МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт математики, физики и информационных технологий
(наименование института полностью)

Кафедра «Прикладная математика и информатика»
(наименование кафедры)

02.03.03 Математическое обеспечение и администрирование информационных систем
(код и наименование направления подготовки, специальности)

Технология программирования
(направленность (профиль)/специализация)

БАКАЛАВРСКАЯ РАБОТА

на тему **Разработка автоматизированных тестов для веб-приложения с использованием библиотеки Selenium WebDriver**

Студент А.С. Мишушина _____
(И.О. Фамилия) (личная подпись)

Руководитель С.В. Мкртычев _____
(И.О. Фамилия) (личная подпись)

Консультант К.А. Селиверстова _____
(И.О. Фамилия) (личная подпись)

Допустить к защите

Заведующий кафедрой к.тех.н., доцент, А.В. Очеповский
(ученая степень, звание, И.О. Фамилия)

_____
(личная подпись)

«_____» _____ 2019 г.

Тольятти 2019

АННОТАЦИЯ

Тема: «Разработка автоматизированных тестов для веб-приложения с использованием библиотеки Selenium WebDriver».

Целью выпускной квалификационной работы является разработка автоматизированных тестов для веб-приложения с использованием библиотеки Selenium WebDriver.

Объектом исследования бакалаврской работы является процесс тестирования веб-приложений.

Предмет исследования – автоматизация тестирования веб-приложения.

Методы исследования – методы автоматизированного тестирования веб-приложений, методы разработки автоматизированных тестов.

Практическая значимость бакалаврской работы заключается в разработке комплекса автоматизированных тестов, обеспечивающих повышение эффективности тестирования веб-приложения.

Структура ВКР представлена введением, тремя главами, заключением и списком литературы.

Проанализированы виды и методы тестирования веб-приложений, рассмотрены готовые решения автоматизации и инструменты создания собственных тестов.

Описаны функциональные и архитектурные особенности Selenium WebDriver.

С использованием данной библиотеки на языке программирования Java разработаны автоматизированные тесты для веб-приложения, и оценена их эффективность.

Данная бакалаврская работа состоит из пояснительной записки на 45 страниц, включая 17 рисунков, 8 таблиц, 20 источников и 2 приложения.

ABSTRACT

The topic of the given graduation work is Development of automated tests for a web-application using Selenium WebDriver.

The aim of the work is to develop automated tests for a web-application using Selenium WebDriver. The object is a process of web-application testing. The subject of the graduation project is automation of web-application testing.

Firstly, we analyse types and methods of web-application testing, consider solutions to automate this process and tools to realize own tests. In the second part, we describe functional and architecture features of Selenium WebDriver: main methods of the library and the components, which are a reason of the tool effectiveness.

The special part of the project gives details about the development of automated tests for a web-application. We design a use-case diagram to define the ways users operate with the application and test cases based on the diagram to form a testing procedure correctly. The automated tests are created using Java programming language for Chrome browser. The software contains GUI additional functions for usability. We also prove the effectiveness of the developed tests with the results of web-application testing.

In conclusion, we have an effectual software, which makes a process of web-application testing faster and easier and minimizes the influence of human factor on it.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ 5

Глава 1 АНАЛИЗ ВИДОВ И МЕТОДОВ ТЕСТИРОВАНИЯ ВЕБ-ПРИЛОЖЕНИЙ 8

1.1 Анализ видов тестирования 8

1.2 Анализ методов тестирования 11

Глава 2 ФУНКЦИОНАЛЬНЫЕ И АРХИТЕКТУРНЫЕ ОСОБЕННОСТИ БИБЛИОТЕКИ SELENIUM WEBDRIVER 17

2.1 Функциональные особенности WebDriver 17

2.2 Архитектурные особенности WebDriver 23

Глава 3 РЕАЛИЗАЦИЯ АВТОМАТИЗИРОВАННЫХ ТЕСТОВ ДЛЯ ВЕБ-ПРИЛОЖЕНИЯ 26

3.1 Разработка диаграммы вариантов использования и тестовых сценариев .. 26

3.2 Разработка автоматизированных тестов 29

3.3 Оценка эффективности автоматизированных тестов 41

ЗАКЛЮЧЕНИЕ 44

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ 45

ПРИЛОЖЕНИЕ А Главный класс приложения 47

ПРИЛОЖЕНИЕ Б Класс, реализующий UI 54

ВВЕДЕНИЕ

Неправильная работа приложения – часто встречающаяся ситуация, причина которой очень проста – все приложения создаются и используются людьми.

Влияние человеческого фактора на программное обеспечение присутствует и на стадии разработки – некорректно составленная документация, неправильная трактовка требований, невнимательность во время реализации – все это приводит к ошибкам в работе продукта.

По причине этого разработка программного обеспечения не обходится без этапа тестирования.

Тестирование – это процесс, во время выполнения которого проводится сравнение между реальным поведением программы и ее ожидаемым результатом, выявляются дефекты и определяется соответствие заявленным требованиям.

Целью тестирования являются повышение качества программного обеспечения, предотвращение появления дефектов, а также предоставление информации о его качестве.

Минимизировать влияние человеческого фактора и ускорить процесс тестирования помогают программные средства автоматизации. Автоматизированные тесты широко используются для тестирования веб-приложений. Для их разработки применяются различные инструменты, один из них – Selenium.

Selenium – это мощный набор инструментов для автоматизации функционального тестирования веб-приложений, эффективен в силу отличной имитации действий обычного пользователя при работе в браузере.

Одним из инструментов данного набора является Selenium WebDriver – программная библиотека для различных языков программирования, позволяющая взаимодействовать с браузером и управлять его поведением.

Таким образом, представляет **актуальность** разработка автоматизированных тестов для веб-приложения с использованием библиотеки Selenium WebDriver.

Объектом исследования бакалаврской работы является процесс тестирования веб-приложения.

Предмет исследования – автоматизация тестирования веб-приложения.

Цель выпускной квалификационной работы – разработка автоматизированных тестов для веб-приложения с использованием библиотеки Selenium WebDriver.

Для достижения данной цели необходимо выполнить следующие задачи:

- проанализировать виды и методы тестирования веб-приложений;
- описать функциональные и архитектурные особенности библиотеки Selenium WebDriver;
- разработать комплекс автоматизированных тестов для веб-приложения с использованием библиотеки Selenium WebDriver;
- подтвердить эффективность разработанных тестов в процессе тестирования веб-приложения.

Методы исследования – методы автоматизированного тестирования веб-приложений, методы разработки автоматизированных тестов.

Практическая значимость бакалаврской работы заключается в разработке комплекса автоматизированных тестов, обеспечивающих повышение эффективности тестирования веб-приложения.

Данная работа состоит из введения, трех глав, заключения, списка используемой литературы и приложений.

Первая глава содержит анализ видов и методов тестирования веб-приложений.

Во второй главе описаны функциональные и архитектурные особенности библиотеки Selenium WebDriver.

Третья глава посвящена разработке автоматизированных тестов для веб-приложения с использованием библиотеки Selenium WebDriver и оценке их эффективности.

В заключении описываются результаты выполнения выпускной квалификационной работы.

Приложения содержат фрагменты программного кода реализации автоматизированных тестов для веб-приложения.

Глава 1 АНАЛИЗ ВИДОВ И МЕТОДОВ ТЕСТИРОВАНИЯ ВЕБ-ПРИЛОЖЕНИЙ

1.1 Анализ видов тестирования

Тестирование веб-приложений – методика тестирования программного обеспечения, предназначенная исключительно для приложений, размещенных в сети Интернет.

Веб-тестирование включает в себя несколько видов:

- функциональное тестирование;
- тестирование удобства использования;
- тестирование интерфейса;
- тестирование совместимости;
- тестирование производительности;
- тестирование безопасности.

Функциональное тестирование (англ. Functional testing) является одним из ключевых видов и направлено на проверку работы основного функционала программного обеспечения в соответствии с заданными требованиями.

Данный вид тестирования включает проверку ссылок, форм, cookie-файлов, HTML и CSS.

Существует несколько видов ссылок: внутренние; внешние; ссылки на адреса электронной почты; ссылки, встроенные в веб-элементы. Каждый из них необходимо проверять на работоспособность и перенаправление на верную страницу.

Формы должны функционировать в соответствии с их предназначением, иметь корректные значения по умолчанию и правильно работающие валидации.

При тестировании cookie-файлов нужно убедиться, что они шифруются перед записью на устройство пользователя. Для cookie-файлов определенной сессии проверяются данные для входа в систему и статистика пользователя после завершения работы.

Оптимизация сайта для поисковых систем требует валидации HTML и CSS, а именно – проверки синтаксиса на наличие ошибок.

Тестирование удобства использования (англ. Usability testing) проводится для определения того, хорошо ли приложение адаптировано для эксплуатации человеком – измеряется эргономичность.

При оценке учитываются следующие показатели:

- легкость использования;
- удобство навигации;
- удовлетворенность пользователя;
- общий внешний вид.

Тестирование интерфейса (англ. Interface testing) направлено на проверку трех составляющих – приложения, веб-сервера и сервера базы данных.

Для приложения проверяется корректность отправки запросов в базу данных и отображения на стороне клиента. Тестирование веб-сервера подразумевает проверку обработки всех запросов от приложения без отказа в обслуживании. Сервер базы данных должен возвращать результаты, соответствующие запросу.

Также стоит проверять реакцию системы в случае отсутствия соединение между всеми составляющими и отображение соответствующего сообщения конечному пользователю.

Тестирование совместимости (англ. Compatibility testing). Основной целью данного вида является проверка совместимости приложения с браузером, операционной системой, а также мобильного просмотра и опций печати.

Совместимость с браузером является важной частью, т.к. некоторые приложения очень зависимы от них. Каждый браузер имеет отличные от других конфигурации и настройки, с которыми веб-страница должна быть совместима.

Некоторый функционал веб-приложения может поддерживаться не всеми операционными системами по причине того, что технологии, используемые в веб-разработке – различные API, графический дизайн, могут быть не доступны. В

следствие этого, целесообразнее тестировать приложение на разных операционных системах.

В связи с ростом используемости портативных устройств присутствует необходимость проверки отображения веб-приложения в мобильных браузерах.

Если приложение поддерживает функцию печати веб-страниц, то нужно проверять корректность выравнивания страниц, шрифтов и т.п. при печати. Страницы должны соответствовать размеру бумаги или размеру, указанному в параметрах.

Тестирование производительности (англ. Performance testing) проводится с целью исследования показателей скорости реакции приложения на внешние воздействия при различной нагрузке. Выделяют несколько подвидов:

− нагрузочное тестирование (англ. Load testing) – представляет собой тестирование системы на сохранность заданных показателей при превышении пределов нагрузки;

− стрессовое тестирование (англ. Stress testing) – позволяет проверить, насколько система работоспособна в условиях нештатных изменений нагрузки или ситуациях, в которых блокируется часть необходимых ресурсов;

− объемное тестирование (англ. Volume testing) – его задачей является получение оценки производительности при увеличении объемов данных.

Тестирование безопасности (англ. Security testing) представляет важность для приложений электронной коммерции, хранящих конфиденциальную информацию.

Для тестирования безопасности необходимо проверить следующее:

− возможность несанкционированного доступа к защищенным страницам;

− загрузку файлов с ограниченным доступом без соответствующих прав;

− автоматическое разъединение после длительного отсутствия пользователя;

− перенаправление на зашифрованные SSL-страницы при использовании SSL-сертификатов.

Основной причиной тестирования безопасности в сети является выявление потенциальных уязвимостей и последующее их устранение.

В зависимости от сложности веб-приложения определяются виды тестирования, которые следует затронуть при проверке и оценке его качества.

Вышеописанные виды тестирования могут проводиться различными методами.

1.2 Анализ методов тестирования

Методов тестирования, в отличии от видов, значительно меньше, их можно разделить на две категории:

- по доступ к коду;
- по необходимости автоматизации.

В зависимости от доступа к программному коду производится разделение на три типа – метод белого ящика, метод черного ящика и метод серого ящика.

