечения. Автоматизированное тестирование имеет свои преимущества, включая быструю скорость выполнения, отсутствие влияния человеческого фактора и возможность выполнения тест-кейсов особенной сложности. Однако, у автоматизированного тестирования есть и недостатки, включая необходимость привлечения высококвалифицированного персонала и затраты на средства автоматизации, разработку и сопровождение тестов, а также риск устаревания тестов при изменении требований.

Автоматизация тестирования не полностью исключает ручное тестирование, но значительно сокращает его долю. Она помогает убрать рутину из процесса тестирования, снизить стоимость и улучшить качество программных продуктов. Тестирование является одной из наиболее трудоемких фаз разработки ПО, и ее стоимость растет с течением времени. Многие специалисты по тестированию работают в условиях ограниченного времени и ограниченных ресурсов, что приводит к проблемам со слишком большим объемом работы и недостаточным

количеством времени для качественного тестирования. Автоматизация тестирования позволяет решить эти проблемы и становится все более востребованной с ростом требований к качеству и сложности программных продуктов.

2.1 **Уровни автоматизации. Место тестирования пользовательского интерфейса в общем процессе автоматизированного тестирования.**

На сегодняшний день существуют различные подходы к автоматизации тестирования. Комплексная стратегия тестирования включает в себя 3 уровня автоматизации:

1. уровень модульного тестирования (Unit Tests Layer) – компонентные тесты, обычно пишутся разработчиками;

2. уровень функционального тестирования (Functional Test Layer, orService / API Tests Layer) – тестирование через доступ к функциональному слою, минуя пользовательский интерфейс;

3. уровень тестирования пользовательского интерфейса, или через пользовательский интерфейс (GUI Test Layer) – дает возможность тестировать не только сам интерфейс пользователя, но и реализовывать функциональное тестирование, выполняя операции, использующие бизнес логику приложения. Для обеспечения наилучшего качества приложения рекомендуется автоматизировать все три уровня. На рисунке 1 приведена классическая пирамида автоматизированного тестирования [7].

Данная пирамида показывает идеальное соотношение тестов различного уровня для среднестатистического проекта. Тесты верхнего уровня считаются более

сложными в разработке и дорогостоящими. Соотношение может меняться в зависимости от специфики тестируемого продукта.



Рисунок 1 – Классическая пирамида автоматизации

2.2 Подходы к автоматизации. Паттерн Page Object.

Существуют 4 основных подхода к автоматизации тестирования пользовательского интерфейса. Первый подход - запуск тестов при помощи утилит записи и воспроизведения. Это неэффективный метод, который требует перезаписи тестов при любых изменениях интерфейса. Второй подход - написание тестовых сценариев на языках программирования. Этот метод решает проблемы масштабирования и поддержки тестов, но более дорогостоящий и требует большего количества ресурсов. Третий подход - управление данными тестирования, который используется для реализации однотипных видов

проверок для множественных вариантов входных данных. Четвертый подход - паттерн Page Object, который предлагает вынести логику работы с элементами пользовательского интерфейса в отдельный слой, что снижает заимствование кода и позволяет легче понимать структуру тестов.

При написании тестовых скриптов обычно применяют технологию фреймворков. Фреймворк – это организация проекта, позволяющая упростить разработку, модификацию и поддержку программного кода. В случае организации проекта по автоматизации тестирования пользовательского интерфейса приходится считаться с тем, что тесты потребуется поддерживать, а возможно и существенно переписывать в случае изменений в интерфейсе тестируемого приложения. Основная задача, решаемая через использование фреймворков – снижение количества изменений, которые требуется внести в тесты при изменениях в тестируемом приложении.

Подобная организация проекта позволяет значительно упростить поддержку тестов и снизить дублирование кода.

Основные идеи данного шаблона проектирования:

- четкое разделение между непосредственно кодом тестов и кодом, специализирующимся на работе с локаторами (путями к элементам интерфейса);
- наличие единого репозитория для всех служб и операций, представленных на странице, а не множества разбросанных по всему тестовому набору. Мартин Фаулер объясняет данный паттерн на простом примере, приведенном на рисунке 2.

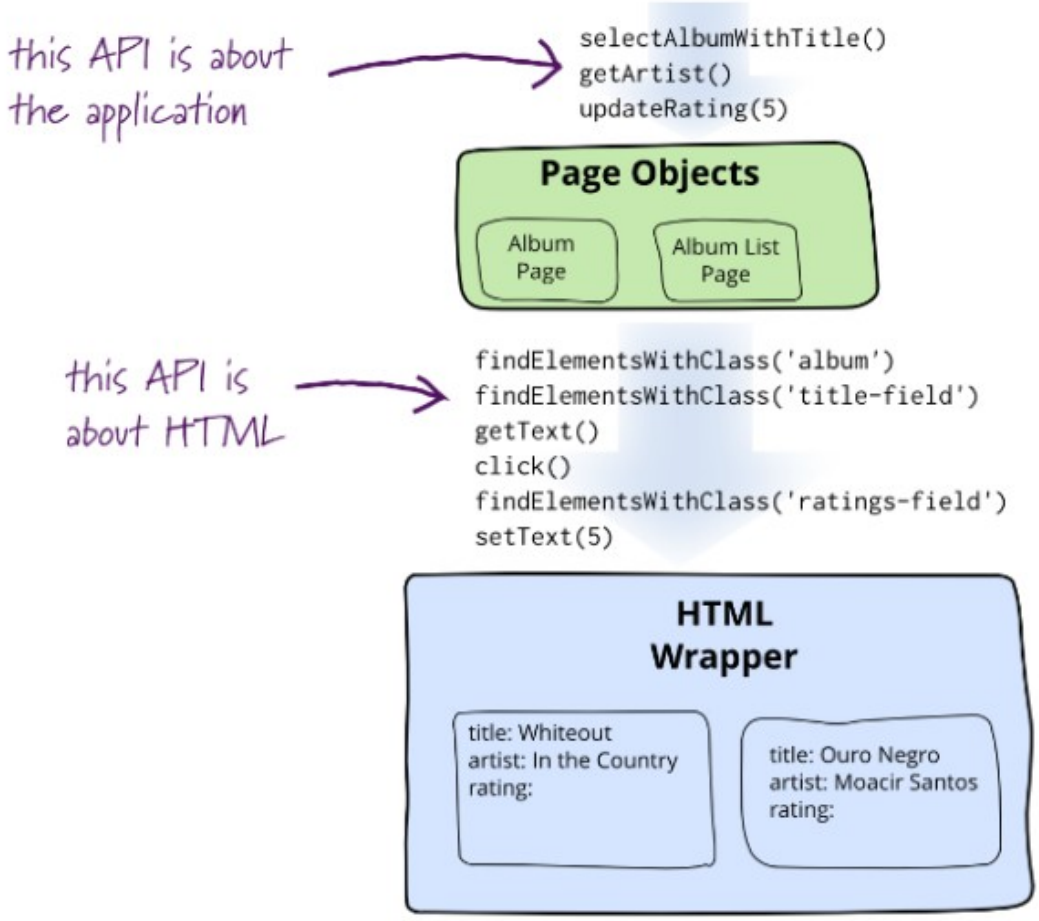


Рисунок 2 – Логика разделения кода в паттерне Page Object

Паттерн Page Object был разработан для тестирования веб-приложений, но его идеи используются сегодня во всех остальных видах автоматизированного тестирования, существуют различные разновидности реализации данного шаблона. Основным преимуществом его использования является то, что при изменениях в пользовательском интерфейсе достаточно модифицировать код тестов только в одном месте

В результате изучения подходов к автоматизации тестирования пользовательского интерфейса были выявлены четыре основных подхода: утилиты записи и воспроизведения, написание тестовых сценариев на языках программирования, управление данными тестирования и паттерн Page Object. Первый подход является неэффективным, так как требует перезаписи тестов при каждом изменении интерфейса. Второй подход решает проблемы масштабирования и поддержки тестов, но более затратен и требует большего количества ресурсов. Третий подход используется в ситуациях, когда необходимо реализовать однотипные виды проверок для множественных вариантов входных данных. Четвертый подход, паттерн Page Object, предлагает вынести логику работы с элементами пользовательского интерфейса в отдельный слой, что уменьшает заимствование кода и обеспечивает более легкое понимание структуры тестов.

