МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»

Кафедра системного анализа и
автоматического управления

АВТОМАТИЗИРОВАННОЕ ТЕСТИРОВАНИЕ ВЕБ-ПРИЛОЖЕНИЙ НА JAVA С ИСПОЛЬЗОВАНИЕМ SELENIUM

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

Студентки 5 курса 551 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Фомкиной Натальи Игоревны

Научный руководитель
доцент, к. ф.-м. н. _____ М. В. Корнилов

Заведующий кафедрой
к. ф.-м. н., доцент _____ И. Е. Тананко

Саратов 2023

ВВЕДЕНИЕ

Актуальность темы. Программирование на современном этапе его развития — это сложная область человеческой деятельности, в которой практически невозможно избежать ошибок. Если раньше процесс отладки программ был достаточно прост в силу их примитивности и не представлял серьезных проблем, то сейчас ситуация изменилась и продолжает меняться. Появляются новые технологии, фреймворки и подходы в разработке. На рынок выходит большое количество похожих по функциональным возможностям продуктов, и чтобы выдерживать конкуренцию, необходимо постоянно модифицировать, развивать и дорабатывать готовые решения. Даже если программа изначально написана без ошибок, после внесения изменений или доработок велик шанс что-то "сломать". А если не заметить дефект на ранних стадиях его появления, цена исправления ошибки будет намного выше. Кроме того, компания может понести репутационные потери.

Программы стали сложнее. Самые популярные подходы к созданию программного обеспечения — с использованием клиент-серверной архитектуры, архитектурного стиля REST и микросервисов — требуют проведения тщательного интеграционного и системного тестирования. С увеличением сложности программ увеличилось и количество тестов, а значит и стоимость проведения тестирования. Проводить проверки вручную стало невыгодно по экономическим соображениям.

Контроль над изменениями приобрел доминирующую степень значимости для сохранения качества программ. Получил популярность подход CI/CD (непрерывная интеграция/непрерывная доставка), при котором задачи сборки, публикации, тестирования продукта полностью или частично автоматизированы. В рамках этого подхода особую пользу принес запуск автотестов на пулл-реквестах, реализуя дополнительно автоматические быстрые проверки перед попаданием кода разработчиков в главный репозиторий.

Автотесты — инструмент, дающий возможность оперативного предоставления информации заинтересованным лицам о качестве продукта в текущий момент времени. Автоматизация позволяет снизить нагрузку на членов команды, разработчиков и тестировщиков, освободив их от выполнения рутинных задач.

Существует пять наиболее значимых этапов жизненного цикла про-

граммного обеспечения:

— спецификация,
— проектирование,
— кодирование,
— отладка,
— сопровождение.

Этап, связанный с поиском ошибок, считается наиболее дорогостоящим. Затраты на него могут даже превосходить в сумме другие этапы жизненного цикла. Особенно это касается систем в сфере здравоохранения, финансовых операций и обеспечения безопасности человека.

Принимая во внимание сказанное выше, становится понятно, почему в последнее время область автоматизации тестирования стала набирать популярность. Начали появляться удобные, интуитивно понятные фреймворки автоматизации, позволяющие легко писать автотесты. Появились научные статьи, книги и исследования на данную тему. Развиваются профессиональные сообщества, в интернете появляется всё больше обучающей информации. Компании готовы внедрять эту практику у себя на проектах, понимая её эффективность.

Цель бакалаврской работы — исследование проблемы автотестирования веб-приложений и создание фреймворка для написания автоматизированных тестов для веб-приложений, используя современные подходы.

Одним из современных и удобных фреймворков является Selenium, который способен работать в среде языка Java и является лидером по популярности в сфере автоматизации Web UI в 2023 году по данным исследования статистики сайта hh.ru.

