УДК 004.05

РАЗРАБОТКА ФРЕЙМВОРКА ДЛЯ ПРОВЕДЕНИЯ АВТОМАТИЗИРОВАННОГО ТЕСТИРОВАНИЯ ФИНАНСОВОГО ВЕБ-ПРИЛОЖЕНИЯ.

Чжао М.Н., магистрант, МОиАИС БИТ, 1 курс
Семёнов В. А., магистрант, МОиАИС БИТ, 1 курс
Борисов И. Д., магистрант, МОиАИС БИТ, 1 курс
Бычкова Я. А., магистрант, МОиАИС БИТ, 1 курс
Балтийский федеральный университет имени Иммануила Канта
г. Калининград

В настоящее время информационные технологии востребованы во многих сферах нашей жизни, огромные бюджеты расходуются на разработку программных решений, востребованных в различных отраслях экономики. В свою очередь высокий спрос на программные продукты породил большое количество компаний-разработчиков. В подобных условиях высокой конкуренции, такая комплексная цель, как снижение стоимости разработки и повышение качества выпускаемого программного продукта является одной из наиболее актуальных в индустрии информационных технологий.

В связи с этим особо пристальное внимание уделяется процессу тестирования, а также методикам, помогающим минимизировать издержки данного процесса. Одной из таких методик является — автоматизация тестирования. Данная методика позволяет компаниям-разработчикам существенно сократить количество ресурсов и времени, затрачиваемых на тестирование, а также снизить риск выпуска на рынок продукта, содержащего дефекты, посредством использования различных программных решений для выполнения тестов, а также сбора информации о результатах их выполнения. В следствии этого технологии автоматизации тестирования получают все более и более широкое распространение. Это и определяет актуальность темы, выбранной мной для выпускной квалификационной работы.

Рынок финансового программного обеспечения является крайне перспективным и прибыльным, поскольку финансовый сектор является одним из крупнейших потребителей информационных технологий в мире, и в России, в частности. Сейчас, финансовые программные продукты, предоставляют широкий спектр функциональных возможностей, включая такие, как: учёт и контроль финансовой деятельности, электронная подпись документов, анализ средств предприятия, а также многие другие. Одним из таких финансовых продуктов является веб-приложение «Автоматизированный Центр Контроля Финансы, которое предназначено для управления общественными финансами и автоматизации процессов исполнения бюджета в субъектах и муниципальных образованиях Российской Федерации. [1] Одной из главных

задач приложения является – централизация финансовых процессов муниципального образования (региона). Эта задача решается за счёт сосредоточения первичной, отчётной, производной финансовой информации в финансовом органе субъекта Российской Федерации. Благодаря этому обеспечивается возможность контроля процесса исполнения бюджета со стороны финансового органа (ФО), главного распорядителя бюджетных средств (ГРБС) и распорядителя бюджетных средств (РБС).

При наличии такого значимого инструментария вопрос качества финансовых программных решений, является, не просто вопросом комфорта использования, потому как любой сбой в данных функциях может привести к серьёзным финансовым и репутационным потерям. Ввиду этого уровень покрытия тестами подобных программных продуктов должно быть высоким, а также тестирование должно проводиться для каждого обновления продукта. Именно для таких задач и подходит автоматизированное тестирование, поскольку оно позволяет, проводить написанные единожды тесты многократно и в сжатые сроки.

Автоматизированный подход к тестированию построен на написании тестовых сценариев на одном из языков программирования. Для организации запуска разработанных сценариев, управления их выполнением, а также для внедрения в процесс непрерывной интеграции и непрерывного развёртывания программного обеспечения разрабатывается фреймворк на выбранном языке программирования для автоматизированного тестирования продукта с использование различных библиотек автоматизации тестирования. Для построения фреймворка автоматизированного тестирования могут применяться различные шаблоны проектирования. Самыми распространёнными шаблонами являются: Page Object, Page Factory, Page Element. [2] При использовании данных паттернов увеличивается упрощается поддержка написанных автоматизированных тестов, а также сокращается время на разработку новых.

Первой задачей при реализации проекта является выбор языка программирования, на котором будет реализован фреймворк. Важно, чтобы язык программирования позволял осуществлять взаимодействие с инструментами для автоматизации тестирования. Более того, необходимо учесть тот факт, что в последующем на основе фреймворка будут разрабатываться новые автоматизированные тесты, а также будет осуществляться поддержка уже существующих. Основываясь на всех вышеперечисленных требованиях, в качестве основного языка программирования для разработки фреймворка был выбран язык Java.

