**ВВЕДЕНИЕ**

На данный момент в сфере тестирования большинство операций осуществляется вручную специалистами по тестированию программного обеспечения (ПО). Процесс функционального тестирования затрагивает анализ требований, составление тестовой документации, осуществление тестирования, обнаружение и описание дефектов.

Такой жизненный цикл тестирования занимает много времени у исполнителей и требует сильной концентрации внимания. Для решения этой проблемы на современных проектах вводится автоматизация тестирования. Роль специалиста по автоматизации тестирования относительно новая и инновационная в индустрии информационных технологий.

Автоматизация тестирования играет ключевую роль в обеспечении качества программных продуктов. Она позволяет проводить тесты быстрее, эффективнее и точнее, чем это возможно при ручном тестировании, также обеспечивает избавление от человеческого фактора в процессе поиска дефектов, возможность переиспользования действий без затрат сил человека, а также гарантирует наглядную отчетность.. Это особенно важно в условиях современных методологий разработки, таких как Agile (гибкая методика разработки) и DevOps (методология разработки и эксплуатации), где скорость и гибкость являются критическими факторами успеха[1].

Также проблемой в этой области является  
отсутствие реализации тестовых сценариев, выходящих за рамки браузера. Например, отсутствуют инструменты для работы с файловыми менеджерами, проверки корректной загрузки файлов, их названий и расширений, отсутствие возможности визуального сравнения изображений и т.д[2].

Дипломный проект, темой которого является разработка фреймворка для автоматизации тестирования и его эргономическое обеспечение, будет выполнен в виде удобного и современного фреймворка, решающего вышеуказанные проблемы, который можно подключить в проект в качестве внешней библиотеки.

При разработке фреймворка был использован язык программирования C# версии 12.0, платформа для разработчиков .NET 8 и Selenium WebDriver в качестве основного ядра.

На этапе разработки фреймворка были определенны следующие задачи:

1 Исследование предметной области;

2 Ознакомление с имеющимися материалами в сфере автоматизации тестирования;

3 Изучение доступных аналогов проектируемой информационной системы в виде фреймворков, библиотек и zero-code приложений для автоматизации тестовых сценариев.

4 Постановка оптимальных целей и задач фреймворка, определение его структуры, разработка технического задания, проработка прототипов модулей, разработка интуитивно-понятных модулей, интеграция с проектом, настройка основных модулей, интеграция с дополнительными системами и сервисами, наполнение сценариями, релиз и тестирование.

5 Определение функций для ролей пользователей в качестве «Пользователя» и «Контрибьютора».

6 Реализация требуемого функционала в готовом программном продукте, поддержка и сопровождение.

Актуальность поставленной темы объясняется тем, что в настоящее время растет спрос на замещение ручного тестирования автоматизацией, а имеющиеся в настоящее время предложения по данной тематике не обладают достаточным для полного покрытия тестами функционалом.

Пояснительная записка имеет следующую структуру:

1 В первой главе «Информационные системы в сфере автоматизации тестирования» содержится описание предметной области дипломного проекта, определенны цели и задачи, которые необходимо выполнить во время разработки информационной системы, доказана актуальность выбранной темы, а также произведен обзор существующих аналогов;

2 Во второй главе, которая называется «Эргономическое проектирование программного средства», представлены готовые UML диаграммы, структура классов, описаны главные модули фреймворка;

3 В третьей главе «Разработка фреймворка для автоматизации тестирования на платформе .NET 8» описаны этапы разработки фреймворка. Также изложена информация о структуре программного обеспечения, алгоритма работы и прототипов модулей, создании программных модулей и реализованных функциях;

4 В четвертой главе «Тестирование фреймворка для автоматизации» произведено функциональное и юзабилити тестирования готового продукта и проверка на системные и функциональные ошибки.

5 В пятой главе «Экономическое обоснование эффективности разработки и реализации фреймворка для автоматизации тестирования» содержатся характеристика информационной системы, а также расчеты сметы затрат, экономического эффекта от реализации информационной системы и эффективности показателей информационной системы.

6 Раздел «Охрана труда. Обоснование выбора системы искусственного освещения в помещении при разработке фреймворка для автоматизации тестирования веб-интерфейсов» содержит аналитику текущей системы освещения и профилактические меры по обеспечению эффективного освещения на рабочем месте, в том числе технические, организационные и режимные мероприятия, которые направлены на устранение или минимизацию факторов, способствующих возникновению проблем со зрением и здоровьем.

7 В заключении подведен итог выполненной работы и анализ реализации поставленных задач.

Текст пояснительной записки построен на основе собственных разработок, учебной и научной литературы, список которой приведен в использованных источниках.

1 ИНФОРМАЦИОННЫЕ СИСТЕМЫ В СФЕРЕ АВТОМАТИЗАЦИИ ТЕСТИРОВАНИЯ

**1.1 Анализ предметной области**

Автоматизация тестирования – это мощный инструмент, который может оказать положительное влияние на бизнес. Рассмотрим, как автоматизация тестирования способствует целям бизнеса[3]:

1. Улучшение качества продукта: Автоматизированные тесты позволяют выявлять дефекты и ошибки в программном продукте более быстро и точно. Это помогает улучшить качество продукта, что в свою очередь способствует удовлетворенности клиентов и повышению лояльности.
2. Сокращение времени релиза: Автоматизация позволяет проводить тестирование быстрее и эффективнее. Это сокращает время, необходимое для выпуска новых версий продукта на рынок. Быстрый релиз может быть критически важным для бизнеса, особенно в быстро меняющейся среде.
3. Экономия ресурсов: Ручное тестирование требует большого количества времени и участия специалистов. Автоматизация позволяет сократить затраты на тестирование, так как тесты могут выполняться автоматически без постоянного вмешательства человека.
4. Повышение надежности: Автоматизированные тесты могут выполняться в любое время, что позволяет выявлять проблемы даже до того, как они повлияют на пользователей. Это способствует повышению надежности продукта.
5. Снижение рисков: Автоматизация позволяет проводить тестирование в различных сценариях, включая крайние случаи и нагрузочное тестирование. Это помогает выявить потенциальные проблемы и снизить риски для бизнеса.
6. Быстрое обнаружение проблем: Автоматизированные тесты могут запускаться после каждого изменения кода. Это позволяет быстро обнаруживать проблемы и устранять их на ранних этапах разработки.

В конечном итоге, автоматизация тестирования помогает бизнесу достичь своих целей, обеспечивая высокое качество продукта, сокращение времени релиза и экономию ресурсов

Разработка фреймворка для автоматизации тестирования – это важный процесс, который обеспечивает эффективное и надежное тестирование программного обеспечения. Вот основные этапы и объекты, которые следует учесть при создании такого фреймворка[4]:

1. Определение объема тестирования:

* Анализ требований и функциональности продукта.
* Определение, какие части продукта будут подвергаться автоматизированному тестированию.

1. Выбор инструментов автоматизированного тестирования:

* Выбор языка программирования (например, C#, Java).
* Использование фреймворков (например, Selenium, XUnit, NUnit).

1. Планирование автоматизированных тестов:

* Создание плана тестирования, включая сценарии и тест-кейсы.
* Определение приоритетов и распределение ресурсов.

1. Разработка дизайна фреймворка:

* Создание архитектуры фреймворка.
* Разработка модулей для управления тестами, данных, отчетами и логами.

1. Разработка тестовых сценариев:

* Написание автоматизированных тестовых сценариев.
* Включение проверок, ассертов и ожиданий.

1. Настройка среды запуска автоматизированных тестов:

* Установка и настройка инструментов (например, Selenium WebDriver).
* Создание конфигурационных файлов.

1. Запуск разработанных автоматизированных тестов и анализ результатов:

* Запуск тестов в автоматическом режиме.
* Анализ результатов, выявление ошибок и дефектов.

1. Поддержка автоматизированных тестов:

* Обновление тестов при изменениях в продукте.
* Поддержка и оптимизация фреймворка

При разработке фреймворка для автоматизации тестирования, максимум внимания требуется уделить взаимодействию каждого модуля между собой, потому что основная потребность пользователей подобной системы – это объединить возможности нескольких инструментов, предназначенных для обеспечения качества через тестирование, в одном единственном месте – в данном случае, в едином цельном фреймворке.

В этом, как раз, и заключается цель данного дипломного проекта – разработка фреймворка для автоматизации тестирования и осуществление анализа преимуществ, которые он может принести как специалистам по тестированию, так и жизненному циклу разработки ПО в целом.

Таким образом, существует целесообразность разработки нового фреймворка для автоматизации тестирования веб интерфейсов с целью совершенствования рабочего процесса тестировщиков, повышения удобства оценки качества ПО, простоты и скорости проведения функционального и интеграционного тестирования в нужных предметных областях, предоставления возможности тестировщикам передавать более полную отчётность о проведении тестирования, при условии обеспечения основных показателей надежности, безопасности и соответствия ожидаемым требованиям на современном этапе развития информационных технологий.

**1.2 Основные аналоги информационной системы для автоматизации тестирования**

В данный момент на коммерческих проектах часто вводится автоматизация тестирования. Бизнесу выгодно доверить информационной системе действия, которые вручную выполняются специалистами по тестированию ПО. В связи с этим остро стоит вопрос по выбору инструмента для автоматизации тестирования.

Перед рассмотрением аналогов важно определить, какие категории фреймворков существуют на рынке. Определение категорий позволит подобрать наиболее подходящий для проекта вариант. Особое внимание уделялось таким параметрам, как мультиплатформенность и поддержка языков программирования, сложность использования, особые функциональные возможности, популярность и совместимость[5].

На рынке существует большое количество фреймворков по автоматизации тестирования, каждый из них глобально выполняет одну и ту же задачу – автоматизировать ручные процессы при тестировании, но аналоги уникальны и имеют как свои преимущества, так и недостатки.

Каждый фреймворк нацелен на конкретную сферу тестирования. Некоторые созданы для автоматизации тестирования пользовательского интерфейса веб-приложений, некоторые оптимизированы для работы с настольными приложениями, некоторые имеют функционал для симуляции действий на мобильных устройствах, другие могут тестировать Application Programming Interface (API) или базы данных.

В результате информационного поиска, были выделены 6 основных категорий, описание каждой из них, их предназначение и примеры инструментов автоматизации, относящихся к этим категориям.

*Инструменты для веб-тестирования*

Эти инструменты предназначены для автоматизации тестирования веб-приложений, симулируют взаимодействие пользователя с веб-элементами страницы. Наиболее популярными и востребованными являются[6]:

* *Selenium:* Наиболее популярный фреймворк для веб-автоматизации;
* *Cypress:* Современный инструмент с уникальными возможностями, работающий напрямую с версткой веб-приложения;
* *WebDriverIO*: Node.js-фреймворк для веб-тестирования;

*Инструменты для мобильного тестирования*

Эти инструменты предназначены для автоматизации тестирования мобильных приложений, взаимодействие пользователя с мобильным устройством. Например[7]:

* *Appium:* Кросс-платформенный инструмент для мобильной автоматизации;
* *XCUITest* (для iOS) и *Espresso* (для Android);

*Инструменты для API-тестирования*

Эти инструменты помогают автоматизировать тестирование API для клиент-серверной и других архитектур. Примеры[8]:

* *Postman:* Инструмент для создания и отправки HTTP-запросов;
* *RestAssured:* Java-библиотека для тестирования REST API;
* *RestSharp:* C#-библиотека для тестирования REST API;

*Инструменты для нагрузочного тестирования*

Эти инструменты позволяют оценить производительность системы под нагрузкой. Примеры[9]:

* *JMeter:* Самый популярный инструмент для нагрузочного тестирования;
* *Gatling:* Scala-инструмент для тестирования производительности;

*Инструменты для тестирования на уровне базы данных*

* Эти инструменты помогают автоматизировать проверку данных в базах данных. Примеры[10]:
* *DbUnit:* Java-библиотека для тестирования баз данных;
* *SQLUnit:* Инструмент для запуска SQL-запросов в тестах;

*Инструменты для функционального тестирования*

* Эти инструменты помогают автоматизировать функциональное тестирование. Примеры[11]:
* *NUnit:* Фреймворк для тестирования на C#;
* *JUnit и TestNG:* Популярные фреймворки для тестирования на Java;

Для определения категории, необходимой для сравнительного анализа был проведён детальный анализ актуальности и популярности на современных проектах каждой из категорий.

В последние время набирают популярность zero-code решения, которые не требуют для своего использования знаний языков программирования, достаточно просто записать действия и иметь возможность воспроизводить их в качестве автоматизированного теста. Примером является платформа TestGrid (Рисунок 1).



Рисунок 1. Интерфейс TestGrid

Но такие решения не будут рассматриваться для сравнения, так как решения, поддерживающие языки программирования более стабильны и эффективны для решения задач по тестированию.

По результатам анализа и опроса был сделан вывод, что самой популярной и важной на данный момент категорией является категория “Инструменты для веб-тестирования”. Далее предоставлено краткое обоснование выбора этой категории.

Веб-приложения продолжают оставаться востребованными и актуальными в современном мире. Вот несколько ключевых аспектов, подчеркивающих их популярность:

Рост числа веб-приложений. Каждый день появляется все больше веб-приложений – от интернет-магазинов до социальных сетей. Это связано с ростом онлайн-бизнеса, цифровой трансформацией и повышенным спросом на онлайн-сервисы [12];

1. Множество устройств и браузеров: Пользователи заходят на веб-сайты с разных устройств (компьютеры, смартфоны, планшеты) и используют разные браузеры. Веб-приложения должны быть адаптированы для всех сценариев;
2. Безопасность и защита данных: Веб-приложения подвергаются угрозам безопасности, таким как взломы, утечки данных и мошенничество. Тестирование и обеспечение безопасности – важные аспекты веб-разработки;
3. Постоянные обновления и новые требования: Веб-приложения постоянно обновляются, добавляются новые функции и исправляются ошибки. Разработчики должны следить за последними тенденциями и адаптироваться к новым требованиям;
4. Пользовательский опыт: Качество веб-приложения напрямую влияет на удовлетворенность пользователей. Отзывчивость, интуитивный интерфейс и быстрая загрузка – ключевые факторы успеха;

После определения точной категории был составлен список самых часто используемых на практике инструментов автоматизации, входящих в эту категорию, которые выполняют свои задачи на современных проектах по разработке ПО и широко используются в сфере тестирования пользовательских интерфейсов и внутренней логики веб-приложений.

Список основных инструментов для автоматизации тестирования интерфейса веб-приложений, начиная от самого популярного к менее популярному:

* Selenium;
* Cypress;
* Playwright;
* Katalon Studio;
* TestCafe;
* WebDriverIO;
* Ranorex;
* Jest;
* Puppeteer;
* SikuliX;
* Applitools Eyes;
* UFT;
* Robot Framework;
* Screenster;
* CasperJS;
* Watir;
* Ghost Inspector;
* TestCraft;
* Cucumber;
* NightwatchJS;
* TestRigor;
* Rapise;
* Aquality framework;
* Testim;
* Subject7;
* Selenide;
* ACCELQ;
* iMacros;
* IBM Rational Functional Tester;
* Watir;
* Telerik Test Studio;
* Virtuoso;

Для сравнительного анализа необходимо определить тройку наиболее подходящих инструментов. Чтобы выбрать наилучшие фреймворки из списка были определены самые влиятельные критерии выбора.

Основные критерии для выбора исследуемых аналогов могут включать следующие аспекты[13]:

* *Функциональность:* Какие функции и возможности предоставляют системы. Особое внимание уделялось на автоматизацию тестов, интеграцию с другими инструментами, отчетность и уникальные особенности инструмента.
* *Возможности:* Оценивается, насколько фреймворк соответствует потребностям проекта. Рассмотрена поддержка различных типов тестов (функциональные, нагрузочные, UI-тесты), интеграция с другими инструментами и возможности взаимодействия с элементами страницы.
* *Производительность:* Как быстро системы выполняют тесты. Есть ли ограничения по объему тестов.
* *Поддержка и сообщество:* Каков уровень поддержки со стороны разработчиков. Есть ли активное сообщество пользователей.
* *Сложность использования:* Насколько легко развернуть и настроить систему. Требуется ли специальная экспертиза.
* *Язык программирования:* Нужно убелиться, что фреймворк поддерживает язык программирования, востребованный для проекта.
* *Отчетность и логирование:*Какие отчеты генерируются после выполнения тестов. Хорошая отчетность поможет анализировать результаты.
* *Совместимость с инфраструктурой:*Как фреймворк интегрируется с существующей инфраструктурой (CI/CD-пайплайны, контейнеры и т. д.).
* *Бюджет и лицензия:*Некоторые фреймворки могут быть платными или иметь ограничения по использованию. Фреймворк должен соответствовать проектному бюджету и требованиям.

На основе выделенных критериев были определены три наиболее влиятельных фреймворка для сравнения: Selenium WebDriver, Cypress и Playwright. Разбор фреймворка Selenium WebDriver представлен в таблице 1.