Тестирование методом белого ящика (англ. White-box testing) учитывает внутреннее строение приложения и внутреннее функционирование, а также логику работы кода. Для проведения тестирования данным методом необходимо обладать достаточным количество соответствующих знаний о приложении.

Тестирование методом черного ящика (англ. Black-box testing) производится без знаний и доступа к структуре приложения. Большинство видов тестирования работают по данному методу, т.к. над системой производятся действия, аналогичные пользовательским.

Тестирование методом серого ящика (англ. Gray-box testing) совмещает в себе два вышеописанных метода и осуществляется с обладанием частичных знаний о приложении, также отсутствует необходимость в доступе к исходному коду.

Методы, связанные с автоматизацией, можно разделить на два типа – ручное и автоматизированное тестирование.

Ручное тестирование (англ. Manual testing) эффективно с точки зрения нахождения ошибок и способствует повышению надежности приложения.

Особенно результативно при проведении дымового тестирования и тестирования удобства использования.

Но данный метод также обладает своими недостатками: при рутинной работе и регулярной проверки одного и того же программного обеспечения внимательность к деталям у инженеров по тестированию снижается, и существует вероятность пропуска ошибки.

Для снижения влияния человеческого фактора на процесс тестирования выполняется его автоматизация.

Автоматизированное тестирование (англ. Test Automation) – проверка программного обеспечения с использованием специальных средств. Эффективно на больших объемах тестовых данных и при необходимости проведения часто повторяющихся тестов.

Автоматизация позволяет сократить временны́е и человеческие ресурсы. Подразумевает написание программного кода – автоматизированных тестов.

Распространенная форма автоматизации – тестирование веб-приложений через графический пользовательский интерфейс. Обусловлено это тем, что используются те же способы, что и человеком при работе с приложением.

Проводить автоматизированное тестирование возможно при использовании готовых инструментов, например, Ixia, SoapUI, Jmeter, HP LoadRunner.

Тестовые решения Ixia эмулируют рабочие характеристики сети, трафик разного рода атак и мультимедийный трафик, давая возможность проверять архитектуру, информационную безопасность и производительность [12].

SoapUI используется для проведения функционального и нагрузочного тестирования путем моделирования необходимых служб и совершения вызовов HTTP и JDBC [19].

Нагрузочное тестирование также может выполняться при помощи инструмента Jmeter. Тесты применяются для JDBC-соединений, JMS, HTTP, FTP, LDAP, SOAP, POP3, IMAP, TCP. С помощью нескольких компьютеров имеется возможность создания большого количества запросов, при этом управляя процессом с одного из них [7].

При помощи HP LoadRunner возможно выполнять тестирование приложений и сайтов различного уровня сложности, эмулировать параллельную работу большого количества пользователей, выполняющих различные действия, и вести контроль системных ресурсов [13].

Но в силу того, что некоторые веб-приложения обладают специфическими особенностями – как функциональными, так и архитектурными, данные средства не всегда действенны при проведении автоматизированного тестирования, т.к. не имеют возможности их учитывать.

В таких случаях разработчикам программного обеспечения целесообразно разрабатывать собственные автоматизированные тесты, беря во внимание своеобразие приложения.

Для решения данной задачи на ИТ-рынке предлагаются специализированные инструменты для разработки автоматизированных тестов:

- Selenium;
- Katalon Studio;
- UFT;
- TestComplete;
- WATIR.

Selenium является самым популярным фреймворком с открытым исходным кодом, предназначенным для автоматизации тестирования веб-приложений. Востребован у инженеров по тестированию, обладающих продвинутыми навыками программирования и опытом написания скриптов.

Поддерживается разными операционными системами – Mac OS, Linux, Windows – и браузерами – Firefox, Chrome, Internet Explorer, Opera, Safari. Тестовые сценарии могут быть написаны на различных языках программирования – Java, C#, Perl, Python, JavaScript, Ruby, PHP.

Включает в себя несколько инструментов, обладающих собственным функционалом для автоматизации, – Selenium IDE, Selenium WebDriver, Selenium RC и Selenium Server [17].

Katalon Studio – эффективный инструмент для автоматизации процесса тестирования веб-приложений, мобильных приложений и веб-сервисов. Не требует углубленных знаний в программировании, поддерживает языки Java и Groovy.

Работает с операционными системами – Windows, Mac OS и Linux – и основными браузерами Может быть интегрирован в CI/CD, обладает функцией, благодаря которой пользователи получают полное представление о ходе тестирования [14].

UFT (Unified Functional Testing) – это популярный инструмент для функционального тестирования. Обладает набором функций для тестирования API, веб-сервисов, тестирования графического интерфейса десктопных, мобильных и веб-приложений.

Не является кроссплатформенным, работает только под операционной системой Windows. Использует Visual Basic Scripting Edition, поддерживает CI [14].

TestComplete является эффективным инструментом для тестирования десктопных, мобильных и веб-приложений. Поддерживает сценарии на таких языках программирования, как: JavaScript, VBScript, Python и C ++.

Как и в случае с UFT, поддерживается одной операционной системой – Windows – и большинством современных браузеров. Обладает функцией распознавания объектов графического пользовательского инструмента [14].

WATIR – инструмент с открытым исходным кодом для автоматизации тестирования веб-приложений, использующий библиотеки Ruby.

Предусмотрена возможность кроссплатформенного – на Windows, Mac OS, Linux – и кросс-браузерного тестирования в большинстве существующих браузеров: Firefox, Opera и Internet Explorer [14].

На основании описания каждого инструмента создана сравнительная таблица для определения наиболее эффективного из них, оценка производится по трехбалльной шкале (таблица 1.1).

Таблица 1.1 – Сравнение инструментов автоматизации

Инструмент	Показатель			Оценка
	Кросс-платформенность	Кросс-браузерность	Языки программирования	
Selenium	3	3	3	9
Katalon Studio	3	3	1	7
UFT	1	3	1	5
TestComplete	1	3	2	6
Watir	3	3	1	7

По результатам сравнения можно сделать вывод о преобладании Selenium WebDriver над другими инструментами благодаря поддержки большого количества операционных систем, браузеров и языков программирования.

Таким образом, в зависимости от доступа к коду приложения и необходимости автоматизации тестирование может быть проведено различными методами.

В случае применения автоматизации в процессе тестирования выбираются специализированные средства, соответствующие специфике приложения.

Выводы по главе 1

В первой главе проведен анализ видов и методов тестирования веб-приложений.

1. Описаны основные виды тестирования – функциональное, удобства использования, интерфейса, совместимости, производительности, безопасности – и методы их проведения – белого ящика, черного ящика, серого ящика, ручного и автоматизированного тестирования.

2. Рассмотрены готовые решения для проведения автоматизированного тестирования.

3. Приведены примеры программных средств автоматизации с описанием, на основании которого можно сделать вывод о преобладании среди них инструмента Selenium благодаря его кроссплатформенности, кросс-браузерности и поддержки большого количества языков программирования.

4. Данные преимущества Selenium дают возможность для написания универсальных автоматизированных тестов, учитывая особенности веб-приложения.

Глава 2 ФУНКЦИОНАЛЬНЫЕ И АРХИТЕКТУРНЫЕ ОСОБЕННОСТИ БИБЛИОТЕКИ SELENIUM WEBDRIVER

2.1 Функциональные особенности WebDriver

Selenium WebDriver является одним из мощнейших и популярнейших инструментов из набора Selenium для проведения автоматизации процесса тестирования. Был разработан как расширенная версия Selenium RC, переняв его преимущества и исключив недостатки.

WebDriver обеспечивает широкий набор функций тестирования для веб-приложений всех типов. Эти операции очень гибкие и позволяют использовать множество опций для определения местоположения элементов пользовательского интерфейса и сравнения ожидаемых результатов тестирования с фактическим поведением приложения.

Отличительные особенности Selenium WebDriver включают в себя:

- поддержку различных языков программирования;
- поддержку многих современных браузеров и операционных систем;
- отсутствие в необходимости запуска Selenium Server;
- нахождение координат любого объекта;
- построение структур на основе ключевых слов;
- использование техник автоматизации, свойственных человеку;
- поддержка тестирования iOS и Android систем.

Основными компонентами для работы с WebDriver являются браузер, драйвер и тест.

В силу того, что WebDriver поддерживает большое количество браузеров, может быть использован любой из них для проведения кросс-браузерного тестирования. Таблица 2.1 содержит список наиболее популярных браузеров и их версий на соответствующих операционных системах.

Таблица 2.1 – Версии браузеров, поддерживаемые Selenium WebDriver

Microsoft Windows	Mac OS X	Linux
Firefox 2, 3, 3.x , 4 — 54	Safari 2, 3, 4	Firefox 2, 3, 3.x
Internet Explorer 6, 7, 8 , 9, 10	Firefox 2, 3, 3.x	Mozilla Suite 1.6+, 1.7+
Safari 2, 3, 4	Camino 1.0a1	Konqueror
Safari 2, 3, 4	Mozilla Suite 1.6+, 1.7+	Opera 8, 9, 10
Google Chrome 12.0.712.0+	Seamonkey 1.0	Google Chrome 12.0.712.0+

Драйвер используется для связи с браузером. Поскольку внутренняя логика браузера и его функциональные возможности не раскрыты, драйвер обеспечивает необходимый уровень «инкапсуляции», чтобы сохранить детализацию уровня выполнения более абстрактной. Для каждого браузера предусмотрен соответствующий драйвер, т.к. команды управления отличаются и реализованы по-разному (рисунок 2.1).

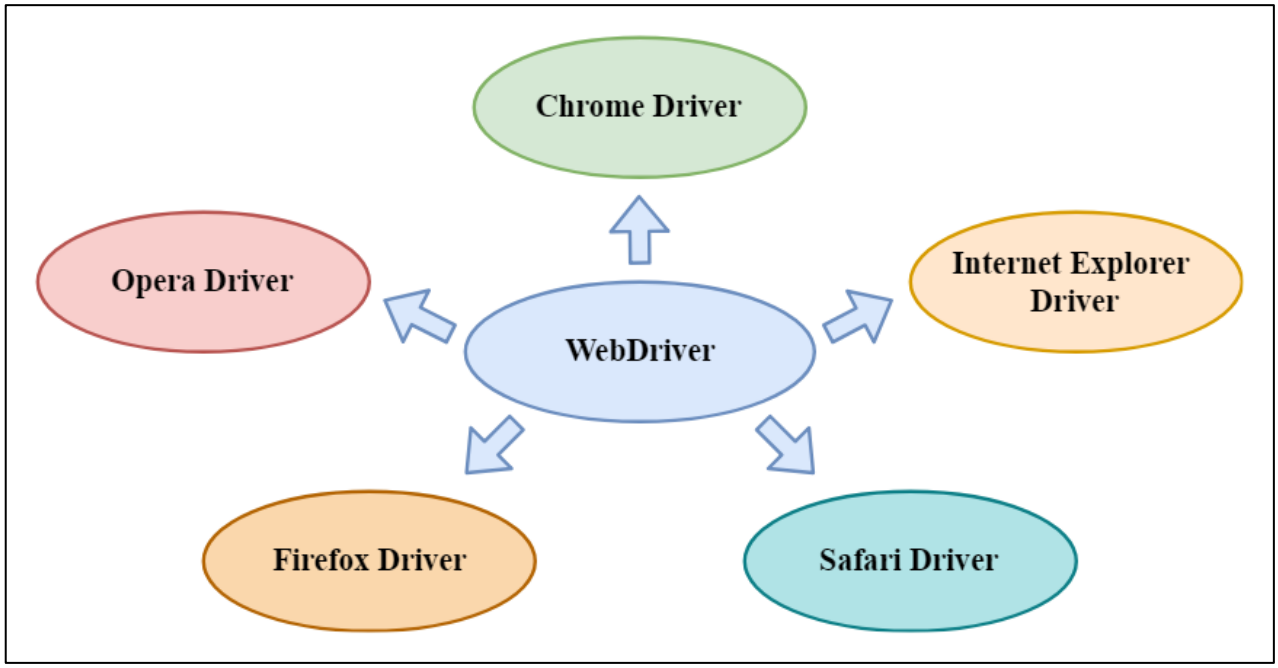

Рисунок 2.1 – Веб-драйвера популярных браузеров

Selenium WebDriver позволяет разработчику выбирать различные языки программирования и создавать свои тестовые сценарии независимо от того, на каком языке написано само приложение. Таким образом, в зависимости от опыта и навыков, может быть выбран один из следующих языков:

- Java;
- C#;
- Ruby;
- Pearl;
- Python;
- Javascript.

