СОДЕРЖАНИЕ

ВВЕДЕНИЕ

На сегодняшний день процессы по обеспечению качества (quality assurance) являются неотъемлемой частью разработки любого хорошо проработанного программного продукта. С первого взгляда трудно осознать всю важность QA-сферы в мире информационных технологий. Многие компании вкладывают в тестирование своих приложений ресурсы, сопоставимые по величине бюджету, выделенному на их разработку. Невольно возникает вопрос: в чем смысл тратить время, деньги и силы на процессы, основная концепция которых заключается лишь в проверке разработанной до текущего момента функциональности?

Как показывает практика, невозможно сходу создать идеальный программный продукт. Чем больше функциональности содержит приложение, тем выше риски возникновения багов в его системе. Так или иначе, если разработчик хочет быть уверен в качестве своего продукта, ему необходимо покрыть тестами всю функциональность приложения. Выходит, что процесс разработки качественного продукта должен, во-первых, начинаться с четкого формирования требований, а во-вторых заканчиваться проверкой данных требований.

Концепция behavior driven development объединяет два данных понятия в единое целое. Что если сами тесты будут задавать требования, которым должен соответствовать разрабатываемый программный продукт? При таком подходе функциональность приложения постоянно должна тестироваться, а значит, необходимо использовать именно автоматизированное тестирование.

Целью данного дипломного проекта является разработка тестового фреймворка в соответствии с BDD концепцией. Программное решение должно стать многофункциональным инструментом, позволяющим в наибольшей мере упростить процесс написания автотестов.

Также целью данного дипломного проекта является проведение автоматизированного тестирования веб-приложения, находящегося в процессе разработки. Необходимо написать ряд тестовых сценариев, покрывающих всю функциональность приложения, на их базе разработать автотесты, а также благодаря CI утилитам сделать процесс проведения тестирования приложения полностью независимым от человека.

В соответствии с поставленной целью были определены следующие задачи:

– разработка фреймворка по автоматизированному тестированию;

– написание тест-кейсов для проверки функциональности веб-приложения в соответствии с концепцией behavior driven development;

– программная реализация тест-кейсов и проведение автоматизированного тестирования веб-приложения;

– автоматизирование процесса проведения тестирования.

1 ОБЗОР ЛИТЕРАТУРЫ

Для достижения поставленных целей по реализации фреймворка автоматизированного тестирования стоит рассмотреть уже существующие аналоги на рынке, а также ознакомиться с понятиями и технологиями, которые понадобятся при разработке вышеуказанного решения.

1.1 Обзор существующих аналогов

Разработку любого программного продукта стоит начинать с обзора аналогов. Фреймворк по автоматизированному тестированию не является исключением.

На данный момент на рынке существует множество как бюджетных, так и коммерческих тестирующих фреймворков. Самым приближенным примером является Cypress. На сайте разработчиков можно найти цикл статей, позволяющих более детально ознакомиться с фреймворком [1]. На рисунке 1.1 представлен интерфейс Cypress.

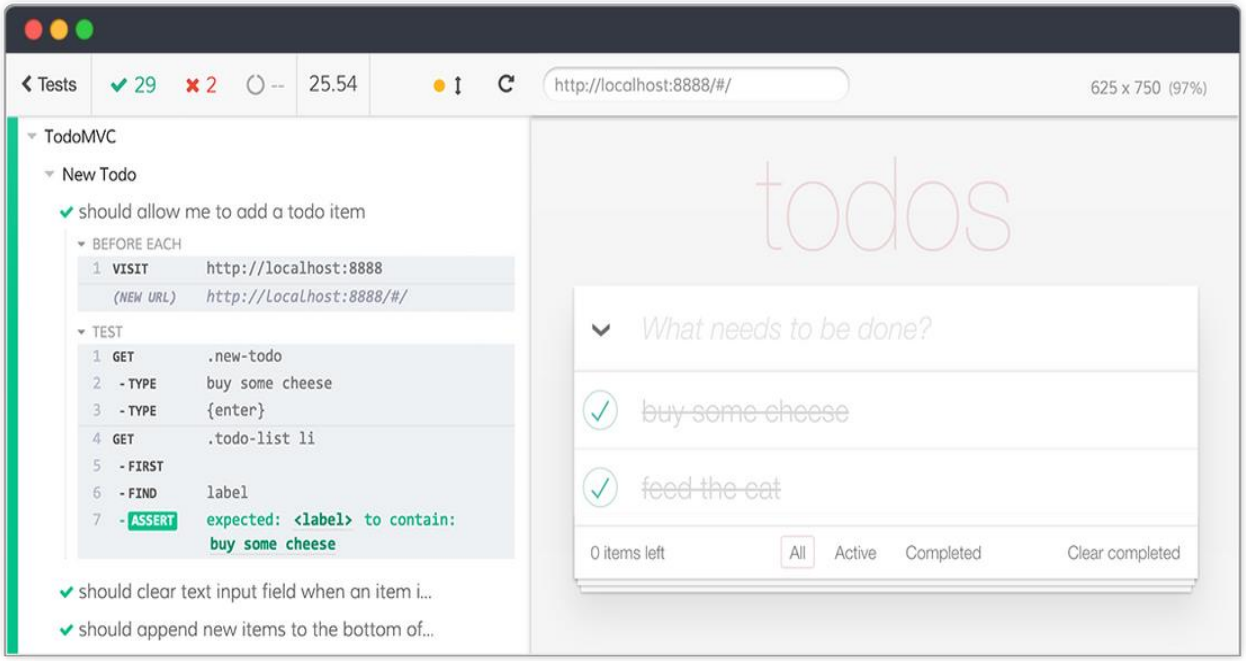

Рисунок 1.1 – Интерфейс фреймворка Cypress [1]

В основе Cypress лежит методология Test Driven Development. В отличие от прочих фреймворков, Cypress использует инновационный подход, основная идея которого заключается в том, что данный фреймворк является окружением для тестируемого продукта. Фактически, веб-приложение запускается в самом Cypress, что позволяет полностью контролировать процесс тестирования, взаимодействовать с любыми веб-элементами на высокой скорости, генерировать детальные отчеты в разных форматах.

Несмотря на все вышеупомянутые достоинства, Cypress также обладает рядом недостатков:

– Cypress позволяет писать тесты исключительно на языке JavaScript;

– в качестве тестовой среды Cypress поддерживает лишь среду Mocha;

– Cypress может взаимодействовать только с браузером Chrome.

Еще одним аналогом является фреймворк Citrus. Особенность данного фреймворка заключается в том, что Citrus позволяет проводить тестирование для любых протоколов обмена сообщениями или форматов данных. Citrus взаимодействует с такими технологиями, как Rest SOAP, JMS, HTTP и другие.

Citrus является open-source решением. Более детально ознакомиться с фреймворком можно на официальном сайте [2]. На рисунке 1.2 изображен принцип, по которому работает данный фреймворк.