Таблица 1 – Описание Selenium WebDriver

|  |  |
| --- | --- |
| **Критерий** | **Описание** |
| **Название и разработчик** | Selenium WebDriver. Разработан командой разработчиков и сообществом, без единого авторства. |
| **Описание и назначение** | Это инструмент для автоматизации тестирования веб-приложений и симуляции действий с браузером.  Он позволяет создавать и запускать тесты, взаимодействуя с веб-страницами и веб-элементами как это делает пользователь.  Особенностью является открытый исходный код фреймворка и бесплатное использование. |
| **Основные компоненты** | *Selenium WebDriver.* Это основной компонент, который позволяет взаимодействовать с браузерами. Есть возможность написать тесты на разных языках программирования (Java, Python, C#, Ruby или JavaScript) и выполнять их с помощью WebDriver.  *Selenium IDE*. Это расширение для браузера Chrome и Firefox, которое позволяет записывать и воспроизводить действия пользователя на веб-страницах.  *Selenium Grid*. Этот компонент позволяет распределить тесты на несколько машин, чтобы ускорить выполнение. |
| **Производительность** | Selenium обеспечивает эффективное выполнение тестов, но производительность может зависеть от объема тестов и настроек. |
| **Поддержка и сообщество** | Selenium имеет активное сообщество разработчиков и пользователей, что обеспечивает поддержку и решение проблем.  Также он является устоявшимся инструментом с обширной документацией и большим количеством примеров использования. |
| **Сложность использования** | Настройка Selenium может быть сложной, особенно для тестирования в разных браузерах и для запуска тестов в параллельных потоках. |
| **Язык программирования** | Поддерживает широкий спектр языков программирования: Java, Python, C#, Ruby, JavaScript. |
| **Отчетность и логирование** | Selenium предоставляет возможности для генерации отчетов после выполнения тестов.  Также имеет большое количество плагинов для генерации отчетов. |
| **Совместимость с инфраструктурой** | Selenium интегрируется с CI/CD-пайплайнами, контейнерами и другими инструментами. |
| **Бюджет и лицензия** | Selenium – это бесплатный и открытый инструмент, доступный для всех. С открытым исходным кодом |

Разбор фреймворка Cypress представлен в таблице 2.

Таблица 2. Описание Cypress

|  |  |
| --- | --- |
| **Критерий** | **Описание** |
| **Название и разработчик** | Cypress разрабатывается Cypress.io, некоммерческой организацией, которая создает инструменты для тестирования веб-приложений. |
| **Функциональность и возможности** | Cypress – это инструмент для автоматизации веб-приложений, специализирующийся на тестировании фронтенда. Он обеспечивает простоту в создании и запуске тестов, а также взаимодействие с веб-страницами.  Особенностью является простой синтаксис и интегрированный тестовый раннер. |
| **Производительность** | Cypress обладает высокой производительностью благодаря своей архитектуре и способности выполнять тесты непосредственно в браузере. |
| **Поддержка и сообщество** | У Cypress есть активное сообщество, которое предоставляет поддержку, обучающие материалы и решения проблем. |
| **Сложность использования** | Cypress обладает простым и интуитивным синтаксисом, что упрощает его использование даже для новичков. |
| **Язык программирования** | Cypress поддерживает только JavaScript, что делает его доступным для менее широкого круга разработчиков. |
| **Отчетность и логирование** | Cypress предоставляет гибкие возможности для создания отчетов и логирования результатов тестов. |
| **Совместимость с инфраструктурой** | Cypress интегрируется с CI/CD-пайплайнами, Docker, и другими инструментами. |
| **Бюджет и лицензия** | Cypress – это бесплатный и открытый инструмент, доступный для всех. |

Разбор фреймворка Playwright практически аналогичен разбору Cypress, поэтому выделены отличительные особенности в таблице 3. А также проанализированы преимущества и недостатки в таблице 4.

Таблица 3. Описание особенностей Playwright

|  |  |
| --- | --- |
| **Критерий** | **Описание** |
| **Функциональность и возможности** | Playwright – это библиотека для автоматизации веб-приложений, специализирующаяся на тестировании фронтенда. |
| **Отличительные особенности** | Она обладает кросс-платформенной, кросс-браузерной и кросс-языковой поддержкой, а также включает полезные функции, такие как автоматическое ожидание.  Она специально разработана для современного веба и обычно выполняет тесты очень быстро, даже для сложных проектов тестирования |

Таблица 4. Преимущества и недостатки Playwright

|  |  |
| --- | --- |
| **Преимущества** | **Недостатки** |
| **Кросс-браузерная поддержка**: Playwright работает с несколькими браузерами, включая Chromium (Chrome, Edge), Firefox и WebKit (Safari). Эта совместимость с разными браузерами позволяет проводить последовательное тестирование в разных средах. | **Сложность использования**: Настройка Playwright может быть сложной, требует документации и опыта работы с подобными системами |
| **Кросс-платформенная поддержка**: Вы можете использовать Playwright для тестирования приложений на разных платформах, таких как мобильные устройства (Android), веб и настольные компьютеры (MacOS, Linux, Windows). | **Отчетность и логирование:** Playwright не предоставляет гибкие возможности для создания отчетов и логирования результатов тестов.Для этого требуются дополнительные плагины. |
| **Эмуляция мобильных устройств**: Playwright может эмулировать мобильные устройства, включая геолокацию, размер экрана и другие характеристики устройств. | **Совместимость с инфраструктурой**: Playwright плохо интегрируется с CI/CD-пайплайнами, Docker и другими инструментами. |
| **Поддержка нескольких языков программирования**: Изначально созданный для Node.js, Playwright теперь предлагает привязки для JavaScript, TypeScript, Python, Java и C#/.NET, что делает его доступным для широкого круга разработчиков и тестировщиков. |  |
| **Режимы без графического интерфейса и с графическим интерфейсом**: Он может запускать браузеры в режиме без графического интерфейса (headless) для более быстрого выполнения в тестовых средах и в режиме с графическим интерфейсом для разработки и отладки. |  |

Исходя из изложенного, каждый из существующих инструментов предоставляет слишком абстрактные и сложные интерфейсы, также они были разработаны до выхода современных сред программирования, поэтому есть необходимость разработки упрощенного фреймворка под конкретную современную платформу, которым смогут пользоваться специалисты по функциональному тестированию без большого опыта в программировании.

1.3 Выводы и постановка задач на дипломное проектирование

Целью дипломного проекта является проектирование набора классов, методов и интерфейсов, разработанных с учетом потребностей специалистов по тестированию.

С его помощью специалисты должны иметь возможность автоматизировать свои тестовые сценарии, соблюдая TDD (Test-driven development) и BDD (Behavior-driven development) подходы. Также он должен позволить производить автоматический запуск тестов для регрессионного тестирования, не тратя рабочее время на повторное выполнение однотипных задач. Подключаемый код фреймворка должен обладать простыми и понятными названиями переменных, методов и классов, своевременно обновлять подключенные пакеты-зависимости и совершенствоваться сообществом тестировщиков. При этом, фреймворк должен иметь возможность решить ряд конкретных задач функционального и интеграционного тестирования, формулируемых на основе требований тестировщиков. Важным является исправление слабых сторон, выявленных при рассмотрении аналогичных инструментов автоматизации[14].

Объект исследования – применение автоматизации процессов с помощью информационных технологий для повышения качества продукта, ускорения процесса разработки, снижения затрат и ресурсов на тестирование, уменьшения рисков появления ошибок в ПО и быстрого обнаружения дефектов.

Предмет исследования – разработка информационной системы для содействия специалистам по тестированию в автоматизации тестовых сценариев.

Исходя из изложенного материала, при разработке дипломного проекта, необходимо решить следующие задачи:

* реализовать метод поиска элемента по Xpath локатору, который позволяет хранить найденный элемент в виде переменной для дальнейшего переиспользования;
* создать классы для элементов разного типа, содержащих уникальный набор функций для симуляции действий над конкретным типом элемента;
* разработать метод симуляции клика и подобных действий, которые совершаются с элементом;
* предусмотреть подсветку активного элемента;
* предоставить возможность получения текста элемента, его HTML атрибутов и CSS значений, состояния отображения;
* разработать алгоритм работы с явными и неявными ожиданиями, а также ожиданий с условием;
* спроектировать функционал для работы с файловой системой для возможности загрузки файлов;
* определить основные функции пользователей в ролях «Пользователя» и «Контрибьютора»;
* обеспечить возможность внесения дополнительного или изменения существующего функционала фреймворка;
* оценить целесообразность финансовых вложений в разработку фреймворка и провести анализ конкурентной среды на коммерческом рынке;
* протестировать каждый метод и класс фреймворка, включая проверку работоспособности всех языковых операторов, ключевых слов и полей, правильность отображения данных при логировании, корректность работы симуляции браузера и т.д., а также совместимость фреймворка с различными браузерами, их версиями, и кроссплатформы, включая проверку корректной работы на разных операционных системах и мобильных устройствах разных версий;
* проанализировать эргономические показатели системы и удобство использования библиотеки с кодом фреймворка для пользователей, включая оценку удобства навигации, доступности, описания функций и др.
* предусмотреть рабочие условия и разработать меры безопасности для снижения рисков возникновения травм, оценить условия работы с компьютером, включая обоснование выбора системы искусственного освещения в помещении, вентилируемости, использования особых дисплеев и других мер безопасности

2 ЭРГОНОМИЧЕСКОЕ ПРОЕКТИРОВАНИЕ ФРЕЙМВОРКА ДЛЯ АВТОМАТИЗАЦИИ ТЕСТИРОВАНИЯ ВЕБ-ИНТЕРФЕЙСОВ

2.1 Разработка алгоритмов работы пользователя

Содержание функций информационной системы можно описать следующим образом:

1. Поиск элемента по Xpath локатору

Каждый элемент на странице имеет своё уникальное положение в иерархии DOM страницы. Для того, чтобы автоматизированный тест взаимодействовал с нужным элементом, используется специальный “путь” на языке Xpath.

1. Создание элементов разного типа

Для эргономичности фреймворка элементы разделены на типы, схожие на типы веб-элементов фронтенд разработчиков. Такие как Кнопка, Текстовое поле, Выпадающий список, Ячейка таблицы и т.д.

1. Клик по элементу

Основное действие над элементом – клик. Встречается в каждом тестовом сценарии, автоматизация позволяет кликать по элементам без задержки, что позволяет находить дополнительные дефекты.

1. Подсветка активного элемента

Во время выполнения автоматизированного теста действия могут происходить со слишком большой скоростью для восприятия человеком, также не всегда заметно, с каким именно элементом взаимодействует автотест. Подсветка решает эти проблемы.

1. Получение текста элемента

Получение текста, который появился после взаимодействия пользователя или уже был на странице нужно для сравнения его с тестовыми данными.

1. Получение HTML атрибутов элемента

Такие HTML аттрибуты как name, class, id, value могут пригодиться в процессе тестирования, т.к. являются динамическими и меняются после действий пользователя

1. Получение CSS значений элемента

Позволяет протестировать визуальную часть элемента, например: цвет элемента, границу, шрифт, размер и т.д.

1. Установка явных ожиданий и ожиданий с условием

Некоторые действия происходят с задержкой по разным причинам, чтобы автоматизированный тест не расценивал эту задержку как провал – необходимо устанавливать ожидания определенных условий, с гибкими настройками времени.

1. Заполнение текстовых полей

Текстовые поля тестируются тщательнее всего в веб-приложениях. Т.к. именно они обычно имеют особые условия, граничные значения и принимаемые паттерны. Заполнение текстовых полей помогает протестировать все возможные варианты.

1. Возможность загрузки файлов

Некоторые тестовые сценарии требуют загрузки файлов либо со страницы на ПК пользователя, либо наоборот на сервер. Специальный функционал поможет упростить этот процесс.

1. Получение состояния видимости и существования в DOM элемента

Даже если элемент нашелся в структуре страницы. Нужно убедиться, что он не перекрыт другими элементами и находится в зоне видимости пользователя. Проверки на видимость элемента решают эту задачу.

1. Прокрутка страницы до определенного элемента

Не всегда элемент находится в видимости пользователя. Также не всегда приложением пользуются со стандартным разрешением сторон. Появляется необходимость симулировать прокрутку колесиком мыши до появления искомого элемента.

1. Наследование базового класса страницы

Принятый в автоматизации паттерн Page Object довольно сложный для реализации с нуля. Поэтому во фреймворке уже реализован этот шаблон в базовом классе страницы, пользователю остается только наследовать его функционал и использовать в своем проекте.

1. Взаимодействие с выпадающим списком

Выпадающий список – особый тип элемента. Для работы с ним нужно реализовать такие функции как: раскрытие, закрытие, выбор элемента, получение всех элементов, получение элемента по умолчанию, получение выбранного элемента.

1. Сравнение ожидаемого и настоящего результата

Основой любого тестирование является сопоставление ожидаемого и актуального состояния. Во фреймворке пользователю предоставлены самые частоиспользуемые проверки, которые встречаются в популярных техниках тест-дизайна.

Для эффективного использования ресурсов разрабатываемого приложения необходимо оптимально распределить функции в системе между пользователем и ПК. Поскольку каждая система обладает своим собственным набором функций и особенностей управления, универсального способа распределения нет. Следовательно, требуется проведение анализа для определения оптимального распределения этих функций в рамках каждой конкретной системы. Основное внимание уделяется уменьшению действий, требуемых от пользователя при взаимодействии с системой. Этот подход обусловлен необходимостью предотвращения возможной негативной реакции пользователя на систему, вызванной излишним взаимодействием[15]. Результат распределения функций представлен в таблице 2.5.

Таблица 2.5 – Распределение функций между человеком и компьютером в проектируемой системе

|  |  |  |
| --- | --- | --- |
| Название функции | Что делает пользователь в системе | Что выполняет в системе компьютер |
| 1. Поиск элемента по Xpath локатору | Составляет путь к необходимому элементу на языке Xpath | Находит необходимый элемент и подготавливает к взаимодействию |
| 1. Создание элементов разного типа | Выбирает логически подходящий тип элемента | Инициализирует объект класса подходящего типа |
| 1. Клик по элементу | Вызывает метод “.Click()” у созданного элемента | Симулирует клик мыши пользователя по элементу |
| 1. Подсветка активного элемента | Вызывает метод “.Highlight()” у созданного элемента | Выделяет границу активного элемента красным цветом |
| 1. Получение текста элемента | Вызывает метод “.GetText()” у созданного элемента | Получает текст внутри HTML тега элемента и возвращает пользователю |
| 1. Получение HTML атрибутов элемента | Вызывает метод “.GetAttribute(string attributeName)” у созданного элемента | Получает HTML атрибут, выбранный пользователем |
| 1. Получение CSS значений элемента | Вызывает метод “.GetCssValue(string valueName)” у созданного элемента | Получает CSS значение элемента, выбранное пользователем |
| 1. Установка явных ожиданий и ожиданий с условием | Составляет условие для ожидания и значения тайм-аута | Устанавливает ожидание по заданному условию на указанное количество секунд |
| 1. Заполнение текстовых полей | Вызывает метод “.SendText(string text)” у созданного элемента | Заполняет текстовое поле выбранным пользователем текстом |
| 1. Возможность загрузки файлов | Вызывает метод “.Upload(string filePath)” у созданного элемента с тегом input | Находит файл на ПК по заданному пути, загружает его на страницу в необходимый элемент |
| 1. Получение состояния видимости и существования в DOM элемента | Обращается к свойству “State” элемента, выбирает необходимое состояние | Проверяет необходимое состояние элемента и возвращает true или false |
| 1. Прокрутка страницы до определенного элемента | Вызывает метод “.ScrollTo()” у созданного элемента | Симулирует прокрутку мыши, пока необходимый элемент не будет в области видимости |
| 1. Наследование базового класса страницы | Создает класс тестируемой страницы, наследует от базового класса “Form” | Передает базовые методы класса “Form” пользователю |
| 1. Взаимодействие с выпадающим списком | Вызывает метод “.SelectOption(string optionName)” у созданного элемента с тегом select | Симулирует открытие выпадающего списка и выбор определенного варианта |
| 1. Сравнение ожидаемого результата | Указывает ожидаемый результат и настоящий результат | Производит сравнение и останавливает тест в случае провала проверки, выводит сообщение с результатом |

Фреймворк должен быть легко масштабируемым и расширяемым для добавления нового и поддержки уже существующего функционала. Он не должен содержать грамматических ошибок и неправильных функциональных переходов[16].

Основываясь на функциональных возможностях информационной системы фреймворк для автоматизации тестирования, были выделены все возможные роли пользователей, а именно роли “Контрибьютор” и “Пользователь”.

Описание выделенных ролей пользователей системы представлено в таблице 2.1.

Таблица 2.1 – Роли пользователей системы

|  |  |  |
| --- | --- | --- |
| № | Роль | Описание |
| 1 | Контрибьютор | Имеет доступ к исходному коду фреймворка, может его модифицировать и предоставлять новую версию пользователям. |
| 2 | Пользователь | Использует фреймворк в качестве библиотеки для автоматизации тестирования своих сценариев. Может использовать фреймворк для своих нужд, но не имеет права изменять его реализацию. |

Для каждой выделенной роли составлен перечень полномочий пользователей в рамках системы, перечень предоставлен в таблице 2.2.

Таблица 2.2 – Полномочия пользователей системы

|  |  |  |
| --- | --- | --- |
| № | Полномочие | Описание полномочий |
| 1 | Получить доступ к исходному коду фреймворка | Имеет возможность склонировать репозиторий с исходным кодом на локальную машину |
| 2 | Создание Pull Request (PR) с изменением исходного кода | Создание предложения на внесение изменений в исходный код фреймворка |
| 3 | Код-ревью PR исходного кода | Возможность оставлять комментарии на предложенные изменения в исходный код, а также отвечать на них |
| 4 | Слияние изменений в основную ветку фреймворка | Возможность применить предложенные изменения в основную версию кода фреймворка и обновить его |
| 5 | Просмотр описания и журнала обновлений фреймворка | Доступ к документации фреймворка |
| 6 | Установить фреймворк в проект или решение | Доступ к добавлению фреймворка в виде зависимости или внешней библиотеки |
| 7 | Использовать доступные классы и методы фреймворка | Доступ к классам и методам фреймворка, не помеченным как private или internal. |
| 8 | Локально расширять фреймворк | Возможность добавлять методы расширения, классы наследники для проекта с использованием фреймворка, не меняя его исходный код |

Составлена матрица ролей и полномочий, на основании таблиц 2.1 и 2.2, отображающая доступность тех или иных возможностей системы в зависимости от роли пользователя. Матрица предоставлена в таблице 2.3.