Основными элементами при написании тестовых сценариев с использованием библиотеки Selenium WebDriver являются:

– WebDriver – главный интерфейс, используемый для проведения тестирования. Классы, имплементирующие данный интерфейс, позволяют создавать объекты для управления браузером;

– WebElement – интерфейс, предоставляющий механизм для принятия HTML-элементов как объектов и выполнения над ними требуемых действий.

Написание автоматизированных тестов начинается с инициализации соответствующего драйвера путем создания объекта определенного класса (ChromeDriver, EdgeDriver, FirfoxDriver и т.п.). Далее вызываются необходимые для работы методы, которые можно разделить на три основные категории:

- контроль браузера;
- взаимодействие с веб-элементами.
- дополнительные функции.

Команды, отвечающие за контроль браузера, выполняют функции для загрузки новой страницы, получения ее адреса, заголовка, завершения работы и т.п.

Полный их перечень представлен в таблице 2.2 [18].

Таблица 2.2 – Методы контроля браузера

Название метода	Пример	Цель
get()	driver.get(url)	Загрузка новой страницы в текущем окне браузера
getCurrentUrl()	driver.getCurrentUrl()	Получение URL текущей страницы
getTitle()	driver.getTitle()	Получение заголовка текущей страницы
getPageSource()	driver.getPageSource()	Получение исходного текста текущей страницы
close()	driver.close()	Закрытие текущего окна. Если оно единственное – закрытие браузера
quit()	driver.quit()	Закрытие всех окон и браузера

После открытия требуемой веб-страницы выполняется работа с ее элементами.

Selenium Webdriver представляет каждый HTML-элемент как WebElement, и любое взаимодействие с экземпляром начинается с его поиска на странице – осуществляется это посредством вызова метода findElement(By by).

В качества параметра данному методу передается механизм By, определяющий местоположение элемента. Объект класса By может использоваться с различными стратегиями локатора, такими как:

- ID;
- Name;
- Class Name;
- Tag Name;
- Link Text;
- Partial Link Text;

– XPATH.

Значение локатора является уникальным, и с его помощью можно идентифицировать веб-элемент. Разработчики веб-приложений обязаны обеспечить уникальную идентификацию элементов с использованием определенных свойств.

После нахождения объекта над ним можно производить различного рода действия путем вызова соответствующих методов, представленных в таблице 2.3 [18].

Таблица 2.3 – Методы для работы с веб-элементами

Название метода	Пример	Цель
click()	driver.findElement(By by).click()	Нажатие на элемент
clear()	driver.findElement(By by).clear()	Если искомый элемент для ввода текста – очищение его значения
isDisplayed()	driver.findElement(By by).isDisplayed()	Проверка отображения элемента
isEnabled()	driver.findElement(By by).isEnabled()	Проверка доступности элемента
isSelected()	driver.findElement(By by).isSelected()	Проверка, выбран ли текущий элемент
sendKeys()	driver.findElement(By by).sendKeys()	Заполнение текстового поля
getAttribute()	driver.findElement(By by).getAttribute ()	Получение значения атрибута
getSize()	driver.findElement(By by).getSize()	Получение ширины и высоты элемента
getTagName()	driver.findElement(By by).getTagName()	Получение имени тега

Название метода	Пример	Цель
getText()	driver.findElement(By by).getText()	Получение внутреннего текста
getLocation()	driver.findElement(By by).getLocation()	Получение расположения верхнего левого угла элемента
getCssValue()	driver.findElement(By by).getCssValue()	Получение CSS-значения

Вызов любого из методов выполняет проверку соединения экземпляра с DOM. Если данный тест не пройден, генерируется исключение StaleElementReferenceException, и все будущие вызовы этого экземпляра будут неудачными.

Работа с браузером не ограничивается веб-страницами и их элементами, пользователи также могут совершать другие, менее значимые, действия, но библиотека Selenium WebDriver предусматривает это – в ней определены методы, выполняющие второстепенные функции.

К ним можно отнести работу с навигационной панелью, cookie-файлами, режимами ожидания, окном браузера и т.д.

В таблице 2.4 приведены наиболее часто используемые методы для выполнения побочных действий [18].

Таблица 2.4 – Вспомогательные методы Selenium WebDriver

Название метода	Пример	Цель
refresh()	driver.navigate().refresh()	Обновление страницы
forward()	driver.navigate().forward()	Перенос на одну страницу вперед, исходя из истории браузера

Название метода	Пример	Цель
back()	driver.navigate().back()	Перенос на одну страницу назад, исходя из истории браузера
to()	driver.navigate().to(url)	Загрузка новой страницы
deleteAllCookies()	driver().manage().deleteAllCookies()	Удаление всех cookie-файлов для текущего домена
getAllCookies()	driver().manage().getAllCookies()	Получение всех cookie-файлов для текущего домена
maximize()	driver().manage().window().maximize()	Открытие текущего окна браузера на весь экран
getSize()	driver().manage().window().getSize()	Получение размера текущего окна браузера
implicitlyWait()	driver.manage().timeouts().implicitlyWait (10, TimeUnit.SECONDS);	Определение времени, выжидаемого драйвером во время поиска элемента
pageLoadTimeout()	driver.manage().timeouts().pageLoadTim eout(10, TimeUnit.SECONDS);	Определение времени ожидания загрузки страниц. По его истечению выдается ошибка

Таким образом, Selenium WebDriver предоставляет широкий спектр возможностей для написания собственных автоматизированных тестов на большинстве популярных языков программирования без привязки к определенным операционным системам или браузерам.

2.2 Архитектурные особенности WebDriver

Selenium WebDriver предоставляет средства связи между языками программирования и браузерами.

Его архитектура состоит из четырех основных компонентов (рисунок 2.2):

1. Selenium WebDriver API;

2. JSON Wire Protocol;
3. драйвера браузеров;
4. браузеры.

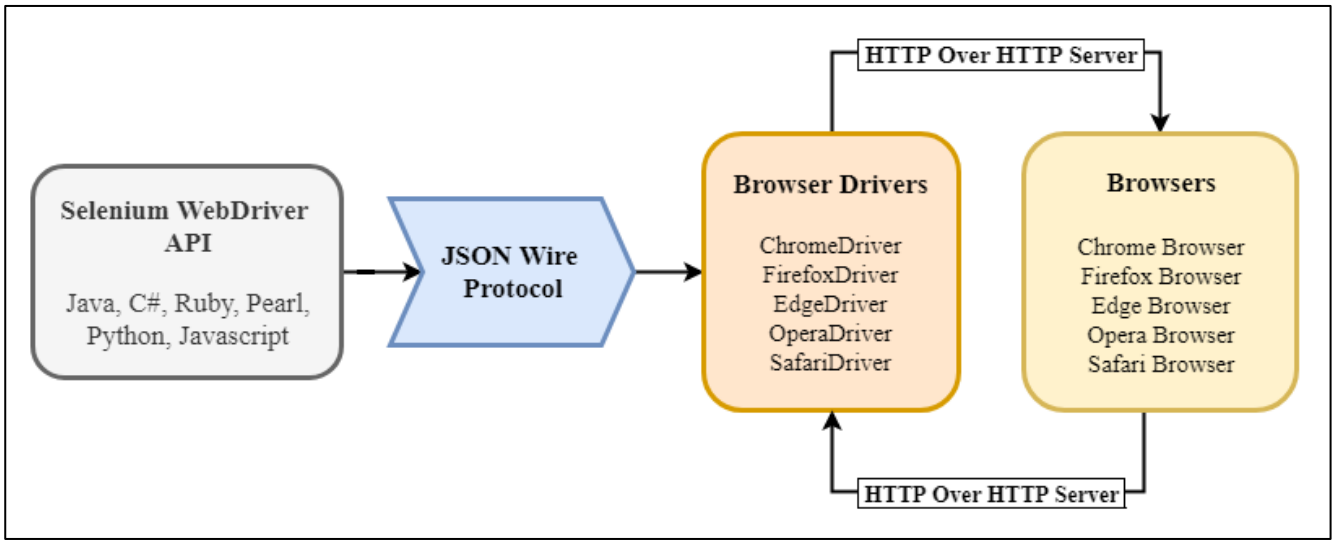

Рисунок 2.2 – Архитектура Selenium WebDriver

Разработчики Selenium создали WebDriver API для поддержки нескольких языков программирования. В случае написания тестов на Java, используются библиотеки, разработанные под данный язык. Каждый из API находится в открытом доступе.

JSON (JavaScript Object Notation) – это открытый стандарт для обмена данными в сети. Он поддерживает структуры данных, такие как объект и массив. Таким образом, легко писать и читать данные из JSON.

Протокол JSON Wire предоставляет транспортный механизм для передачи данных между сервером и клиентом и служит отраслевым стандартом для различных веб-сервисов REST.

Selenium WebDriver использует индивидуальные драйверы для каждого браузера, чтобы установить с ним безопасное соединение, не раскрывая внутреннюю логику функциональности. Драйвер браузера также зависит от языка, используемого для работы.

При выполнении тестового скрипта выполняются следующие операции:

1. HTTP-запрос генерируется и отправляется драйверу браузера для каждой команды;

2. драйвер получает HTTP-запрос через HTTP-сервер;

3. HTTP-сервер решает все шаги для выполнения инструкций, которые выполняются в браузере;

4. статус выполнения отправляется обратно на HTTP-сервер, который затем отправляется обратно в скрипт автоматизации.

С помощью вышеописанных компонентов Selenium WebDriver предоставляет возможность написания эффективных автоматизированных тестов.

Выводы по главе 2

Во второй главе были описаны функциональные и архитектурные особенности библиотеки Selenium WebDriver.

1. Рассмотрены основные составляющие работы с данной библиотекой.

2. Описаны наиболее важные методы для написания тестовых сценариев.

3. Представлены компоненты архитектуры WebDriver.

Глава 3 РЕАЛИЗАЦИЯ АВТОМАТИЗИРОВАННЫХ ТЕСТОВ ДЛЯ ВЕБ-ПРИЛОЖЕНИЯ

3.1 Разработка диаграммы вариантов использования и тестовых сценариев

Диаграмма вариантов использования моделирует функциональность системы с использованием действующих лиц и прецедентов.

Действующие лица – это люди или организации, работающие под определенными ролями в системе.

Варианты использования – это набор действий, сервисов и функций, которые должна выполнять система.

В рассматриваемом контексте «система» – это веб-приложения, предоставляющее доступ к расписанию Вуза. «Действующие лица» – это пользователи данного приложения.

Диаграмма вариантов использования полезна для визуализации функциональных требований, которые помогут в составлении тестовых сценариев для проверки веб-приложения.

Возможности, которые приложение предоставляет пользователям, следующие:

- выбор института;
- просмотр расписания определенного преподавателя, группы или в аудитории;
- использование поиска;
- смена дня и недели расписания;
- редактирование дополнительных опций отображения расписания – цветовая маркировка, название предмета, время, преподаватель, группа, вид занятия, аудитория.

На основе действующих лиц и прецедентов составляется диаграмма вариантов использования веб-приложения (рисунок 3.1).