Поставленная цель определила **следующие задачи**:

1. Изучить теоретический материал по проблеме;
2. Написать тест-план автоматизации тестирования;
3. Выбрать технологии, с помощью которых будет написан фреймворк автотестов на основе Selenium и языка Java;
4. Разработать этот фреймворк;
5. Пошагово показать как использовать созданный фреймворк на примере написания автотеста.

Методологические основы автоматизированного тестирования и, в

частности, разработки автотестов на Selenium представлены в работах Г. Майерса[1], Р. Блэка[2], У. Гундеча[3], А. А Романова[5], Ю. Б Поповой[6].

Практическая значимость бакалаврской работы. В ходе выполнения выпускной квалификационной работы был разработан фреймворк для написания автотестов, с помощью которого можно создавать автоматизированные тесты для веб-приложения с использованием Selenium.

Структура и объем работы. Бакалаврская работа состоит из введения, 3 разделов, заключения, списка использованных источников, приложения с кодом программы и цифрового носителя с кодом программы. Общий объем работы — 51 страница, из них 40 страниц — основное содержание, включая 16 рисунков, список использованных источников информации — 21 наименование.

КРАТКОЕ СОДЕРЖАНИЕ РАБОТЫ

Первый раздел «Анализ проблемы автоматизированного тестирования» посвящен теоретическому исследованию проблемы автоматизированного тестирования, а также смежных с этой проблемой тем.

В подразделе 1.1 приведено описание особенностей процессов разработки веб-приложений, архитектурного стиля REST, Agile методологии[9] и процессов CI/CD как наиболее популярных подходов к созданию веб-приложений.

Подраздел 1.2 посвящен сравнению двух подходов: ручного и автоматизированного тестирования. Представлены описания подходов, существующих в рамках ручного и автоматизированного тестирования, выявлены плюсы и минусы. Дан обзор API[7], UI, исследовательского видов тестирования.

В подразделе 1.3 рассмотрены цели и задачи автоматизированного тестирования.

В подразделе 1.4 сделан акцент на преимуществах и недостатках автоматизированного тестирования в целом.

В подразделе 1.5 рассмотрена целесообразность применения автотестирования, в каких случаях стоит прибегать к автоматизации тест-кейсов, и когда она может оказаться нерациональной.

В подразделе 1.6 приведен обзор экономической эффективности автотестов, способы её подсчёта и основная формула расчёта.

В подразделе 1.7 представлен обзор существующих способов автоматизации тестирования и имеющихся библиотек. Рассмотрена пирамида тестирования 1, рекомендуемое соотношение Unit, API и UI(E2E) тестов согласно пирамиде.

Рисунок 1 – Пирамида тестирования

В подразделе 1.8 приведён анализ возможностей языка Java в контексте выбранной проблемы. Рассмотрены требования, предъявляемые к автотестам, преимущества Java, позволяющие реализовывать эти требования. Перечислены фреймворки автотестирования, которые поддерживает Java[4].

В подразделе 1.9 рассмотрен фреймворк Selenium, его основные составляющие, основные преимущества и недостатки. Приведены примеры фреймворков, реализованных на базе Selenium.

В подразделе 1.10 описаны способы повышения надежности автотестов.

В подразделе 1.11 приведены техники тест-дизайна при автоматизации тестирования (тестирование потока управления, тестирование потока данных, разбиение на классы эквивалентности, анализ граничных значений, попарное тестирование, таблицы принятия решений, диаграмма изменения состояний, тестирование пользовательских сценариев)[8].

В подразделе 1.12 рассмотрена приоритизация тест-кейсов в контексте автоматизации тестирования.

Второй раздел «Разработка тест-плана по автоматизации тестирования» посвящен понятию тест-план, дано определение и рассмотре-

ны ключевые секции тест-плана:

1. Цель;
2. Области, подвергаемые тестированию;
3. Области, которые не будут подвергаться тестированию;
4. Компоненты 3х лиц;
5. Тестовая стратегия;
6. Критерии: приемочные, критерии начала и окончания тестирования, критерии возобновления тестирования, критерии завершения тестирования;
7. Ресурсы: программные, аппаратные, человеческие, временные, финансовы;
8. Расписание;
9. Роли и ответственности;
10. Оценка рисков;
11. Документация.