После выбора основного языка программирования для фреймворка необходимо определиться с инструментарием автоматизации тестирования, доступными для данного языка. Основываясь на том, что тестируемая система веб-приложение, есть необходимость в автоматической генерации отчётов о выполнении автоматизированных тестов, а также учитывая факт того, что

данные тесты будут внедряться в CI/CD, то был сформирован следующий список инструментов для построения фреймворка:

- Система сборки проекта – для реализации возможности автоматизации сборки проекта. Для языка программирования Java существует целый ряд различных систем сборки проекта. Наибольшее распространение получили три следующих системы [3]: Apache Maven, Gradle, Apache Ant. Сборщик Apache Ant является менее предпочтительным, поскольку не имеет жизненного цикла, использует императивный подход, а также сценарии в Apache Ant не могут быть использованы повторно. В свою очередь сборщики Gradle и Apache Maven являются более гибкими системами и предоставляют широкие возможности, включая такие как: управление проектом, разрешение зависимостей, жизненный цикл проекта, а также возможность использования различных плагинов. Поэтому Gradle и Apache Maven, являются более предпочтительными к использованию в проекте. Выбирая между ними, стоит отметить, что Apache Maven обеспечивает простое и эффективное управление зависимостями, поддерживает широкий спектр этапов жизненного цикла сборки и что, крайне важно, легко интегрируется со сторонними инструментами, различными серверами непрерывной интеграции. Поэтому для реализации проекта была выбрана система сборки проекта Apache Maven;
- Фреймворк для создания отчётов – для автоматического формирования и визуализации результатов тестирования. Одним из таких программных продуктов для языка программирования Java, решающих задачу автоматического построения отчётов о проведении тестирования, является Allure Framework. Стоит отметить, что помимо Allure, также существует широко применяемый Java-фреймворк Thucydides [4], однако данный инструмент имеет ряд архитектурных недостатков по сравнению с Allure, включая такие недостатки как: монолитная архитектура, ориентированность исключительно на приёмочное тестирование веб-приложений. Поэтому для реализации проекта был выбран именно Allure Framework;
- Среда автоматизации тестирования – для реализации функций: формирования различных наборов тестов, а также для управления параллельным запуском автоматизированных тестов. Для языка программирования Java существует большое количество различных сред автоматизации тестирования. Самыми широко применяемыми из них являются системы TestNG и Junit [5]. Оба инструмента могут применяться для автоматизации тестирования на различных уровнях, от unit тестов до UI тестов. Выбирая между данными системами, важно отметить, что TestNG изначально разрабатывался для функционального тестирования и поддерживает большее количество

функций тестирования [5]. Поэтому для реализации проекта была выбрана именно библиотека TestNG;

- Драйвер браузера – для автоматизации действий в веб-браузере. С помощью данного инструмента фреймворк взаимодействует с тестируемым веб-приложением через браузер. В настоящее время рекомендованным веб-драйвером от консорциума всемирной паутины (World Wide Web Consortium) является Selenium WebDriver [6]. В связи с этим для проекта был выбран именно Selenium WebDriver.

При реализации проекта для автоматизированных тестов был выбран шаблон проектирования – Page Object Model (POM). Данный шаблон помогает инкапсулировать работу с отдельными элементами страницы, что позволяет уменьшить количество кода при написании автоматизированных тестов, а также упрощает поддержку проекта. Согласно POM для каждой страницы веб-приложения должен быть определён соответствующий класс. В этом классе будет находиться описание элементов страницы, а также методы для работы с описываемыми элементами. Названия методов должны соответствовать действиям, которые они выполняют. К основным преимуществам такого подхода можно отнести:

- В POM элементы объявляются отдельно от реализации теста;
- Независимость класса объектов от реализации теста позволяет использовать этот код в разных целях и с разными инструментами для выполнения тестов. Например, для функционального тестирования можно интегрировать POM с TestNG;
- Уменьшение объема кода за счёт повторного использование;
- Объединение всех действий по работе с веб-страницей в одном месте.

Для описания элементов страниц был выбран язык запросов к элементам XML-документов – Xpath.

В связи с выбранным шаблоном проектирования была сформирована следующая структура проекта (см. рисунок 2).

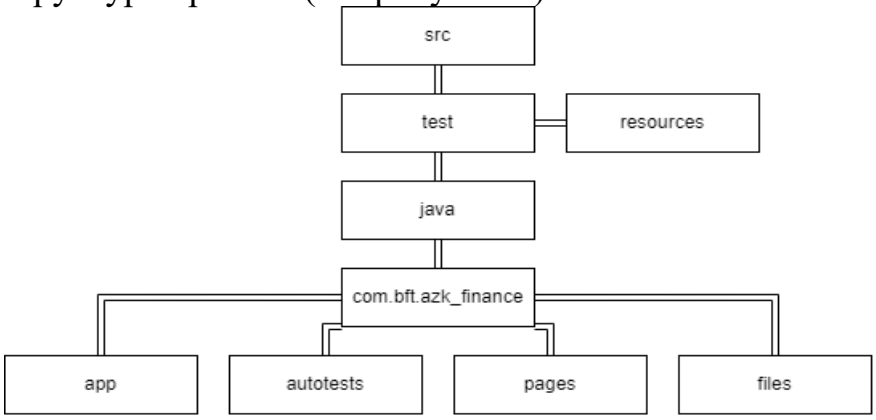

Рис. 2 Структура проекта

В разделе resources хранятся файлы свойств проекта: configuration.properties – файл свойств используемый в различных классах проекта, в котором содержится информация: для генерации паролей, о ссылках на различные версии веб-приложения и иные статичные данные требующиеся для проекта, а также в данном разделе содержится файл allure.properties – содержащий свойства Allure Framework.