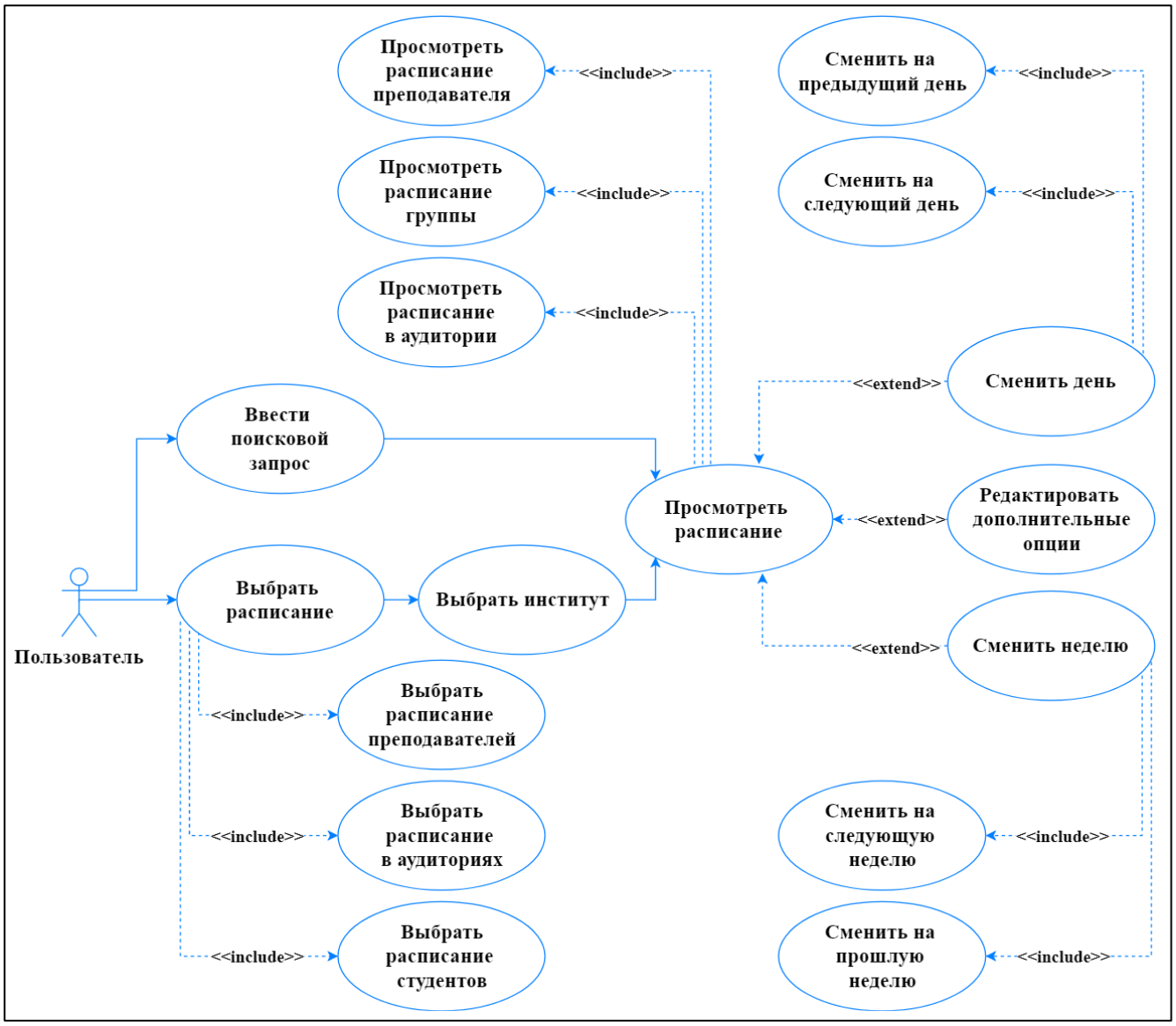

Рисунок 3.1 – Диаграмма вариантов использования веб-приложения

Исходя из действий, которые могут выполнять пользователи при работе с приложением, составляются тестовые сценарии.

Тестовый сценарий (англ. Test Case) – это последовательность действий, выполняя которую можно проверить соответствие функционала заявленным требованиями. Основные составляющие сценария следующие:

– идентификатор (англ. ID) – уникальный номер сценария;

– краткое описание (англ. Summary) – краткое описание тестируемого функционала;

– предусловия (англ. Preconditions) – список шагов либо данных, необходимых для выполнения данного сценария;

– выполняемые действия (англ. Action) – последовательность действий для тестирования;

– ожидаемый результат (англ. Expected Result) – результат, соответствующий заявленным требованиям;

– результат прохождения (англ. Passed/Failed) – результат прохождения теста: passed – успешно, failed – неудачно.

С использованием данной структуры составляются сценарии для проведения тестирования веб-приложения. Примеры тестовых сценариев представлены в таблице 3.1.

Таблица 3.1 – Примеры тестовых сценариев

ID	Summary	Pre-conditions	Action	Expected Result	P/F
01.TSU	Просмотреть расписание группы	Группа «ОНб-1501»	1. Нажать на кнопку «Расписание студентов»; 2. Нажать на кнопку «ИП»; 3. Нажать на кнопку «ОНб-1501».	Расписание группы «ОНб-1501» отображено	Passed
02.TSU	Найти расписание в аудитории	Аудитория «УЛК-408»	1. Нажать на поисковую строку; 2. Ввести запрос; 3. Нажать на кнопку «УЛК-408».	Расписание в аудитории «УЛК-408» отображено	Passed

Создание диаграммы вариантов использования помогло составить тестовые сценарии, описывающие действия, которые совершают пользователи при взаимодействии с веб-приложением.

Таким образом, появляется возможность автоматизации данного процесса путем разработки автоматизированных тестов на основании заготовленных сценариев.

3.2 Разработка автоматизированных тестов

Разработка автоматизированных тестов с использованием Selenium WebDriver начинается с подготовки необходимых компонентов – драйвера браузера и API языка программирования.

Согласно статистики использования браузеров (рисунок 3.2), Chrome является наиболее популярным, что дает основание для написания под него тестов.

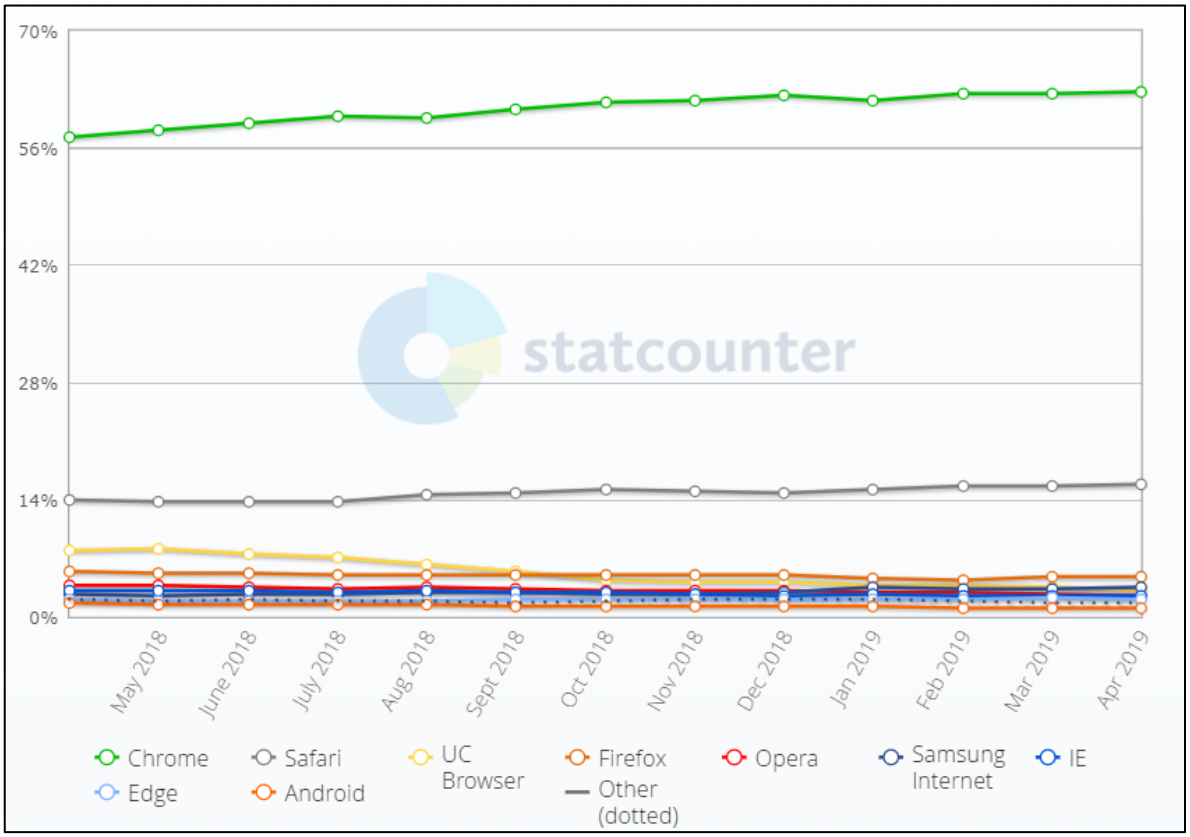

Рисунок 3.2 – Статистика использования браузеров

Также следует определить структуру программы: какие действия необходимо выполнить для непосредственного тестирования, какие вспомогательные операции нужно провести, в каком образе программа будет представлена.

Для проведения тестирования целесообразно разбить функционал веб-приложения на модули и испытывать их по отдельности для более тщательной проверки (рисунок 3.3):

1. выбор основных секций – это расписание преподавателей, студентов и в аудиториях, а также расписание институтов;
2. поисковая система;
3. просмотр расписания, включая смену дня, недели и редактирование дополнительных опций отображения – цветовая маркировка, название предмета, время, преподаватель, группа, вид занятия, аудитория.

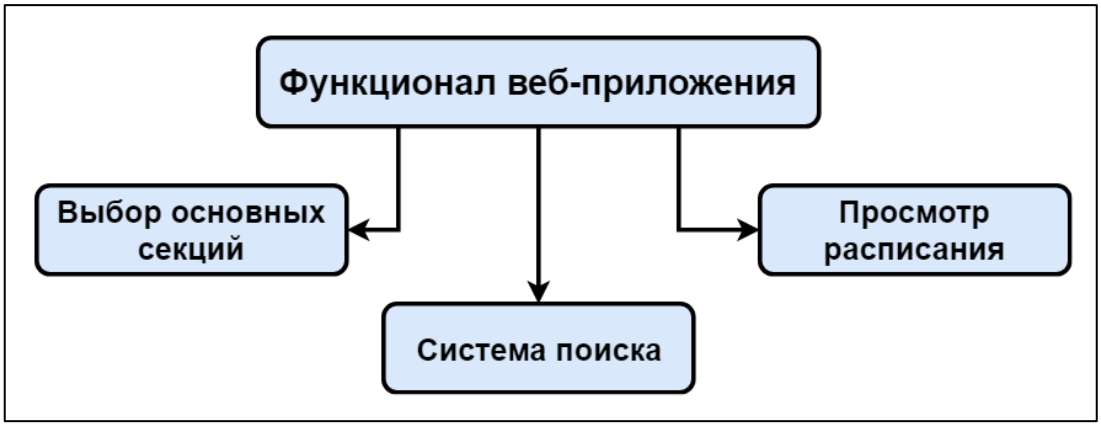

Рисунок 3.3 – Функционал веб-приложения

Результаты прохождения тестовых сценариев будут фиксироваться путем создания скриншотов и помещения их в специально отведенную директорию.

Также следует предусмотреть отслеживание ошибок, записывая их в лог с описанием проблемы. Лог – это автоматически создаваемый хронологический протокол работы программы или устройства. Располагаться данный файл будет в той же директории, что и скриншоты выполнения сценариев.

В связи с тем, что для результатов тестирования будет выделена одна папка, перед каждым запуском программы автоматизированных тестов необходимо ее очищать во избежание путаницы в файлах.

Для удобства использования взаимодействие с программой должно осуществляться через графический пользовательский интерфейс, все функции для проведения тестирования будут вызываться посредством нажатия соответствующих кнопок.