К каждой секции тест-плана написан пример ее создания.

Третий раздел «Реализация тест-плана» посвящен тому, как данный план можно реализовать на основе использования фреймворка, созданного поверх фреймворка Selenium.

Для написания фреймворка был выбран сайт `https://ft-sandbox.workbench.lanit.ru/`. Функционал сайта позволяет создавать тикеты в службу поддержки, указывая описание проблемы, приоритет, срок исполнения, прикреплять файлы, указать адрес электронной почты и некоторые другие данные. Также можно просматривать тикет, его статус и основные данные. На сайте предусмотрена авторизация. В случае, если пользователь обладает правами администратора, ему становится доступен список всех созданных тикетов.

В подразделе 3.1 были описаны технологии, выбранные для реализации фреймворка:

1. Язык Java, Java SDK версии 11.0.10;
2. Среда разработки — IntelliJ IDEA;
3. Maven версии 3.8.1;
4. Allure версии 2.13;
5. Паттерн Page Object;

6. Selenium WebDriver версии 3.141.5;
7. JUnit версии 5.8.2;
8. библиотека TypeSafe версии 1.4.1;
9. вкладка Инструменты разработчика внутри браузера.

В подразделе 3.2 показано, как настроить проект в IntelliJ IDEA и использовать Maven, а также подключить зависимости в файле pom.xml. Приведен пример подключения библиотеки Selenium.

В подразделе 3.3 говорится о создании конфигурационного файла и интерфейса для его чтения. Была использована библиотека TypeSafe. В конфигурационном файле описываются параметры пользователей, такие как логин, пароль, права доступа и некоторые другие. Также хранится ссылка на веб-приложение, которое будет тестироваться. Создан интерфейс *ConfigInterface* для чтения конфигурационного файла.

В подразделе 3.4 описано создание базовых классов. В классе *BaseTestSelenium* происходит работа с драйвером браузера. Описано создание методов *setUp()* и *tearDown()* для настройки и завершения работы с драйвером браузера. Создан абстрактный класс *BasePage*, который служит основой написания тестовых классов.

В подразделе 3.5 показано, как искать элементы на веб-страницах с помощью вкладки Инструменты разработчика внутри браузера. Также описано, как программно создавать элементы страницы внутри тестовых классов.

В подразделе 3.6 говорится об использовании PageFactory для инициализации элементов на странице внутри создаваемого фреймворка.

В подразделе 3.7 описано создание дополнительных переиспользуемых методов на тестовой странице для того, чтобы сделать сами автотесты более компактными.

В подразделе 3.8 показано создание страницы тестирования авторизации *AuthorizationPage*.

В подразделе 3.9 говорится о создании классов-помощников. В частности, был создан класс *StringModifyHelper*, помогающий в работе со строками.

В подразделе 3.10 описано создание тестового класса *AllTicketsPage*, содержащего все тикеты тестируемого веб-сайта.

В подразделе 3.11 был создан класс *TicketInfoPage*, описывающий создаваемый на тестируемом веб-сайте тикет.

В подразделе 3.12 сказано о выносе используемых строковых констант в отдельный класс *TestConstants*.

В подразделе 3.13 показано написание автотеста на основе использования созданного фреймворка. Был автоматизирован следующий сценарий:

1. Заполнение полей формы создания тикета и его отправка (см. рисунок 2).