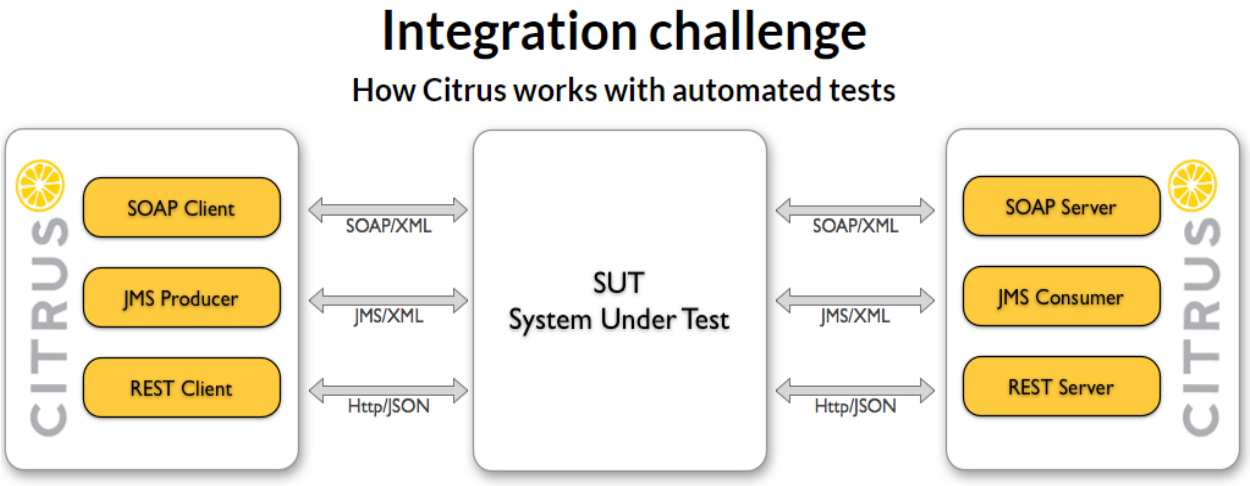

Рисунок 1.2 – Принцип работы фреймворка Citrus [2]

Citrus не создавался как самостоятельный инструмент проведения автоматизированного тестирования веб-приложений. Данный фреймворк является одним из лучших вариантов в случае необходимости проведения тестирования запросов, однако, на этом его функциональность ограничивается. Тем не менее, трудно назвать данную особенность недостатком фреймворка, так как тестирование UI-функциональности попросту не входит в концепцию его создания.

1.2 Понятие тестирования

Для написания тестового фреймворка в первую очередь необходимо разобраться с понятием тестирования. Существует множество различных определений, но все они сводятся к одному: тестирование – это проверка соответствия ожидаемого результата с фактическим. На рисунке 1.3 приведена подробная система классификаций, опираясь на которую можно выделить ряд

понятий, относящихся к тестированию, под которое и разрабатывается автотестовый фреймворк:

– автоматизированное;
– регрессионное;
– модульное;
– функциональное;
– динамическое.

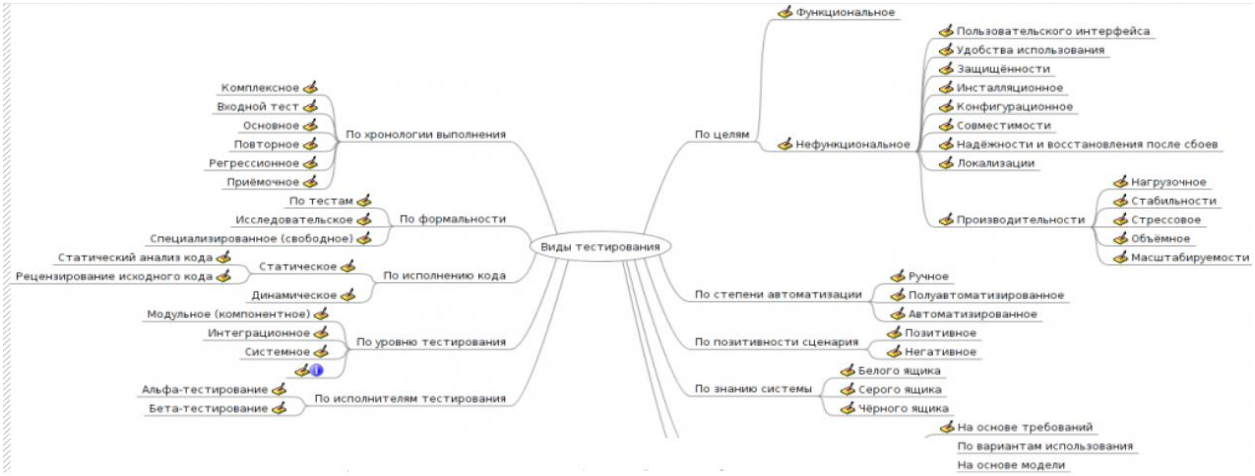

Рисунок 1.3 – Виды тестирования [3]

Стоит также иметь представление о процессах тестирования: по каким принципам разрабатываются тестовые сценарии, в каких ситуациях уместно использовать тестирование того или иного вида, как взаимодействовать с багами и многое другое. Во всех выше перечисленных вопросах поможет разобраться книга Романа Савина «Тестирование DOT COM» [4]. В книге приведено множество наглядных примеров, которые в полной мере отвечают на вопрос: почему процесс обеспечения качества (quality assurance) настолько важен при разработке любого программного продукта.

1.3 Среда юнит-тестирования nUnit

Среди множества сред юнит-тестирования, использующихся на базе платформы .NET, на данный момент выделяются две наиболее развитые: MSTest и nUnit. Документацию к MSTest можно найти на официальном сайте Microsoft [5], документацию к nUnit, соответственно, на сайте разработчиков среды [6]. Делать выбор стоит, руководствуясь следующими факторами: функциональность, возможность генерации отчетов в различных форматах (таких как trx и html), удобство интеграции со сторонними инструментами.

Учитывая все выше изложенные параметры, было решено остановится на nUnit. Не все актуальные фреймворки и окружения осуществляют поддержку MSTest. Также, nUnit обладает большим количеством полезных атрибутов, что значительно расширяет его функциональность.

1.4 Инструмент для автоматизации Selenium WebDriver

Фреймворк ориентирован на автоматизацию веб-приложений. Для достижения поставленной задачи необходимо выбрать соответствующий инструмент взаимодействия с веб-браузером. Selenium WebDriver является наиболее подходящим вариантом. Среди прочих альтернатив на рынке он обладает следующими преимуществами:

– широкий функционал – Selenium предоставляет возможность взаимодействия со всеми элементами веб-страницы, вызывать определенные функции браузера (работа с всплывающими окнами, вкладками, cookie и многое другое), использовать различные JavaScript-методы;

– кроссплатформенность – Selenium работает на разных языковых платформах, а также дает возможность взаимодействовать со всеми актуальными на сегодняшний день браузерами: Chrome, Firefox, Safari, Edge, Internet Explorer;

– Selenium WebDriver является бесплатным решением.

Всю необходимую документацию можно найти на официальном сайте разработчиков [7]. В качестве альтернативы можно воспользоваться циклом статей с русскоязычного сайта «kreisfahrer» [8], рассказывающих об особенностях работы с Selenium’ом. Здесь в первую очередь стоит обратить внимание на статью, посвященную работе с так называемыми ожиданиями. Ожидания позволяют обрабатывать ситуации, при которых элементы веб-страницы загружаются с медленной скоростью или не загружаются вовсе. Грамотная расстановка ожиданий в фреймворке улучшит скорость его работы, а также качество тестов, написанных с его помощью.

Еще одна полезная статья описывает все виды локаторов, с которыми может взаимодействовать Selenium. Локаторы – адреса веб-элементов, которые позволяют взаимодействовать с ними на уровне программного фреймворка. Selenium предоставляет инструменты для работы со следующими видами локаторов:

– XPath – язык запросов к элементам XML;

– CssSelector – позволяет находить элементы с определенными CSS-свойствами;

– Id – поиск элемента по уникальному идентификатору (если таковой есть в верстке веб-страницы);

– ClassName – поиск всех элементов в верстке веб-страницы, принадлежащих к определенному классу;

– LinkText – поиск элемента по ссылке, указывающей на адрес данного элемента;

– прочие, менее широко применяемые локаторы: Name, TagName, PartialLinkText.

Работа с локаторами является крайне важной частью при проведении автоматизированного тестирования любого веб-приложения. Вся сложность заключается в том, что верстка тестируемого сайта может меняться в процессе разработки. Отсюда вытекает необходимость правильного выбора локаторов, которые будут стабильно указывать на необходимые веб-элементы даже при условиях изменения верстки веб-приложения.

1.5 RestSharp

Большинство веб-приложений так или иначе реализовывают базу API-запросов. API-запросы необходимы для взаимодействия клиентской и серверной частей приложения. Также API-запросы позволяют оперативно пользоваться сторонними сервисами (примеры: Cloudinary, TestRail, Amazon и другие).