При написании тестовых сценариев используются технологии фреймворков, которые позволяют упростить разработку, модификацию и поддержку программного кода. Организация проекта по автоматизации тестирования приложений позволяет снизить количество изменений, которые требуется вносить в тесты при изменениях в тестируемом приложении, и уменьшить дублирование кода.

Использование паттерна Page Object и фреймворков при автоматизации тестирования web-сервисов позволяет существенно повысить качество и надежность тестов, сократить время выполнения и уменьшить трудозатраты на их поддержку. Более того, фреймворки позволяют проводить автоматизированное тестирование на различных конфигурациях и устройствах, что увеличивает эффективность и практическую ценность данного процесса.

2.3 Исследование техник и методологий тестирования приложений

2.3.1 Техники тестирования веб-приложений

Тестирование программного обеспечения (Software Testing) – проверка соответствия между реальным и ожидаемым поведением программы, осуществляемая на конечном наборе тестов, выбранном определенным образом. В более широком смысле, тестирование является одной из техник контроля качества, которая включает в себя активности по планированию работ, проектированию тестов, выполнению тестирования и анализу полученных результатов. Рассмотрим виды тестирования ПО. Все виды тестирования программного обеспечения, в зависимости от преследуемых целей, можно условно разделить на следующие группы:

1. Функциональные;
2. Нефункциональные;
3. Связанные с изменениями.

Рассмотрим каждый отдельный вид тестирования, его назначение и использование при тестировании ПО. Функциональные тесты основываются на функциях и взаимодействии с другими системами. Тесты также могут быть представлены на всех уровнях тестирования: компонентном или модульном, интеграционном, системном и приемочном. Функциональные виды тестирования рассматривают внешнее поведение системы. Рассмотрим одни из самых распространенных видов функциональных тестов:

1. Тестирование взаимодействия;

2. Тестирование безопасности;

Нефункциональные тесты, необходимые для определения характеристик программного обеспечения, которые могут быть измерены различными величинами. В широком смысле, нефункциональное тестирование – это тестирование того, как работает система. Основные виды нефункциональных тестов:

1. Тестирования производительности:

a. нагрузочное тестирование;

b. тестирование стабильности или надежности;

c. объемное тестирование.

2. Тестирование установки;

3. Тестирование удобства использования;

4. Тестирование на отказ и восстановление;

5. Конфигурационное тестирование.

После проведения необходимых изменений, таких как исправление дефекта, ПО должно быть перетестировано для подтверждения, что проблема была действительно исправлена.

2.3.2 Методологии тестирования web приложения

В соответствии со стандартом ИСО/МЭК 12207-2010 [4] процессы жизненного цикла программных средств включают в себя следующие этапы (рассматриваются этапы, касающиеся процессов реализации и технических процессов):

1. Определение требований.

2. Анализ требований.

3. Проектирование архитектуры.

4. Детальное проектирование.

5. Конструирование программного средства.

6. Комплексирование программных средств.

7. Квалификационное тестирование.

8. Инсталляция.

9. Приемка.

10. Сопровождение.

11. Прекращение применения программного средства.

Данная работа касается рассмотрения этапов квалификационного тестирования (7 пункт), а также приемки программных средств (9 пункт). «Цель процесса квалификационного тестирования системы заключается в подтверждении того, что реализация каждого системного требования тестируется на соответствие и система готова к поставке» [4]. «Цель процесса поддержки приемки программных средств заключается в содействии приобретающей стороне в обеспечении уверенности в том, что продукт соответствует заданным требованиям» . Таким образом, оба этапа – и квалификационное тестирование и приемка служат схожим целям – проверка готовности ПО и соответствия его требованиям. В соответствии со стандартом IEEE SWEBOK v3.0 «тестирование программного обеспечения (Software Testing) – это динамическая проверка соответствия между реальным и ожидаемым поведением программы, осуществляемая на конечном наборе тестов, выбранном определенным

образом» [12]. Тестирование является одной из составляющих процесса контроля качества и включает в себя:

- планирование процесса тестирования (Test Management);

- разработку и проектирование тестов (Test Design);

- прогон тестовых сценариев (Test Execution);

- анализ результатов, полученных после прогонов тестов (Test Analysis).

В соответствии с пособием по стандартизации и сертификации программного обеспечения «все виды тестирования программного обеспечения, в зависимости от преследуемых целей, можно условно разделить на следующие группы: 1) функциональные; 2) нефункциональные; 3) связанные с изменениями» [7]. «Функциональные тесты базируются на функциях и особенностях, а также взаимодействии с другими системами, и могут быть представлены на всех уровнях тестирования: компонентном или модульном (Component/Unit testing), интеграционном (Integration testing), системном (System testing) и приемочном (Acceptance testing). Функциональные виды тестирования рассматривают внешнее поведение системы. Далее перечислены самые распространенные виды функциональных тестов:

- Тестирование безопасности (Security and Access Control Testing);

- Тестирование взаимодействия (Interoperability Testing)» [7].

«Нефункциональное тестирование описывает тесты, необходимые для определения характеристик программного обеспечения, которые могут быть измерены различными величинами. В целом, это тестирование того, "Как" система работает.

2.3.3 Проблема автоматизации тестирования

Проблема автоматизации тестирования программного обеспечения заключается в том, что при разработке и тестировании программного обеспечения возникает множество задач, которые должен решать тестировщик. Одной из эффективных практик для решения этой проблемы является автоматизированное тестирование программного обеспечения. Автоматизированное тестирование программного обеспечения представляет собой процесс верификации программного обеспечения, который выполняется при помощи инструментов для автоматизированного тестирования. Технический специалист, отвечающий за создание, отладку и поддержку тестовых скриптов, тестовых наборов и инструментов для автоматизированного тестирования, называется специалистом по автоматизированному тестированию программного обеспечения. Инструмент для автоматизированного тестирования - это программное обеспечение, при помощи которого специалист по автоматизированному тестированию создает, отлаживает, запускает и анализирует результаты прогона тестовых скриптов.

Тестовый скрипт - это набор инструкций, используемых для автоматической проверки определенной части программного обеспечения.

Тестовый набор - это комбинация тестовых скриптов, объединенных общей функциональностью или целями, преследуемыми при запуске данного набора.

Тесты для запуска (Test Run) - это комбинация тестовых скриптов или тестовых наборов для последующего совместного запуска (последовательного или параллельного, в зависимости от преследуемых целей и возможностей инструмента для автоматизированного тестирования)» [8].

Для написания и работы данных артефактов удобнее всего применять общий фреймворк автоматизированного тестирования. Основной проблемой при автоматизации является подготовка и контроль перечисленных артефактов, а также настройка необходимой среды для запуска и выполнения автоматических тестов. При этом есть проблема вовлечения в процесс поддержки автоматических тестов как можно большего количества специалистов, занимающихся тестированием, связанная с возможным недостатком квалификации или специализированных знаний языков и сред разработки.

В ходе анализа деятельности по тестированию программного обеспечения была разработана модель, которая показывает текущую технологию тестирования, которая предполагает ручное тестирование, как нового функционала, так и смоук и регрессионных тестов, что требует значительных человеческих ресурсов. Вы можете рассмотреть схему деятельности IDEF0 отдела тестирования ПО и декомпозицию текущих этапов/ операций деятельности отдела тестирования DFD, которые представлены на рисунках 3.1 и 3.2.