Алгоритм процесса тестирования представлен на рисунке 3.4

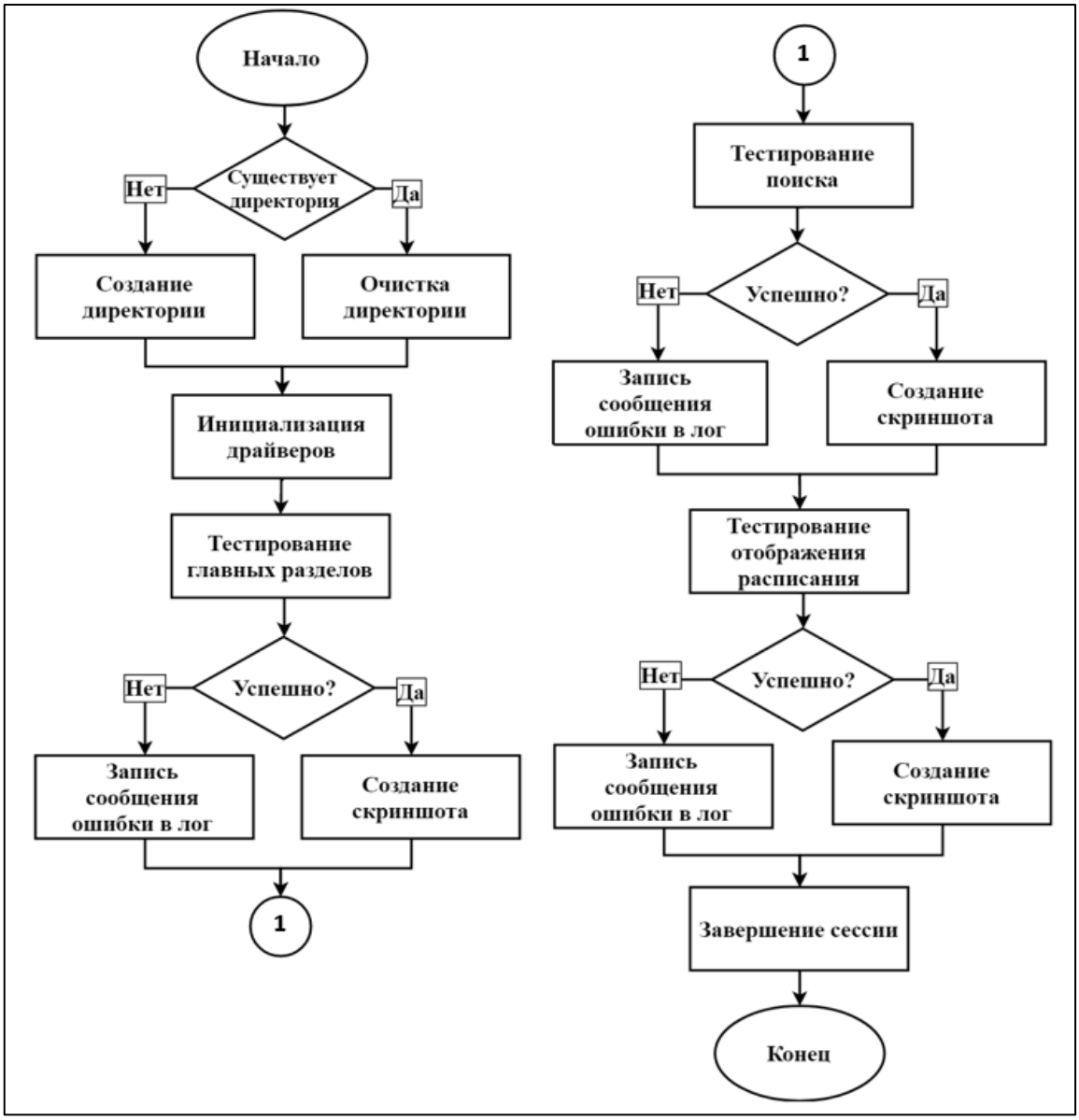

Рисунок 3.4 – Алгоритм тестирования

Основой для написания автоматизированных тестов служит объекта класса ChromeDriver. Вокруг данного объекта будут совершаться все последующие события.

В первую очередь определяются параметры драйвера – тип и место расположения.

Далее создаются объекты классов ChromeDriver и WebDriverWait. Объект второго класса необходим для вызова функций ожидания совершения определенных событий, а создание объекта ChromeDriver запускает работу браузера.

После открытия окна браузера производится максимизация его размеров, и задается таймер ожидания перед поиском элементов.

Завершается работа данного метода открытием страницы веб-приложения.

В случае появления ошибки при выполнении осуществляется ее перехват методом фиксации ошибок.

Фрагмент данного кода представлен на рисунке 3.5.

```
173     // Инициализация драйверов
174     private static void initDrivers() throws Exception{
175         try{
176             // Задание параметров веб-драйвера
177             System.setProperty("webdriver.chrome.driver", pathToDriver);
178             // Создание объектов классов ChromeDriver и WebDriverWait
179             driver = new ChromeDriver();
180             wait = new WebDriverWait(driver, 10);
181             // Максимимзация размера окна браузера
182             driver.manage().window().maximize();
183             // Задание времени ожидания перед поиском веб-элементов
184             driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
185             // Открытие страницы веб-приложения
186             driver.get(page);
187         } catch(WebDriverException e){
188             // Перехват исключения методом логирования ошибок
189             errLog(e);
190         }
191     }
```

Рисунок 3.5 – Фрагмент кода инициализации драйверов и открытия страницы веб-приложения

Следующий метод предназначен для тестирования главных секций приложения – расписания преподавателей, студентов, аудиторий в определенных институтах – всего их десять. Отображение содержимого каждого раздела осуществляется через нажатие на соответствующую кнопку.

Процесс взаимодействия с секциями проходит по следующему сценарию:

1. выбрать категория расписания – преподавателей, студентов или в аудиториях;

2. выбрать институт – ИФЭиУ, ИХиИЭ, ГумПИ, ИнМаш, ИЭиЭ, ИП, ИФКиС, АСИ, ИИиДПИ, ИМФИТ.

Таким образом, необходимо протестировать для каждой категории расписания выбор всех институтов.

Поиск соответствующих элементов будет производиться по идентификаторам.

Сопоставление идентификатора каждому из тринадцати элементов представлено в таблице 3.2.

Таблица 3.2 – Идентификаторы элементов главных разделов

Веб-элемент	Идентификатор
Расписание преподавателей	lecture
Расписание студентов	group
Расписание в аудиториях	classroom
ИФЭиУ	2
ИХиИЭ	3
ГумПИ	4
ИнМаш	5
ИЭиЭ	6
ИП	7
ИФКиС	8
АСИ	9
ИИиДПИ	11
ИМФИТ	12

Тестирование главных разделов проходит следующим образом:

1. идентификаторы, соответствующие расписанию преподавателей, групп и в аудиториях, помещаются в первый массив;

2. идентификаторы, соответствующие институтам, помещаются во второй массив;

3. посредством цикла for производится поиск элементов первого массива и нажатие на них;

4. с использованием встроенного цикла for осуществляется поиск элементов второго массива и нажатие на них;

5. вызывается метод создания скриншота с указанием номеров элементов каждого из массивов;

6. в случае появления исключения вызывается метод его перехвата для каждого из циклов.

Фрагмент кода, реализующего данный процесс, представлен на рисунке 3.6.

```
192  // Тестирование главных разделов
193  private static void testMainSections() throws Exception{
194      // Помещение идентификаторов в массивы
195      String section[] = new String[] {"lecturer", "group", "classroom"};
196      String inst[] = new String[] {"2","3","4","5","6","7","8","9","11","12"};
197      try{
198          for (String sec_id : section) {
199              try{
200                  // Нажатие на кнопку, соответствующую элементам первого массива
201                  driver.findElement(By.id(sec_id)).click();
202                  for(String inst_id : inst) {
203                      // Нажатие на кнопку, соответствующую элементам второго массива
204                      driver.findElement(By.id(inst_id)).click();
205                      // Создание скриншота
206                      takeScr(sec_id + "_section_" + inst_id);
207                  }
208              } catch (WebDriverException e){
209                  // Перехват исключения методом логирования ошибок
210                  errLog(e);
211              }
212          }
213      } catch(WebDriverException e){
214          // Перехват исключения методом логирования ошибок
215          errLog(e);
216      }
217  }
```

Рисунок 3.6 – Фрагмент кода тестирования главных разделов

Поиск в веб-приложении производится по наименованию необходимой группы, номера аудитории или ФИО преподавателя, после получения результата осуществляется нажатие на выданное расписание.

В качестве поисковых запросов выбраны названия групп из разных институтов, один преподаватель и аудитория.

Проверка работоспособности поиска расписания происходит по следующему алгоритму:

1. поисковые запросы помещаются в массив;
2. с использованием цикла for проводится навигация на текстовое поле и передаются значения элементов массива;
3. результат фиксируется вызовом функции создания скриншота;
4. в том же цикле совершается нажатие на найденное расписание;
5. вызывается функция создания скриншота;
6. в случае появления исключения вызывается метод его перехвата.

Фрагмент кода, реализующий данный процесс, представлен на рисунке 3.7.

```
218  // Тестирование поиска
219  private static void testSearch() throws Exception{
220      // Помещение поисковых запросов в массив
221      String srch[] = new String[] {"МОб-1501","ЛИНб-1601б","ФЗКп-1601а",
222                                    "Бобровский Николай Михайлович","УЛК-418"};
223      for (String srch_i : srch){
224          try{
225              driver.get(page);
226              //  Передача запроса в функцию поиска
227              driver.findElement(By.className("searchTextField")).sendKeys(srch_i);
228              // Создание скриншота
229              takeScr("search_" + srch_i);
230              // Навигация на найденное расписание
231              driver.findElement(By.linkText(srch_i)).click();
232              // Создание скриншота
233              takeScr("searchResult_" + srch_i);
234          } catch(WebDriverException e){
235              // Перехват исключения методом логирования ошибок
236              errLog(e);
237          }
238      }
239  }
```

Рисунок 3.7 – Фрагмент кода тестирования функции поиска

После навигации на расписание предоставляется возможность задействовать дополнительные функции – сменить день или неделю, редактировать опции отображения – цветовую маркировку, название предмета, время, ФИО преподавателя, название группы, вид занятия, аудиторию.

Предусловиями для данного тестового сценария являются открытое расписание и массив с XPath кнопок для смены дня и недели (рисунок 3.8).

```
240  // Тестирование дополнительных функций для просмотра расписания
241  private static void testTimetable() throws Exception{
242      // Название группы для поиска
243      String tt1 = "ОНБ-1501";
244      String tt2 = "ОНБ-1502";
245      // Массивы для xpath кнопок переключения дней и недель
246      String week[] = new String[3];
247      String day[] = new String[3];
248      try{
249          // Поиск и открытие расписания
250          driver.get(page);
251          driver.findElement(By.className("searchTextField")).sendKeys(tt1);
252          driver.findElement(By.cssSelector("body")).sendKeys(Keys.ENTER);
253          wait.until(ExpectedConditions.
254                                  invisibilityOf(driver.findElement(By.linkText(tt2))));
255          driver.findElement(By.linkText(tt1)).click();
256          // Помещение xpath кнопок переключения дней и недель в массив
257          for(int i=0; i<3; i++){
258              int j = i+1;
259              week[i] = "//*[@id=\"weekBars\"]/a[" + j + "]";
260              day[i] = "//*[@id=\"dayBars\"]/a[" + j + "]";
261          }
```

Рисунок 3.8 – Фрагмент кода подготовки предусловий для тестирования дополнительных функций просмотра расписания

Тестирование функции смены недели производится следующим образом:

1. фиксируется текущая неделя созданием скриншота;
2. выполняется переход на предыдущую неделю;
3. создается скриншот;
4. выполняется переход на следующую неделю;
5. создается скриншот;
6. в случае появления исключения вызывается метод его перехвата.