Возможность реализации API-запросов влечет за собой необходимость проверки их корректной работоспособности. Именно поэтому фреймворк по автоматизированному тестированию должен содержать в себе функциональность, позволяющую тестировать API-запросы.

Среди множества библиотек языка C#, реализующих API-запросы, для поставленной задачи больше других альтернатив подходит библиотека RestSharp. Данная библиотека позволяет не только реализовывать POST, GET, PUT и прочие запросы, но и проводить сериализацию и десериализацию в соответствии с форматами json и xml. С этими и прочими особенностями библиотеки более детально можно ознакомиться на официальном сайте RestSharp [9].

1.6 Концепция Behavior Driven Development

Подход BDD возник на основе другой, более распространенной концепции – Test Driven Development (TDD). Суть данной концепции заключается в том, что тесты для разрабатываемого приложения пишутся еще до непосредственного создания самого приложения. TDD имеет как положительные, так и отрицательные стороны. Среди плюсов ключевым является тот факт, что тесты при таком подходе участвуют в разработке приложения как такового. Множество дефектных ситуаций обрабатывается еще до начала имплементации логики, что может заметно ускорить разработку и улучшить ее качество.

Однако, такой подход сильно замедляет разработку самого приложения. Многие разработчики считают написание тестов еще до создания кода

приложения неоправданно сложным процессом. В первую очередь это связано с формированием требований: не всегда ясно что и как необходимо тестировать. Именно для этого было создано BDD. Отличие Behavior Driven Development от Test Driven Development заключается в том, что при первом подходе больший уклон делается именно на создание требований, на основе которых уже и пишутся тесты для приложения. Подробнее про особенности использование BDD можно прочитать в книге Джона Смарта «BDD in action: Behavior-driven development for the whole software lifecycle» [10]. Также много важной информации можно найти на сайте компании по предоставлении услуг обеспечения качества A1QA [11].

Ключевой особенностью данного подхода стала необходимость четкого взаимодействия между разработчиками бизнес-логики приложения и разработчиками программного кода. Нередко люди первого типа не являются высококвалифицированными программистами, и, напротив, людям второго типа трудно сходу воспринимать бизнес-требования. Возникла необходимость создания единого языка, понятного в равной степени для всех. Таким языком стал Gherkin. Именно на языке Gherkin при использовании таких ключевых слов, как «Scenario» «Given», «When» и «Then» будут писаться тестовые сценарии на базе разрабатываемого фреймворка. На рисунке 1.4 предоставлена образная схема тестового сценария, написанного при поддержке данного языка.

Рисунок 1.4 – Схема тестового сценария на языке Gherkin [12]

1.7 Specflow

Specflow служит инструментом реализации BDD подхода на базе языка программирования C#. Принцип работы Specflow заключается в следующем: тестовые сценарии, написанные на языке Gherkin, хранятся в файлах с расширением «feature»; команды «Given», «When» и «Then» реализовываются на программном уровне при помощи соответствующих атрибутов; по необходимости добавляются хуки (предусловия и постусловия), а также необходимые теги в «feature» файлы.

Со всей необходимой документацией по Specflow можно ознакомиться на официальном сайте разработчиков. На рисунке 1.5 изображен пример «feature» файла.

```
1   Feature: Purchase
2   Account holder makes a purchase
3
4   Scenario Outline: Check balance before purchase
5
6   Given an account with <balance>
7   When the owner attempts a purchase of <cost>
8   Then the transaction should be <status>
9   And the balance should be <new balance>
10
11  Examples:
12  | balance | cost | status   | new balance |
13  | 9.98    | 9.99 | rejected | 9.98        |
14  | 10      | 9.99 | approved | 0.01        |
15
```

Рисунок 1.5 – Пример «feature» файла [13]

1.8 Понятие CI

Процесс проведения автоматизированного тестирования любого приложения по концепции BDD подразумевает постоянный, непрекращающийся запуск разработанных автотестов. Отсюда вытекает необходимость создания условий автоматизации самого процесса тестирования. Для подобных целей была создана методология непрерывной интеграции (Continuous Integration).

Концепция методологии CI проста и эффективна. В ходе создания приложения разработчикам постоянно приходится имплементировать новую функциональность. В связи с этим необходимо все время проверять корректную работоспособность программного продукта. После добавления новых изменений (или при прочих условиях – например, в случае обновления окружения), стоит проверять все приложение на наличие появившихся дефектов и багов. Для подобных целей и применяется автоматизированное тестирование. Ручной подход при таком методе создания программного продукта занимал бы невероятно много человеческого времени. В то же время, при наличии разработанных автотестов нет необходимости в задействовании человека вовсе. Вся функциональность приложения проверяется автоматически, что позволяет разработчикам в случае обнаружения неисправностей оперативно найти их источник и решить возникшую проблему.

В своей книге «Continuous Integration: Improving Software Quality and Reducing Risk» [14] Стив Матыас и Эндрю Гловер детально изложили особенности разработки по методологии непрерывной интеграции. На рисунке 1.6 показана образная схема взаимодействий процессов CI.

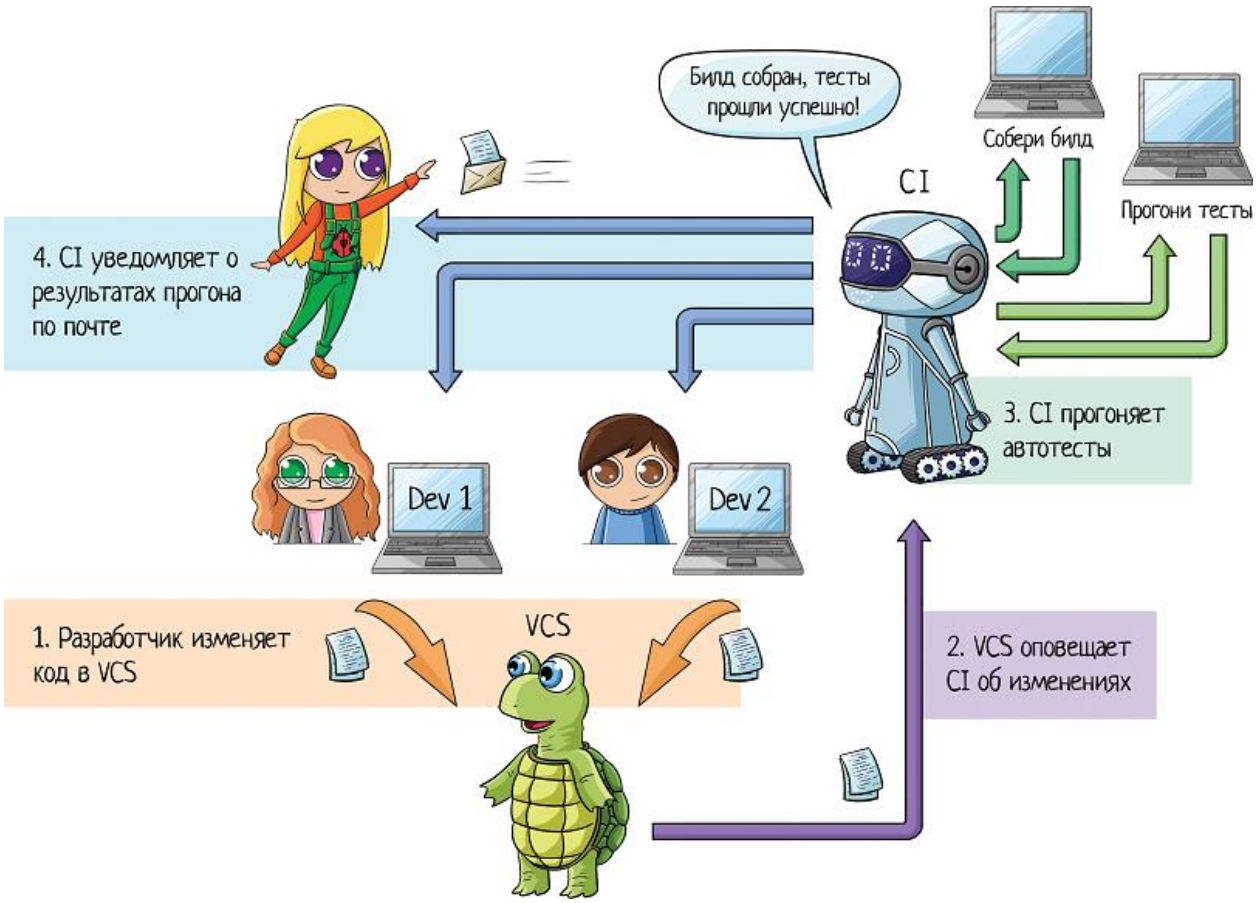

Рисунок 1.6 – Пример взаимодействия по методологии CI [15]