Рисунок 3.1. Схема процесса работы отдела тестирования

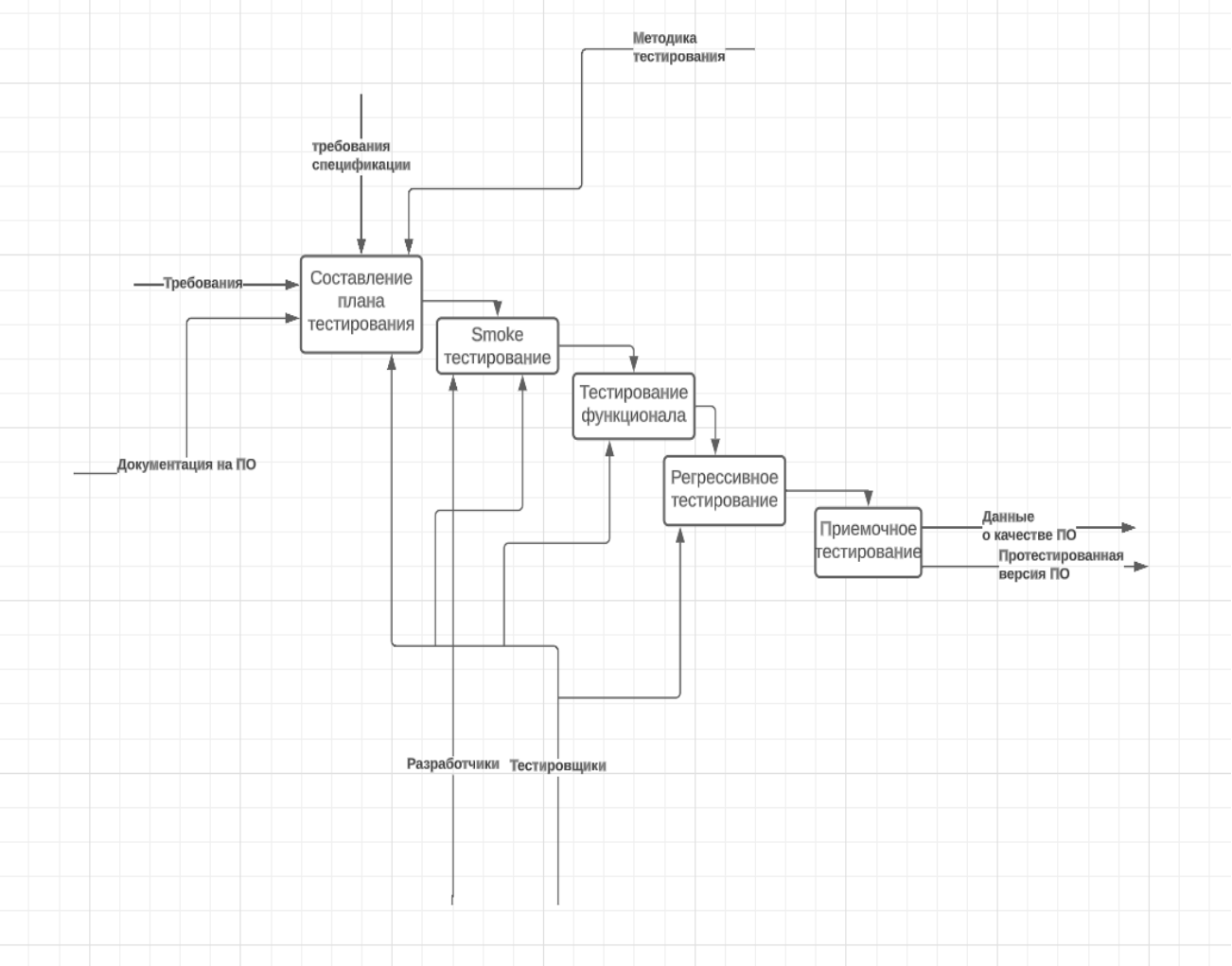


Рисунок 1.2. Декомпозиция схемы процесса работы отдела тестирования

Выявленные недостатки по данной схеме процесса:

- низкая производительность труда специалистов по тестированию;

- низкое качество отчетной информации из-за возможных ошибок специалистов при ручном отражении результатов прогонов тестов;

- высокая трудоемкость тестирования крупных релизов;

- несовершенство организации первичных тестовых данных разными сотрудниками;

- несовершенство процесса анализа возникающих при тестировании проблем и ошибок;

- разработчикам приходится принимать участие в процессе тестирования на этапе смоук тестирования, так как установка поставляемого ПО проводится в ручном режиме без использования методики Continuous Integration/Delivery.

Заказчик представлен на схеме в качестве участника приемочного тестирования продукта. Проблема увеличения эффективности приемочного тестирования также может рассматриваться в рамках проблематики данной работы

Как итог можно сделать вывод что для написания автоматических тестов, большинство инструментов автоматизации достаточно сложны и требуют понимания скриптового языка (VB Script, Java Script, и т.д.). Обычно инструменты позволяют создавать тесты при помощи записи и воспроизведения, но, как правило, такие скрипты не очень эффективны, не могут быть переиспользованы и тяжелы в поддержке. Исходя из ряда описанных выше проблем было решено использовать готовый фреймворк автоматического тестирования на языке Python. Чаще всего фреймворк - это набор условий, концепций и практик, направленный на переиспользование, уменьшение затрат на поддержку и повышение надежности использования тестов. Использование фреймворка решает большинство описанных проблем, что предполагает использование инструмента широким кругом специалистов, включая разработчиков и специалистов по ручному тестированию.

2.4 Выбор языка программирования, фреймворка, среды разработки для тестирования

2.4.1 Выбор языка программирования

Выбор языка программирования для написания комплекса автотестов очень важный аспект работы и ему необходимо уделить особое внимание. Важными характеристиками языка программирования в данной сфере деятельности являются быстродействие, гибкость и простота освоения, а также популярность языка среди разработчиков.
Исходя из данных сайта https://www.statista.com/, а конкретно опроса рекрутеров летом 2022 года о самых популярных языках программирования среди тестировщиков можно выделить три основных языка:

Python, Java, C# и JavaScript - языки программирования, которые широко используются для тестирования web-сервисов.Рассмотрим каждый по отдельности

Java - популярный язык программирования для тестирования web-сервисов. Java имеет большое количество инструментов для тестирования, таких как JUnit и TestNG, и может быть использован для настройки и запуска тестов на различных платформах. Java также имеет множество фреймворков и библиотек для тестирования web-сервисов, что делает его удобным и гибким выбором для написания тестовых сценариев. Однако Java может быть сложным в изучении и требует больше времени и усилий для работы с ним. JavaScript - это язык программирования, который может быть использован для тестирования web-сервисов, особенно для тестирования пользовательского интерфейса (UI). JavaScript имеет множество библиотек и фреймворков для тестирования UI, таких как Jest и Mocha. Он также может быть использован для написания тестов для API и других протоколов, включая REST и SOAP. Однако JavaScript может

быть менее эффективным при работе с более сложными задачами, и его синтаксис менее лаконичным, чем у аналогов, что может сделать его неудобным для написания тестовых сценариев. Также JavaScript не имеет стандартной библиотеки, как Java, и для работы с API и другими протоколами, будут требоваться сторонние библиотеки.

C# - это язык программирования, который может быть использован для тестирования web-сервисов. C# имеет множество библиотек и фреймворков, которые могут быть использованы для тестирования, включая библиотеку NUnit и фреймворк MSTest. Он также имеет возможность работы с многопоточностью и асинхронностью, что делает его эффективным для написания тестовых сценариев.

Однако, C# может быть не таким гибким, и его использование может быть ограничено в зависимости от платформы. Также C# может быть менее удобным для работы с REST API. Кроме того, C# может быть менее популярным и менее доступным, чем некоторые из других языков программирования, в зависимости от региона и отрасли. В целом, C# может быть хорошим выбором для тестирования web-сервисов, особенно если у вас уже есть опыт работы с этим языком программирования и доступны все необходимые инструменты и библиотеки. Однако, если у вас нет такого опыта или если вы хотите использовать более гибкий и доступный язык программирования, то Python может быть более оптимальным выбором.

Python предоставляет простой и лаконичный синтаксис, который позволяет разрабатывать тесты для web-сервисов более эффективно и быстро. Он также предоставляет большое количество библиотек и фреймворков для тестирования web-сервисов, что делает его более удобным и гибким при написании тестовых сценариев. Python также имеет высокую

производительность и может быть использован для тестирования различных протоколов и форматов, включая работу с API.