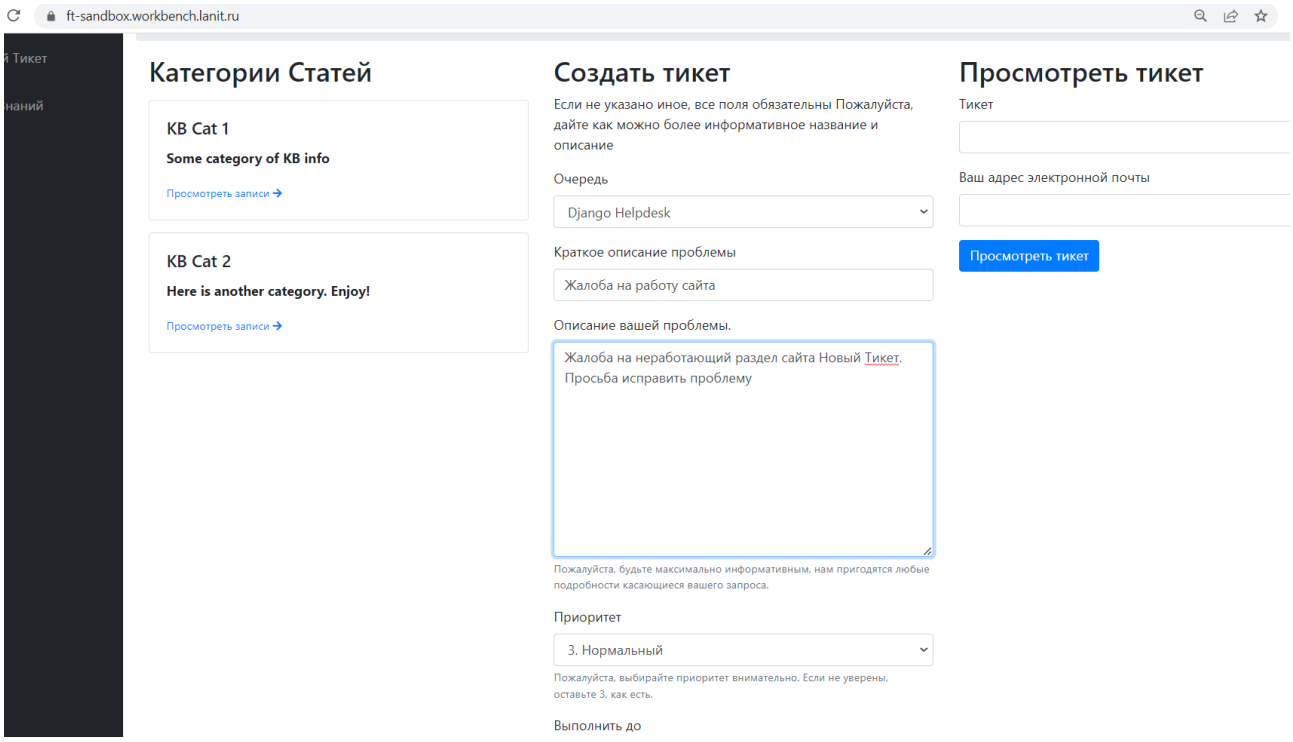

Рисунок 2 – Создание тикета

2. Авторизация пользователем с правами администратора (см. рисунок 3).