Таблица 2.3 – Полномочия пользователей системы

|  |  |  |  |
| --- | --- | --- | --- |
| № | Полномочие | Контрибьютор | Пользователь |
| 1 | Получить доступ к исходному коду фреймворка | + | - |
| 2 | Создание Pull Request (PR) с изменением исходного кода | + | - |
| 3 | Код-ревью PR исходного кода | + | - |
| 4 | Слияние изменений в основную ветку фреймворка | + | - |
| 5 | Просмотр описания и журнала обновлений фреймворка | + | + |
| 6 | Установить фреймворк в проект или решение | + | + |
| 7 | Использовать доступные классы и методы фреймворка | + | + |
| 8 | Локально расширять фреймворк | + | + |

Более наглядное представление функций информационной системы и взаимодействие пользователей отображено на диаграмме вариантов использования информационной системы, изображенной на рисунке 2.1



Рисунок 2.1 – Диаграмма вариантов использования

Деятельность пользователя зависит от шагов теста, который он хочет автоматизировать. Тест может включать в себя различные тестируемые страницы и веб-элементы. Деятельность зависит от необходимых действий над элементом для симуляции: клик, получение текста , подсветка, прокрутка, ожидание, получение состояния.

На основе анализа распределения функций в информационной системе можно сделать вывод, что структура системы имеет 2 подсистемы: пользователь и контрибьютор. Следовательно, необходимо разработать алгоритмы работы человека в двух подсистемах “тестировщик-фреймворк-среда” и “контрибьютор-фреймворк-среда” в виде таблиц и блок-схем. Алгоритм работы пользователей представлен в таблице 2.6

Структурная схема для подсистемы “тестировщик-фреймворк-среда” представлена на рисунке 2.2.

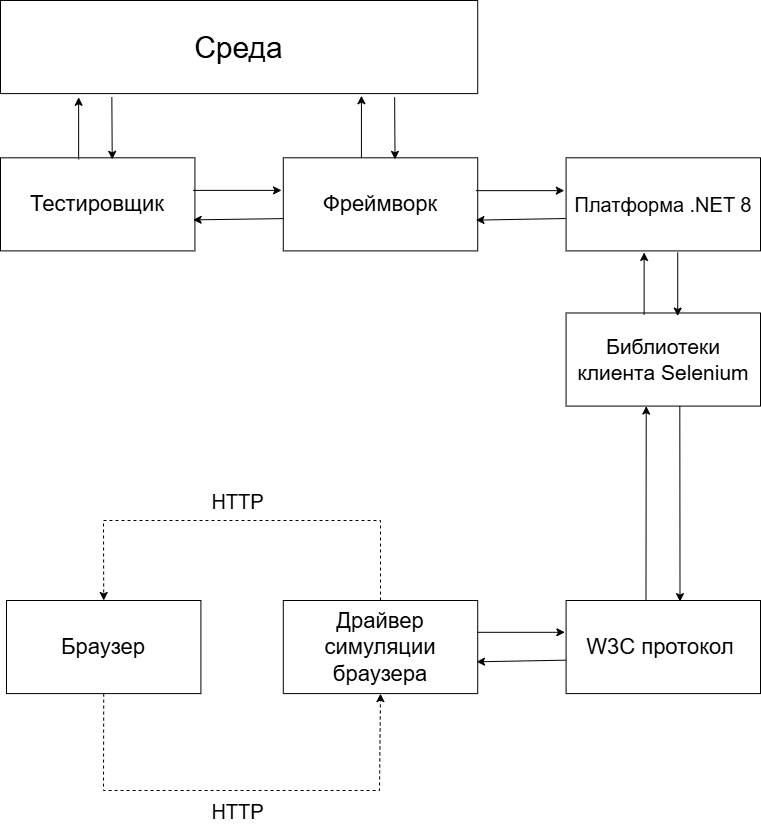


Рисунок 2.2 - Структурная схема для подсистемы “тестировщик-фреймворк-среда”

Структурная схема для подсистемы “контрибьютор-фреймворк-среда” представлена на рисунке 2.3.



Рисунок 2.3 - Структурная схема для подсистемы “тестировщик-фреймворк-среда”

По структурным схемам можно сделать вывод о том, что для тестировщика важнейшим аспектом взаимодействия с фреймворком является работа с симулируемым браузером, на котором и происходит автоматизированное тестирование, в то время как для контрибьютора первостепенно взаимодействие с исходным кодом проекта через репозиторий.

Таблица 2.6 – Алгоритм работы пользователя в процессе взаимодействия с приложением в подсистемах «тестировщик-фреймворк-среда» и “контрибьютор-фреймворк-среда”.

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Содержание операции | Обращение к средствам отображения информации | Обращение к органам управления |
| Алгоритм для роли “Пользователь” | | | |
| 1 | Подключение фреймворка | Открытие окна менеджера пакетов, добавление библиотеки фреймворка | Нажатие кнопки “Добавить пакет”, кнопка изменила состояние нажатия |
| 2 | Запуск автоматизированного теста | Открытие окна обозревателя тестов, выбор автоматизированного теста для запуска | Нажатие кнопки “Run”, прогресс бар показывает состояние запуска автоматизированного теста |
| 3 | Выбор браузера для симулируемой среды | Конфигурационный json файл, заполнение ключей и значений файла | Добавление значения к полю “browserName” |

Продолжение таблицы 2.6

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Содержание операции | Обращение к средствам отображения информации | Обращение к органам управления |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 4 | Дополнительные настройки браузера | | Конфигурационный json файл, заполнение ключей и значений файла | | Добавление значения к полю “browserVersion” | |
| 5 | Выбор тестируемой страницы | | Класс наследник базовой страницы, выбранный для проведения автоматизированного тестирования | | Ввод названия выбранной страницы в базовый конструктор класса страницы | |
| 6 | Добавление элемента | | У драйвера метод добавления элемента | | Ввод пути Xpath в параметр “locator” | |
| 7 | Симуляция клика | | У необхоимого элемента отображение метода симуляции“Click()” | | Вызов метода у объекта наследника базового элемента | |
| 8 | Подсветка элемента | | У необхоимого элемента отображение метода симуляции “Highlight()” | | Вызов метода у объекта наследника базового элемента | |
| 9 | Получение текста | | Отображение метода симуляции “GetText()” | | Вызов метода у объекта наследника базового элемента | |

Продолжение таблицы 2.6

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Содержание операции | Обращение к средствам отображения информации | Обращение к органам управления |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 10 | Получение HTML атрибута | | У необхоимого элемента отображение метода симуляции “GetAttribute()” | | Ввод необходимого аттрибута в параметр “attribute” | |
| 11 | Получение CSS значения | | У необхоимого элемента отображение метода симуляции “GetCSSValue()” | | Ввод необходимого значения в параметр “value” | |
| 12 | Установка ожидания с условием | | Объект класса ожидания, отображение метода симуляции ожидания состояния | | Передача необходимого условия в параметр “condition” | |
| 13 | Заполнение текстового поля | | У необхоимого элемента отображение метода симуляции “SendText()” | | Передача необходимого текста в параметр “text” | |
| 14 | Загрузка файла в папку фреймворка | | У необхоимого элемента отображение метода симуляции “UploadFile()” | | Передача пути к файлу в параметр “filePath” | |

Продолжение таблицы 2.6

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Содержание операции | Обращение к средствам отображения информации | Обращение к органам управления |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 15 | Получение состояния элемента | | У необхоимого элемента обращение к свойству “Condition” | | Выбор необходимого состояния в выпадающем списке вариантов |
| Алгоритм для роли “Контрибьютор” | | | | | |
| 16 | | Создание Pull Request | | Ветка для разработки обновлений исходного кода фреймворка по автоматизации | Нажатие кнопки “Create pull request”, отображается созданный запрос на изменения в виде слияния веток |
| 17 | | Получение доступа к исходному коду проекта | | Репозиторий исходного кода проекта с ветвями разработки и файлами конфигурации проекта | Обращение к команде “Clone”, отображение загрузки файлов проекта |
| 18 | | Добавление Code-review | | Отображение запроса на внесение изменений с подсветкой добавленных и удаленных частей кода | Добавление комментария на предложенные изменения, нажатие кнопки “Approve”. |

Алгоритмы, приведенные выше, описывают работу пользователя-тестировщика со всеми функциями фреймворка для автоматизации тестирования на платформе .NET 8.

2.2 Разработка эргономических требований и сценария информационного взаимодействия

Эргономические требования (ЭТ) – это требования к системе в целом, ее отдельным подсистемам, оборудованию, рабочей среде, а также человеку-пользователю компьютера, определяемые свойствами человека и устанавливаемые для обеспечения его эффективной и безопасной деятельности[17]. Задачей эргономического проектирования является проектирование деятельности пользователей.

Для разрабатываемой системы выделены психофизиологические, психологические и социально-психологические группы эргономических требований, в соответствии с принятой в инженерной психологии и эргономике классификацией.

Групповые эргономические показатели определяют эргономические свойства системы. В общем случае принято выделять 4 таких свойства, а именно: «управляемость», «обслуживаемость», «освояемость» и «обитаемость». В связи с тем, что формат фреймворка отличается от стандартного веб-сайта или настольного приложения, свойства «обслуживаемость» и «обитаемость» не являются значимыми для проектируемой системы. Для фреймворка можно ограничиться только двумя соответствующими формату свойствами: «управляемость» и «освояемость».

Основываясь на рекомендациях и требованиях, приведенных в нормативной, справочной и научной литературе, разработана спецификация эргономических требований, проведена их группировка. Дальнейший анализ представляет собой определение общих эргономических требований. Эргономические требования к системе приведены в таблице 2.7

Таблица 2.7 – Эргономические требования к проектируемой системе

|  |  |  |
| --- | --- | --- |
| Эргономические свойства | Эргономические показатели | |
| Управляемость | Группа ЭТ | Номенклатура ЭТ |
| Психофизиологические | * Возможность настройки контрастности кода, ключевых слов и операторов для обеспечения необходимой видимости; * оптимальный размер и толщина знаков на экране с учетом порога чувствительности зрения человека; * Оптимальный размер шрифта, межбуквенное расстояние (кернинг), межсловное расстояние, межстрочный интервал в соответствии чувствительности зрения человека; * соответствие количества одновременно предъявляемых стимулов объему восприятия человека; * соответствие объемов информации, требующей запоминания, возможностям памяти человека. |
| Психологические | * Соответствие цветов надписей и знаков стереотипам восприятия; * пространственное расположение информации на экране должно соответствовать оптимальному порядку ее восприятия; * отсутствие неоднозначного толкования требований, инструкций и команд; * соответствие количества одновременно предъявляемых сигналов возможностям внимания человека; * соотношение формата составления и размера тестовых сценариев, позволяющим воспринимать их как однотипные; * один и тот же характер команд на протяжении всего периода работы в системе в схожих ситуациях; * наличие подсказок о следующих шагах работы в системе; * наличие предупреждений о нежелательных последствиях некоторых действий. |
| Социально-психологические | * Наличие ограничений доступа к определенным функциям в зависимости от роли в конкретной ситуации. |
| Освояемость | Психологические | * Соответствие сложности инструкций времени, отводимому на их усвоение; * логичная и интуитивно понятная структура фреймворка. |

Разработанный фреймворк должен полностью соответствовать техническому заданию и позволять успешно решать поставленные перед ним цели и задачи, а в процессе использования – надежно и устойчиво функционировать.

Рассмотрим подробнее эргономические требования к информации, представляемой на экране дисплея. Для точного считывания информации и комфортных условий ее восприятия работа с дисплеями должна проводиться при таких характеристиках, которые входят в оптимальные или предельно допустимые (при кратковременной работе) диапазоны [17].

Контраст изображения должен быть не менее 3:1. Ширина контура знака должна быть в пределах от 0,25 до 0,5 мм.

При необходимости точной идентификации цвета знака в рядах буквенно-цифровых символов высота знака не должна быть менее 20` и 30` для обособленных знаков при проектном расстоянии наблюдения.

Поля восприятия графической информации должны иметь следующие размеры:

* поле точного восприятия: 3⁰ вверх-вниз (или 2,6-2,7 см), 7⁰ вправо-влево (или 4,8-5,2 см) от оси зрения;
* поле опознания расположения: вверх 25⁰ (или 24-28 см), вниз 35⁰ (или 34 ˗ 40 см), вправо и влево по 32⁰ (или 31-37 см) от оси зрения;
* поле высокозначимой информации: 15⁰ (или 14-16 см) во все стороны от оси зрения;
* поле главного объекта: 10⁰ (или 9-10 см) во все стороны от оси зрения.

В поле главного объекта не должно находиться более 4-6 второстепенных объектов. Надписи, обозначающие объекты или органы управления, должны быть краткими, однозначно воспринимаемыми и читаться слева направо. В них допускается использование только тех слов, которые хорошо известны пользователю.

Параметры предъявляемого на экране текста должны удовлетворять следующим требованиям:

* высота знака – не менее 3 мм;
* отношение ширины буквы, цифры к высоте – в пределах 0,76-0,8;
* толщина линии обводки в прямом контрасте – в пределах 10-15% от высоты знака, в обратном контрасте – в пределах 12-16% от высоты знака;
* расстояние между буквами от 20 до 50 % ширины знака, а между словами от 1 до 1,5 ширины знака;
* расстояние между строками – 1,5-2 высоты знака;
* длина строки – 40-80 знакомест.

Эскиз рабочего окна класса тестируемой страницы и окна тестового сценария в виде прототипов представлены соответственно на рисунках 2.4 и 2.5.



Рисунок 2.4 - Прототип класса тестируемой страницы



Рисунок 2.5 - Прототип тестового сценария

С учётом приведенных эргономических свойств и требований, разработан фреймворк для автоматизации тестирования веб-интерфейсов, который удовлетворяет условиям технического задания и рассмотренным выше эргономическим требованиям.

2.3 Эргономическая оценка проектируемой системы и выводы

Заключительным этапом эргономического проектирования системы является эргономическая оценка проекта. Эргономическая оценка представляет собой определение степени реализации эргономических требований. В данном случае она выполняется экспертным методом, при этом в качестве эксперта выступает сам разработчик. Сущность эргономической оценки проекта через экспертный метод заключается в проведении интуитивно-логического анализа с количественной оценкой суждений и обработкой результатов.

Последующий анализ включает в себя выявление единичных эргономических показателей, которые составляют каждую из перечисленных групп показателей.

На основе сформулированных ранее эргономических требований составим спецификацию единичных эргономических показателей (ЕЭП) (таблица 2.8)

Таблица 2.8 – Общие эргономические требования к проектируемой системе и соответствующие им единичные эргономические показатели

|  |  |  |
| --- | --- | --- |
| Эргономическое свойство «Управляемость» | | |
| Группа ЭТ | Эргономическое требование | Единичный эргономический показатель |
| Психофизиологические | 1. Возможность настройки контрастности кода, ключевых слов и операторов для обеспечения необходимой видимости | Наличие шаблонов контрастности разработанных с учетом оптимального контраста, что обеспечивает четкость различения между определенными частями кода |
| 1. Оптимальный размер и толщина знаков на экране с учетом порога чувствительности зрения человека | Наличие регулировки размера и толщины шрифтов на экране пользователем с учетом личного уровня зрения |
| 1. Оптимальное межбуквенное расстояние (кернинг), межсловное расстояние, межстрочный интервал в соответствии чувствительности зрения человека | Отступы между буквами, словами и строками в текстовых элементах настроенное так, чтобы обеспечить комфортное чтение кода для различных категорий пользователей |

Продолжение таблицы 2.8

|  |  |  |
| --- | --- | --- |
| Эргономическое свойство «Управляемость» | | |
| Группа ЭТ | Эргономическое требование | Единичный эргономический показатель |

|  |  |  |
| --- | --- | --- |
| Психофизиологические | 1. Соответствие количества одновременно предъявляемых стимулов объему восприятия человека | Наличие контроля одновременного количество стимулов – при желании можно разбить большие конструкции на несколько методов или файлов, уменьшив количество стимулов |
| 1. Соответствие объемов информации, требующей запоминания, возможностям памяти человека | Наличие подсказок, описывающих каждый используемый компонент или использование письменных комментариев в коде |
| Психологические | 1. Соответствие цветов надписей и знаков стереотипам восприятия | Использование зеленого цвета для уведомления о успешном прохождении теста и красного при наличии ошибки в коде или неуспешном завершении теста. Синий цвет используется как цвет активных элементов или для выделения фрагмента кода |
| 1. Пространственное расположение информации на экране должно соответствовать оптимальному порядку ее восприятия | Порядок и расположение модулей тестового сценария относительно друг друга |
| 1. Отсутствие неоднозначного толкования требований, инструкций и команд | Однозначность формулировки требований, инструкций и команд |
| 1. Соответствие количества одновременно предъявляемых сигналов возможностям внимания человека | Наличие инструмента контролирования предъявляемых сигналов для пользователя, имеется возможность уменьшать |

Продолжение таблицы 2.8

|  |  |  |
| --- | --- | --- |
| Эргономическое свойство «Управляемость» | | |
| Группа ЭТ | Эргономическое требование | Единичный эргономический показатель |

|  |  |  |
| --- | --- | --- |
| Психологические | 1. Соотношение формата составления и размера тестовых сценариев, позволяющим воспринимать их как однотипные | Наличие шаблона формата сценария. Размеры подобных сценариев варьируются в диапазоне 0-5%, что позволяет воспринимать их как однотипные |
| 1. Один и тот же характер команд на протяжении всего периода работы в системе в схожих ситуациях | Наличие однообразных структур и команд в схожих сценариях |
| 1. Наличие подсказок о следующих шагах работы в системе | Наличие подсказок, описывающих все возможные шаги пользователю |
| 1. Наличие предупреждений о нежелательных последствиях некоторых действий | Выделение компилятором опасного кода желтым цветом при возникновении возможности нежелательных последствий. |
| Социально-психологические | СП-1. Наличие ограничений доступа к определенным функциям в зависимости от роли в конкретной ситуации | Предоставление прав доступа к внутреннему коду фреймворка только для соучастников репозитория |
| Эргономическое свойство «Освояемость» | | |
| Психологические | 1. Соответствие сложности инструкций времени, отводимому на их усвоение | Представление инструкции по пользованию фреймворком с примерами, время ознакомления контролирует пользователь |
| 1. Логичная и интуитивно понятная структура приложения | Наличие структуры фреймовка и названий переменных, классов, методов в соответствии с общепринятыми конвенциями чистого кода |

Проведем оценку значений единичных эргономических показателей и занесем результат в таблицу 2.9.