Таким образом, каждый из этих языков программирования предоставляет ряд преимуществ и недостатков при тестировании web-сервисов, и выбор нужного языка будет зависеть от конкретной задачи и условий проекта.

Для данной работы понимая значительный ряд преимуществ и анализируя поставленную задачу был выбран язык программирования Python, теперь необходимо выбрать фреймворк и соответствующую среду программирования.

2.4.2 Выбор фрейм-ворка

В языке Python существует множество различных инструментов тестирования, но основными являются Pytest и Unittest. Это два наиболее популярных фреймворка для тестирования на языке Python.

unittest - это модуль стандартной библиотеки Python, который предоставляет набор инструментов и методов для написания тестов. Он позволяет создавать тестовые кейсы и проверять поведение функций, методов и классов.

pytest - это более современный и более гибкий фреймворк для тестирования на языке Python. Он обладает более удобным синтаксисом и множеством полезных функций. Он позволяет быстро запускать тесты и автоматически находить и запускать все файлы с тестами в проекте.

В рамках данной дипломной работы мы сравним эти два инструмента и проанализируем их достоинства и недостатки. Будут рассмотрены следующие аспекты:

- Синтаксис и удобство написания тестов
- Возможности для создания фикстур и моков

- Возможности для организации тестов в тестовые сьюты и группы
- Интеграция с другими инструментами автоматизации тестирования

Также будут рассмотрены примеры кода и описаны основные методы и функции, которые могут быть использованы при тестировании различных типов приложений.

1.Синтаксис и удобство написания тестов

pytest предоставляет более простой и удобный синтаксис для написания тестов. Например, в pytest можно использовать декораторы для указания параметров тестов и для запуска тестов с определенными условиями. Также pytest обладает удобной системой выравнивания вывода тестов, которая помогает быстро понимать, какие тесты прошли, а какие нет.

unittest, напротив, требует более избыточного и сложного синтаксиса для написания тестов. Однако, unittest позволяет создавать более сложные и гибкие тестовые сьюты и группы, что может быть полезно для больших проектов.

2.Возможности для создания фикстур и моков

pytest позволяет создавать более гибкие и эффективные фикстуры (fixture) и моки (mock), которые могут быть использованы для ускорения и упрощения тестирования. Фикстуры и моки в pytest выполняются в определенном порядке и могут быть переиспользованы в разных тестах. Это позволяет сильно ускорить процесс тестирования.

unittest также позволяет создавать фикстуры и моки, однако, синтаксис их создания более сложный, и они не так гибки.

3.Возможности для организации тестов в тестовые сьюты и группы

unittest позволяет более гибко организовывать тестовые сьюты и группы, что может быть полезно для больших проектов. Однако, синтаксис создания этих сьютов и групп более сложный, и они требуют больше времени и усилий для создания.

pytest, напротив, предоставляет более легкий и простой способ создания тестовых сьютов и групп, что делает его более удобным для маленьких и средних проектов.

4.Интеграция с другими инструментами автоматизации тестирования

pytest имеет более обширный набор инструментов и плагинов, которые можно использовать для интеграции с другими инструментами автоматизации тестирования, такими как Selenium, Pytest-BDD, Pytest-HTML, и многими другими. Это делает его более гибким и удобным для интеграции с другими инструментами.

unittest тоже поддерживает интеграцию с другими инструментами автоматизации тестирования, однако, не так широко их поддерживает, как pytest.

Итак, с учетом вышеперечисленного, pytest является более современным, легким и гибким фреймворком для тестирования на языке Python, который предоставляет более простой и удобный синтаксис для написания тестов. unittest же более избыточен, но при этом более гибок и позволяет создавать более сложные фикстуры и группы тестов.

2.4.3 Выбор среды программирования

Существует множество сред программирования для Python, которые обеспечивают различные функциональности для разработки, отладки,

тестирования и совместной работы. Рассмотрим некоторые из наиболее популярных сред программирования для Python и сравним их:

PyCharm - один из наиболее популярных интегрированных сред программирования для Python, который обеспечивает множество функциональностей. Он имеет широкую поддержку языка Python, поддержку отладки, подсветку синтаксиса, а также мощный редактор кода с возможностью автодополнения и рефакторинга. PyCharm также интегрируется с различными системами контроля версий, такими как Git, Mercurial и SVN и обеспечивает возможность создания и выполнения юнит-тестов, а также интегрируется с другими инструментами разработки, такими как Docker и Kubernetes.

VS Code - это бесплатный редактор кода, который обеспечивает мощные функциональности для разработки Python-программ. Имеет широкую поддержку языка Python, поддержку отладки, интеграцию с системами контроля версий и т.д. Также VS Code имеет большое количество плагинов и расширений, которые обеспечивают большую гибкость в работе.

Sublime Text - это легкий и быстрый редактор кода, который поддерживает множество языков программирования, в том числе Python. Он обеспечивает функциональности, такие как автодополнение кода, рефакторинг, поддержку отладки и т.д. Однако у Sublime Text отсутствует многих функций PyCharm и VS Code.

Spyder - это IDE для Python, который специально разработан для научных вычислений и анализа данных. Он имеет встроенные библиотеки, такие как NumPy и SciPy, обеспечивающие высокую производительность. Spyder поддерживает подсветку синтаксиса, автодополнение, интеграцию с пакетными менеджерами (pip, conda), а также отладку и тестирование кода.

В итоге, я считаю, что PyCharm является наиболее мощной и полноценной IDE для разработки на Python. Он обеспечивает более широкую поддержку языка Python, а также наиболее полные функциональности для разработки, отладки и тестирования кода. Тем не менее, в зависимости от потребностей и стиля работы, вы можете выбрать другую среду разработки, которая будет лучше соответствовать вашим нуждам.

3 ПРАКТИЧЕСКАЯ ЧАСТЬ

3.1 Разработка логической модели автоматизации тестирования

Для создания объектной модели фреймворка был использован язык UML, который является стандартизированным и позволяет в виде чертежей и схем визуализировать, специфицировать, конструировать и документировать программные системы любого масштаба и направления. UML применяется для моделирования информационных систем предприятий, Web-приложений, встроенных систем реального времени и других систем. Для построения логической модели данных фреймворка была выбрана методология IDEF1x, которая используется для создания реляционных моделей данных. Далее происходит переход от структурной диаграммы, описанной в главе 1, к диаграмме вариантов использования (Use case Diagram), которая также называется диаграммой прецедентов. В соответствии с руководством пользователя по языку UML, субъект(актор) - это любая сущность, взаимодействующая с системой извне или множество логически связанных ролей, исполняемых при взаимодействии с прецедентами. Субъект может быть любым: человек, устройство, программа или другая система, которая воздействовует на моделируемую систему. Стандартным графическим обозначением субъекта на диаграммах является фигура человека, под которой записывается имя субъекта, но это не обязательно, так как субъектом может быть представлен любой объект, в зависимости от того, что определит разработчик.