Фрагмент данного кода представлен на рисунке 3.9.

```
262     // Тестирование функции переключения недель
263     try{
264         // Создание скриншота текущей недели
265         takeScr("timeTableBar1_currWeek");
266         // Переключение на предыдущую неделю
267         driver.findElement(By.xpath(week[0])).click();
268         // Создание скриншота предыдущей недели
269         takeScr("timeTableBar2_prevWeek");
270         // Переход на следующую неделю
271         for(int i=0; i<2; i++){
272             driver.findElement(By.xpath(week[2])).click();
273         }
274         // Создание скриншота следующей недели
275         takeScr("timeTableBar3_nextWeek");
276     } catch(WebDriverException e){
277         // Перехват исключения методом логирования ошибок
278         errLog(e);
279     }
```

Рисунок 3.9 – Фрагмент кода тестирования функции переключения недель

После проверки переключения между неделями проводится тестирование смены дней: выполняется переход на режим отображения расписания по дням и ожидается скрытие панели для смены недели, выполняется переход на предыдущий день, выполняется переход на следующий день, возвращение к режиму отображения расписания по неделям. Каждое открытие расписания сопровождается созданием скриншота, в случае появления исключения вызывается метод его перехвата. Фрагмент данного кода представлен на рисунке 3.10.

```
280     // Тестирование функции переключения дней
281     try{
282         // Переход на режим отображения расписания по дням
283         driver.findElement(By.xpath("//*[@id=\"weekBars\"]/a[2]")).click();
284         // Ожидание скрытия панели переключения недель
285         wait.until(ExpectedConditions
286                 .invisibilityOf(driver.findElement(By.id("weekBars"))));
287         // Создание скриншота текущего дня
288         takeScr("timeTableBar4_currDay");
289         // Переключение на предыдущий день
290         driver.findElement(By.xpath(day[0])).click();
291         // Создание скриншота предыдущего дня
292         takeScr("timeTableBar5_prevDay");
293         // Переключение на следующий день
294         for(int i=0; i<2; i++){
295             driver.findElement(By.xpath(day[2])).click();
296         }
297         // Создание скриншота следующего дня
298         takeScr("timeTableBar6_nextDay");
299         // Переход на режим отображения расписания по неделям
300         driver.findElement(By.xpath(day[1])).click();
301         // Ожидание скрытия панели переключения дней
302         wait.until(ExpectedConditions
303                 .invisibilityOf(driver.findElement(By.id("dayBars"))));
304     }catch(WebDriverException e){
305         // Перехват исключения методом логирования ошибок
306         errLog(e);
307     }
```

Рисунок 3.10 – Фрагмента кода тестирования функции переключения дней

Последняя дополнительная функция для просмотра расписания – редактирование опций отображения.

По идентификаторам опций будет производиться взаимодействие с ними. Сопоставление идентификатора каждому из восьми элементов представлено в таблице 3.3.

Таблица 3.3 – Идентификаторы элементов опций редактирования отображения расписания

Веб-элемент	Идентификатор
Цветовая маркировка пар	color
Полное название предмета	nameSize
Название предмета	SUBJECT_NAME
Вид занятия	PARA_TYPE
Группы	GROUPS
Преподаватель	LECTURER_NAME
Аудитория	CLASSROOM_NAME
Время	PARA_TIME

Тестирование данной функции начинается с открытия панели редактирования опций. После чего все идентификаторы помещаются в массив.

С использованием цикла for производится поиск и нажатие на кнопки, соответствующие элементам массива.

После проверки каждой из опций создается скриншот результата выполнения.

В случае появления исключения вызывается метод его перехвата.

Фрагмент кода представлен на рисунке 3.11.

```
308  // Тестирование функции редактирования опций отображения расписания
309  try{
310      // Нажатие на кнопку открытия панели редактирования опций
311      driver.findElement(By.xpath("/html/body/div[2]/div[2]/a")).click();
312      // Ожидание отображения панели
313      wait.until(ExpectedConditions.visibilityOf(driver.findElement(By.id("extra"))));
314      // Помещение идентификаторов элементов в массив
315      String extr[] = new String[]{"color","nameSize","SUBJECT_NAME",
316                                   "PARA_TYPE","GROUPS","LECTURER_NAME",
317                                   "CLASSROOM_NAME","PARA_TIME"};
318      // Поиск и нажатие на кнопки, соответствующие элементам массива
319      for(String extr_i : extr){
320          driver.findElement(By.id(extr_i)).click();
321          takeScr("timeTableBarExtra_" + extr_i);
322          driver.findElement(By.id(extr_i)).click();
323      }
324  } catch(WebDriverException e){
325      // Перехват исключения методом логирования ошибок
326      errLog(e);
327  }
```

Рисунок 3.11 – Фрагмент кода тестирования функции редактирования опций отображения расписания

Финальный метод для работы с браузером выполняет закрытие всех окон и завершение сессии. Как и в предыдущих методах, предусмотрен перехват исключений. Фрагмент кода завершения работы представлен на рисунке 3.12.

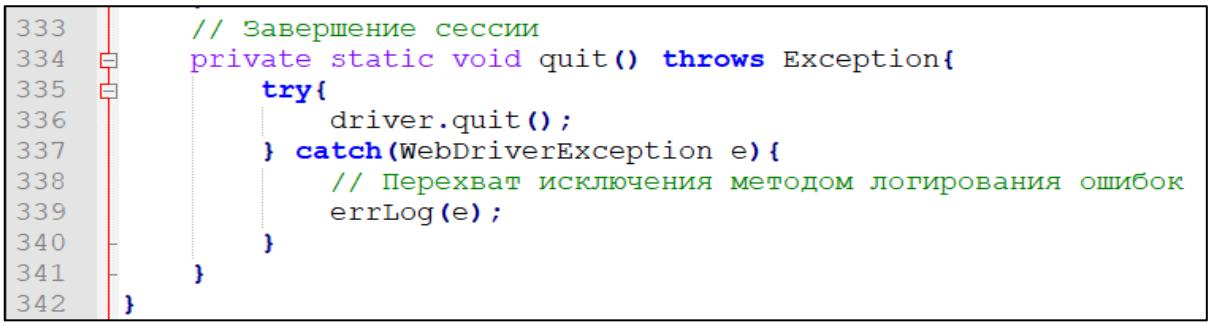

Рисунок 3.12 – Фрагмент кода завершения сессии

Помимо основных методов для проведения тестирования в программе реализованы вспомогательные функции – создание скриншота, очищение директории с результатами тестирования, фильтрация разрешения файлов, логирование ошибок.

Метод создания скриншота принимает в качестве параметра строковое значение, которое будет использовано в качестве названия файла. Создание изображение производится через вызов функции getScreenshotAs().

Метод очищения директории выполняет две задачи – очистку папки с результатами и, в случае ее отсутствия, создание по заданному пути.

Логирование ошибок принимает текст сообщения об ошибке в качестве параметра и производит следующие действия:

1. путем вызова метода фильтрации разрешения файлов определяет наличие в директории файла с разрешением «.log»;

2. в случае отсутствия лог-файла создает его и записывает сообщение об ошибке;

3. если лог присутствует, то в конец файла дописывает сообщение об ошибке.

Программный код главного класса приведен в приложении А.

Подготовив основную часть программы, следует приступить к созданию графического пользовательского интерфейса. Класс, в котором реализуется данный интерфейс, приведен в приложении Б. Создание объекта данного класса запускает форму вида, представленного на рисунке 3.13.

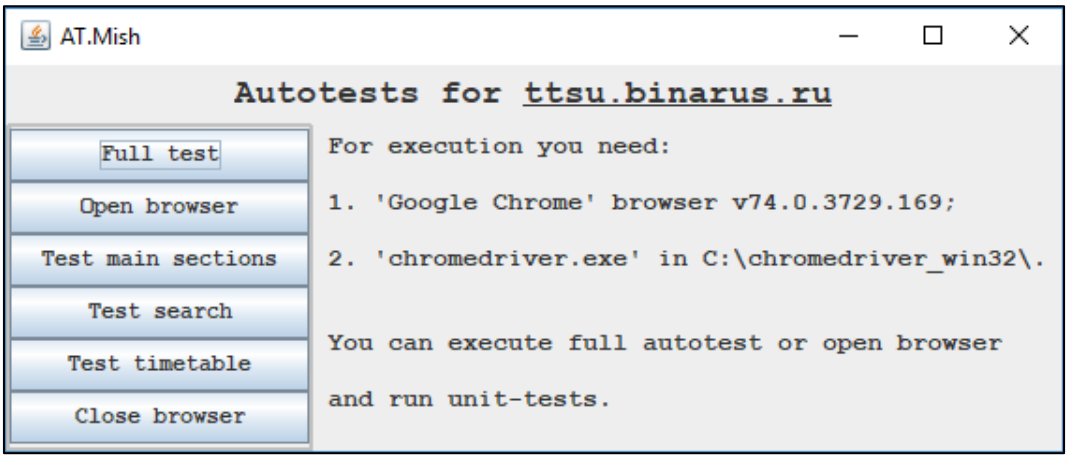

Рисунок 3.13 – Пользовательский интерфейс приложения

Пользовательский интерфейс содержит рекомендации по работе с приложением – требуемую версию драйвера и путь, по которому драйвер должен быть расположен, а также предоставляет возможность проведения тестирования как отдельных модулей, так и веб-приложения в целом.

Далее приведен перечень названия кнопок и действий, следующих за их нажатием:

– «Full test» – открытие браузера, исполнение всех авто-тестов;

– «Open browser» – инициализация драйвера и открытие браузера;

– «Test main sections» – тестирование главных секций;

– «Test search» – тестирование поиска;

– «Test timetable» – тестирование дополнительных функций для просмотра расписания;

– «Close browser» – закрытие браузера и завершение сессии.

Далее созданная программа экспортируется в исполняемый JAR-файл.

Таким образом, разработка автоматизированных тестов для веб-приложения завершена, и требуется оценка их эффективности.

3.3 Оценка эффективности автоматизированных тестов

Оценка эффективности разработанных автоматизированных тестов будет проводиться по сохраненным результатам – скриншотам и логу ошибок.

Отсутствия лог-файла после завершения тестирования говорит о том, что процесс прошел без ошибок и весь функционал был проверен. Общее количество созданных скриншотов должно быть равно пятидесяти пяти.

При запуске исполняемого JAR-файла должны открываться форма с корректно отображенными элементами и удаляться результаты прошлого теста, а нажатие соответствующих кнопок выполняет тестирование веб-приложения.

Ниже представлены результаты выполнения автоматизированных тестов:

1. корректно отображены элементы формы;
2. удалены предыдущие результаты тестов;
3. отсутствует лог ошибок;
4. скриншоты отображают результаты проверки каждого функционального модуля.

На рисунке 3.14 представлен конечный результат выполнения тестов.

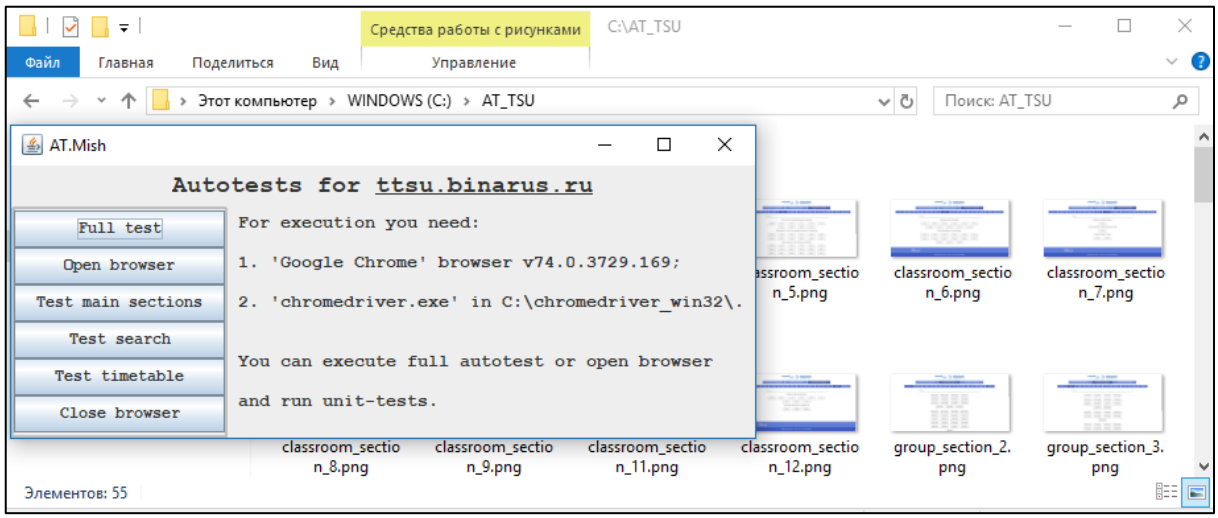

Рисунок 3.14 – Конечный результат проведения тестирования

Лог позволяет определить, каким методом была вызвана ошибка и, сымитировав проблемную ситуация, а именно использовав несуществующие идентификаторы веб-элементов, он выглядит следующим образом (рисунок 3.15).