Таблица 2.9 – Оценка единичных и групповых эргономических показателей проектируемой системы

|  |  |  |
| --- | --- | --- |
| Эргономическое свойство «Управляемость» | | |
| Группа ЭП | Значение единичных ЭП | Значение групповых ЭП |
| Психофизиологические | ПФ-5 = 0  ПФ-1, ПФ-2, ПФ-3, ПФ-4 = 1 | 4 x 1 / 5 = 0,8 |
| Психологические | П-2, П-7 = 0  П-1, П-3, П-4, П-5, П-6, П-8 = 1 | 6 x 1 / 8 = 0,75 |
| Социально-психологические | СП-1 = 1 | 1 x 1/1 = 1 |
| Эргономическое свойство «Освояемость» | | |
| Психологические | П-1, П-2 = 1 | 1 ∙ 1 / 1 = 1 |

Проведем оценку эргономических свойств, значимых для проектируемой системы. Эргономические свойства системы определяются как совокупность групповых эргономических показателей. Различные групповые эргономические показатели играют разную роль в реализации конкретного эргономического свойства, поэтому групповые эргономические показатели имеют свои весовые коэффициенты. Сумма весовых коэффициентов по каждому эргономическому свойству должна равняться единице. Значения весовых коэффициентов групп ЭП проектируемой системы представлены в таблице 2.10.

Таблица 2.10 – Распределение значений весовых коэффициентов групп ЭП

|  |  |
| --- | --- |
| Эргономическое свойство «Управляемость» | |
| Групповой ЭП | Значение весового коэффициента |
| Психофизиологический | 0,3 |
| Психологический | 0,6 |
| Социально-психологический | 0,1 |
| Эргономическое свойство «Освояемость» | |
| Психологический | 1 |

Для расчета эргономических свойств воспользуемся формулой:

ЭСв = ∑ 𝛼𝑖 ∙ ГЭПi,

где:

ЭСв – эргономическое свойство;

𝛼𝑖–весовые коэффициенты, присвоенные различным групповым эргономическим показателям;

ГЭПi – групповые эргономические показатели.

ЭСвуправляемость = (0,3 ∙ 0,8) + (0,6 ∙ 0,75) + (0,1 ∙ 1) = 0,24 + 0,45 + 0,1 = 0,79

ЭСвосвояемость = 1 ∙ 1 = 1

Различные эргономические свойства играют разную роль в формировании эргономичности интерфейса проектируемой системы, поэтому эргономическим свойствам также присваиваются весовые коэффициенты, соответствующие их вкладу в эргономичность. Значения весовых коэффициентов эргономических свойств проектируемой системы представлены в таблице 7.4.

Таблица 2.11 – Распределение значений весовых коэффициентов эргономических свойств

|  |  |
| --- | --- |
| Эргономическое свойство | Значение весового коэффициента |
| Управляемость | 0,7 |
| Освояемость | 0,3 |

Эргономичность вычисляется по формуле:

Э = ∑ 𝛽𝑖 ∙ ЭСвi,

где:

Э – эргономичность;

𝛽𝑖–весовые коэффициенты, присвоенные различным эргономическим свойствам;

ЭСв – эргономическое свойство.

Э = (0,7 ∙ 0,79) + (0,3 ∙ 1) = 0,553 + 0,3 = 0,853

На основании полученного значения показателя эргономичности можно сделать вывод о высокой степени реализации эргономических требований.

Полученное значение группового эргономического показателя оценивается с учетом того, что показатели в рамках 0,8–1,0 расцениваются как «отлично» – эргономические характеристики разработки соответствуют необходимым значениям.

Далее произведем анализ единичных показателей, значения которых не соответствуют эргономическим требованиям (получили «нулевые» оценки). Результаты данного этапа представлены в таблице 2.7.

Таблица 2.7 – Рекомендации по улучшению эргономичности проектируемой системы

|  |  |
| --- | --- |
| Невыполненное эргономическое требование | Предложение по улучшению эргономичности |
| Соответствие объемов информации, требующей запоминания, возможностям памяти человека | Добавить всплывающие подсказки для пользователя, которые визуально отображают информацию о используемом методе фреймворка, без необходимости запоминать эту информацию |
| Пространственное расположение информации на экране должно соответствовать оптимальному порядку ее восприятия | Разработать шаблон расположения информации тестового сценария. Описание элементов в начале класса, а действий над элементов в конце позволит быстро анализировать страницу. |
| Наличие подсказок о следующих шагах работы в системе | Добавить дорожную карту использования фреймворка с автоматической рекомендацией оптимального следующего шага для автоматизации сценария |

Таким образом, в процессе эргономического проектирования фреймворка для автоматизации тестирования веб-интерфейсов на платформе .NET 8. Были выделены роли для взаимодействия с системой, описаны их полномочия, разработаны алгоритмы работы пользователя, проанализрованы функции и их распределение в информационной системе, разработаны эргономические требования и сценарии информационного взаимодействия. Эргономическая оценка показала, что эргономичность системы в целом отличная.

3 РАЗРАБОТКА ФРЕЙМВОРКА ДЛЯ АВТОМАТИЗАЦИИ ТЕСТИРОВАНИЯ НА ПЛАТФОРМЕ .NET 8.

3.1 Разработка структуры программного обеспечения фреймворка

Согласно разделу 1.3 «Выводы и постановка задач на дипломное проектирование», на этапе проектирования необходимо указать взаимодействие модулей фреймворка между собой. Для отображения и детальной визуализации необходимо создать следующие диаграммы:

* структурная схема системы;
* диаграмма классов фреймворка;
* диаграмма деятельности пользователей фреймворка;
* диаграмма вариантов использования;
* эскизы рабочих окон программы.

Диаграмма классов фреймворка отображает наличие различных модулей системы, их взаимосвязи между собой, основные методы, свойства и поля каждого класса, а также необходимые конфигурационные файлы. Диаграмма классов представлена на рисунке 3.1.

Диаграмма классов фреймворка вынесена на чертеж ГУИР 502900.002 ПД.

Модуль класса «BrowserFactory» является основной частью фреймворка и содержит в себе объект самого веб-драйвера, который открывает необходимый браузер заданной версии и является средой для симуляции действий пользователя в тестируемой системе. Также этот класс предоставляет публичные методы GetDriver и QuitBrowser для передачи объекта драйвера в остальные классы фреймворка и для завершения работы браузера с очисткой процесса драйвера из диспетчера задач. Получение драйвера реализовано по порождающему паттерну “фабрика”. Также в классе представлены приватные методы InitDriver и InitDriverByType для инициализации браузера в текущий поток с учетом заданной конфигурации. Для драйвера предоставлена возможность перехода по заданной ссылке, получения списка текущих вкладок и переключение между ними, возможность сделать и сохранить скриншот экрана, возможность управления размером окна, возможность добавления и удаления куки браузера, предоставление логов действий, управление временем ожидания,

Класс «BrowserSettings» представляет из себя модель с основными параметрами браузера, которые считываются из конфигурационного файла, данные можно просмотреть и изменить при необходимости. Примером основного параметра является свойство BrowserName, которое в первую очередь передается в метод создания драйвера определенного типа. Дополнительными параметрами являются: язык браузера, инкогнито режим, измененный путь к папке загрузок, режим подсвечивание активного элемента.

Модуль «BrowserJsActions» содержит в себе объект драйвера и предоставляет возможность использования в симулируемом браузере особого кода на языке JavaScript. Самыми используемыми скриптами являются скрипты работы с загрузкой страницы, например, StopPageLoading, который принудительно останавливает загрузку страницы или наоборот скрипт, который ожидает полной загрузки страницы. Дополнительно через JavaScript можно работать со вкладками, ссылками, свойством document страницы.

Утилитарный класс «BrowserSettingReader» предоставляет возможность считывать тестовые или конфигурационные данные с файлов формата json. Предоставляет возможность получить конкретного значения по названию ключа, либо список объектов по названию ключа, либо коллекцию типа ключ-значения со всеми данными из файла. Класс содержит в себе изменяемый путь к конфигурационному файлу, объект настроек браузера, который заполняется значениями по мере считывания файла, а также приватные метод считывания конфигурационных параметров.

Абстрактный класс «BaseElement» является вторым по важности модулем в системе. Базовый класс для каждого типа элемента, который описывает базовую логику работы с элементами, такую как клик, нахождение элемента на странице, получение текста, получение состояния видимости элемента, получения аттрибута и т.д.. Хранит в себе Xpath локатор, по которому можно найти элемент на странице, свойство Actions, предоставляющее дополнительные действия с мышью и клавиатурой, такие как клик правой кнопкой мыши или одновременное зажатие функциональных клавиш на клавиатуре, свойство JsActions хранит в себе объект типа «ElementJsActions», описанный далее, свойство Element хранит уже найденный на тестируемой странице веб-элемент, свойство Visibility хранит в себе объект класса «ElementVisibility».

Наследники «BaseElement» в виде «TextBox», «Button», «ImageLabel», «Dropdown», «RadioButton», «TextLabel», «CheckBox», «Container», предоставляют пользователю дополнительные возможности взаимодействия с конкретными типами элементов. Например, ввод текста в элемент типа TextBox или выбор элемента по значению в «Dropdown».

Статический класс «WaitConditions» предоставляет возможность создания ожидания с условием. Содержит в себе объект веб-драйвера, необходимый для проверки выполнения условия ожидания, числовое значения количества секунд для ожидания, интервал повторной попытки совершения действия перед проверкой на ожидаемый результат, генератор сообщения об ошибке. Основными методами являются WaitForDisplayed, WaitForDisappeared, WaitForNotExist, которые позволяют прверить видимость или существование элемента на странице симулируемой среды. Также класс предоставляет пользователю методы для ожидания видимости элемента на экране, ожидания состояния возможности кликнуть на элемент, ожидания любого условия, возвращающего true или false, а также самый гибкий способ ожидания любого события, переданного пользователем через аргументы.

Универсально-обобщенный (Generic) класс «ElementsCollection» представляет из себя коллекцию нескольких элементов, которые являются наследниками класса «BaseElement» в виде списка. Хранит в себе экземпляр драйвера для поиска элементов, общий локатор для всех элементов, а также методы для передачи коллекции в другие модули и метод получения количества элементов.

Класс «ElementsVisibility» предоставляет информацию о состоянии элемента на странице, с его помощью можно узнать находится ли элемент в зоне видимости, готов ли элемент к взаимодействию, существует ли элемент в коде страницы, отсутствует ли элемент.

Класс «ElementsJsActions» предоставляет набор дополнительных действий над элементом с использованием языка JavaScript. Методы этого класса позволяют удалить элемент, подсветить его красной рамкой и прокрутить страницу до элемента.

Абстрактный класс «PageObject» представляет собой базовый класс для любого созданного класса страницы или формы в проекте по автоматизированному тестированию. Создан для соблюдения одноименного паттерна «Page Object», широко используемого в тестировании. Содержит в себе уникальный элемент страницы, название и методы проверки на успешное открытие конкретной страницы, также дает возможность получения названия страницы пользователю.

На диаграммах деятельности будет отражен функционал, доступный каждому из участников веб-сайта по ролям.

Алгоритм работы пользователя в роли «Пользователь» с системой представлен на рисунке 3.2 и и предполагает следующую последовательность действий:

* пользователь выбирает браузер и его версию;
* если требуются дополнительные настройки браузера, то заполняются необходимые параметры браузера в конфигурационном файле проекта;
* выбирается тестируемая страница или форма, на которой будут симулироваться действия тестировщика;
* на выбранную страницу добавляется уникальный элемент для проверки открытия нужной страницы в тесте;
* при необходимости, добавляются дополнительные элементы страницы, над которыми нужно произвести определенные действия, в рамках тестового сценария;
* после добавления элементов выбираются те, для котороых необходима симуляция действия в тесте;
* затем предоставляется выбор действия для симуляции над текущим элементом;
* если пользователь не добавляет больше элементов для симуляции, то взаимодействие с фреймворком заканчивается и пользователь может запускать созданный автоматизированный тест.



Рисунок 3.2 – Алгоритм работы пользователя в роли «Пользователь»

Одной из самых важных частей алгоритма является момент выбора действия для симуляции над текущим элементом. Алгоритм работы пользователя в роли «Пользователь» с выбором действия для симуляции предоставляет обширные возможности для взаимодействия с элементом и вынесен в отдельную блок-схему, представленную на рисунке 3.3, и предполагает следующую последовательность действий:

* пользователь заполняет путь XPath, необходимый для нахождения элемента на веб-странице;
* пользователю предоставлены варианты действия для симуляции, из которого он выбирает подходящий. У пользователя есть возможность выбрать один вариант из списка: «Действие с элементом», «Ввести текст в элемент», «Получить CSS значение», «Получить аттрибут», «Получить состояние элемента»;
* при выборе «Действие с элементом» имеется возможность выбрать один из вариантов: «Клик», «Подсветить элемент», «Прокрутить экран до элемента», «Получить текст», «Ожидать состояние элемента», «Навести мышь на элемент», «Удалить элемент». После выбора пользователь переходит к симуляции выбранного действия;
* при выборе «Ввести текст в элемент» указывается текст для ввода и происходит симуляция ввода текста;
* при выборе «Получить CSS значение» пользователем указывается название необходимого CSS свойства и он получает его значение;
* при выборе «Получить аттрибут» указывается название необходимого аттрибута и пользователю предоставляется его значение;
* при выборе «Получить состояние элемента» пользователю предоставляется выбор одного из трёх возможных состояний элемента: «Видимый», «Существует в верстке», «Активный». После чего получает в ответ true или false, в зависимости значения элемента;
* когда пользователь симулировал выбранное действие или получил запрошенные данные, то предлагается добавить ещё один элемент или завершить выполнение выбора действия для симуляции.



Рисунок 3.3 – Алгоритм работы пользователя в роли «Пользователь» с выбором действия для симуляции

Алгоритм работы пользователя в роли «Контрибьютора» с системой представлен на рисунке 3.4 и предполагает следующую последовательность действий:

* Контрибьютор клонирует репозиторий с исходным кодом фреймворка для синхронизации с текущей версией фреймворка;
* В исходный код фреймворка вносятся необходимые изменения;
* Создается Pull Request для применения внесенных изменений;
* Производится слияние изменений;
* Если произошел конфликт слияния, то на конфликтирующий код добавляется Code-review;
* Если конфликт не произошел или Code-review добавлено, то алгоритм завершается.

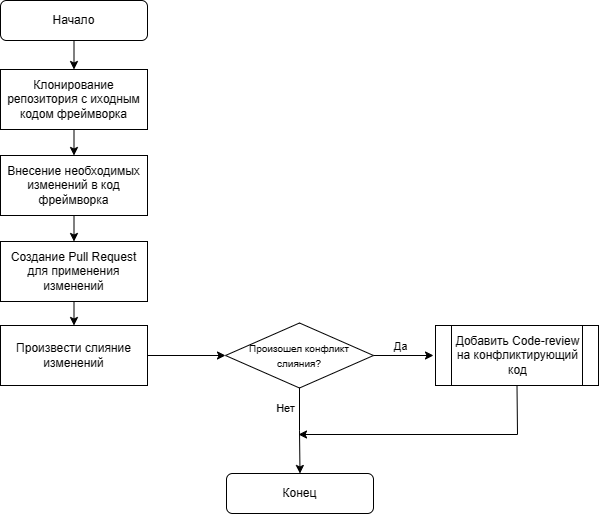


Рисунок 3.4 – Алгоритм работы пользователя в роли «Контрибьютор»

Данные алгоритмы работы пользователя дают общее представление о том, каким образом происходит автоматизация тестирования с помощью фреймворка. Описываются основные шаги и действия пользователей-тестировщиков, которые необходимо выполнить, чтобы получить доступ к основным функциональным возможностям фреймворка и автоматизировать тестовый сценарий.

Поскольку фреймворк содержит двух актеров – для каждого из них будет составлено подробное описание вариантов использования. Каждый вариант использования определяет определенный набор действий, которые система выполняет при взаимодействии с актером.

Актеру в роли пользователя «Пользователь» доступны возможности:

1. Выбор одного из действий для симуляции над элементом. Для этого пользователю необходимо выбрать один из вариантов: «Кликнуть», «Подсветить», «Прокрутить», «Получить текст», «Ожидать состояние», «Навести мышь», «Удалить»;
2. Симулировать выбранное действие над элементом;
3. Получить состояние выбранного элемента;
4. Найти элемент на странице по заданному пути XPath;
5. Заполнить путь XPath для нахождения элементов;
6. Добавить уникальный элемент страницы;
7. Выбрать страницу для произведения тестирования.

Актеру в роли «Контрибьютор», доступен такой функционал, как:

Создание Pull Request с внесенными изменениями для обновления функционала фреймворка;

Получение доступа к исходному коду фреймворка для его изучения и внесения изменений;

Произвести слияние в основную ветку фреймворка для публикации новой версии библиотеки;

Добавить Code-review на чужой Pull Request с исправлениями.

Диаграмма вариантов использования вынесена на чертеж ГУИР 508900.002 ПД.

3.2 Разработка алгоритма работы фреймворка

Первый шаг, который выполняет пользователь при начале работы с фреймворком – это подключает фреймворк к своему проекту автоматизации в виде дополнительной зависимости (библиотеки). Подключение возможно несколькими способами и зависит от используемого IDE, но универсальным является подключение через командную строку. Пример подключения через Windows PowerShell предоставлен на рисунке 3.5.



Рисунок 3.5 – Пример подключения фреймворка для автоматизации веб-интерфейсов в проект.