1.9 Jenkins

Jenkins – инструмент непрерывной интеграции, позволяющий реализовать на практике все принципы данной методологии. Jenkins оперирует понятием «Jobs» - процессы, отвечающие за сборку и запуск программных проектов, хранящихся в любой системе контроля версий (например, git), или с любого подключенного машинного устройства (при должном подключении и наличии необходимого окружения). Jenkins позволяет осуществлять сборку проекта по следующим критериям:

– сборка по запуску «Job»'ы (разработчик запускает сборку проекта вручную);

– сборка по ссылке (переход по ссылке осуществляет запуск сборки);

– сборка по изменению состояния репозитория в системе контроля версий (пример: добавление коммита в git-репозиторий);

– сборка по расписанию.

Последнему способу сборки стоит уделить отдельное внимание. Для реализации сборки по расписанию Jenkins использует так называемые «крон-выражения». На рисунке 1.7 представлена схема, взятая с официального сайта разработчиков. Например, выражение «* * 1,15 1-11 *» будет осуществлять сборку по одному разу первого и пятнадцатого числа каждого месяца, кроме декабря.

MINUTE	HOUR	DOM	MONTH	DOW
Minutes within the hour (0–59)	The hour of the day (0–23)	The day of the month (1–31)	The month (1–12)	The day of the week (0–7) where 0 and 7 are Sunday.

To specify multiple values for one field, the following operators are available. In the order of precedence,

- `*` specifies all valid values
- `M-N` specifies a range of values
- `M-N/X` or `*/X` steps by intervals of `X` through the specified range or whole valid range
- `A,B,…,Z` enumerates multiple values

Рисунок 1.7 – Описание синтаксиса «крон-выражений» [16]

2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

Изучив предметную область разрабатываемого тестового фреймворка, были разработаны основные требования, которые должны быть выполнены при реализации дипломного проекта. Для упрощения разработки системы разобьем ее на структурные блоки.

2.1 Описание основных блоков программы

В разрабатываемом фреймворке по автоматизированному тестированию были выделены следующие блоки:

– блок конфигурации браузера;
– блок веб-элементов браузера;
– блок сущностей браузера;
– блок API-функциональности;
– блок расширений;
– блок тестовых сценариев;
– блок тестовой функциональности;
– блок модели тестируемого приложения.

Структурная схема, иллюстрирующая перечисленные блоки и связи между ними приведена на чертеже ГУИР.400201.020 С1.

Каждый из блоков содержит в себе определенную функциональность, необходимую для корректной работы фреймворка и стабильного запуска автотестов. Рассмотрим более детально работу и задачи каждого блока отдельно, а также их взаимодействие и обмен данных между блоками.

2.2 Блок конфигурации браузера

Блок конфигурации браузера является сердцем автотестового фреймворка. Здесь находится логика, отвечающая за создание объекта браузера, его настройку, взаимодействие с инструментами самого браузера (например, переключение между вкладками, работа с всплывающими окнами, взаимодействие с куками, вызов JavaScript-методов, открытие и закрытие браузера).

Объект браузера создается в соответствии с параметрами, указанными в ресурсных файлах. Разработчик автотестов может выбрать один из четырех видов браузера, в котором будет проводится тестирование веб-приложения: Chrome. Safari, Firefox или Edge. Предоставляется возможность настройки времени ожидания открытия веб-страницы или появления элемента на ней, можно запускать тесты в headless (фоновом) режиме.

Данный блок взаимодействует с блоками веб-элементов и веб-сущностей браузера, а также с блоком тестовой функциональности. Первые два используют инструменты браузера в целях взаимодействия с объектами

веб-страниц. В блоке тестовой функциональности реализованы хуки (запуск и закрытие браузера).

2.3 Блок веб-элементов браузера

Фреймворк осуществляет взаимодействие с веб-страницами браузера через элементы, из которых они состоят. К наиболее часто используемым веб-элементам относятся:

– текстовые поля (поля для ввода текстовой информации);

– лейблы (неизменяемые текстовые поля страницы – заголовки, статьи и тому подобное);

– чекбоксы (элементы управления параметрами в двух состояниях - включение и отключение);

– комбобоксы (поля выбора одного из нескольких вариантов);

– кнопки.

У всех элементов есть схожие свойства. Например, на все элементы можно нажать кнопкой мыши, у всех элементов можно взять размеры и координаты, а также их состояния на странице (можно ли кликнуть на элемент, виден ли элемент на веб-странице, существует ли элемент вовсе). В то же время, каждый элемент должен обладать собственной функциональностью. В текстовое поле, в отличие от кнопки, можно передавать определенные значения; комбобокс позволяет выбирать 1 из вариантов, содержащихся в нем и так далее.

Данный блок тесно взаимодействует с блоком страниц тестируемого приложения – все страницы так или иначе содержат в себе различные элементы, через взаимодействие с которыми будет осуществляться тестирование веб-приложения.

2.4 Блок сущностей браузера

Помимо веб-элементов, из которых состоят все веб-приложения интернет-пространства, существуют сущности более высокого уровня, которые присутствуют у большинства сайтов. Хорошим примером является форма создания профиля пользователя. У всех подобных форм можно выделить ряд общих черт: есть поля для заполнения логина, пароля, почты, даты рождения, часто используются поля для заполнения личной информации и выбора пола, всегда есть кнопка подтверждения. Более того, всем формы создания профиля соответствуют общему шаблону поведения, которое часто приходится тестировать. Например, имя не может быть слишком коротким, в поле email обязательно использование определенных спецсимволов и тому подобное.

Подобные сущности будет использовать в себе модель тестируемого приложения. При разработки тестирующей части это значительно упростит

процесс создания моделей страниц и ускорит написание автотестов. Сам блок сущностей браузера зависит от блока веб-элементов, так как любая сущность полностью состоит из них.

2.5 Блок API-функциональности

Блок API-функциональности содержит в себе логику реализации API-запросов. Данный блок должен решать две задачи. Во-первых, необходимо предоставить инструментарий для создания различных типов запросов с любыми входными параметрами. Должна быть реализована поддержка передачи данных не только в текстовом, но и в графическом формате. Для достижения данной цели стоит разработать механизм, преобразующий изображение в формат «multipart/form-data», и способный совершать обратный процесс.

Второй целью данного блока является реализация механизмов сериализации и десериализации по двум наиболее часто используемым на сегодняшний день форматам обмена данных посредством API механизма: json и xml. Все запросы должны быть представлены в виде моделей на программном уровне для удобного взаимодействия с ними.

Данный блок должен быть разработан в первую очередь для проведения тестирования API-функциональности, следовательно, он имеет прямое взаимодействие с блоком тестовой функциональности.

2.6 Блок расширений

Данный блок содержит в себе множество полезных утилит, которые используются в различных прочих блоках фреймворка. В первую очередь здесь присутствует функциональность логирования. Это необходимо для генерации максимально детальных и точных тестовых отчетов. В случае получения негативного результата теста, грамотная система логирования поможет быстро разобраться в каком месте веб-приложение функционирует некорректно. Логи должны быть задействованы во всех блоках автотестового фреймворка. Особенно это актуально для блока веб-элементов. Отчеты должны содержать в себе подробную информацию о том, какие действия были совершены с теми или иными элементами веб-страницы.