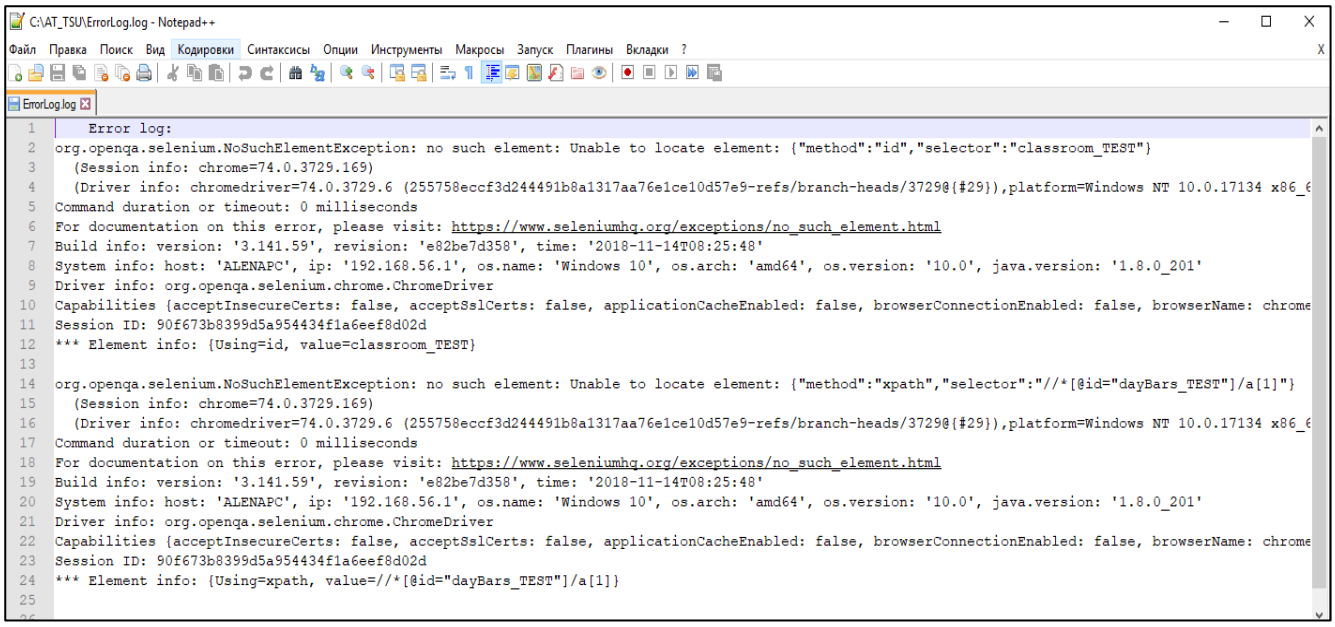

Рисунок 3.15 – Результат работы метода отслеживания ошибок

Веб-приложение было успешно протестировано, а метод отслеживания ошибок корректно выполняет свои задачи, что подтверждает эффективность разработанных автоматизированных тестов.

Выводы по главе 3

В третьей главе были разработаны диаграмма вариантов использования, тестовые сценарии и автоматизированные тесты для веб-приложения, и подтверждена эффективность данных тестов.

1. На основании возможностей веб-приложения, предоставляющего доступ к расписанию Вуза, была разработана диаграмма вариантов использования, послужившая базой для написания тестовых сценариев.

2. С использованием созданных сценариев разработаны автоматизированные тесты, учитывающие специфику приложения.

3. Подтверждена эффективность реализованных тестов благодаря успешному проведению функционального тестирования веб-приложения.

ЗАКЛЮЧЕНИЕ

Выпускная квалификационная работа посвящена повышению эффективности тестирования веб-приложений путем автоматизации данного процесса.

В ходе выполнения работы достигнуты следующие результаты:

1. проведен анализ видов тестирования веб-приложений – функционального, удобства использования, интерфейса, совместимости, производительности, безопасности, а также методов их проведения – белого, серого, черного ящиков, ручного и автоматизированного;

2. рассмотрены готовые решения для проведения автоматизированного тестирования, а также инструменты, позволяющие создавать собственные тесты;

3. на основании сравнения инструментов автоматизации был сделан вывод о преобладании Selenium в силу поддержки большого количества операционных систем, браузеров и языков программирования;

4. описаны функциональные и архитектурные особенности Selenium WebDriver, включая основные методы данной библиотеки и компоненты, обеспечивающие эффективность ее применения;

5. разработаны диаграмма вариантов использования приложения, предоставляющего доступ к расписанию Вуза, а на ее базе – тестовые сценарии;

6. с использованием созданных сценариев реализованы автоматизированные тесты, выполняющие проверку функционала веб-приложения;

7. подтверждена эффективность разработанных тестов путем успешного проведения тестирования.

Реализованное программное обеспечение может быть использовано разработчиками данного веб-приложения для выполнения функционального и регрессионного тестирования.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

Нормативно-правовые акты

1. ГОСТ 19.402–78. Единая система программной документации. Описание программы, 1980. – Москва, 2010. – 1 с.;

2. ГОСТ 34.603–92. Информационная технология. Виды испытаний автоматизированных систем, 1993. – Москва, 2009. – 6 с.;

Научная и методическая литература

3. Грегори Джанет. Гибкое тестирование. Практическое руководство для тестировщиков ПО и гибких команд / Грегори Джанет,Криспин Лиза. – Вильямс, 2016. – 464 с.;

4. Гленфорд Майерс. Искусство тестирования программ / Гленфорд Майерс, Том Баджетт, Кори Сандлер. – Вильямс, 2016. – 272 с.;

Электронные ресурсы

5. Старолетов, С.М. Основы тестирования и верификации программного обеспечения [Электронный ресурс] : учебное пособие / С.М. Старолетов. — Электрон. дан. — Санкт-Петербург : Лань, 2018. — 344 с. — Режим доступа: https://e.lanbook.com/book/110939. — Загл. с экрана;

6. Виссер, Д. Разработка обслуживаемых программ на языке Java [Электронный ресурс] / Д. Виссер ; пер. с англ. Р. Н. Рагимова. — Электрон. дан. — Москва : ДМК Пресс, 2017. — 182 с. — Режим доступа: https://e.lanbook.com/book/105834. — Загл. с экрана;

Литература на иностранном языке

7. Bayo Erinle. Performance Testing with JMeter 3 – Third Edition: Enhance the performance of your web application / Bayo Erinle. – Packt Publishing, 2017. – 166 p.;

8. Brian Hambling. Software Testing: An ISTQB–BCS Certified Tester Foundation guide 4th edition Edition / Brian Hambling, Peter Morgan, Angelina Samaroo, Geoff Thompson, Peter Williams. – BCS, The Chartered Institute for IT, 2019. – 225 p.;

9. Carl Cocchiaro. Selenium Framework Design in Data–Driven Testing: Build data–driven test frameworks using Selenium WebDriver, AppiumDriver, Java, and TestNG / Carl Cocchiaro. – Packt Publishing, 2018. – 354 p.;

10. Diego Molina. Selenium Fundamentals: Speed up your internal testing by automating user interaction with browsers and web applications / Diego Molina. – Packt Publishing, 2018. – 206 p.;

11. Glenford J. Myers ET AL. Art Of Software Testing, 3Ed 3rd Edition / Glenford J. Myers ET AL. – WILEY INDIA, 2015. – 185 p.;

12. Greg Paskal. Test Automation in the Real World: Practical Lessons for Automated Testing / Greg Paskal. – MissionWares, 2015. – 103 p.;

13. Jamie L Mitchell. Advanced Software Testing – Vol. 3, 2nd Edition: Guide to the ISTQB Advanced Certification as an Advanced Technical Test Analyst 2nd Edition / Jamie L Mitchell, Rex Black. – Rocky Nook, 2015. – 480 p.;

14. Jonathan Rasmusson. The Way of the Web Tester: A Beginner's Guide to Automating Tests 1st / Jonathan Rasmusson. – Pragmatic Bookshelf, 2016. 258 p.;

15. Parveen. Software Testing: Selenium for Beginners / Parveen. – Parveen, 2016. – 28 p.;

16. Philip Conrod. Learn Java GUI Applications: A JFC Swing Tutorial /Philip Conrod, Lou Tylee. – Kidware Software LLC, 2017. – 1470 p.;

17. Pinakin Chaubal. Selenium WebDriver Quick Start Guide: Write clear, readable, and reliable tests with Selenium WebDriver 3 / Pinakin Chaubal. – Packt Publishing, 2018. – 192 p.;

18. Rex Allen Jones II. Java 4 Selenium WebDriver: Come Learn How To Program For Automation (Part 2) / Rex Allen Jones II. – Test 4 Success, 2016. – 175 p.;

19. Rupert Anderson. SoapUI Cookbook / Rupert Anderson. – Packt Publishing, 2015. – 330 p.;

20. Zhimin Zhan. Selenium WebDriver Recipes in Java: The problem solving guide to Selenium WebDriver in Java (Web Test Automation Recipes Series) (Volume 3) / Zhimin Zhan. – CreateSpace Independent Publishing Platform, 2015. – 184 p.