После подключения фреймворка пользователю необходимо отдельно импортировать типы из пространства имен фреймворка с использованием директивы using. Пример подключения типов страниц и элементов фреймворка представлен на рисунке 3.6.



Рисунок 3.6 – Пример импортирования классов страниц и элементов в проект

После необходимой подготовки пользователь создает класс тестируемой страницы. Для этого после директив using определяется пространство имен текущего класса страницы, которое зависит от положения тестируемой страницы относительно других страниц тестируемого веб-сайта. Внутри области пространства имен определяется сам класс тестируемой страницы. Для его создание описывается модификатор доступа, при необходимости, дополнительные ключевые слова для классов, затем имя выбранной для тестирования страницы. После определения класса он наследуется от базового класса страниц фреймворка для получения полей и методов, необходимых для полноценного тестирования страницы. Внутри класса определяется необходимое количество приватных элементов текущей страницы, отдельно выбирается тип элемента и составляется XPath локатор к нему. После описания элементов определяется конструктор класса тестируемой страницы, а также заполняется конструктор базовой страницы, который принимает параметром путь к уникальному элементу данной страницы. В конце класса страницы добавляются публичные методы для взаимодействия с созданными элементами. Методы могут иметь пустой возвращаемый тип или возвращать данные о элементе, такие как его состояние видимости или текст. Также для каждого метода определяется уникальное название. Для визуального представления модулей класса страницы был разработан прототип класса страницы, представленный на рисунке 3.7.



Рисунок 3.7 – Прототип класса тестируемой страницы

Созданный класс страниц используется в тестовом классе или классе определения шагов BDD сценария. Сами сценарии описываются в отдельном файле типа feature. Данный файл описывает сценарий понятным человеку языку, без использования языка программирования. В сценарии описывается его название, описание, включающее название роли, желаемого действия и условия выполнения. Также в сценарии возможно добавление предусловия с действиями, которые будет выполнены перед началом основных шагов.

После предусловия описывается сам сценарий с использованием языка документации Gherkin. Gherkin - язык документации, который описывает поведение системы (BDD) при помощи сценариев. Каждый сценарий написан понятным для бизнеса (неискушенного пользователя) языком и полностью описывает какую-то часть функционала[18].

Каждый сценарий состоит из трех ключевых слов: Given, When, и Then. Эта структура помогает четко и последовательно описывать поведение системы.

* Given: этот шаг описывает начальное состояние системы перед выполнением каких-либо действий. Это контекст, в котором происходит тестирование.
* When: здесь описываются действия, которые выполняются над системой: когда что-то происходит или когда пользователь взаимодействует с системой.
* Then: шаг описывает ожидаемое состояние системы после выполнения определенных действий. Это результат, который ожидается после того, как система прошла через начальное состояние и определенные действия [19].

Структура сценария представлена в виде прототипа на рисунке 3.8



Рисунок 3.8 – Прототип тестового сценария

3.3 Выводы и оценка результатов разработки

В рамках выполнения разработки фреймворка для автоматизации тестирования были разработаны программный модуль системы, структура классов фреймворка, диаграмма деятельности пользователя и контрибьютора, диаграмма вариантов использования. Был описан алгоритм работы фреймворка с подробной схемой взаимодействия пользователя с фреймворком, фреймворка с драйвером и драйвера с браузером. При разработке фреймворка был использован язык программирования C# 12 версии, а также инструменты платформы .NET 8. Для подключения фреймворка в качестве библиотеки был использован nuget packet manager. В качестве среды разработки (IDE) была использована Visual Studio 2022. Для создания проекта с автоматизированными тестами использовалась библиотека NUnit и встроенный Test Explorer. Как основа фреймворка для взаимодействия c браузером использовался Selenium WebDriver. Для осуществления проверки ожидаемых результатов тестовых сценариев использовался класс Assert и дополнительная библиотека FluentAssertions. Для взаимодействия с файловой системой используются встроенные в .NET библиотеки пространства имен System.

Дополнительно был реализован функционал взаимодействия фреймворка с API. Для отправки HTTP запросов и получения ответов использовалась библиотека RestSharp. Для поддержки составления BDD сценариев подключено расширение SpecFlow. Для осуществления логирования действий, выполняемых во время работы автоматического теста использовалась библиотека Serilog. В качестве CI/CD системы для запусков тестов использовался сервер AzureDevOps. Для доступа к некоторым функциям браузера был использован язык JavaScript, который позволил создать необходимые пользователю скрипты взаимодействия с элементами, недоступным при использовании Selenium. Для реализации дополнительной логики работы с клавиатурой и мышью, такой как одновременное нажатие нескольких клавиш, нажатие правой кнопки мыши, взаимодействия с колесиком мыши, была использована библиотека Actions.

Разработка фреймворка была выполнена с учетом требований к качеству кода по соответствию принципам чистого кода и ООП, соблюдением принципов SOLID, а также с поддержкой многопоточности и параллельного запуска тестов.

Фреймворк создан для обеспечения удобного и эффективного способа специалистам по тестированию ПО автоматизировать свои тестовые сценарии, ускорить процесс обеспечения качества ПО, без участия человека запускать проект с тестами и получать результаты с историей прошлых запусков.

4 ТЕСТИРОВАНИЕ ФРЕЙМВОРКА ДЛЯ АВТОМАТИЗАЦИИ ТЕСТИРОВАНИЯ ВЕБ-ИНТЕРФЕЙСОВ

Согласно разделу 1.3 «Выводы и постановка задач на дипломное проектирование» информационная система – фреймворк для автоматизации тестирования веб-интерфейсов на платформе .NET 8 была протестирована и проверена на наличие дефектов.

В качестве наиболее подходящего метода оценки качества было выбрано комбинированное тестирование, которое включает в себя функциональное тестирование и юзабилити тестирование.

Функциональное тестирование включает в себя чек-листы и тест-кейсы.

Юзабилити-тестирование включает в себя эвристики Нильсона.

Чек-лист был создан для систематизации функций информационной системы и представляет собой список проверок определенных функций, содержащий ряд необходимых проверок во время тестирования. При взаимодействии с данным списком, разработчик или тестировщик могут получить представление о текущем состоянии выполненной работы и о качестве продукта в целом[20].

Чек-лист разрабатываемого фреймворка представлен в таблице 4.1.

Таблица 4.1 – Чек-лист разрабатываемой программы

|  |  |
| --- | --- |
| Проверка | Результат |
| Поиск по локатору с аттрибутом id | Найден элемент, соответствующий заданному локатору. |
| Поиск по локатору с аттрибутом class | Найден элемент, соответствующий заданному локатору. |
| Поиск по локатору с аттрибутом name | Найден элемент, соответствующий заданному локатору. |
| Поиск по локатору с аттрибутом value | Найден элемент, соответствующий заданному локатору. |
| Поиск по локатору с аттрибутом href | Найден элемент, соответствующий заданному локатору. |
| Поиск по локатору с пользовательским аттрибутом | Найден элемент, соответствующий заданному локатору. |
| Поиск элементов по значению текста | Найден элемент, соответствующий заданному локатору. |
| Поиск элемента с использованием ключевого слова contains | Найден элемент, соответствующий заданному локатору. |
| Поиск элемента с использованием ключевых слов ancestor, sibling, child | Найден элемент, соответствующий заданному локатору. |

Продолжение Таблицы 4.1 – Чек-лист разрабатываемой программы

|  |  |
| --- | --- |
| Проверка | Результат |
| Выбор браузера для тестирования | Инициализирован драйвер с выбранными настройками |
| Выбор версии браузера | Инициализирован драйвер с выбранными настройками |
| Заполнение дополнительных настроек | Инициализирован драйвер с выбранными настройками |
| Настройка размера окна | Инициализирован драйвер с выбранными настройками |
| Выбор языка браузера | Инициализирован драйвер с выбранными настройками |
| Настройка пути папки для загрузки файлов | Инициализирован драйвер с выбранными настройками |
| Запуск автоматизированного теста | Запущен браузер в соответствии с инициализированным драйвером |
| Наследование от базового класса страниц | Отображается родительский конструктор класса страниц |
| Добавление уникального элемента страницы | Уникальный элемент страницы закреплен в конструкторе класса страницы |
| Запуск автоматизированного теста | Запущен браузер в соответствии с инициализированным драйвером |
| Наследование от базового класса страниц | Отображается родительский конструктор класса страниц |
| Добавление уникального элемента страницы | Уникальный элемент страницы закреплен в конструкторе класса страницы |
| Запросить состояние Displayed элемента | Состояние элемента доступно пользователю в виде переменной со значениями true или false, в зависимости от текущего состояния. |
| Запросить состояние Exist элемента | Состояние элемента доступно пользователю в виде переменной со значениями true или false, в зависимости от текущего состояния. |
| Запросить состояние Enabled элемента | Состояние элемента доступно пользователю в виде переменной со значениями true или false, в зависимости от текущего состояния. |

Продолжение Таблицы 4.1 – Чек-лист разрабатываемой программы

|  |  |
| --- | --- |
| Запросить состояние NotDisplayed элемента | Состояние элемента доступно пользователю в виде переменной со значениями true или false, в зависимости от текущего состояния. |
| Симулировать клик по элементу | Произведен клик на выбранный элемент |
| Подсветить текущий элемент | Активный элемент выделен красной рамкой в браузере |
| Получить текст элемента | Получен текст выбранного элемента в виде строки |
| Симулировать прокрутку страницы | Браузер прокручивается до выбранного элемента |
| Ожидать состояние элемента Displayed | Браузер ожидает видимость элемента заданное время |
| Ожидать состояние элемента Exist | Браузер ожидает существование элемента заданное время |
| Ожидать состояние элемента Enabled | Браузер ожидает активность элемента заданное время |
| Ожидать состояние элемента NotDisplayed | Браузер ожидает исчезновение элемента заданное время |
| Навести мышь на элемент | Курсор мыши наведен на выбранный элемент |
| Удалить элемент | Выбранный элемент не отображается в DOM браузера |
| Добавить элемент типа Label | В классе страницы отображается элемент заданного типа |
| Добавить элемент типа Button | В классе страницы отображается элемент заданного типа |
| Добавить элемент типа RadioButton | В классе страницы отображается элемент заданного типа |
| Добавить элемент типа Dropdown  Добавить элемент типа ImageLabel | В классе страницы отображается элемент заданного типа |
| Добавить элемент типа Container | В классе страницы отображается элемент заданного типа |
| Добавить элемент типа TextBox | В классе страницы отображается элемент заданного типа |
| Добавить элемент типа CheckBox | В классе страницы отображается элемент заданного типа |

Чтобы протестировать основные функции фреймворка, были проведены тесты вида Smoke, которые представляют собой поверхностный тест для определения пригодности сборки для дальнейшего тестирования и охватывают основные функции программного обеспечения[21].

Для выполнения тестов на уровне Smoke-тестирования составляются низкоуровневые тестовые примеры, которые представляют собой набор входных данных, условий производительности и ожидаемых результатов, предназначенных для тестирования конкретного свойства.

Для низкоуровневого теста составляются тест-кейсы, содержащие конкретные входные данные и ожидаемые результаты.

Тест-кейсы представлены в таблице 4.2

Таблица 4.2 – Тест-кейсы для основных функциональных проверок

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | Приоритет | Модуль | Название | Описание теста | Ожидаемый результат | Фактический результат |
| TC1 | High | Класс страницы. Поле элемента | Поиск элемента по аттрибуту XPath | Перечень аттрибутов (attribute):  id, name, class, name, value, href, пользовательский.  1.Открыть класс страницы  2. Выбрать поиск элемента по XPath  3. Добавить в XPath значение аттрибута {attribute} | 1. Открыт класс выбранной страницы  2. Доступен ввод значения XPath  3. Найден элемент на странице с соответствующим аттрибутом | Не соответствует ожидаемому. Дефект D2 |

Продолжение таблицы 4.2 – Тест-кейсы для основных функциональных проверок

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | Приоритет | Модуль | Название | Описание теста | Ожидаемый результат | Фактический результат |
| TC3 | Medium | Класс страницы. Поле элемента | Поиск элемента по значению текста | 1.Открыть класс страницы  2. Выбрать поиск элемента по XPath  3. Добавить в XPath полный текст элемента на странице | 1. Открыт класс выбранной страницы  2. Доступен ввод значения XPath  3. Найден элемент на странице с соответствующим текстом | Соответствует ожидаемому |
| TC4 | Medium | Класс страницы. Поле элемента | Поиск элемента c использованием ключевого слова contains | 1.Открыть класс страницы  2. Выбрать поиск элемента по XPath  3. Добавить в XPath часть текста элемента в качестве параметра contains | 1. Открыт класс выбранной страницы  2. Доступен ввод значения XPath  3. Найден элемент на странице, включающий часть введенного текста | Соответствует ожидаемому |

Продолжение таблицы 4.2 – Тест-кейсы для основных функциональных проверок

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | Приоритет | Модуль | Название | Описание теста | Ожидаемый результат | Фактический результат |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| TC5 | Low | Класс страницы. Поле элемента | Поиск элемента c использованием ключевого слова ancestor | 1.Открыть класс страницы  2. Выбрать поиск элемента по XPath  3. Добавить в XPath путь к стабильному элементу  4. Добавить ключевое слово ancestor | 1. Открыт класс выбранной страницы  2. Доступен ввод значения XPath  3. Найден элемент на странице по заданному XPath  4. Найден элемент-предок предыдущего элемента | Соответствует ожидаемому |
| TC6 | Medium | Класс страницы. Поле элемента | Поиск элемента c использованием ключевого слова sibling | 1.Открыть класс страницы  2. Выбрать поиск элемента по XPath  3. Добавить в XPath путь к стабильному элементу  4. Добавить ключевое слово sibling | 1. Открыт класс выбранной страницы  2. Доступен ввод значения XPath  3. Найден элемент на странице по заданному XPath  4. Найден последующий элемент, включенный в тот же контейнер | Соответствует ожидаемому |

Продолжение таблицы 4.2 – Тест-кейсы для основных функциональных проверок

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | Приоритет | Модуль | Название | Описание теста | Ожидаемый результат | Фактический результат |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| TC7 | Low | Класс страницы. Поле элемента | Поиск элемента c использованием ключевого слова child | 1.Открыть класс страницы  2. Выбрать поиск элемента по XPath  3. Добавить в XPath путь к стабильному элементу  4. Добавить ключевое слово child | 1. Открыт класс выбранной страницы  2. Доступен ввод значения XPath  3. Найден элемент на странице по заданному XPath  4. Найден элемент-потомок текущего элемента | Соответствует ожидаемому |
| TC8 | Critical | Базовый тестовый класс | Запуск браузера определенного типа | Варианты типов браузера: Chrome, Firefox, Edge.  1. Открыть конфигурационный файл фреймворка  2. Добавить тип браузера  3. Запустить автоматизированный тест | 1. Файл формата JSON открыт  2. Файл изменен, настройки сохранены  3. Открыт браузер заданного типа | Не соответствует ожидаемому. Дефект D3 |
| TC9 | Medium | Базовый тестовый класс | Запуск браузера определенной версии | 1. Открыть конфигурационный файл фреймворка  2. Добавить версию браузера  3. Запустить автоматизированный тест | 1. Файл формата JSON открыт  2. Файл изменен, настройки сохранены  3. Открыт браузер выбранной версии | Соответствует ожидаемому |

Продолжение таблицы 4.2 – Тест-кейсы для основных функциональных проверок

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | Приоритет | Модуль | Название | Описание теста | Ожидаемый результат | Фактический результат |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| TC10 | Medium | Базовый тестовый класс | Запуск браузера с дополнительными настройками | Варианты аргументов: incognito, headless, no-pop-up, disable-extensions.  1. Открыть конфигурационный файл фреймворка  2. Добавить один или несколько вариантов аргументов  3. Запустить автоматизированный тест | 1. Файл формата JSON открыт  2. Файл изменен, настройки сохранены  3. Открыт браузер с применением настроек выбранных аргументов | Соответствует ожидаемому |
| TC11 | Medium | Базовый тестовый класс | Запуск браузера с пользовательским размером окна | 1. Открыть конфигурационный файл фреймворка  2. Изменить параметр размера окна браузера  3. Запустить автоматизированный тест | 1. Файл формата JSON открыт  2. Файл изменен, настройки сохранены  3. Открыт браузер с заданным в пикселях размером окна | Соответствует ожидаемому |

Продолжение таблицы 4.2 – Тест-кейсы для основных функциональных проверок

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | Приоритет | Модуль | Название | Описание теста | Ожидаемый результат | Фактический результат |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| TC12 | Low | Базовый тестовый класс | Запуск браузера с измененным языком | 1. Открыть конфигурационный файл фреймворка  2. Изменить параметр языка браузера  3. Запустить автоматизированный тест | 1. Файл формата JSON открыт  2. Файл изменен, настройки сохранены  3. Открыт браузер с интерфейсом на выбранном языке | Соответствует ожидаемому |
| TC13 | Medium | Базовый тестовый класс | Запуск браузера с измененной папки загрузки | 1. Открыть конфигурационный файл фреймворка  2. Изменить путь папки загрузки файлов  3. Запустить автоматизированный тест  4. В тесте скачать любой файл | 1. Файл формата JSON открыт  2. Файл изменен, настройки сохранены  3. Открыт браузер  4. Файл скачан в заданную папку | Соответствует ожидаемому |

Продолжение таблицы 4.2 – Тест-кейсы для основных функциональных проверок

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | Приоритет | Модуль | Название | Описание теста | Ожидаемый результат | Фактический результат |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| TC14 | High | Класс тестируемой страницы | Выбор тестируемой страницы | 1. Создать класс с названием страницы для тестирования  2. Наследовать класс страницы от базового класса всех страниц  3. Добавить уникальный элемент страницы | 1. Пустой класс создан в проекте по автоматизации  2. Класс унаследовал функционал базового класса, отображается базовый конструктор  3. Уникальный элемент страницы закреплен в конструкторе класса  4. Класс отображается в списке классов страниц | Соответствует ожидаемому |
| TC15 | High | Класс страницы. Метод работы с элементом | Получение состояние элемента | Варианты состояний: Displayed, Exist, Enabled, NotDisplayed.  1.Открыть класс страницы  2. В методе обратиться к необходимому элементу  3. Вызывать метод получения одного из состояний | 1. Открыт класс выбранной страницы  2. Предложены варианты методов для выбранного элемента  3. Значение состояния сохранено в виде true или false | Соответствует ожидаемому |