Другой немаловажной частью блока расширений является функциональность, направленная на взаимодействие с ресурсами. Тут стоит упомянуть два важных фактора. Во-первых, работа с ресурсами позволяет получать всю необходимую информацию для конфигурации тестового фреймворка и самих тестов. Во-вторых, ресурсные файлы могут хранить в себе данные, используемые для проведения тех или иных автотестов. Хорошим примером являются креды веб-приложения – логины и пароли пользователей с различными правами доступа.

Также в блоке расширения должны содержаться инструменты взаимодействия с файлами, изображениями, датами и прочими подобными сущностями, так как это значительно упростит процесс написания автотестов.

2.7 Блок тестовых сценариев

Данный блок должен напрямую иллюстрировать behavior driven development концепцию в действии. Весь блок будет состоять из «feature»-файлов, написанных при поддержке фреймворка Specflow.

Каждый из «feature»-файлов должен содержать в себе тестовый сценарий, написанный на языке Gherkin. Все тестовые сценарии должны покрывать основную функциональность тестируемого приложения.

В соответствии с языком Gherkin, «feature»-файлы содержат в себе так называемые степы тестирования. «Given» степы отвечают за создание окружения, в котором будет проходить тест. «When» степы задают последовательность действий, которая должна привести к определенному результату. «Then» степы должны проводить само тестирование, то есть проверяют соответствие ожидаемого результата с фактическом.

Все тесты должны быть заимплементированы в блоке тестовой функциональности. Таким образом, данный блок выполняет две принципиально важные для проведения тестирования задачи: содержит в себе всю тестовую документацию, участвует в программной разработке самих автотестов.

2.8 Блок тестовой функциональности

Блок тестовой функциональности содержит в себе сами тесты в виде программного кода. Каждому степу из «feature» файлов соответствует определенный метод, который должен выполнять соответствующую функцию. Отсюда возникает необходимость разработать четкую структуру и разделить все тестовые методы на группы, соответствующие их назначению («given», «when» и «then» методы).

Особое внимание стоит уделить логике, отвечающей за конфигурацию тестового окружения. Необходимо разработать множество хуков, которые будут контролировать открытие браузера в соответствии с заданными параметрами, его корректное закрытие с очисткой использовавшихся ресурсов, реализовывать ряд прочих преднастроек, которые будут актуальны для многих тестов (переход на необходимую страницу браузера, получение данных из ресурсных файлов и тому подобное).

По понятным причинам блок тестовой функциональности наиболее тесно взаимодействует с блоком тестовых сценариев. По факту, блок тестовой функциональности всего лишь инструмент, который разрабатывается для реализации логики, хранящейся в «feature» файлах. Логика взаимодействия с браузером берется из блока конфигурации браузера.

Еще один блок, с которым взаимодействует тестовая функциональность – блок модели тестируемого приложения. Тестовые степы в большинстве случаев должны лишь собирать в себе методы, которые реализовывают страницы в соответствии со своей функциональностью.

2.9 Блок модели тестируемого приложения

Блок модели тестируемого приложения – полная абстракция сайта, над которым необходимо провести автоматизированное тестирование. Каждая страница, существующая в веб-приложении, должна иметь свою модель в рамках автотестового фреймворка. Сами же модели должны хранить в себе объекты страницы. Отсюда появляется связь с блоком веб-элементов браузера. Более того, так как все страницы-модели создаются для проверки функциональности своих оригиналов, необходимо добавить возможность скриптового воспроизведения данной функциональности. Хорошим примером тут может стать форма авторизации. В модели необходимо создать объекты страницы – текстовые поля логина и пароля, кнопку подтверждения, прочие элементы в случае их наличия на странице. Также модель должна содержать в себе функцию проведения авторизации – заполнение полей логина и пароля, нажатие на кнопку подтверждения. В целях проведения тестирования функциональности страницы авторизации, стоит также добавить методы проверки отображения как элементов страницы, так и самой страницы.

Структура модели веб-приложения должна соответствовать своему действительному аналогу. В случае наличия связей, одни страницы должны ссылаться на другие, общие сущности необходимо выносить в отдельные модели, а страницы их содержащие должны иметь методы перехода на данные сущности. Для достижения данных целей хорошей практикой будет использование паттерна «компоновщик». Возвращаясь к примеру со страницей авторизации, ее модель также должна содержать метод перехода на другую страницу, которая открывается в случае совершения успешной авторизации.

Как было упомянуто выше, блок модели тестируемого приложения должен предоставлять всю необходимую функциональность для блока тестовой функциональности. Основная идея заключается в том, что созданная абстракция тестируемого сайта должна содержать в себе эмуляцию любых действий реального пользователя, которые затрагивают функциональность веб-приложения.

3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

Рассмотрим подробно функционирование программы. Для этого проведем анализ основных модулей фреймворка и рассмотрим их зависимости. При разработке используется объектно-ориентируемый подход, следовательно, весь код состоит из классов и их объектов.

Условно программу можно разделить на следующие части:

– блок тестирующего фреймворка;
– блок тестирования приложения.

Структурная схема, иллюстрирующая перечисленные блоки и связи между ними приведена на чертеже ГУИР.400201.020 РР.

3.1 Блок тестирующего фреймворка

Данный блок является универсальным тестирующим инструментом, предназначенным для упрощения процесса проведения автоматизированного тестирования. Подразумевается реализовать логику, полностью независимую от веб-приложения, для которого будут разрабатываться автотесты.

Рассмотрим подробнее все классы, перечисления и сущности, присутствующие в данном классе.

3.1.1 Класс `Browser`

Данный класс является моделью браузера, в котором будут запускаться тесты. В целях экономии ресурсов, здесь уместно использование паттерна Singleton, так как любой тест будет задействовать лишь один браузер. Класс должен содержать в себе поля и методы, отвечающие за взаимодействие с инструментами самого браузера. Рассмотрим их подробнее.

– `IWebDriver Driver` – драйвер браузера, позволяющий проводить с ним взаимодействие;

– `int WaiterSize` – длительность ожидания появления страниц и элементов браузера;

– `IWebDriver CreateBrowser(BrowserFactory browser)` – метод создания браузера и контроля его уникальности;

– `void OpenUrl(string url)` – метод перехода по url-адрессу;

– `void SwitchTab(string tabName)` – метод переключения на определенную вкладку;

– `void CloseTab()` – метод закрытия текущей вкладки;

– `void SwitchTab(string tabName)` – метод переключения на определенную вкладку;

– `WebElement FindElement(By locator)` – метод поиска элемента на странице по локатору;

– `void SendCookie(string key, string value)` – метод отправки Cookie на сайт;

– `string GetCookie(string key)` – метод получения Cookie с сайта;

– `void DeleteCookie(string key)` – метод удаления Cookie с сайта;

– `void MakeJSACtion(string script)` – метод выполнения JavaScript метода сайта;

– `void CloseBrowser()` – метод закрытия и очистки браузера.

3.1.2 Класс `BrowserFactory`

Данный класс содержит в себе методы конфигурации браузеров разных типов. Логика данного класса используется в классе Browser для создания объекта браузера определенного типа. Для упрощения взаимодействия добавлено перечисление BrowserName, содержащее в себе следующие поля: Chrome, Firefox, Safari, Edge. Ниже описана вся функциональность класса BrowserFactory:

– `void CreateChromeDriver()` – метод создания конфигурации браузера Chrome;

– `void CreateFirefoxDriver()` – метод создания конфигурации браузера Firefox;

– `void CreateSafariDriver()` – метод создания конфигурации браузера Safari;

– `void CreateEdgeDriver()` – метод создания конфигурации браузера Edge;

– `void CreateDriver(BrowserName driver)` – метод одного из четырех вышеперечисленных в соответствии с входным параметром.