Прецеденты (use case) - описание множества последовательностей действий, которые выполняются системой для того, чтобы актор получил нужный результат. Диаграмма прецедентов функционального назначения системы предназначена для описания того, как система взаимодействует со своими пользователями (актерами) и что она должна делать, чтобы удовлетворить их потребности. Как уже было сказано, на диаграммах прецедентов стандартным

обозначением является эллипс с кратким названием прецедента или его именем в форме глагола и пояснительными словами. Например, на диаграмме прецедентов для автоматизации тестирования могут быть прецеденты, такие как "Выбрать тест-кейс для выполнения", "Запустить тест-кейс", "Сравнить ожидаемый результат с фактическим", и т.д. Прецеденты могут иметь отношения между собой, такие как включение, расширение, обобщение и т.д., что помогает организовать диаграмму и сделать её более понятной.Диаграмма прецедентов фреймворка автоматизации тестирования приведена на рисунке 4.1

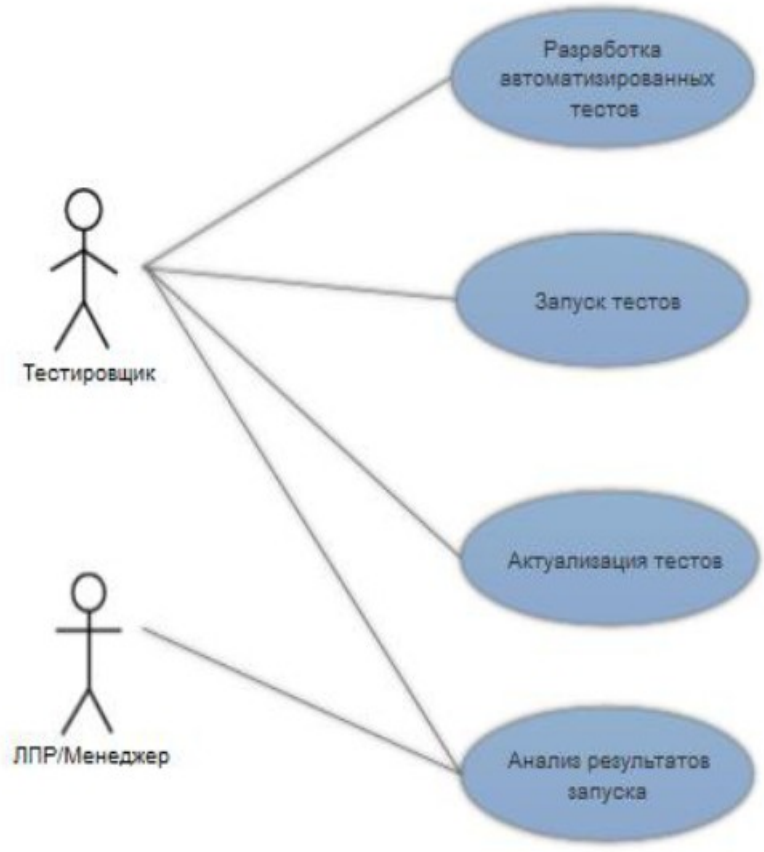


Рисунок 4.1. Диаграмма прецедентов использования автоматизации тестирования

Далее, разработаем диаграмму классов для фреймворка (Class Diagram).

Класс – это основной составной элемент информационной системы. Данное понятие является одним из основополагающих в объектно- ориентированных языках программирования. Соответственно, между классами в диаграммах UML и классами в программном коде есть соответствие, поэтому большинство средств построения диаграмм UML позволяют генерировать программный код на выбранном языке программирования на основе диаграммы классов. Каждый класс обязательно имеет уникальное название, а также атрибуты и методы. Атрибутами называются свойства класса, а методами – действия, которые может выполнять объект выбранного класса, чаще всего методы класса предназначены для изменения его свойств.

Абстрактная диаграмма классов для автоматизации представлена на рисунке 4.2

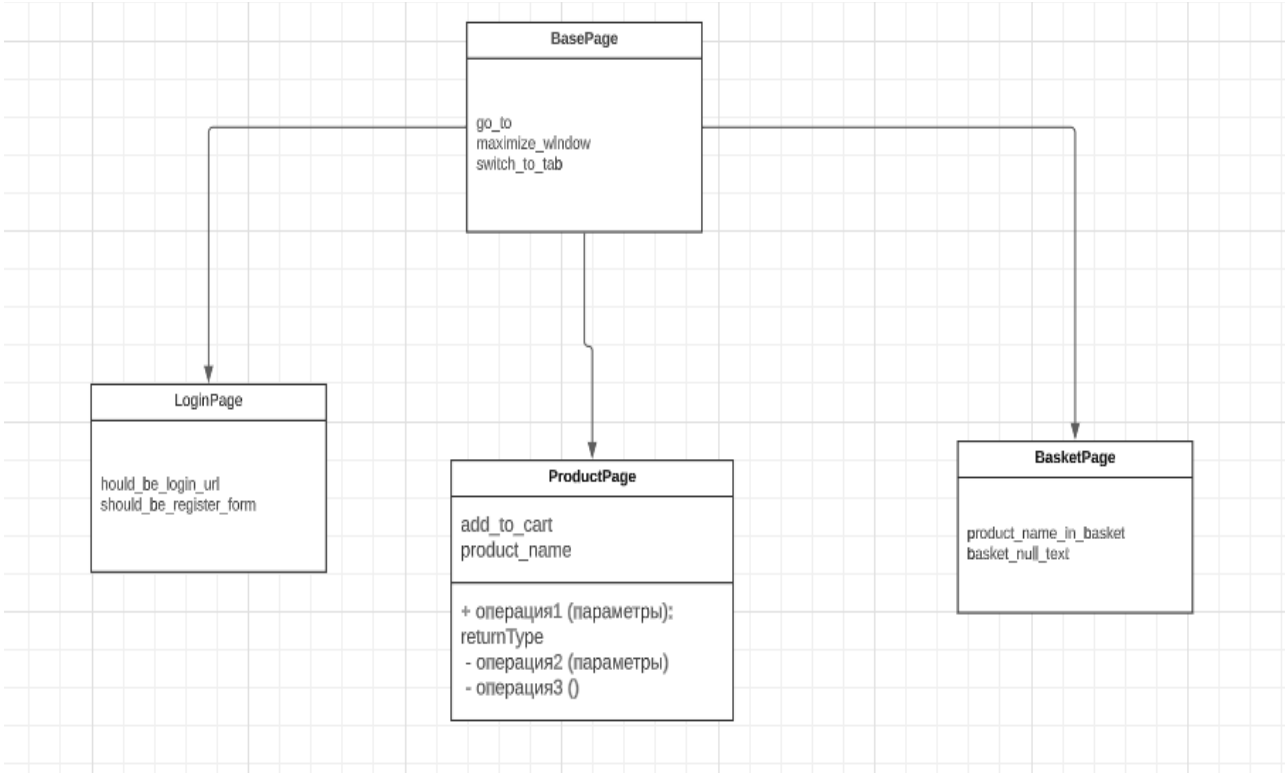


Рисунок 4.2 Диаграмма классов для автоматизации тестирования

3.2 Выбор технологий и спецификации

Помимо выбора языка программирования, среды разработки и фреймворка необходимо выбрать технологии и спецификации для проекта. Так как в данной работе рассматривается тестирования веб приложений придется взаимодействовать с различными браузерами. В этом поможет Selenium WebDriver.

Selenium WebDriver — это инструмент, который позволяет производить кросс-браузерное тестирование, то есть проверять, как отображается сайт в разных браузерах. Он входит в набор для автоматизированного тестирования Selenium. Всего в Selenium три компонента: Selenium WebDriver, Selenium IDE и Selenium Grid.

Тестировщики используют Selenium, чтобы автоматизировать действия браузеров, проверить работоспособность программы и получить данные с сайтов.

У программных средств Selenium разные подходы к автоматизации тестирования. WebDriver нужен для управления браузером, как локальным, так и удаленным. Это гибкий инструмент: он легко интегрируется с тестовыми фреймворками и другими инструментами. На его основе можно создать узконаправленные инструменты для кроссбраузерного тестирования.

WebDriver — главный вектор развития Selenium. Вот его основные преимущества:

1. Поддерживает разные языки программирования (Java, C#, PHP, Ruby, Perl, Python), а значит, его может использовать большое количество разработчиков.
2. Облегчает кроссбраузерное тестирование и поддерживает различные браузеры: Firefox, Opera, Chrome, Edge.

3. Есть фреймворки, облегчающие разработку.

4. Позволяет проводить параллельное тестирование в нескольких браузерах одновременно.

5. Имеет открытый исходный код, он бесплатный для любого разработчика.

6. Большое сообщество пользователей — при возникновении трудностей в работе есть у кого попросить помощи.