Продолжение таблицы 4.2 – Тест-кейсы для основных функциональных проверок

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | Приоритет | Модуль | Название | Описание теста | Ожидаемый результат | Фактический результат |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| TC16 | High | Класс страницы. Область полей класса | Добавление элемента определенного типа | Варианты типов элемента: Label, Button, RadioButton, Dropdown, ImageLabel, Container, TextBox, CheckBox.  1.Открыть класс страницы  2. Добавить приватное поле одного из типа элементов в класс  3. Добавить локатор XPath к элементу  4. Обратиться к созданному элементу в публичном методе | 1. Открыт класс выбранной страницы  2. Добавлено поле выбранного типа, выводится предупреждение о необходимости добавления XPath  3. Локатор закреплен в параметрах элемента  4.Предложены варианты методов для выбранного элемента | Соответствует ожидаемому |
| TC17 | High | Класс страницы. Метод работы с элементом | Метод симуляции клика | 1.Открыть класс страницы  2. Добавить публичный метод для  3. Внутри созданного метода обратиться к элементу для симуляции  4. Вызвать для выбранного элемента метод Click | 1. Открыт класс выбранной страницы  2. Пустой публичный метод отображается в классе  3.Предложены варианты методов для выбранного элемента  4.Элемент кликнут в симулируемой среде | Соответствует ожидаемому |

Продолжение таблицы 4.2 – Тест-кейсы для основных функциональных проверок

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | Приоритет | Модуль | Название | Описание теста | Ожидаемый результат | Фактический результат |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| TC18 | Low | Класс страницы. Метод работы с элементом | Метод подсветки текущего элемента | 1.Открыть класс страницы  2. Добавить публичный метод для  3. Внутри созданного метода обратиться к элементу для симуляции  4. Вызвать для выбранного элемента метод Highlight | 1. Открыт класс выбранной страницы  2. Пустой публичный метод отображается в классе  3.Предложены варианты методов для выбранного элемента  4.Активный элемент подсвечен красной рамкой в симулируемой среде | Не соответствует ожидаемому. Дефект D1. |
| TC19 | High | Класс страницы. Метод работы с элементом | Метод получения текста элемента | 1.Открыть класс страницы  2. Добавить публичный метод для  3. Внутри созданного метода обратиться к элементу для симуляции  4. Вызвать для выбранного элемента метод GetText | 1. Открыт класс выбранной страницы  2. Пустой публичный метод отображается в классе  3.Предложены варианты методов для выбранного элемента  4.Текст выбранного элемента сохранен в виде строки | Соответствует ожидаемому |

Продолжение таблицы 4.2 – Тест-кейсы для основных функциональных проверок

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | Приоритет | Модуль | Название | Описание теста | Ожидаемый результат | Фактический результат |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| TC20 | Medium | Класс страницы. Метод работы с элементом | Метод симуляции прокрутки страницы | 1.Открыть класс страницы  2. Добавить публичный метод для  3. Внутри созданного метода обратиться к элементу для симуляции  4. Вызвать для выбранного элемента метод ScrollIntoView | 1. Открыт класс выбранной страницы  2. Пустой публичный метод отображается в классе  3.Предложены варианты методов для выбранного элемента  4.Окно браузера в симулируемой среде прокручено до выбранного элемента | Соответствует ожидаемому |
| TC21 | Medium | Класс страницы. Метод работы с элементом | Метод ожидания состояния | Варианты состояний для ожидания: Displayed, Exist, Enabled.  1.Открыть класс страницы  2. Добавить публичный метод для  3. Внутри созданного метода обратиться к элементу для симуляции  4. Вызвать для элемента метод WaitFor с одним из вариантов ожидания | 1. Открыт класс выбранной страницы  2. Пустой публичный метод отображается в классе  3.Предложены варианты методов для выбранного элемента  4.Симулируемая среда находится в ожидании выбранного состояния элемента | Не соответствует ожидаемому. Дефект D4 |

Продолжение таблицы 4.2 – Тест-кейсы для основных функциональных проверок

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | Приоритет | Модуль | Название | Описание теста | Ожидаемый результат | Фактический результат |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| TC22 | Low | Класс страницы. Метод работы с элементом | Метод удаления элемента | 1.Открыть класс страницы  2. Добавить публичный метод для  3. Внутри созданного метода обратиться к элементу для симуляции  4. Вызвать для выбранного элемента метод DeleteElement | 1. Открыт класс выбранной страницы  2. Пустой публичный метод отображается в классе  3.Предложены варианты методов для выбранного элемента  4.Элемент исчез из DOM симулируемой среды | Соответствует ожидаемому |

В ходе прохождения чек-листа и тест-кейсов были выявлены некоторые дефекты информационной системы, более подробно описанные в таблице 4.3

Таблица 4.3 – Описание найденных дефектов

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Название дефекта | Важность | Алгоритм воспроизведения | Ожидаемый результат | Фактический результат |
| D1 | Подсветка элемента работает для предыдущего элемента в симулируемой среде, при вызове метода Highlight. | Minor | 1.Открыть класс страницы  2. Добавить публичный метод для  3. Внутри созданного метода обратиться к элементу для симуляции  4. Вызвать для выбранного элемента метод Highlight | 1. Открыт класс выбранной страницы  2. Пустой публичный метод отображается в классе  3.Предложены варианты методов для выбранного элемента  4.Активный элемент подсвечен красной рамкой в симулируемой среде | Не соответствует ожидаемому -  1. Открыт класс выбранной страницы  2. Пустой публичный метод отображается в классе  3.Предложены варианты методов для выбранного элемента  4.Предыдущий элемент подсвечен красной рамкой в симулируемой среде |
| D2 | XPath с использованием пользовательского аттрибута выводится ошибка-исключение в методе нахождения элемента при добавлении пути | Major | 1.Открыть класс страницы  2. Выбрать поиск элемента по XPath  3. Добавить в XPath значение пользовательского аттрибута | 1. Открыт класс выбранной страницы  2. Доступен ввод значения XPath  3. Найден элемент на странице с соответствующим пользовательским аттрибутом | 1. Открыт класс выбранной страницы  2. Доступен ввод значения XPath  3. Автоматизированный тест прекратил выполнение и пробросил исключение: CustomAttrubutesAreNotSupported |
| D3 | При попытке запуска Edge открывается Chrome. В конфигурационном файле. При попытке смены типа браузера | Critical | 1. Открыть конфигурационный файл фреймворка  2. Добавить тип браузера Edge  3. Запустить автоматизированный тест | 1. Файл формата JSON открыт  2. Файл изменен, настройки сохранены  3. Открыт браузер типа Edge | 1. Файл формата JSON открыт  2. Файл изменен, настройки сохранены  3. Открыт браузер другого типа (Chrome) |

Продолжение таблицы 4.3 – Описание найденных дефектов

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Название дефекта | Важность | Алгоритм воспроизведения | Ожидаемый результат | Фактический результат |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| D4 | Вместо Enabled работает ожидание NotDisplayed. В методе работы с элементом. При попытке использовать ожидание. | Average | .1.Открыть класс страницы  2. Добавить публичный метод для  3. Внутри созданного метода обратиться к элементу для симуляции  4. Вызвать для выбранного элемента метод WaitForEnabled  5. Удалить выбранный элемент | 1. Открыт класс выбранной страницы  2. Пустой публичный метод отображается в классе  3.Предложены варианты методов для выбранного элемента  4.Симулируемая среда находится в ожидании состояния Enabled  5. Ожидание остановило выполнение автоматизированного теста | 1. Открыт класс выбранной страницы  2. Пустой публичный метод отображается в классе  3.Предложены варианты методов для выбранного элемента  4.Симулируемая среда находится в ожидании неверного состояния (NotDisplayed)  5. Ожидание завершено успешно |

Итоговый отчёт с описанием дефектов представлен в таблице 4.4.

Таблица 4.4 – Итоговый отчёт качества фреймворка

|  |  |  |  |
| --- | --- | --- | --- |
| Дата тестирования | | | |
| 29.04.2024 | | | |
| Выставленная отметка качества – High | | | **Smoke test** Acceptable |
| **Acceptance test** High |
| **New defects** 4 |
| **Test documentation** Test Cases |
| Детальный анализ качества | | | |
|  | | | |
| Выводы по модулям | | | |
| Модуль | Качество | Комментарий | |
| Методы работы с элементом | High | Обратите внимание на типы состояний и специфику исключений | |
| Интерфейс симулируемого браузера | Medium | Обратите внимание на выделение именно активного элемента | |
| Конфигурационный файл | Low | Обратите внимание на поддержку браузера Edge | |
| **Рекомендации:**  Обратите внимание на то, что при работе с ожиданием состояния активируется режим ожидания, но не для выбранного состояния | | | |

Для проведения тестирования использовалась операционная система Windows версии 22H2 и среда разработки Visual Stuido. Наиболее важными являются дефекты с ID D3 и D2 со степенью критичности Critical и Major. По модулям 2 дефекта были обнаружены в методе работы с элементом, 1 в интерфейсе симулируемого браузера и 1 в конфигурационном файле.

Для проведения юзабилити-тестирования была выбрана методика на основе экспертного подхода, которая руководствуется эвристиками Якоба Нильсена. Для сравнения наличия эвристик Нильсена в разработанном фреймворке и наличия эвристик в аналогах были использованы аналоги, рассмотренные в разделе 1.2, а именно веб-сервисы: Cypress и PlayWright. Сравнительная таблица с эвристиками представлена в таблице 4.5.

Таблица 4.5 – Сравнение наличия эвристик Нильсена в разработанном веб-сайте и аналогах

|  |  |  |  |
| --- | --- | --- | --- |
| Эвристики Нильсона | Описание | Наличие | |
| Фреймворк для автоматизации тестирования на платформе .NET 8 | Аналоги |
| 1. Явное состояние системы | Фреймворк должен четко отображать текущий статус автоматизированного теста и предоставлять пользователю информацию о его действиях | Реализовано в данной информационной системе. В виде прогресс-бара, который меняет цвет на зеленый, желтый или красный в зависимости от статуса | Реализовано только у PlayWright. В виде прогресс-бара, который меняет цвет на зеленый, желтый или красный в зависимости от статуса |
| 2. Соответствие реальному миру | Использование терминологии и понятий, которые ясно и понятно связаны с тестированием и соответствуют реальным ожиданиям специалистов по тестированию ПО. | Реализовано в данной информационной системе. В соответствии с наименованием классов и переменных по подходу чистого кода. | Не реализовано в аналогах |

Продолжение таблицы 4.5

|  |  |  |  |
| --- | --- | --- | --- |
| Эвристики Нильсона | Описание | Наличие | |
| Фреймворк для автоматизации тестирования на платформе .NET 8 | Аналоги |
| 3. Свобода и контроль | Пользователи должны иметь возможность свободно автоматизировать свои тестовые сценарии и изменять конфигурацию браузера | Реализовано в данной информационной системе. В виде изменяемого конфигурационного файла и блоков для составления сценария автоматического теста | Реализовано только у PlayWright. В виде изменяемого конфигурационного файла и блоков для составления сценария автоматического теста |
| 4. Согласованность и стандарты | Фреймворк должен представлять информацию о результатах автоматизированных тестов в стандартизированном формате, чтобы пользователи могли легко сравнивать и принимать решения | Не реализовано в данной информационной системе | Реализовано только у Cypress. В виде стандартизированного html отчета с результатами автоматизированного теста. |
| 5. Предотвращение ошибок | Фреймворк должен предоставлять четкие инструкции и предупреждения для предотвращения ошибок при работе с симуляцией взаимодействия с элементами. | Реализовано в данной информационной системе. В виде описания каждого используемого пользователем метода и сообщения с предупреждением об ошибке | Реализовано у Cypress и PlayWright. В виде окна, отображающего предупреждения по улучшению кода и ошибками, которые блокируют выполнение автоматизированного теста. |
| 6. Гибкость и эффективность | Фреймворк должен предоставлять гибкие варианты фильтрации и сортировки результатов автоматизированных тестов, а также эффективные средства навигации по проекту | Реализовано в данной информационной системе. В виде меню для фильтрации и сортировки результатов тестов и обозревателя решения для навигации. | Не реализовано в аналогах |

Продолжение таблицы 4.5

|  |  |  |  |
| --- | --- | --- | --- |
| Эвристики Нильсона | Описание | Наличие | |
| Фреймворк для автоматизации тестирования на платформе .NET 8 | Аналоги |
| 7. Удовлетворение пользовательских потребностей | Фреймворк должен предлагать разнообразные услуги и функции, чтобы удовлетворить различные потребности специалистов по тестированию ПО при автоматизации своих тестовых сценариев | Реализовано в данной информационной системе. В виде классов и методов для симуляции различных действий в симулируемой среде. | Реализовано только у Cypress. В виде классов и методов для симуляции различных действий в симулируемой среде. |
| 8. Понятность и интуитивность | Фреймворк должен быть интуитивно понятным и легким в использовании, чтобы пользователи могли быстро разобраться в его функциях и возможностях. | Реализовано в данной информационной системе. В виде инструкции к фреймворку и примеров кода с его использованием | Реализовано только у PlayWright. В виде инструкции к фреймворку и примеров кода с его использованием |
| 9. Оперативность | Фреймворк должен обеспечивать быстрые загрузки симулируемых страниц, оперативно симулировать заданные действия. | Не реализовано в данной информационной системе | Реализовано только у Cypress. В виде ускоренной загрузки страницы с использованием кэширования |
| 10. Предоставление обратной связи | Фреймворк должен предоставлять возможность для оставления отзывов о найденных дефектах и качестве его работы, а также обеспечивать коммуникацию между пользователями и контрибьюторами | Не реализовано в данной информационной системе | Не реализованно в упомянутых аналогах |

В представленной таблице были сравнены эвристики Нильсона для фреймворка для автоматизации тестирования веб-интерфейсов на платформе .NET 8 и их использование, а также описание аналогов фреймворков и реализация этих эвристик. Анализ показал, что юзабилити системы хорошее, в разработанном фреймворке 7 из 10 эвристик соблюдены. Самый сильный конкурент из аналогов – Cypress, в этом фреймворке соблюдено 4 эвристик из 10.

В ходе тестирования фреймворка для автоматизации тестирования веб-интерфейсов на платформе .NET 8 было проведено комбинированное тестирование, которое включало в себя функциональное и юзабилити тестирование. Были составлены тест-кейсы и чек-листы, разработан визуальный отчёт качества фреймворка. На различных этапах разработки были выявлены функциональные дефекты и дефекты пользовательского интерфейса. Дефекты были оформлены в виде детального баг-репорта с шагами воспроизведения, ожидаемым и актуальным результатом. Данные дефекты были воспроизведены в работе системы и исправлены. А также по результатам тестирования был разработан итоговый отчет качества проекта с информацией о распределении дефектов по типу, степени критичности и модулям. При проведении итогового тестирования фреймворк работает корректно и в соответствии с целями и заданиями, указанными в разделе 1.3. Фреймворк готов к подключению в проект с автоматизированными тестами, готов к симуляции симулируемой среды в виде различных типов браузеров и к симуляции работы с элементами.

5 ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ЭФФЕКТИВНОСТИ РАЗРАБОТКИ И РЕАЛИЗАЦИИ ФРЕЙМВОРКА ДЛЯ АВТОМАТИЗАЦИИ ТЕСТИРОВАНИЯ

5.1 Характеристика информационной системы

Информационная система, разработанная в дипломном проекте, представляет собой фреймворк для автоматизации тестирования веб-интерфейсов на платформе .NET 8. Помимо основной функции симуляции действий над элементами веб-страницы, фреймворк также позволяет внедрять в сценарии тестов взаимодействие с API ресурсом тестируемой системы и предоставляет отчёты о пройденных тестах и динамике положительных или отрицательных результатов в показателях качества ПО.

Информационная система – фреймворк для автоматизации тестирования разработана для обеспечения удобного и эффективного способа сократить затраты человеческих сил специалистов по тестированию ПО и проектных ресурсов на осуществление ручного тестирования при помощи автоматизации процессов.

Для создания информационной системы использовалась операционная система Windows 11, такие языки программирования, как C# и JavaScript. Разработка происходила в таких средах, как VisualStudio и GitHub, для хранения данных используются конфигурационные JSON файлы и система управления базами данных класса NoSQL с типом Key-value.

Преимуществом фреймворка является его универсальные возможности использования, ведь он совмещает в себе функционал тестирования интерфейса веб-приложений, клиент-серверной архитектуры приложения, позволяет автоматизировать как функциональные и интеграционные тесты, так и end-to-end тестовые сценарии[22].

Его можно использовать как разработчику приложения, так и специалисту по тестированию ПО, не имеющему большого опыта в программировании.

Целевая аудитория – все люди, вовлеченные в сферу тестирования: QA инженеры, автоматизаторы тестирования, разработчики ПО, DevOps инженеры.

Экономическим эффектом для разработчика будет чистая прибыль, полученная от комиссии на лицензионное использование фреймворка в коммерческих проектах.

В данном разделе необходимо определить все затраты, произведенные на каждом из этапов создания описанной информационной системы. Также необходимо рассчитать экономию основных видов ресурсов в связи с использованием данного фреймворка.

5.2 Расчет затрат на разработку ПО

5.2.1 Затраты на основную заработную плату команды разработчиков

Для расчета затрат на разработку фреймворка были учтены система налогообложения и различных расходов на дополнительные нужды.