3.1.3 Класс `BaseElement`

Абстрактный класс BaseElement содержит в себе поля и методы, отвечающие за общую логику, присущую любому элементу веб-страницы. Данный класс является вершиной иерархии наследования всех классов-моделей элементов. К каждому из данных классов будет прилагаться одноименный интерфейс. Это необходимо для соблюдения двух паттернов проектирования – композиции (удобство обращения к элементам) и фабрики (реализация общего интерфейса всех элементов). Подробнее данные классы будут рассмотрены ниже. Класс BaseElement имеет следующие поля и методы:

– `string name` – название элемента (в первую очередь необходимо для логирования);

– `By selector` – локатор поиска элемента на веб-странице;

– `bool IsVisible` – поле, отвечающее за состояние видимости элемента;

– `bool IsClickable` – поле, описывающие возможность нажатия на элемент при помощи кнопки мыши;

– `bool IsExist` – поле, описывающее факт существования элемента на веб-странице;

– `void Click()` – метод нажатия на элемент левой кнопкой мыши;

– `void DoubleClick()` – метод двойного нажатия на элемент левой кнопкой мыши;

– `void RightClick()` – метод нажатия на элемент правой кнопкой мыши;

– `void Focus()` – метод наведения на элемент левой кнопкой мыши;

– `string GetAttribute(string attributeType)` – метод получения определенного атрибута элемента;

– `void WaitFor()` – метод ожидания появления элемента на странице;

– `void Scroll(Direction direction)` – метод скроллинга к элементу (принимает в качестве параметра один из объектов перечисления Direction: up, down, left, right);

– `string GetText()` – метод получения текста элемента.

3.1.4 Класс `TextField`

Класс TextField является моделью текстового поля браузера. Текстовые поля предназначены в первую очередь для передачи в них определенного текстового значения. В данном классе содержатся методы взаимодействия, относящиеся исключительно к текстовым полям:

– `void SendKeys(string text, bool makeClear)` – метод передачи данных в текстовое поле (с возможностью первоочередной очисткой поля);

– `void Clear()` – метод очистки текстового поля.

3.1.5 Класс `ComboBox`

Комбобоксы – элементы, предоставляющие функциональность выбора варианта (или вариантов) из списка предложенных. Функциональность комбобоксов включает в себя взаимодействие со внутренними элементами. Данный класс включает в себя следующую функциональность:

– `void Select(string item)` – метод выбора элемента комбобокса;

– `bool Contains(string item)` – метод определения наличия в комбобоксе элемента;

– `List<string> GetItems()` – метод получения списка элементов комбобокса.

3.1.6 Класс `CheckBox`

Чекбоксы – эдементы, предназначенные для выбора. Они хранят в себе два состояния – либо чекбокс выбран, либо нет. Функциональность данного элемента не сильно отличается от функциональности BaseElement. Данный класс обладает следующими двумя методами:

– `void Select()` – метод выбора чекбокса;

– `void Unselect()` – метод отмены выбора чекбокса.

3.1.7 Классы `Button и Label`

Кнопки и лейблы не обладают функциональностью, которая может относиться исключительно к ним. Все действия, которые можно совершить с данными веб-элементами, уже описаны в классе BaseElement. Однако, с целью соблюдения принципа абстракции объектно-ориентированного программирования, данные элементы (как и их интерфейсы), должны иметь соответствующие модели в фреймворке (классы Button и Label соответственно).

3.1.8 Класс `ElementFactory`

Класс ElementFactory необходим для внесения логики взаимодействия с элементами по всему проекту. Подразумевается, что при возникновении необходимости обращения к тому или иному элементу, данное обращение будет происходит через фабрику элементов. ElementFactory работает с интерфейсами моделей элементов, описанных выше. Данный класс содержит в себе следующие методы:

– `IButton GetButton(By locator, string name)` – метод, позволяющий взаимодействовать с элементом Button;

– `ILabel GetLabel(By locator, string name)` – метод, позволяющий взаимодействовать с элементом Label;

– `ITextField GetTextField(By locator, string name)` – метод, позволяющий взаимодействовать с элементом TextField;

– `ICheckBox GetCheckBox(By locator, string name)` – метод, позволяющий взаимодействовать с элементом CheckBox;

– `IComboBox GetComboBox(By locator, string name)` – метод, позволяющий взаимодействовать с элементом ComboBox.

3.1.9 Класс `BasePage`

Следующие три класса реализовывают в себе модели сущностей веб-страницы браузера. Класс BasePage является базовой моделью страницы. Любое веб-приложение состоит из определенной перечни страниц. В блоке тестирования приложения описаны модели всех страниц приложения, над которым осуществляется тестирование. Каждая из страниц должна наследоваться от BasePage. BasePage, в свою очередь, содержит в себе поля и методы, которые могут быть применимы абсолютно к любой странице веб-пространства:

– `bool IsOpened` – поле, описывающее открыта ли в браузере на текущий момент данная страница;

– `By BaseLocator` – поле, содержащее в себе уникальный локатор, относящийся к данной странице;

– `string name` – название текущей страницы;

– `string address` – адрес текущей страницы;

– `bool IsElementExist(BaseElement element)` – метод, описывающий существование определенного веб-элемента на текущей странице;

– `string GetHtml(By locator)` – метод, возвращающий html-код элемента, находящегося на странице;

–

– `BaseForm GoToForm(BaseForm form)` – метод, обращающийся к форме страницы;

– `BaseMenu GoToMenu(BaseMenu form)` – метод, обращающийся к меню страницы.

3.1.10 Класс `BaseForm`

Множество веб-страниц, хранящихся в интернете, состоят из в той или иной степени шаблонных блоков, имеющих ряд общих свойств. BaseForm является сущностью, которая описывает поведение тех самых шаблонных блоков. От данного класса будут наследоваться все формы-модели тестируемого приложения.

Поля и методы, реализованные в классе BaseForm:

– `bool IsVisible` – поле, описывающее состояние видимости формы на текущей странице браузера;

– `By BaseLocator` – поле, содержащее в себе уникальный локатор, относящийся к данной форме;

– `bool IsElementExist(BaseElement element)` – метод, описывающий существование определенного веб-элемента внутри текущей формы;

– `string GetHtml(By locator)` – метод, возвращающий html-код элемента, находящегося внутри формы;

– `void InputData(BaseElement element, string data)` – метод введения данных в веб-элемент, присутствующий на форме;

– `void Submit(BaseElement element)` – метод подтверждения формы.

3.1.11 Класс `BaseMenu`

Отдельное внимание стоит уделить меню веб-страниц. Большинство сайтов в том или ином виде имеют свое меню. Учитывая тот факт, что с большой долей вероятности тесту придется взаимодействовать с меню сайта в целях навигации по нему, оправдано использование класса BaseMenu. Следующие поля и методы относятся к данному классу:

– `bool IsVisible` – поле, описывающее состояние видимости формы на текущей странице браузера;

– `By BaseLocator` – поле, содержащее в себе уникальный локатор, относящийся к данной форме;

– `bool Select(string menuItem)` – метод, отвечающий за выбор элемента меню;

– `List <string> GetListOfEntries()` – метод, возвращающий список элементов меню;

– `bool IsMenuItemExist(string menuItem)` – метод, проверяющий наличие элемента меню.

3.1.12 Класс `ApiGetRequest`

Следующие два класса описывают логику реализации API-запросов. ApiGetRequest позволяет выполнять GET-запросы. Здесь уместно использование механизмов сериализации и десериализации, так как большинство ответов, получаемых с сервера, возвращаются в формате json. Данный формат имеет довольно простую систему ключей и значений, что позволяет без особых трудностей преобразовать сообщения в модель и взаимодействовать на уровне методов данной модели.

Большинство методов данного класса должны быть перегружены, так как в GET-запрос может поступать множество комбинаций различных параметров. Рассмотрим методы класса ApiGetRequest:

– `string Get(string url)` – стандартный метод реализации GET-запроса (имеет три перегрузки);

– `string GetFile(string url, string FilePath)` – метод получения файла через GET-запроса (имеет три перегрузки);

–

– `List <string> GetList(string url)` – метод получения списка объекта через GET-запрос (имеет три перегрузки).