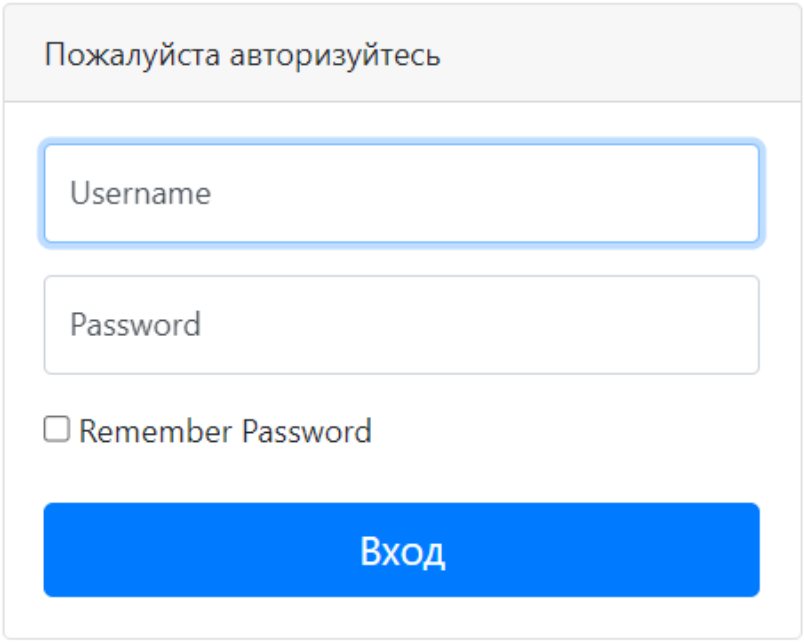

Рисунок 3 – Авторизация

3. Поиск созданного тикета в списке всех тикетов по уникальному описанию (см. рисунок 4).

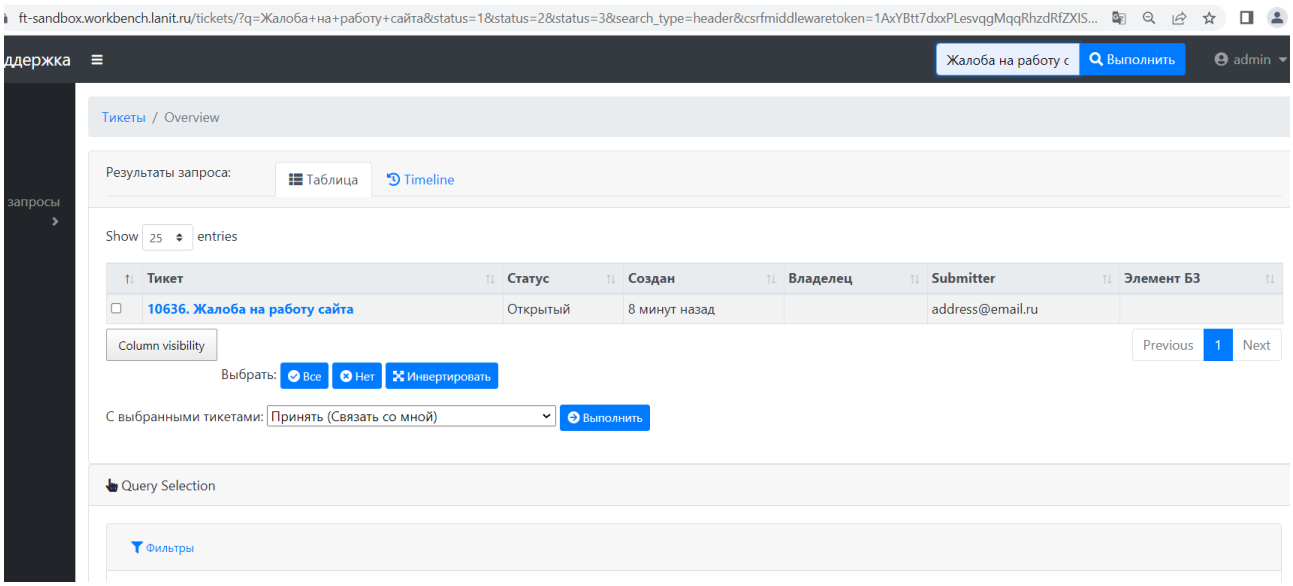

Рисунок 4 – Поиск созданного тикета

4. Проверка корректности полей email, описание и title найденного тикета (см. рисунок 5).