Для исполнителей, занятых разработкой основная заработная плата () определяется исходя из численности и состава команды проекта, а также величины месячной заработной платы у каждого из участников команды, также учитывается общая трудоемкость разработки программного обеспечения.

В результате анализа предметной области и оценки рынка труда, в команду проектировщиков были привлечены следующие специалисты:

* разработчик, трудоемкость выполнения работ 80 часов, месячный оклад 2200 рублей.

– тестировщик, трудоемкость 20 часов, месячный оклад 1900 рублей.

Значение основной заработной платы команды получим по формуле:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | | | (5.1) |

где n – количество исполнителей, занятых разработкой фреймворка;

= коэффициент премий участников (составляет 1,2);

– часовая заработная плата i-го исполнителя, руб.;

– трудоемкость работ i-го исполнителя, ч.

Трудоёмкость проекта была оценена в 2,5 рабочие недели, или 100 часов.

Часовая оплата определяется путем деления месячного оклада на количество рабочих часов в месяце (для апреля 175 ч).

Расчет затрат на основную заработную плату команды представлен   
в таблице 5.2.

Таблица 5.2 – Расчет затрат на основную заработную разработчиков фреймворка

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Разработ-чик | Вид работы | Оклад месячный, руб. | Оклад часовой,  руб. | Трудоем-  кость работ, ч. | Зарплата по тарифу, руб. |
| Разработ-чик | Разработка фреймворка, исправление дефектов | 2200 | 12,57 | 80 | 1005,7 |
| Тестиров-щик | Тестирование фреймворка | 1900 | 10,85 | 20 | 217,14 |
| Итого | | | | | 1222,84 |
| Премия и иные стимулирующие выплаты (20%) | | | | | 244,57 |
| Всего затрат на основную заработную плату разработчиков, руб. | | | | | 1467,4 |

5.2.2 Затраты на дополнительную заработную плату команды разработчиков

Дополнительная заработная плата исполнителей проекта определяется по формуле:

|  |  |
| --- | --- |
|  | (5.3) |

где, Зо – затраты на основную заработную плату, руб.;

Нд – норматив дополнительной заработной платы (примем норматив 10%).

Дополнительная заработная плата исполнителей составит:

Согласно формуле (5.3), затраты на дополнительную заработную плату команды составляют рублей.

5.2.3 Отчисления в фонд социальной защиты населения и обязательное страхование

Отчисления в фонд социальной защиты населения и на обязательное страхование определяются в соответствии с действующим с действующими законодательными актами Республики Беларусь по формуле:

|  |  |
| --- | --- |
|  | (5.4) |

где – норматив отчислений в фонд социальной защиты населения и обязательное страхования, где отчисления в фонд социальной защиты составляют 34%, а обязательное страхование – 0,6%.

Итого отчисления в ФСЗН и на обязательное страхование составят:

Согласно формуле (5.4), размер отчислений в фонд социальной защиты населения и на обязательное страхование составляет рублей.

5.2.4 Прочие затраты

Расходы по статье прочих затрат включают в себя затраты на приобретение специальной научно-технической информации, литературы, а также использование лицензионного платного программного обеспечения в ходе разработки, определяются по формуле:

|  |  |
| --- | --- |
|  | (5.5) |

где, Нp – норматив прочих затрат, (примем значение равным 30%).

Прочие затраты составят:

Общая сумма инвестиций на разработку считается по формуле:

|  |  |
| --- | --- |
|  | (5.6) |

Полная сумма затрат на разработку программного обеспечения находится путем суммирования всех рассчитанных статей затрат. Расчет общей суммы затрат представлен в таблице 5.7.

Таблица 5.7 – Затраты на разработку веб-сайта

|  |  |
| --- | --- |
| Статья затрат | Сумма, руб. |
| Основная заработная плата команды разработчиков ( |  |
| Дополнительная заработная плата команды разработчиков (д) |  |
| Отчисления на социальные нужды ( |  |
| Прочие затраты ( |  |
| Общая сумма затрат на разработку | 2612,85 |

Таким образом, полная себестоимость разрабатываемого фреймворка для автоматизации тестирования составит 2612,85 руб. Основные статьи расходов это – основная заработная плата программистов, а также отчисления на социальные нужды.

5.3 Расчет экономического эффекта от реализации фреймворка для автоматизации тестирования веб-интерфейсов

Для проведения оценки экономического эффекта от реализации информационной системы необходимо определить предполагаемый объем специалистов по тестированию ПО, которые будут использовать фреймворк на коммерческих проектах и платить лицензионную комиссию.

|  |  |
| --- | --- |
|  | (5.8) |

где – прибыль, руб.;

Ц ком – цена комиссии за использование на 1 проекте,

*N* – количество пользователей, подключивших библиотеку с фреймворком.

НДС – сумма налога на добавленную стоимость, р.;

Рпр – рентабельность продаж (40%);

Нп – ставка налога на прибыль, согласно законодательству   
(стандартная составляет 18%).

По данным официального пакетного менеджера nuget от компании Microsoft, аналогичными фреймворками для автоматизации пользуются в среднем 22000 пользователей в день на платформе .NET. Разрабатываемым фреймворком планируется охватить 0,27% всего рынка, с учётом фокусирования на крупные компании, которые используют более дорогие аналоги[23].

Исходя из аналогов на рынке и предполагаемого объема рынка, за первый год ожидается привлечь 20 компаний по разработке коммерческих проектов, у каждой из которых по 3 проекта в год, следовательно за год общее число лицензий составит 60.

Для повышения конкурентноспособности и быстрого продвижения продукта была взята цена в среднем в 2 раза ниже представленных на рынке аналогов. Примерная средняя цена аналогов составляет 400 рублей. В связи с этим, комиссия за лицензионное использование разрабатываемого фреймворка составляет 180 рублей.

Налог на добавленную стоимость при таком подходе к ценообразованию рассчитывается по формуле:

|  |  |
| --- | --- |
|  | (5.9) |

где Нд.с. – ставка налога на добавленную стоимость согласно действующему законодательству РБ..

Налог на добавленную стоимость составит:

Прирост чистой прибыли составит:

5.4 Расчет эффективности показателей информационной системы

Для расчета экономической эффективности разработки и использования фреймворка для автоматизации необходимо провести сравнение затрат на его разработку и полученный годовой прирост чистой прибыли.

Для проведения расчетов эффективности показателей информационной системы использовалась формула расчета рентабельности инвестиций (Return on Investment, *ROI*):

|  |  |
| --- | --- |
|  | (5.10) |

где – прирост чистой прибыли, полученной от реализации фреймворка на коммерческих проектах, р;

– затраты на разработку и реализацию фреймворка, р.

Рентабельность инвестиций будет равна:

На разработку фреймворка для автоматизации тестирования было задействовано 100 человеко-часов и затраты составили рублей. За год планируется привлечь 60 лицензионных пользователей, которые оплачивают комиссию 180 руб. При таких условиях прирост чистой прибыли за год составит рублей. Сумма затрат намного меньше, чем сумма годового экономического эффекта, рентабельность инвестиций составляет очень высокий показатель %, следовательно, при успехе привлечения нужного количества клиентов, инвестиции будут весьма эффективными и целесообразно разрабатывать данный фреймворк.

Актуальность разработки объясняется тем, что современные проекты по разработке ПО все больше автоматизирует процессы, которые занимают время у специалистов, что позволяет экономить ресурсы бизнеса, перенаправить сэкономленное время на более важные задачи, а также уменьшить человеческий фактор в процессе обеспечения качества ПО[24].

При создании эффективного фреймворка уникальность продукта достигнута за счет определенных факторов. В первую очередь, это возможность использовать новейшую версию языка программирования C# и современную платформу разработки .NET 8, чего не могут предложить альтернативные системы. Это позволяет использовать современные подходы программирования и улучшить скорость прохождения автоматических тестов за счёт усовершенствованной среды разработки. Во вторую очередь, фреймворк покрывает сразу несколько областей в процессе тестирования. В-третьих, это низкая стоимость лицензии по сравнению с конкурентами, что является определяющим фактором при выбора инструмента для бизнеса по разработке ПО.

Таким образом, фреймворк для автоматизации тестирования на платформе .NET 8 среди специалистов по тестированию ПО является актуальным продуктом, который может быть внедрен в проект, благодаря совершенствованию технических возможностей для повышения удобства, простоты и скорости выполнения тестовых сценариев и анализа соответствия требованиям разрабатываемого программного средства.

6 ОХРАНА ТРУДА. ОБОСНОВАНИЕ ВЫБОРА СИСТЕМЫ ИСКУССТВЕННОГО ОСВЕЩЕНИЯ В ПОМЕЩЕНИИ ПРИ РАЗРАБОТКЕ ФРЕЙМВОРКА ДЛЯ АВТОМАТИЗАЦИИ ТЕСТИРОВАНИЯ ВЕБ-ИНТЕРФЕЙСОВ

Целью дипломного проекта является разработка фреймворка для автоматизации тестирования веб-интерфейсов на платформе .NET 8. Данная система представляет из себя удобный фреймворк в виде библиотеки-зависимости для проекта по автоматизации тестирования, позволяющий пользователю симулировать на компьютере тестовые сценарии, которые до внедрения фреймворка воспроизводятся вручную. Информационная система позволяет автоматически запускать тесты, сохранять и предоставлять визуально их результаты, анализировать дефекты в программном обеспечении, а также интегрироваться с пайплайн-инструментами жизненного цикла разработки ПО[24]. Фреймворк предназначен для опытных и начинающих специалистов по тестированию ПО, разработчиков, проектных менеджеров и бизнес-аналитиков.

Во время разработки фреймворка на разработчика могут оказывать неблагоприятное воздействие опасные и вредные производственные факторы, связанные с выбором системы искусственного освещения в помещении. Факторы неправильного выбора системы искусственного освещения при работе с ЭВМ могут привести к ухудшению зрения, повышенной усталости и ошибкам в работе, вызвать дискомфорт и затруднить выполнение задач и сложных операций, а также иногда привести к ухудшению здоровья глаз и кожи, в крайнем случае ослеплению. В настоящем разделе рассмотрим вопросы, связанные с обоснованием выбора системы искусственного освещения в помещении при разработке фреймворка для автоматизации тестирования веб-интерфейсов, а также разработкой профилактических мероприятий.

Важнейшими устройствами, на которые влияют факторы выбора системы освещения, являются рабочие мониторы. Необходимо учитывать такие характеристики как размер дисплея, мерцание экрана, разрешение экрана, фильтр синего света, расстояние до монитора, регулировка высоты и наклона, тип матрицы, плотность пикселей, частота обновления экрана, тип поверхности экрана, яркость и время отклика пикселей.

При разработке информационной системы использовался стационарный компьютер c двумя подключенными мониторами модели ASUS TUF Gaming VG259QM и BenQ GW2470H. Рабочим местом разработчика является жилое помещение, где установлен стол с мониторами на ножках, на котором основной монитор находится прямо напротив разработчика, а дополнительный с левой стороны под небольшим наклоном, при этом позади мониторов находится лампа для обеспечения дополнительного фонового освещения.

Диагональ экрана, разрешение, плотность пикселей и тип поверхности экранов идентичны и соответственно равны 24 дюймам, 1920x1080 пикселям, 90 пикселей на дюйм и матовой поверхности. Также оба монитора обладают технологией отсутствия мерцания (Flicker-free) и фильтром синего света. Основной монитор обладает матрицей IPS, яркостью 400 кд/м2, временем отклика 1 мс, частотой обновления 280 Гц, а также оснащен ножкой с возможностью поворота экрана на 360 градусов в стороны, углом наклона на 33 градуса и поворотом в портретный режим. В то же время дополнительный монитор обладает матрицей VA, яркостью 250 кд/м2, временем отклика 4 мс, частотой обновления 60 Гц и статичной ножкой, с единственной возможностью наклона экрана на 20 градусов. Лампа представляет из себя настольный светильник, питаемый от сети, с мощностью лампочки 60 Вт и возможностью регулировки высоты и наклона плафона[25].

Для конкретного рабочего места разработчика фреймворка было выделено несколько потенциальных причин плохого освещения или освещенности, которые могут быть связаны с использованием компьютера с двумя мониторами и дополнительными приборами искусственного освещения[26]:

1. Недостаточное освещение рабочего места. Недостаточное освещение может привести к напряжению глаз, утомлению и затруднению чтения текста или работы с мелкими деталями. Это может вызвать ухудшение зрения и повысить риск возникновения глазных проблем.
2. Блики и отражения от естественных и искусственных источников света. Они могут стать причиной повышенного стресса, привести к усталости и негативно отразиться на психоэмоциональном состоянии и общей производительности разработчика.
3. Неравномерное освещение, создающее тени и пятна. Усложняет восприятие информации и может быть опасно при выполнении сложных операций.
4. Ультрафиолетовое излучение от источников освещения. Может негативно сказываться на здоровье глаз и кожи.
5. Недостаточная яркость или контрастность экрана монитора. Приводит к утомлению глаз и затруднияет восприятие текста или изображений.
6. Слишком большое расстояние или неправильный угол наклона монитора. Может привести к напряжению шеи и спины разработчика.
7. Использование монитора с низким временем отклика и частотой обновления кадров. Может ухудшить плавность и четкость изображения в движении, затруднить работу с кодом или графикой.
8. Длительное время работы за монитором без перерывов. Что может привести к сухости глаз, усталости, снижению концентрации и в целом увеличивает риск несчастных случаев.
9. Нарушение эргономичности рабочего места. Плохое освещение может привести к неправильному положению тела разработчика, чтобы лучше видеть на экране или на рабочем столе. Это может вызвать напряжение в шее, спине или руках, что в свою очередь может привести к различным мускульно-скелетным заболеваниям.

Нормы и требования к искусственному освещению в помещениях разработчиков в Беларуси определяются в соответствии с ГОСТ 12.2.003-91 "Системы охраны труда. Освещение на производстве". В этом стандарте устанавливаются общие требования к искусственному освещению в рабочих помещениях, включая помещения разработчиков.

Согласно ГОСТ 12.2.003-91, нормы и требования к искусственному освещению включают в себя следующие параметры[27]:

* Уровень освещенности: Уровень освещенности помещения для разработчиков должен обеспечивать комфортные условия работы, предотвращать утомление глаз и обеспечивать безопасность труда. Нормы уровня освещенности могут зависеть от типа работ, выполняемых в помещении, но обычно рекомендуется уровень освещенности от 300 до 500 люкс.
* Равномерность освещения: Освещение должно быть равномерным по всей рабочей поверхности, чтобы избежать ярких пятен или теней, что может вызвать напряжение глаз и утомление.
* Наличие бликов и отблесков: Необходимо предотвращать блики и отблески на экранах мониторов и других поверхностях, чтобы обеспечить комфортные условия работы и предотвратить возможные проблемы со зрением.
* Цветовая температура: Также важно учитывать цветовую температуру искусственного света, чтобы создать приятную атмосферу в помещении и обеспечить хорошую цветопередачу.

Еще одним важным аспектом при обеспечении искусственного освещения в помещениях разработчиков является выбор типа источников света. В соответствии с ГОСТ 12.2.003-91, для работы в помещениях разработчиков рекомендуется использовать источники света, обеспечивающие хорошую цветопередачу и минимальное мерцание.

Наиболее распространенными типами источников света, которые часто применяются в офисных помещениях и для работы за компьютером, являются[28]:

1. Лампы люминесцентные. Энергоэффективные источники света, обеспечивающие равномерное освещение и хорошую цветопередачу. Они могут быть использованы как основной источник освещения.
2. Светодиодные лампы. Имеют высокую энергоэффективность, долгий срок службы и хорошую цветопередачу. Светодиодные лампы также могут быть использованы для создания уютной атмосферы в помещении.
3. Галогенные лампы: Обладают высокой яркостью и хорошей цветопередачей, но менее энергоэффективны по сравнению с другими типами ламп.

Для разработки данного фреймворка использовались исключительно светодиодные лампы. При выборе источников света для помещения разработчика фреймворка были учтены требования к уровню освещенности, равномерности освещения, цветовой температуре и минимизации бликов на экранах мониторов.

Выбор искусственного освещения в помещении разработчика решает несколько важных задач, включая[29]:

1. Обеспечение комфортных условий работы. Правильно подобранное искусственное освещение помогает создать комфортную рабочую атмосферу, улучшает концентрацию и повышает производительность разработчика.
2. Повышение эффективности работы. Освещение влияет на зрительное восприятие, что в свою очередь может повлиять на качество работы. Хорошее освещение помогает избежать усталости глаз, снижает риск возникновения ошибок и повышает точность выполнения задач.
3. Создание безопасной среды. Правильно спроектированное искусственное освещение помогает предотвратить травмы зрения и тела человека, связанные с неправильным положением разработчика относительно мониторов.
4. Улучшение эргономики помещения: Выбор подходящих типов источников света, цветовой температуры и яркости позволяет создать эргономичную среду в помещении для разработки.

5. Энергосбережение. Использование энергоэффективных источников света и оптимального распределения света помогает снизить энергопотребление и экономить ресурсы.

Профилактические меры по обеспечению системы искусственного освещения в помещении разработчика включают в себя технические, организационные и режимные мероприятия. Они направлены на устранение или минимизацию факторов, которые могут вызвать ухудшение работоспособности разработчика фреймворка и вызвать нарушения здоровья. Некоторые из таких мероприятий могут включать:

Технические мероприятия:

1. Установка дополнительных источников освещения (настенные, потолочные, настольные, мониторные лампы и торшеры);
2. Приобретение мониторов с улучшенными характеристиками яркости и контрастности изображения;
3. Использование подставок или кронштейнов для мониторов, обеспечивающих возможность изменения высоты, наклона и расстояния до дисплея;
4. Переход на энергоэффективные решения для экономии энергии и снижения нагрузки на систему электроснабжения;
5. Использование эргономичного рабочего стола и кресла для правильного расстояния глаз до монитора и источников света.