3.1.13 Класс `ApiPostRequest`

Данный класс схож по своей структуре с ApiGetRequest. Единственное его отличие – реализация POST-запросов. Следующие методы содержаться в данном классе:

– `string Post(string url)` – стандартный метод реализации POST-запроса (имеет три перегрузки);

– `string PostFile(string url, string FilePath)` – метод отправки файла через POST-запрос (имеет три перегрузки);

– `string PostList(List <string> objects)` – метод отправки списка объекта через POST-запрос (имеет три перегрузки).

3.1.14 Класс `JsonUtils`

Роль данного класса заключается в работе с данными в формате json. Класс JsonUtils должен активно использоваться в классах, отвечающих за реализацию API-запросов, и быть тем самым механизмом сериализации и десериализации для обеспечения удобного взаимодействия с ответами запросов.

Данный класс имеет следующие методы:

– `JObject SerialazeToJObject(string jsonString)` – метод сериализации json строки в объект JObject;

– `string DeserialazeFromJObject(JObject jsonobject)` – метод десериализации JObject в строковой формат;

– `List <JObject> GetListOfJObject (string jsonString)` – метод сериализации json строки в массив объектов JObject.

3.1.15 Класс `Logger`

Данный класс играет ключевую роль в формировании отчетности по результатам тестов, созданных на базе разрабатываемого фреймворка. Логирование необходимо добавить во всех местах взаимодействия с браузером. Во-первых, логи должны присутствовать в методах взаимодействия с браузером и оповещать об открытии браузера, переключении между вкладками, добавлении или удалении куков и так далее. Во-вторых, логирование должно вестись при взаимодействии с элементами веб-страницы. Для этих целей класс BaseElement содержит в себе поле – название элемента. Все методы взаимодействия с веб-элементами должны содержать в себе лог, дающий полную информацию о том, с каким элементом

страницы и какого рода было совершено взаимодействие. В-третьих, необходимо логировать API-запросы (в случае с API-запросами чаще всего результат успешности прохождения теста можно увидеть наглядно исключительно через логи).

Класс Logger имеет следующую структуру:

Данный класс имеет следующие методы:

– `ILog logger` – объект инициализации логгера;

– `void InitLogger()` – метод инициализации логера;

– `void MakeWorningLog(string message)` – метод создания предупреждающего лога ;

– `void MakeInfoLog(string message)` – метод создания информирующего лога ;

– `void MakeErrorLog(string message)` – метод создания лога, предупреждающего об ошибке.

3.1.16 Класс `ResourcesUtils`

Роль класса ResurcesUtils – упростить работу с ресурсными и конфигурационными файлами. Ресурсные файлы содержат в себе тестовые данные, которые используются в приложении. Роль конфигурационных файлов – кастомизировать запуск тестов (тип браузера, длительность ожидания элементов и прочее).

Данный класс имеет следующие методы:

– `string GetValueFromFile(string key, string filePath)` – метод получения данных по ключу из заданного файла;

– `string GetConfigValue(string key)` – метод получения данных по ключу из конфигурационного файла;

– `string GetTestDataValue(string key)` – метод получения данных по ключу из файла тестовых данных;

– `string GetCreditalsValue(string key)` – метод получения данных по ключу из файла, содержащего креды к приложению.

7 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ ТЕСТОВОГО ФРЕЙМВОРКА С ПОДДЕРЖКОЙ BDD ТЕСТИРОВАНИЯ

7.1 Характеристика программного средства

Разрабатываемый тестовый фреймворк в первую очередь предназначен для использования при проектировании и создании прочих веб-приложений. Стоит иметь ввиду, что фреймворк является именно программным решением и рассчитан исключительно на аудиторию разработчиков, в чьи цели входит тестирование их программного продукта.

Данное решение по автоматизации представляет собой набор удобной функциональности, облегчающей процесс разработки автоматизированных тестов. Фреймворк включает в себя инструменты тестирования пользовательского интерфейса веб-приложений, позволяющие эмулировать деятельность настоящих пользователей, ряд методов, направленных на тестирование функциональности API-запросов, несколько расширений, упрощающих работу с датой, файлами, изображениями.

Среди прочих конкурентов фреймворк выделяется поддержкой тестирования в соответствии с behavior driven development концепцией. Это может привлечь тех разработчиков, которые помимо проведения тестирования также задаются целью сформировать требования к приложению, находящемуся на ранней стадии разработки. Интегрирована возможность написания тестовых сценариев на языке gherkin с последующей их реализацией, что упростит процессы взаимодействия бизнес-аналитиков, отвечающих за создание спецификации к приложению, и тестировщиков, разрабатывающих автотестовые скрипты для проверки функциональности тестируемого приложения.

7.2 Расчет инвестиций в разработку программного средства для реализации его на рынке

Рассчитаем цену программного продукта (ПП) основываясь на полных затратах на разработку организацией-разработчиком, где оценка стоимости создание ПП со стороны разработчика предполагает создание плана затрат, который включает соответствующие расходы:

– дополнительную ($З_д$) и основную заработную плату ($З_о$);
– отчисления на социальные нужды ($Р_{соц}$);
– прочие расходы ($Р_{пр}$);
– общую сумму затрат на разработку ($З_р$);
– плановую прибыль, включаемую в цену программного продукта ($П_{пс}$);
– отпускную цену программного средства ($Ц_{пс}$).

Основная заработная плата исполнителей на конкретный программный продукт рассчитывается по формуле:

$$З_о = \sum_{i=1}^{n} З_{чi} \cdot t_i \cdot К_{пр}, \quad (7.1)$$

где n – количество исполнителей, занятых разработкой конкретного программного продукта;

$З_{чi}$ – часовая тарифная ставка *i*-го исполнителя, руб;

t_i – трудоемкость работ, выполняемых *i*-ым исполнителем, час;

$К_{пр}$ – коэффициент премирования, равен 1,5.

Размер месячной заработной платы каждого исполнителя соответствует установленному в организации-разработчике фактическому ее размеру, зависящему от должности, занимаемой сотрудником. Часовая заработная плата каждого исполнителя определяется путем деления его месячной зарплаты на количество рабочих часов в месяце. Количество рабочего времени в месяц по данным Министерства труда и социальной защиты составляет 168 часов.

Трудоемкости проекта была дана оценка в 3 месяца, или 504 часа, при 168 рабочих часа в месяц. Необходимо иметь ввиду, что архитектор решений участвует только на этапе проектирования, (первые два месяца – 336 часов).

Расчет затрат на основную заработную плату разработчиков программных средств клиент-серверного приложения для проведения онлайн анкетирования представлена в таблице 7.1.

Таблица 7.1 – Расчет основной заработной платы исполнителей

Категория исполнителя	Месячная заработная плата, руб.	Часовая заработная плата, руб.	Трудоемкость работ, ч.	Итого, руб.
1	2	3	4	5
Архитектор решений	1075,00	6,40	336	2153,49
Проектный менеджер	1500,00	8,92	504	4495,68
Инженер-программист	1600,00	9,52	504	4798,08
QA-инженер	1600,00	9,52	504	4798,08
Итого				16245,33
Премия, 150 %				8122,665
Всего затраты на основную заработную плату сотрудников				24368,40

Дополнительная заработная плата включает выплаты, предусмотренные законодательство о труде, и определяется по нормативу в процентах от основной заработной платы:

$$З_д = \frac{З_о \cdot Н_д}{100}, \quad (7.2)$$

где $З_о$ – затраты на основную заработную плату с учетом премии, руб;
$Н_д$ – норматив дополнительной заработной платы, 10%.

Отчисления на социальные нужды ($Р_{соц}$), в фонд социальной защиты населения и на обязательное страхование, определяется в соответствии с действующими законодательными актами по формуле:

$$Р_{соц} = \frac{(З_о + З_д) \cdot Н_{соц}}{100}, \quad (7.3)$$

где $Н_{соц}$ – норматив отчислений в фонд социальной защиты, 34,6%.