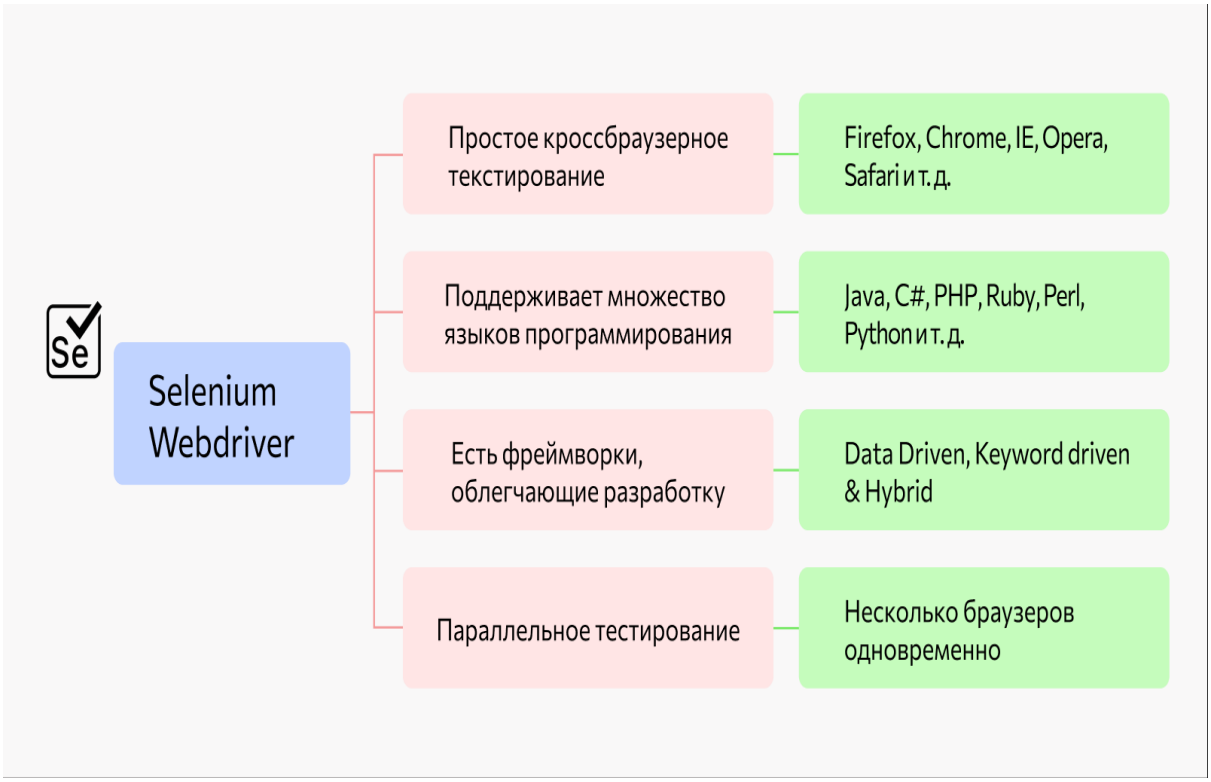


Рисунок 4.3 Selenium Webdriver

Для начала работы с Selenium WebDriver нужно определить три ключевых элемента: браузер, драйвер браузера и скрипт для драйвера браузера. Браузер, работу которого пользователю необходимо автоматизировать

Для браузера нужно определить параметры:

версия браузера;

операционная система, на которой браузер установлен. WebDriver поддерживает три операционные системы: Microsoft Windows, macOS и Linux;

настройки, которые должны быть выставлены в браузере — по умолчанию или кастомные.

Для более качественного тестирования нужно создавать как можно больше комбинаций этих параметров.

Список браузеров, которые поддерживает WebDriver:

Microsoft Edge;
Firefox;
Internet Explorer;
Safari;
Opera;
Chrome.

Драйвер браузера

Драйвер — это веб-сервис, который отправляет команды браузеру. У каждого браузера свои команды управления, которые реализованы по-разному, поэтому и драйверы нужны разные. Скрипт с набором команд для драйвера браузера В этом скрипте прописан алгоритм действий для браузера, с помощью него WebDriver эмулирует поведение пользователя.
Архитектура WebDriver

Selenium WebDriver использует собственные API и API браузера для работы с ними. Это значит, что для корректной работы WebDriver нужно участие

команды разработчиков браузера. Такой подход позволяет делегировать разработку драйверов создателям браузеров, но, когда выходят новые версии браузера, бывают задержки в обновлении.

Архитектура WebDriver позволяет разрабатывать драйверы для браузера независимо от разработки драйверов для других браузеров и от разработки клиентского приложения.

Взаимодействовать с драйвером будем посредством готовой библиотеки Python Selenium

Исходя из перечисленного выше набора инструментов можно построить архитектуру проекта. Данные о тестируемом веб-сервисе будем получать из браузера посредством веб-драйвера, далее при помощи фреймворка Pytest будем тестировать полученные данные в проекте, результаты на текущем этапе разработки будут выводиться в терминал выбранной среды разработки. В дальнейшем можно реализовать экспорт результатов в различные форматы, в рамках данной работы будет достаточно готового шаблона, предоставленного фреймворком. Схема разработанной архитектуры проекта представлена на рисунке 4.5

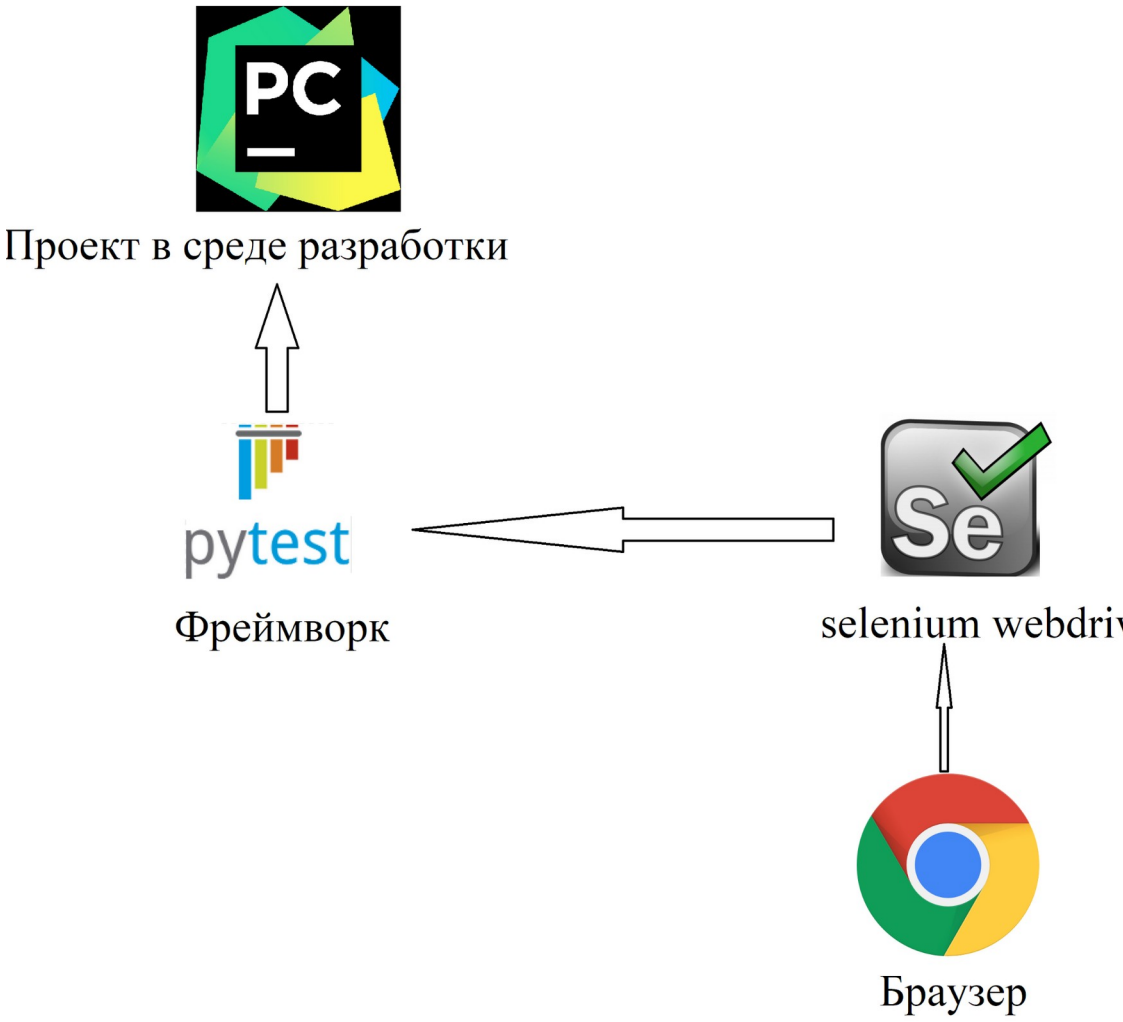


Рисунок 4.5 – Разработанная архитектура системы

3.3 Выбор тестируемого проекта

Подход к тестированию и его автоматизации сильно зависит от выбранного проекта. В зависимости от специфики веб-сервиса и его архитектурных особенностей набор автотестов может значительно меняться. Собрать универсальный комплекс автотестов на данный момент невозможно, так как при тестировании учитывается строение html-страниц и набор доступных запросов к API. Задача данной работы выстроить подход к созданию автотестов, основываясь на котором можно тестировать различные веб-сервисы, модифицируя текущий проект.