Основной раздел проекта содержит в себе четыре подраздела:

- Раздел app – в данном разделе содержатся классы, отвечающие за: настройку и инициализацию WebDriver, управление данными для входа в приложение, создание скриншотов для отчётов, выполнение конфигураторов before и after описанных ранее;
- Раздел autotests – в данном разделе содержится код автоматизированных тестов;
- Раздел pages – в этом разделе содержится классы описывающие страницы веб-приложения. Каждый такой класс включает в себя описание различных элементов страницы, а также методы для выполнения различных действий с ними;
- Раздел files – содержит в себе различные файлы, например, если при выполнении проверки необходимо прикрепить определённый, заранее подготовленный файл, то он должен быть расположен в данной директории.

Автоматизированный тест представляет из себя публичный метод данного класса, не возвращающий какого-либо значения. Для каждого теста, используется аннотация из TestNG - @Test [7], которая обозначает для системы автоматизации тестирования, что данный метод является тестом, также у этой аннотации используется параметр description, необходимый для формирования наименования теста в отчёте.

Первым тестовым сценарием для реализации автоматизированного теста был выбран сценарий проверяющий поведение тестируемого веб-приложения при попытке входа в систему с неверно указанными данными пользователя. Сценарий представлен в таблице 1.

Табл. 1 Сценарий «Выполнение логина с неверными данными»

Сценарий:	Выполнение логина с неверными данными	
Шаг сценария	Данные для ввода	Ожидаемый результат
Перейти на страницу входа в веб-приложение	-	Страница входа в веб-приложение загрузилась
Ввести в поле ввода логина значение	User#4#*@#*97$01	Указанные данные введены в поле логина
Ввести в поле ввода пароля значение	&*#(@#&(!@)&!@&	Указанные данные введены в поле пароля
Нажать кнопку войти	-	Появилось предупреждающее сообщение

		«Аутентификация не пройдена!»
Закрыть предупреждение	-	Предупреждение скрыто

Данный сценарий был реализован в виде метода negativeTestLoginForm, представленный на рисунке 3.

```
@Test(description = "Выполнение логина с неверными данными")
public void negativeTestLoginForm(){
    startPage().clickButtonQuit();
    loginPage().fillLogin( user: "User#4#*@#*97$01");
    loginPage().fillPassword("&*#(@#&(!@)&!@&");
    loginPage().clickLoginButton();
    loginPage().checkTextIsVisible( expectedText: "Аутентификация не пройдена!");
    loginPage().checkTextIsVisible( expectedText: "Неверное имя пользователя или пароль.");
    loginPage().checkTextIsVisible( expectedText: "Проверьте правильность написания имени пользователя.");
    loginPage().checkTextIsVisible( expectedText: "Проверьте соответствие языка ввода для пароля.");
    loginPage().checkTextIsVisible( expectedText: "Посмотрите, не нажат ли [Caps Lock].");
    loginPage().clickCommonSpanButton( buttonName: "Закрыть");
    loginPage().checkTextIsNotVisible( expectedText: "Аутентификация не пройдена!");
    loginPage().checkTextIsNotVisible( expectedText: "Неверное имя пользователя или пароль.");
    loginPage().checkTextIsNotVisible( expectedText: "Проверьте правильность написания имени пользователя.");
    loginPage().checkTextIsNotVisible( expectedText: "Проверьте соответствие языка ввода для пароля.");
    loginPage().checkTextIsNotVisible( expectedText: "Посмотрите, не нажат ли [Caps Lock].");
    loginPage().clickCommonSpanButton( buttonName: "Закрыть");
}
```

Рис. 3 Метод negativeTestLoginForm

В результате работы был разработан фреймворк для автоматизированного тестирования финансового веб-приложения «АЦК-Финансы» на языке программирования Java с применением шаблона проектирования Page Object Model (POM).

Список литературы:

1. ООО «Бюджетные и Финансовые Технологии Холдинг». АЦК-Финансы/ АСУ БП АЦК-Финансы. – URL: https://bftcom.com/products/upravlenie-gosudarstvennymi-finansami/ispolnenie-byudzheta/. Текст: электронный.
2. Хабр, Паттерны проектирования в автоматизации тестирования – URL: https://habr.com/ru/company/jugru/blog/338836/. Текст: электронный.
3. Baeldung, Ant vs Maven vs Gradle – URL: https://www.baeldung.com/ant-maven-gradle. Текст: электронный.
4. Хабр, Allure — фреймворк от Яндекса для создания простых и понятных отчётов автотестов [для любого языка] – URL: https://habr.com/ru/company/yandex/blog/232697/. Текст: электронный.
5. Baeldung, A Quick JUnit vs TestNG Comparison – URL: https://www.baeldung.com/junit-vs-testng. Текст: электронный.
6. Selenium Documentation, WebDriver – URL: https://www.selenium.dev/documentation/webdriver/. Текст: электронный.
7. TestNG Documentation, 5 - Test methods, Test classes and Test groups – URL: https://testng.org/doc/documentation-main.html. Текст: электронный.