ПРИЛОЖЕНИЕ А

Главный класс приложения

```
package tsu;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.util.concurrent.TimeUnit;
import javax.imageio.ImageIO;
import java.awt.event.ActionEvent;
import java.io.FileWriter;
import java.io.FilenameFilter;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.openqa.selenium.By;
import org.openqa.selenium.Keys;
import org.openqa.selenium.OutputType;
import org.openqa.selenium.TakesScreenshot;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;
import org.openqa.selenium.WebDriverException;
import org.openqa.selenium.chrome.ChromeDriverService;

public class TSU{
    private static WebDriver driver;
    private static ChromeDriverService service;
    private static WebDriverWait wait;
    private static String pathToDriver = "C:\\chromedriver_win32\\chromedriver.exe";
    private static String pathToScreens = "C:\\AT_TSU";
    private static String page = "http://ttsu.binarus.ru";
    private static String logName = "ErrorLog.log";
    private static String ext = ".log";

    public static void main(String[] args) throws Exception {
        // Создание формы
        Menu menu = new Menu();
        // Очистка директории с результатами
        clearFolder();
        // Кнопка инициализации драйвера и навигации на страницу
        menu.getBOpen().addActionListener((ActionEvent ev) -> {
            if (driver == null){
                try{
                    initDrivers();
                } catch (Exception ex) {
                    Logger.getLogger(TSU.class.getName()).log(Level.SEVERE, null, ex);
                }
            }
        });
```

```
// Кнопка зпуска полного теста
menu.getBFull().addActionListener((ActionEvent ev) -> {
    if (driver == null){
        try {
            initDrivers();
            testMainSections();
            testSearch();
            testTimetable();
            //quit();
        } catch (Exception ex) {
            Logger.getLogger(TSU.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
});
// Кнопка тестирования главных секций
menu.getBMain().addActionListener((ActionEvent ev) -> {
    if (driver != null){
        try {
            testMainSections();
        } catch (Exception ex) {
            Logger.getLogger(TSU.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
});
// Кнопка тестирования поиска
menu.getBSeacrh().addActionListener((ActionEvent ev) -> {
    if (driver != null){
        try {
            testSearch();
        } catch (Exception ex) {
            Logger.getLogger(TSU.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
});
// Кнопка тестирования дополнительных функций просмотра расписания
menu.getBTT().addActionListener((ActionEvent ev) -> {
    if (driver != null){
        try {
            testTimetable();
        } catch (Exception ex) {
            Logger.getLogger(TSU.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
});
// Кнопка заверешения работы
menu.getBQuit().addActionListener((ActionEvent ev) -> {
    if (driver != null){
        try {
            quit();
        } catch (Exception ex) {
            Logger.getLogger(TSU.class.getName()).log(Level.SEVERE, null, ex);
        }
```

```
        }else {}
    });
}

 // Создание скриншота
private static void takeScr(String screenshotName) throws IOException{
                File screen = ((TakesScreenshot) driver).getScreenshotAs(OutputType.FILE);
                try{BufferedImage img = ImageIO.read(screen);
        File file = new File(pathToScreens, screenshotName + ".png");
        ImageIO.write(img, "png", file);
                } catch (IOException e) {
        errLog(e);}
        }
// Очищение директории
private static void clearFolder() throws IOException{
    File files = new File(pathToScreens);
    if(!files.exists()){
        files.mkdir();
    }
                String[] entries = files.list();
    for(String s: entries){
        File currentFile = new File(files.getPath(), s);
        if (currentFile.exists()) {
            currentFile.delete();}}
}
// Фильтр разрешения файлов
public static class FileFilter implements FilenameFilter{
    private String ext;
    public FileFilter(String ext){
        this.ext = ext.toLowerCase();
    }
    @Override
    public boolean accept(File dir, String name) {
        return name.toLowerCase().endsWith(ext);
    }
}
// Лог ошибок
private static void errLog(Exception errMessage) throws IOException{
    File eDir = new File(pathToScreens);
    File[] listFiles = eDir.listFiles(new FileFilter(ext));
    if(listFiles.length == 0){
        File eFile = new File(pathToScreens, logName);
        try (FileWriter eWriter = new FileWriter(eFile)) {
            eWriter.write("\tError log:\r\n");
            String err = errMessage.toString();
            eWriter.write(err + "\r\n\r\n");
        } catch(IOException e){
            e.printStackTrace();
        }
    }else{
        for (File f : listFiles){
            try (FileWriter eWriter2 = new FileWriter(f, true)) {
```

```
            String err = errMessage.toString();
            eWriter2.write(err + "\r\n\r\n");
        }catch(IOException e){
            e.printStackTrace();
        }
    }
  }
}
// Инициализация драйверов
private static void initDrivers() throws Exception{
  try{
    // Задание параметров веб-драйвера
    System.setProperty("webdriver.chrome.driver", pathToDriver);
    // Создание объектов классов ChromeDriver и WebDriverWait
    driver = new ChromeDriver();
    wait = new WebDriverWait(driver, 10);
    // Максимимзация размера окна браузера
    driver.manage().window().maximize();
    // Задание времени ожидания перед поиском веб-элементов
    driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
    // Открытие страницы веб-приложения
    driver.get(page);
  } catch(WebDriverException e){
    // Перехват исключения методом логирования ошибок
    errLog(e);
  }
}
// Тестирование главных разделов
private static void testMainSections() throws Exception{
    // Помещение идентификаторов в массивы
  String section[] = new String[] {"lecturer", "group", "classroom"};
  String inst[] = new String[] {"2","3","4","5","6","7","8","9","11","12"};
  try{
    driver.get(page);
    for (String sec_id : section) {
      try{
        // Нажатие на кнопку, соответствующую элементам первого массива
        driver.findElement(By.id(sec_id)).click();
        for(String inst_id : inst) {
                // Нажатие на кнопку, соответствующую элементам второго массива
          driver.findElement(By.id(inst_id)).click();
                // Создание скриншота
          takeScr(sec_id + "_section_" + inst_id);
        }
      } catch (WebDriverException e){
        // Перехват исключения методом логирования ошибок
        errLog(e);
      }
    }
  } catch(WebDriverException e){
    // Перехват исключения методом логирования ошибок
    errLog(e);
```

```
    }
}
// Тестирование поиска
private static void testSearch() throws Exception{
      // Помещение поисковых запросов в массив
   String srch[] = new String[] {"МОб-1501","ЛИНб-1601б","ФЗКп-1601а",
                                 "Бобровский Николай Михайлович","УЛК-418"};
   for (String srch_i : srch){
      try{
         driver.get(page);
               //  Передача запроса в функцию поиска
         driver.findElement(By.className("searchTextField")).sendKeys(srch_i);
         // Создание скриншота
               takeScr("search_" + srch_i);
               // Навигация на найденное расписание
         driver.findElement(By.linkText(srch_i)).click();
         // Создание скриншота
               takeScr("searchResult_" + srch_i);
      } catch(WebDriverException e){
               // Перехват исключения методом логирования ошибок
         errLog(e);
      }
   }
}
// Тестирование дополнительных функций для просмотра расписания
private static void testTimetable() throws Exception{
      // Название группы для поиска
   String tt1 = "ФИЛб-1801а";
      // Массивы для xpath кнопок переключения дней и недель
   String week[] = new String[3];
   String day[] = new String[3];
   try{
      // Поиск и открытие расписания
      driver.get(page);
      driver.findElement(By.className("searchTextField")).sendKeys(tt1);
      takeScr("timeTableBar0_foundTT");
      driver.findElement(By.linkText(tt1)).click();
      // Помещение xpath кнопок переключения дней и недель в массив
      for(int i=0; i<3; i++){
         int j = i+1;
         week[i] = "//*[@id=\"weekBars\"]/a[" + j + "]";
         day[i] = "//*[@id=\"dayBars\"]/a[" + j + "]";
      }
      // Тестирование функции переключения недель
      try{
         // Создание скриншота текущей недели
               takeScr("timeTableBar1_currWeek");
               // Переключение на предыдущую неделю
         driver.findElement(By.xpath(week[0])).click();
               // Создание скриншота предыдущей недели
         takeScr("timeTableBar2_prevWeek");
               // Переход на следующую неделю
```

```
   for(int i=0; i<2; i++){
      driver.findElement(By.xpath(week[2])).click();
   }
         // Создание скриншота следующей недели
   takeScr("timeTableBar3_nextWeek");
} catch(WebDriverException e){
                        // Перехват исключения методом логирования ошибок
   errLog(e);
}
// Тестирование функции переключения дней
try{
   // Переход на режим отображения расписания по дням
         driver.findElement(By.xpath("//*[@id=\"weekBars\"]/a[2]")).click();
         // Ожидание скрытия панели переключения недель
   wait.until(ExpectedConditions.invisibilityOf(driver.findElement(By.id("weekBars"))));
   // Создание скриншота текущего дня
         takeScr("timeTableBar4_currDay");
   // Переключение на предыдущий день
         driver.findElement(By.xpath(day[0])).click();
   // Создание скриншота предыдущего дня
         takeScr("timeTableBar5_prevDay");
   // Переключение на следующий день
         for(int i=0; i<2; i++){
      driver.findElement(By.xpath(day[2])).click();
   }
         // Создание скриншота следующего дня
   takeScr("timeTableBar6_nextDay");
         // Переход на режим отображения расписания по неделям
   driver.findElement(By.xpath(day[1])).click();
         // Ожидание скрытия панели переключения дней
   wait.until(ExpectedConditions.invisibilityOf(driver.findElement(By.id("dayBars"))));
} catch(WebDriverException e){
   // Перехват исключения методом логирования ошибок
         errLog(e);
}
// Тестирование функции редактирования опций отображения расписания
try{
   // Нажатие на кнопку открытия панели редактирования опций
         driver.findElement(By.xpath("/html/body/div[2]/div[2]/a")).click();
   // Ожидание отображения панели
         wait.until(ExpectedConditions.visibilityOf(driver.findElement(By.id("extra"))));
   // Помещение идентификаторов элементов в массив
         String extr[] = new String[]{"color","nameSize","SUBJECT_NAME",
                        "PARA_TYPE","GROUPS","LECTURER_NAME",
                        "CLASSROOM_NAME","PARA_TIME"};
   // Поиск и нажатие на кнопки, соответствующие элементам массива
   for(String extr_i : extr){
      driver.findElement(By.id(extr_i)).click();
      takeScr("timeTableBarExtra_" + extr_i);
   driver.findElement(By.id(extr_i)).click();
   }
} catch(WebDriverException e){
```

```
            // Перехват исключения методом логирования ошибок
                errLog(e);
            }
        } catch(WebDriverException e){
            errLog(e);
        }
    }
    // Завершение сессии
    private static void quit() throws Exception{
        try{
            driver.quit();
        } catch(WebDriverException e){
            // Перехват исключения методом логирования ошибок
                errLog(e);
            }
    }
}
```

ПРИЛОЖЕНИЕ Б

Класс, реализующий UI

```
package tsu;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.FlowLayout;
import java.awt.Font;
import java.awt.GridLayout;
import javax.swing.BorderFactory;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;

public class Menu {
   // Создание кнопок
   private static JButton bOpen = new JButton("Open browser");
   private static JButton bQuit = new JButton("Close browser");
   private static JButton bFull = new JButton("Full test");
   private static JButton bMain = new JButton("Test main sections");
   private static JButton bSearch = new JButton("Test search");
   private static JButton bTT = new JButton("Test timetable");
   // Создание шрифтов
   private static Font font12 = new Font("Courier New", Font.BOLD, 12);
   private static Font font13 = new Font("Courier New", Font.BOLD, 13);
   private static Font font16 = new Font("Courier New", Font.BOLD, 18);

   public Menu (){
      // Создание формы
      JFrame frame = new JFrame("AT.Mish");
      frame.setBounds(100, 200, 580, 245);
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setLocationRelativeTo(null);
      frame.setLayout(new BorderLayout());
      // Заголовок и описание
      String headline = "<html>Autotests for <u>ttsu.binarus.ru<u><html>";
      String description = "<html> For execution you need: <br><br>"
                  + "1. 'Google Chrome' browser v74.0.3729.169;<br><br>"
                  + "2. 'chromedriver.exe' in C:\\chromedriver_win32\\.<br><br><br>"
                  + "You can execute full autotest or open browser<br><br>"
                  + "and run unit-tests.<br><br>";
      // Создание панелей
      JPanel pNorth = new JPanel(new FlowLayout());
      JPanel pWest = new JPanel(new GridLayout(6,1));
      JPanel pCent = new JPanel(new FlowLayout());
      // Создание лейблов
      JLabel hl = new JLabel(headline);
      JLabel desc = new JLabel(description);
      desc.setFont(font13);
      hl.setFont(font16);
```

```
        pWest.setBorder(BorderFactory.createLineBorder(Color.LIGHT_GRAY, 2, true));
        // Добавление элементов на панели
        pNorth.add(hl);
        pCent.add(desc);
        pWest.add(bFull).setFont(font12);
        pWest.add(bOpen).setFont(font12);
        pWest.add(bMain).setFont(font12);
        pWest.add(bSearch).setFont(font12);
        pWest.add(bTT).setFont(font12);
        pWest.add(bQuit).setFont(font12);
        // Добавление панелей в форму
        frame.add(pNorth, BorderLayout.NORTH);
        frame.add(pCent, BorderLayout.CENTER);
        frame.add(pWest, BorderLayout.WEST);
        frame.setVisible(true);
    }
    public JButton getBOpen(){
        return bOpen;
    }
    public JButton getBQuit(){
        return bQuit;
    }
    public JButton getBMain(){
        return bMain;
    }
    public JButton getBSeacrh(){
        return bSearch;
    }
    public JButton getBFull(){
        return bFull;
    }
    public JButton getBTT(){
        return bTT;
    }
}
```