Для примера был выбран проект интернет-магазина написанный на базе ресурса pythonanywhere.

Pythonanywhere.com — сервис, позволяющий вести разработку на Python онлайн, прямо в браузере. Данный пример был выбран не случайно, я считаю, что интернет-магазин один из более обширных вариантов для автоматизации тестирования. На примере интернет-магазина можно рассмотреть тестирования функционала отображения каталогов, корзины товаров, регистрации и авторизации пользователей, что делает его самым удачным вариантом, по сравнению с другими веб-сервисами

3.4 Основные классы проекта

Для начала процесса тестирования необходимо определить основные элементы веб-сервиса и понять, какие виды тестов необходимо провести. В данном разделе главные объекты веб-сервиса будут рассмотрены в качестве классов с определенным набором атрибутов учитывая описанную выше диаграмму классов. Каждому классу будут присвоен ряд методов, описывающих процесс тестирования данного объекта.

3.4.1 Класс BasePage

Основным классом в проекте будет базовый класс, который будет содержать методы для управления окнами браузера и работы с текущей страницей. Он будет использован в качестве базового класса для других классов, которые будут работать с конкретными страницами

Конструктор класса инициализирует WebDriver и URL, с которым будет работать экземпляр класса.

```
def __init__(self, browser, url):
    self.browser = browser
    self.url = url
```

Метод go_to открывает заданную страницу в браузере.

```
def go_to(self, url):
    self.browser.get(self.url)
```

Метод maximize_window открывает браузер на весь экран.

```
def maximize_window(self):
    self.browser.maximize_window()
```

Методы switch_to_newly_opened_tab, switch_to_first_opened_tab и switch_to_last_opened_tab позволяют переключаться между вкладками браузера.

```
self.browser.switch_to.window(self.browser.window_handles[1])
```

Метод switch_wait_to_newly_tab_select позволяет переключаться на нужную вкладку. Он ожидает появления нужной вкладки и затем переключается на неё.

```
def switch_wait_to_newly_tab_select(self, number):
    while True:
    try:
        self.browser.window_handles[number]
    except IndexError:
        continue
    else:
        time.sleep(3)
        break
    self.browser.switch_to.window(self.browser.window_handles[number])
```

Метод close_tab закрывает текущую вкладку браузера.

```
    def close_tab(self):
        self.browser.close()
```

Метод reload_page перезагружает страницу в браузере.

```
    def reload_page(self):
        time.sleep(3)
        self.browser.refresh()
```

Метод get_title возвращает заголовок текущей страницы.

```
    def get_title(self):
        return self.browser.title
```

Метод get_url возвращает текущий адрес страницы.

```
    def get_url(self):
        return self.browser.current_url
```

Метод alert_accept обрабатывает аллерты в браузере. Если аллерт есть, то он принимает его сообщение и возвращает его значение. Если аллерта нет, то метод возвращает строку "no alert".

solve_quiz_and_get_code - метод, который позволяет решить задачу в алерте, вычисляя логарифм. Он получает значение переменной x из текста алерта, вычисляет ответ на задачу и вводит его в алерт. Затем метод получает текст из второго алерта и выводит его в консоль.

```
    def solve_quiz_and_get_code(self):
    alert = self.browser.switch_to.alert
    x = alert.text.split(" ")[2]
    answer = str(math.log(abs((12 * math.sin(float(x))))))
    alert.send_keys(answer)
```

```
        alert.accept()
        try:
        alert = self.browser.switch_to.alert
        alert_text = alert.text
        print(f"Your code: {alert_text}")
        alert.accept()
        except NoAlertPresentException:
        print("No second alert presented")
```

is_element_present - метод, который проверяет наличие элемента на странице.

```
        def is_element_present(self, how, what):
        try:
                self.browser.find_element(how, what)
        except NoSuchElementException:
                return False
        return True
```

is_not_element_present - метод, который проверяет отсутствие элемента на странице. Он ожидает определенное время и возвращает False, если элемент все же появился.

```
        def is_not_element_present(self, how, what, timeout=4):
                try:
                        WebDriverWait(self.browser,
timeout).until(EC.presence_of_element_located((how, what)))
        except TimeoutException:
                return True
        return False
```

go_to_login_page - метод, который кликает на кнопку "Войти или зарегистрироваться" на странице.

```
def go_to_login_page(self):
        self.browser.find_element(*BasePageLocators.LOGIN_LINK).click()
```

element_text - метод, который получает текст элемента на странице.

```
        def element_text(self, by, value):
                try:
                        element = self.browser.find_element(by, value)
                        text = element.text
                except NoSuchElementException:
                        return False
                return text
```

go_to_cart - метод, который кликает на иконку корзины на странице.

```
def go_to_cart(self):
        self.browser.find_element(*BasePageLocators.BASKET_LINK).click()
```

3.4.2 класс BasketPage

BasketPage наследует класс BasePage и является классом для тестирования функционала корзины интернет-магазина,а конкретно для проверки информации о товарах в корзине и наличии сообщения о пустой корзине. Класс содержит два метода: basket_null_text и product_name_in_basket. Метод basket_null_text проверяет, что на странице корзины отображается сообщение "Your basket is empty." Если это сообщение не отображается, метод вызовет ошибку и проверка не будет пройдена.

```
def basket_null_text(self):
        assert empty_basket in self.element_text(
```

*BasketPageLocators.EMPTY_TEXT)

Метод product_name_in_basket получает информацию о названии продукта, который находится в корзине. Он ищет элемент на странице, используя селектор из файла locators.py и возвращает текст в найденном элементе. В данном случае, возвращается название продукта, находящегося в корзине.

```
def product_name_in_basket(self):
        return self.browser.find_element(
                *BasketPageLocators.PRODUCT_NAME).text
```

3.4.3 LoginPage

Класс LoginPage наследует BasePage. Этот класс определяет методы для проверки соответствия страницы авторизации/регистрации определенным критериям. Если страница не соответствует этим критериям, методы в этом классе вызовут ошибку и проверка не будет пройдена. В этом классе определены три метода, которые выполняют проверки на странице авторизации/регистрации.

Метод should_be_login_url проверяет, что URL-адрес страницы соответствует ожидаемому значению. Если URL-адрес отличается, метод вызовет ошибку и проверка не будет пройдена.

```
        def should_be_login_page(self):
                self.should_be_login_url()
                self.should_be_login_form()
                self.should_be_register_form()
```

Метод should_be_login_form проверяет, что на странице есть форма авторизации. Если форма не найдена, метод вызовет ошибку и проверка не будет пройдена.

```
    def should_be_login_url(self):
        assert self.get_url() == f"{TEST_URL}/en-gb/accounts/login/",
"Страница не совпадает"
```

Метод should_be_register_form проверяет, что на странице есть форма регистрации. Если форма не найдена, метод вызовет ошибку и проверка не будет пройдена.

```
    def should_be_register_form(self):
        assert self.is_element_present(
*LoginPageLocators.REGISTRATION_FORM),
 "Форма регистрации нет на странице "
```

3.4.4 ProductPage

Класс ProductPage наследует BasePage. Класс определяет методы для добавления товара в корзину и получения названия товара со страницы товара. В этом классе определены два метода: add_to_cart и product_name.
Метод add_to_cart выполняет нажатие на кнопку, которая отвечает за добавление товара в корзину. Кнопка находится на странице товара и выбирается с помощью селектора ADD_BUTTON из файла locators.py.

```
def add_to_cart(self):
        button =
self.browser.find_element(*ProductPageLocators.ADD_BUTTON)
```

```
        button.click()
```

Метод product_name возвращает название товара, находящегося на странице товара. Он использует селектор PRODUCT_NAME и возвращает текст элемента, содержащего название товара.

```
    def product_name(self):
        return self.browser.find_element(
*ProductPageLocators.PRODUCT_NAME).text
```

3.4.5 TestPage

Данный класс содержит три тестовых метода, которые выполняют проверки на странице сайта.