Организационные мероприятия:

1. Проведение инструктажа сотрудников по использованию мониторов и осветительных электроприборов, описание оптимальных условий освещения;
2. Установление требований к настройке мониторов для обеспечения необходимой яркости, контрастности и четкости изображения;
3. Установление правил по использованию светового электрооборудования и устройств для его защиты.

Режимные мероприятия:

1. Регулярное обслуживание и чистка источников света;
2. Калибровка цветопередачи и других настроек дисплеев;
3. Контроль освещенности рабочего помещения разработчика;
4. Обеспечение перерывов при долгой работе с монитором, а также уменьшение времени работы при недостаточном освещении.

Обоснование выбора таких мероприятий основывается на их эффективности и реалистичности для данного рабочего места. Они помогут минимизировать риски возникновения проблем со зрением и общим самочувствием разработчика фреймворка и обеспечить его эффективность.

Таким образом, изложенные выше мероприятия обеспечат безопасные условия труда для разработчиков фреймворка и предоставят эргономичные условия для работы, которые не приводят к переутомлению и снижению концентрации работников, минимизируют риски возникновения осложнений, связанных с ухудшением зрения. Регулярное проведение проверок дисплеев и световых приборов, обучение сотрудников оптимальным условиям освещения и использование подходящего оборудования на рабочем месте помогут предотвратить возможные опасные и вредные производственные факторы.

ЗАКЛЮЧЕНИЕ

В рамках дипломного проектирования был разработан фреймворк для автоматизации тестирования веб-интерфейсов на платформе .NET 8, предназначенный для специалистов по тестированию ПО, желающих сократить затраты времени и сил на ручное тестирование и доверить процесс оценки качества автоматизации. Информационная система создана в формате фреймворка, тем самым предоставляя доступ к необходимому функционалу в любое время при наличии интернета и установленного пакета с фреймворком.

Фреймворк реализован в полном объеме и соответствует техническому заданию, полностью отлажен и протестирован. Поставленные задачи на разработку выполнены.

В ходе разработки были достигнуты следующие результаты:

– исследована предметная область автоматизации тестирования;

– изучен необходимый теоретический материал в сфере автоматизации тестирования веб-интерфейсов;

– рассмотрены доступные аналоги проектируемой информационной системы в виде фреймворков, библиотек и zero-code приложений для автоматизации тестовых сценариев.;

– спроектированы UML диаграммы деятельности, варианты использования и структурные схемы фреймворка;

– реализован и протестирован задуманный функционал в готовом программном продукте.

Во время проектирования были определены достоинства и недостатки существующих программных аналогов. Благодаря изучению и анализу аналогов были определены требования к фреймворку, которые отражают лучшие стороны разработанного фреймворка для автоматизации тестирования.

Спроектированный фреймворк обладает рядом положительных качеств, связанных, прежде всего с расширенным функционалом, позволяющим специалистам по тестированию ПО автоматизировать самые сложные и полезные тестовые сценарии, симулировать необходимые действия по взаимодействию с браузером и элементами веб-страниц, увеличить тестовое покрытие и переиспользовать разработанные шаги.

Таким образом, в результате выполнения дипломного проекта, поставленные задачи выполнены в полном объеме.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Atlassian статьи. Agile vs Devops [Электронный ресурс]. – Режим доступа: <https://www.atlassian.com/devops/what-is-devops/agile-vs-devops>

[2] Lamdatest блоги сообщества. Common Challenges In Selenium Automation & How To Fix Them? [Электронный ресурс]. – Режим доступа: <https://www.lambdatest.com/blog/common-challenges-in-selenium-automation-how-to-fix-them/>

[3] Lamdatest блоги сообщества. How Test Automation Can Accelerate Business Transformation [Электронный ресурс]. – Режим доступа: <https://www.lambdatest.com/blog/how-test-automation-can-accelerate-business-transformation/>

[4] DevQA документация для тестировщиков. How to Develop a Test Automation Framework From Scratch? [Электронный ресурс]. – Режим доступа: <https://devqa.io/develop-test-automation-framework-scratch/>

[5] BrowserStack гайдлайны по автоматизации тестирования. Best Test Automation Frameworks [Электронный ресурс]. – Режим доступа: <https://www.browserstack.com/guide/best-test-automation-frameworks>

[6] Lamdatest блоги сообщества. 35 Best Test Automation Frameworks for 2024. [Электронный ресурс]. – Режим доступа: <https://www.lambdatest.com/blog/best-test-automation-frameworks/>

[7] BrowserStack гайдлайны по автоматизации тестирования.Best Automation Mobile Testing Tools and Frameworks [Электронный ресурс]. – Режим доступа: <https://www.browserstack.com/guide/mobile-application-testing-frameworks>

[8] SoftwareTestingHelp репозиторий помощи разработчикам. 10 Best API Testing Tools In 2024 (SOAP And REST API Testing Tools) [Электронный ресурс]. – Режим доступа: https://www.softwaretestinghelp.com/api-testing-tools/

[9] GURU99 библиотека знаний. 11 BEST Performance Testing Tools (2024) [Электронный ресурс]. – Режим доступа: <https://www.guru99.com/performance-testing-tools.html>

[10] SoftwareTestingHelp репозиторий помощи разработчикам. 40+ Best Database Testing Tools – Popular Data Testing Solutions [Электронный ресурс]. – Режим доступа: <https://www.softwaretestinghelp.com/tools/40-best-database-testing-tools/>

[11] LogRocket блог разработчиков. Comparing the best unit testing frameworks [Электронный ресурс]. – Режим доступа: <https://blog.logrocket.com/comparing-best-node-js-unit-testing-frameworks/>

[12] Amplitude сборник аналитики. [Электронный ресурс]. – Режим доступа: <https://amplitude.com/guides/2022-app-vs-website-report>

[13] Lamdatest блоги сообщества. 58 Top Automation Testing Tools For 2024. [Электронный ресурс]. – Режим доступа: <https://www.lambdatest.com/blog/automation-testing-tools/>

[14] Lamdatest блоги сообщества. Integration Testing vs Functional Testing: Key Differences [Электронный ресурс]. – Режим доступа: <https://www.lambdatest.com/blog/integration-testing-vs-functional-testing/>

[15] Хранилище студенческих работ StudRef. ЧЕЛОВЕКО-МАШИННЫЙ ИНТЕРФЕЙС [Электронный ресурс]. – Режим доступа: <https://studref.com/387955/tehnika/cheloveko_mashinnyy_interfeys>

[16] Пресс-релизы Microsoft. Человеко-компьютерное взаимодействие. Часть 1 [Электронный ресурс]. – Режим доступа: https://news.microsoft.com/ru-ru/cheloveko-komp-yuternoe-vzaimodejstv-2/

[17] Справочник научных статей НаучныеСтатьи.Ру. Эргономические требования: суть, принципы и влияние на производительность и безопасность [Электронный ресурс]. – Режим доступа: <https://nauchniestati.ru/spravka/ergonomicheskie-trebovaniya>

[18] AutomationPanda блог для разработки и тестирования. BDD 101: GHERKIN BY EXAMPLE. [Электронный ресурс]. – Режим доступа: <https://automationpanda.com/2017/01/27/bdd-101-gherkin-by-example/>

[19] Словарь Agile. Agile Alliance [Электронный ресурс]. – Режим доступа: <https://www.agilealliance.org/glossary/given-when-then>

[20] GURU99 библиотека знаний. What is Usability Testing? Software UX. [Электронный ресурс]. – Режим доступа: https://www.guru99.com/usability-testing-tutorial.html

[21] GURU99 библиотека знаний. What is Smoke Testing?. [Электронный ресурс]. – Режим доступа: https://www.guru99.com/smoke-testing.html

[22] BrowserStack гайдлайны по автоматизации тестирования. What is End To End Testing? . [Электронный ресурс]. – Режим доступа: <https://www.browserstack.com/guide/end-to-end-testing>

[23] Пакетный менеджер Nuget. Статистика использования библиотек [Электронный ресурс]. – Режим доступа: <https://www.nuget.org/packages>

[24] BrowserStack гайдлайны по автоматизации тестирования. Benefits of Automation Testing. [Электронный ресурс]. – Режим доступа: <https://www.browserstack.com/guide/benefits-of-automation-testing>

[25] Открытый набор спецификаций дисплеев. [Электронный ресурс]. – Режим доступа: <https://www.displayspecifications.com/en>

[26] Организация оптимизации рабочих мест visitdesk. Optimizing Workplace Lighting for Health and Productivity – A Comprehensive Guide [Электронный ресурс]. – Режим доступа: <https://visitdesk.io/optimizing-workplace-lighting-for-health-and-productivity-a-comprehensive-guide>

[27] База ГОСТов. ГОСТ 12.2.003-91 Система стандартов безопасности труда. Оборудование производственное. Общие требования безопасности [Электронный ресурс]. – Режим доступа: https://allgosts.ru/13/100/gost\_12.2.003-91

[28] Статистика магазина освещения Home Senator. 5 Types of Lighting and Light Sources to Consider for Your Home [Электронный ресурс]. – Режим доступа: <https://homesenator.com/5-types-of-lighting-and-light-sources-to-consider-for-your-home>

[29] Медицинский журнал MedicalNewsToday Light and mood: Scientists find brain pathway that may explain sensitivity [Электронный ресурс]. – Режим доступа: <https://www.medicalnewstoday.com/articles/light-and-mood-scientists-find-brain-pathway-that-may-explain-sensitivity>

[30] Шупейко И.Г. /Эргономическое проектирование систем «человек–компьютер–среда». Курсовое проектирование // Учебно-методическое пособие. – Минск: БГУИР, 2012. – С. 92

ПРИЛОЖЕНИЕ А (ОБЯЗАТЕЛЬНОЕ) ЛИСТИНГ ПРОГРАММЫ

BaseElement.cs

using MySeleniumFramework.Browser;

using OpenQA.Selenium;

using OpenQA.Selenium.Interactions;

namespace MySeleniumFramework.Elements.ElementTypes

{

public abstract class BaseElement

{

public Actions Actions { get => new Actions(\_driver); }

public IWebElement Element { get => GetWebElement(); }

internal By Locator { get; private set; }

private IWebDriver \_driver = BrowserFactory.GetDriver();

protected BaseElement(By locator)

{

Locator = locator;

}

public IWebElement GetWebElement()

{

IWebElement element = \_driver.FindElement(Locator);

new ElementJsActions(element).HighlightElement();

return element;

}

public ElementVisibility Visibility => new ElementVisibility(this);

public ElementJsActions JsActions => new ElementJsActions(GetWebElement());

public int SameElementsCount() => \_driver.FindElements(Locator).Count;

public string GetText() => GetWebElement().Text;

public void Click() => GetWebElement().Click();

}

}

ElementCollection.cs

using MySeleniumFramework.Browser;

using OpenQA.Selenium;

namespace MySeleniumFramework.Elements.ElementTypes

{

public class ElementsCollection<T> where T : BaseElement, new()

{

public List<T> Elements { get; } = new();

internal By CommonLocator { get; }

private IWebDriver \_driver = BrowserFactory.GetDriver();

public ElementsCollection(By locator)

{

CommonLocator = locator;

var type = typeof(T);

var elementConstructor = type.GetConstructor(new Type[] { typeof(By) });

int webElementsCount = GetFoundCount();

for (int xpathIndex = 1; xpathIndex < webElementsCount; xpathIndex++)

{

string xpathWithIndex = $"({locator})[{xpathIndex}]";

T elementWithIndex = (T)elementConstructor.Invoke(new object[] { xpathWithIndex });

Elements.Add(elementWithIndex);

}

}

public int GetFoundCount() => \_driver.FindElements(CommonLocator).Count;

}

}

ElementJsActions.cs

using MySeleniumFramework.Browser;

using OpenQA.Selenium;

namespace MySeleniumFramework.Elements

{

public class ElementJsActions

{

private readonly IWebElement \_webElement;

private readonly IWebDriver \_driver = BrowserFactory.GetDriver();

internal ElementJsActions(IWebElement element)

{

\_webElement = element;

}

public void HighlightElement()

{

var js = (IJavaScriptExecutor)\_driver;

js.ExecuteScript("arguments[0].style.border='3px solid red'", \_webElement);

}

public void ScrollToElement()

{

var js = (IJavaScriptExecutor)\_driver;

js.ExecuteScript("arguments[0].scrollIntoView(true);", \_webElement);

}

public void DeleteElement()

{

var js = (IJavaScriptExecutor)\_driver;

js.ExecuteScript("arguments[0].remove();", \_webElement);

}

}

}

ElementVisibility.cs

using MySeleniumFramework.Elements.ElementTypes;

namespace MySeleniumFramework.Elements

{

public class ElementVisibility

{

internal BaseElement Element { get; }

internal ElementVisibility(BaseElement element)

{

Element = element;

}

public bool IsDisplayed() => Element.GetWebElement().Displayed;

public bool IsEnabled() => Element.GetWebElement().Enabled;

public bool IsExistInDOM() => Element.SameElementsCount() > 0;

public bool IsNotExist() => Element.SameElementsCount() == 0;

}

}

WaitConditions.cs

using MySeleniumFramework.Browser;

using OpenQA.Selenium;

using OpenQA.Selenium.Support.UI;

namespace MySeleniumFramework.Elements

{

public static class WaitConditions

{

private static IWebDriver \_driver = BrowserFactory.GetDriver();

private static string ExpireMessage(By locator, string waitType, int totalSeconds)

=> $"Element with locator '{locator}' wasn't in '{waitType}' state after '{totalSeconds}' seconds";

public static bool WaitForDisplayed(this ElementVisibility visibility, TimeSpan timeout)

{

WebDriverWait webDriverWait = new(\_driver, timeout);

webDriverWait.Message = ExpireMessage(visibility.Element.Locator, "Displayed", timeout.Seconds);

return webDriverWait.Until(\_driver => visibility.IsDisplayed());

}

public static bool WaitForDisappeared(this ElementVisibility visibility, TimeSpan timeout)

{

WebDriverWait webDriverWait = new(\_driver, timeout);

webDriverWait.Message = ExpireMessage(visibility.Element.Locator, "Displayed", timeout.Seconds);

return webDriverWait.Until(\_driver => !visibility.IsDisplayed());

}

public static bool WaitForNotExist(this ElementVisibility visibility, TimeSpan timeout)

{

WebDriverWait webDriverWait = new(\_driver, timeout);

webDriverWait.Message = ExpireMessage(visibility.Element.Locator, "Displayed", timeout.Seconds);

return webDriverWait.Until(\_driver => !visibility.IsNotExist());

}

}

}

PageObject.cs

using MySeleniumFramework.Elements;

using MySeleniumFramework.Elements.ElementTypes;

namespace MySeleniumFramework.PageObjects

{

public abstract class PageObject

{

protected BaseElement \_uniqueElement;

protected PageObject(BaseElement uniqueElement)

{

\_uniqueElement = uniqueElement;

}

public bool WaitForOpened(TimeSpan timeout)

{

return \_uniqueElement.Visibility.WaitForDisplayed(timeout);

}

}

}

BrowserConfigurations.cs

namespace MySeleniumFramework.Configurations

{

internal class BrowserSettings

{

private IReadOnlyDictionary<string, object> \_browserSettings;

internal BrowserSettings()

{

\_browserSettings = new BrowserSettingsReader().BrowserSettings;

}

public string BrowserName => \_browserSettings["browserName"].ToString();

}

}

BrowserSettingsReader.cs

using System.Text.Json;

namespace MySeleniumFramework.Configurations

{

internal class BrowserSettingsReader

{

private IReadOnlyDictionary<string, object> \_browserSettings;

internal IReadOnlyDictionary<string, object> BrowserSettings => \_browserSettings ??= ReadSettingsFile();

private string browserSettingsPath;

public BrowserSettingsReader()

{

string defaultSettingsPath = Path.GetFullPath(@"Resources/BrowserSettings.json");

browserSettingsPath = defaultSettingsPath;

}

private IReadOnlyDictionary<string, object> ReadSettingsFile()

{

string settingsJson = File.ReadAllText(browserSettingsPath);

\_browserSettings = JsonSerializer.Deserialize<Dictionary<string, object>>(settingsJson);

return \_browserSettings;

}

}

}

BrowserFactory.cs

using MySeleniumFramework.Configurations;

using OpenQA.Selenium;

using OpenQA.Selenium.Chrome;

using OpenQA.Selenium.Edge;

using OpenQA.Selenium.Firefox;

namespace MySeleniumFramework.Browser

{

public class BrowserFactory

{

private static ThreadLocal<IWebDriver> driverThreadLocal = new ThreadLocal<IWebDriver>();

public static IWebDriver GetDriver()

{

if (driverThreadLocal.Value == null)

{

driverThreadLocal.Value = InitDriver();

}

return driverThreadLocal.Value;

}

public static void QuitBrowser()

{

if (driverThreadLocal.Value != null)

{

driverThreadLocal.Value.Quit();

driverThreadLocal.Value = null;

}

}

private static IWebDriver InitDriver()

{

BrowserSettings browserSettings = new();

string browserType = browserSettings.BrowserName.ToLower();

return InitDriverByType(browserType);

}

private static IWebDriver InitDriverByType(string browserName, DriverOptions options = null)

{

ChromeOptions chromeOptions = new();

chromeOptions.PageLoadStrategy = PageLoadStrategy.Eager;

return browserName switch

{

"chrome" => new ChromeDriver(chromeOptions),

"firefox" => new FirefoxDriver(),

"edge" => new EdgeDriver(),

\_ => throw new PlatformNotSupportedException($"Browser type '{browserName}' is not supported. Please pick another browser type in your config.")

};

}

}

}

BrowserJsActions.cs

using OpenQA.Selenium;

namespace MySeleniumFramework.Browser

{

public class BrowserJsActions

{

private readonly IWebDriver \_driver = BrowserFactory.GetDriver();

public void StopPageLoading(TimeSpan stopAfterTime)

{

Thread.Sleep(stopAfterTime);

var js = (IJavaScriptExecutor)\_driver;

js.ExecuteScript("window.stop();");

}

}

}