Расходы по статье «Прочие расходы» ($Р_{пр}$) включают затраты на приобретение специальной литературы, необходимой для разработки программного продукта. Определяются по формуле:

$$Р_{пр} = \frac{З_о \cdot Н_{пр}}{100}, \quad (7.4)$$

где $Н_{пз}$ – норматив прочих затрат в целом по организации, 30%.

Определим расходы на реализацию:

$$Р_р = \frac{З_о \cdot Н_р}{100}, \quad (7.5)$$

где $Н_р$ – норматив расходов на реализацию, 3%.

Определим общую сумму инвестиций на разработку:

$$З_р = З_о + З_д + Р_{соц} + Р_{пр} + Р_р, \quad (7.6)$$

где $Р_{пс}$ – рентабельность затрат, 30%.

Формирование цены на основе затрат, приведены в таблице 7.2.

Таблица 7.2 – Расчет основной заработной платы исполнителей

Наименование статьи затрат	Расчет	Значение, руб.
1	2	3
Основная заработная плата разработчиков	См. табл.7.1	24368,40
Дополнительная заработная плата разработчиков	$\frac{24368,40 \cdot 10}{100}$	2436,84
Отчисления на социальные нужды	$\frac{(24368,40 + 2436,84) \cdot 34,6}{100}$	9274,61
Прочие расходы	$\frac{24368,40 \cdot 30}{100}$	7310,52
Расходы на реализацию	$\frac{24368,40 \cdot 3}{100}$	731,05
Общая сумма инвестиций на разработку	$24368,40 + 2436,84 + +9274,61 + 7310,52 + 731,05$	44121,42

7.3 Расчет экономического эффекта от реализации программного средства на рынке

Тестовый фреймворк предназначен в первую очередь для коммерческого распространения. На данный момент существует большое количество дорогостоящих аналогов, следовательно, упор стоит делать на позиционирование продукта как решения низкой ценовой категории.

Для определения ожидаемого количества продаж лицензий воспользуемся информацией с официального сайта аналога - автотестового фреймворка Ranorex [17]. Разработчики предоставляют 3 вида лицензии – «Runtime», «Studio» и «Enterprise» по 690, 2890 и 4790 евро за год использования соответственно. На главной странице [18] указано, что с момента создания решения (2007 год) было продано более 14000 лицензий среди которых 4000 «Enterprise». Стоит ориентироваться на самую бюджетную лицензию - «Studio», экземпляров которой было продано около пяти тысяч за 14 лет (около 360 в год). Учитывая тот факт, что разрабатываемый фреймворк не обладает графическим интерфейсом, имеет меньше функциональности и по своей сути позиционируется как бюджетное решение, что и выделяет его из конкурентов, было решено уменьшить его стоимость в 3 раза – 650 рублей за годовую лицензию.

Прирост чистой прибыли, полученной разработчиком от реализации программного средства на рынке, рассчитаем по формуле:

$$\Delta П_{ч}^{р} = (Ц_{отп} \cdot N - НДС) \cdot \frac{Р_{пр}}{100} \cdot (1 - \frac{Н_{п}}{100}), \quad (7.7)$$

где $Ц_{отп}$ – отпускная цена копии (лицензии) программного средства, руб;

N – количество копий (лицензий) программного средства, реализуемое за год, шт;

НДС – сумма налога на добавленную стоимость, руб;

$Р_{пр}$ – рентабельность продаж копий (лицензий), 30%;

$Н_{п}$ – ставка налога на прибыль согласно действующему законодательству, 18%.

Налог на добавленную стоимость определяется по формуле:

$$НДС = \frac{Ц_{отп} \cdot N \cdot Н_{дс}}{100\% + Н_{дс}}, \quad (7.8)$$

где $Н_{дс}$ – ставка налога на добавленную стоимость в соответствии с действующим законодательством, 20%.

$$НДС = \frac{650,00 \cdot 360 \cdot 20}{100 + 20} = 39000,00, \quad (7.9)$$

$$\Delta П_{ч}^{р} = (650,00 \cdot 360 - 39000,00) \cdot \frac{30}{100} \cdot (1 - \frac{18}{100}) = 47970,00. \quad (7.10)$$

7.4 Расчет показателей экономической эффективности разработки и реализации программного средства на рынке

Оценка экономической эффективности разработки и реализации программного средства на рынке определяется простой нормой прибыли:

$$Р_{и} = \frac{\Delta П_{ч}^{р}}{З_{р}} \cdot 100\%, \quad (7.11)$$

где $\Delta П_{ч}^{р}$ – прирост чистой прибыли, полученной от реализации программного средства на рынке, руб;

$З_{р}$ – затраты на разработку программного средства, руб.

$$Р_{и} = \frac{47970,00}{44121,42} \cdot 100\% = 108,72\%. \quad (7.12)$$

ЗАКЛЮЧЕНИЕ

В ходе прохождения преддипломной практики была проведена усердная работа над разработкой тестового фреймворка. Были изучены различные подходы к тестированию, тема актуальности автоматизированного тестирования на сегодняшний день, особенности процессов по обеспечению качества.

Также при проектировании дипломного проекта был изучена тема behavior driven development подхода при написании автотестов. На сегодняшний день данная концепция становится все более популярной при проведении тестирования крупных проектов. Был изучен синтаксис и особенность языка gherkin на примере BDD-фреймворка Specflow.

По результатам прохождения преддипломной практики было спроектировано решение по автоматизации тестирование, позволяющее взаимодействовать как с API-функциональностью приложения, так и скрипты с имитацией действий реальных пользователей.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] cypress.io [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.cypress.io – Дата доступа: 09.04.2021.

[2] citrusframework.org [Электронный ресурс]. – Электронные данные. – Режим доступа: https://citrusframework.org – Дата доступа – 09.04.2021.

[3] qaevolution.ru [Электронный ресурс]. – Электронные данные. – Режим доступа: https://qaevolution.ru – Дата доступа 09.04.2021.

[4] Савин, Р. Тестирование DOT COM. – М. : Издательство «Дело», 2007. – 139 с.

[5] docs.microsoft.com [Электронный ресурс]. – Электронные данные. – Режим доступа: https://docs.microsoft.com/en-us/dotnet/core/testing/unit-testing-with-mstest – Дата доступа 09.04.2021.

[6] nunit.org [Электронный ресурс]. – Электронные данные. – Режим доступа: https://nunit.org – Дата доступа 09.04.2021.

[7] selenium.dev [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.selenium.dev/documentation/en/webdriver – Дата доступа 09.04.2021.

[8] kreisfahrer.gitbooks.io [Электронный ресурс]. – Электронные данные. – Режим доступа: https://kreisfahrer.gitbooks.io/selenium-webdriver – Дата доступа: 09.04.2021.

[9] restsharp.dev [Электронный ресурс]. – Электронные данные. – Режим доступа: https://restsharp.dev – Дата доступа: 09.04.2021.

[10] BDD in Action: Behavior-driven development for the whole software lifecycle/ J. Smart – New York, 2014.

[11] a1qa.by [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.a1qa.by – Дата доступа 09.04.2021.

[12] functionize.com [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.functionize.com/blog/ Дата доступа: 09.04.2021.

[13] specflow.org [Электронный ресурс]. – Электронные данные. – Режим доступа: https://specflow.org – Дата доступа: 09.04.2021.

[14] Continuous Integration: Improving Software Quality and Reducing Risk / St. M. Matyas III, A. Glover – Pearson Education, 2007.

[15] Что такое CI (Continuous Integration) [Электронный ресурс]. – Электронные данные. – Режим доступа: https://habr.com/ru/post/508216 – Дата доступа: 09.04.2021.

[16] jenkins.io [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.jenkins.io – Дата доступа: 09.04.2021.

[17] ranorex.com [Электронный ресурс]. – Электронные данные. – Режим доступа: https://ranorex.com – Дата доступа 09.04.2021.

ПРИЛОЖЕНИЕ А
(обязательное)
Вводный плакат

ПРИЛОЖЕНИЕ Б
(обязательное)
Схема структурная

ПРИЛОЖЕНИЕ В
(обязательное)
Диаграмма классов