Метод test_go_to_login_page проверяет, что пользователь может перейти на страницу авторизации/регистрации. Он создает объект MainPage, переходит на главную страницу, затем переходит на страницу авторизации/регистрации с помощью метода go_to_login_page. Затем создается объект LoginPage и выполняются три проверки на корректность URL-адреса страницы, наличие формы логина и наличие формы регистрации.

```
    def test_go_to_login_page(self, browser):
    page = MainPage(browser, TEST_URL)
    page.go_to(TEST_URL)
    page.go_to_login_page()
    login_page = LoginPage(browser, browser.current_url)
    login_page.should_be_login_page()
```

Метод test_guest_cant_see_product_in_basket_opened_from_main_page проверяет, что гость не видит продукты в корзине при открытии корзины со страницы главной страницы. Он создает объект MainPage, переходит на страницу главной страницы, затем переходит на страницу корзины с помощью метода go_to_cart. Затем создается объект BasketPage и проверяется, что на странице отображается сообщение "Your basket is empty."

```
    def  test_guest_cant_see_product_in_basket_opened_from_main_page(
self, browser):
    page = MainPage(browser, TEST_URL)
    page.go_to(TEST_URL)
    page.go_to_cart()
        basket_page = BasketPage(browser, browser.current_url)
        basket_page.basket_null_text()
```

Метод test_guest_can_add_product_to_basket проверяет, что гость может добавить продукт в корзину и проверить, что продукт находится в корзине. Он выполняет параметризованный тест на добавление продукта в корзину со страницы товара. Для каждой ссылки продукта создается объект MainPage, переходит на страницу товара, добавляет товар в корзину с помощью метода add_to_cart, решает задачу и получает код, получает название продукта и переходит на страницу корзины. Затем создается объект BasketPage и проверяется, что название продукта, который находится в корзине соответствует добавленному продукту.

```
    @pytest.mark.parametrize("link",
                ["/catalogue/coders-at-work_207/?promo=offer0",
                "/catalogue/coders-at-work_207/?promo=offer1",
```

```
                "/catalogue/coders-at-work_207/?promo=offer2",
                "/catalogue/coders-at-work_207/?promo=offer3"
                ])

    def test_guest_can_add_product_to_basket(self, browser, link):

            page = MainPage(browser, f"{TEST_URL}{link}")
            page.go_to(f"{TEST_URL}{link}")
            product_page = ProductPage(browser, browser.current_url)
            product_page.add_to_cart()
            product_page.solve_quiz_and_get_code()
            product_name = product_page.product_name()
            page.go_to_cart()
            basket_page = BasketPage(browser, browser.current_url)
            product_name_in_basket_page =
basket_page.product_name_in_basket()
            assert product_name == product_name_in_basket_page
```

3.4.6 TestUser

Класс TestUser содержит два тестовых метода для проверки функционала API. Метод test_create_user тестирует создание нового пользователя с использованием POST-запроса, который отправляется на URL, указанный в константе USER. В качестве тела запроса используется user_body. Метод также содержит проверку на успешный ответ с кодом 200, а затем отправляет DELETE-запрос на удаление созданного пользователя.

```
  def test_create_user(self):
      response = send_request("POST", USER, json=user_body)
```

```
        print(response.json())
        assert response.status_code == 200
        send_request("DELETE", f"{USER}/{user_body['username']}")
```

Метод test_get_data_user тестирует получение данных пользователя с использованием GET-запроса. Текст URL-адреса строится используя имя пользователя, которое было указано при создании пользователя. Метод выполняет проверку, что код ответа равен 404, что означает, что данный пользователь не был найден, и после этого производит дополнительную проверку на наличие в JSON-ответе текста "User not found".

```
    def test_get_data_user(self):
        response = send_request("GET", f"{USER}/{user_body['username']}")
        print(response.json())
        assert response.status_code == 404
        assert response.json()["message"] == "User not found"
```

4 ЗАКЛЮЧЕНИЕ

В результате данной работы был разработан комплекс автотестов для автоматизации тестирования web-сервисов. Были изучены существующие методы и инструменты тестирования web-сервисов, выбран оптимальный инструмент для автоматизации тестирования, разработаны автотесты для проверки различных функциональных возможностей web-сервисов, а также проведено тестирование созданного комплекса автотестов на различных web-сервисах. Анализ результатов показал, что использование комплекса автотестов для тестирования web-сервисов может улучшить качество и производительность работы web-сервисов, а также повысить удобство их использования для конечных пользователей. На основании проделанной работы можно сделать вывод о том, что автоматизация тестирования web-сервисов является актуальной и перспективной задачей, которая может быть решена с помощью современных инструментов и методов. Созданный в ходе исследования комплекс автотестов может быть использован для тестирования web-сервисов различной сложности и типов. Дальнейшее развитие данного направления может быть связано с расширением комплекса автотестов и созданием новых инструментов для автоматизации тестирования web-сервисов. Результаты работы могут быть полезны для разработчиков web-сервисов, тестировщиков и любых специалистов, связанных с созданием и поддержанием web-сервисов.

5 СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Бек, К. Шаблоны реализации корпоративных приложений / К. Бек. – М. : Вильямс, 2017. – 176 с.

2. Буч, Г. Язык UML. Руководство пользователя: Пер. с англ. Мухин Н. / Г. Буч, Д. Рамбо, А. Якобсон – М. : ДМК Пресс, 2006. – 496 с.

3. Введение в BDD [Электронный ресурс]. – Режим доступа : http://agilerussia.ru/practices/introducing-bdd/.

4. Информационная технология. Системная и программная инженерия. Процессы жизненного цикла программных средств [Текст] : ГОСТ Р ИСО/МЭК 12207-2010; введ. 01.03.12

5. Канер, С. Тестирование программного обеспечения. Фундаментальные концепции менеджмента бизнес-приложений: Пер. с англ. / С. Канер, Д. Фолк, Е. К. Нгуен. – К. : ДиаСофт, 2001. – 544 с.

6. Макконнелл, С. Совершенный код. Мастер класс. Пер. с англ. / С. Макконнелл. – М. : Русская редакция, 2010. — 896 с.

7. Осипенко, Н. Б. Стандартизация и сертификация программного обеспечения. Тексты лекций. / Н. Б. Осипенко. – Министерство образования РБ, Гомельский гос. ун-т им. Ф. Скорины. – Гомель: ГГУ им. Ф. Скорины, 2012. – 155 с.

8. Портал Про Тестинг – Тестирование Программного Обеспечения [Электронный ресурс]. – Режим доступа : http://www.protesting.ru.

9. Тамре, Л. Введение в тестирование программного обеспечения / Л. Тамре. – М. : Вильямс, 2003. – 368 с.

10. Шаблоны корпоративных приложений / М. Фаулер [и др.]. – М. : Вильямс, 2016. – 544 с.

11. Guide to the Software Engineering Body of Knowledge [Текст] :

SWEBOK, Version 3.0, IEEE; введ. 20.12.13
71
12. PageObject. Martin Fowler [Электронный ресурс]. – Режим доступа : https://martinfowler.com/bliki/PageObject.html.
13. TIOBE Index [Электронный ресурс]. – Режим доступа : https://tiobe.com/tiobe-index/.