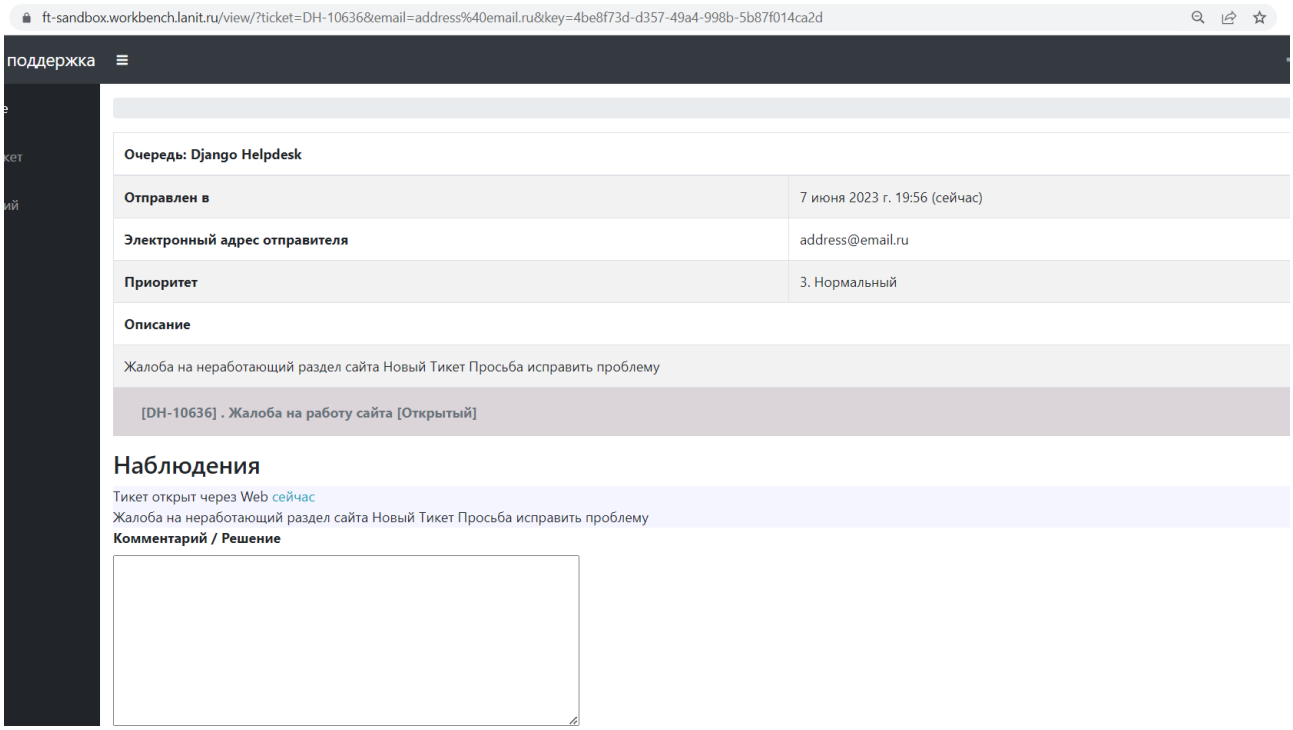

Рисунок 5 – Проверка корректности полей созданного тикета

Благодаря использованию фреймворка, код автотеста получился компактным и легко читаемым.

```
@Test
public void verifyTicket(){
String titleValue = getModifiedUniqueString(TestConstants.TICKET_TITLE);
String bodyValue = TestConstants.MESSAGE_DATA;
String userEmail = TestConstants.TICKET_EMAIL;
MainAppPage mainAppPage = new MainAppPage();
TicketInfoPage ticketInfoPage = mainAppPage.createOneTicket(titleValue, bodyValue,
    userEmail)
.openAuthorizationPage()
.authorize(ConfigInterface.NATALIA_LOGIN, ConfigInterface.NATALIA_PASSWORD)
.findCreatedTicket(titleValue);
Assertions.assertTrue(ticketInfoPage.getTicketTitle().contains(titleValue));
Assertions.assertEquals(ticketInfoPage.getData(), TestConstants.MESSAGE_DATA);
Assertions.assertEquals(ticketInfoPage.getValueEmail(), TestConstants.TICKET_EMAIL);
}
```

Показан результат запуска автотеста (см. рисунок 6). Тест успешно пройден менее, чем за 6 секунд.

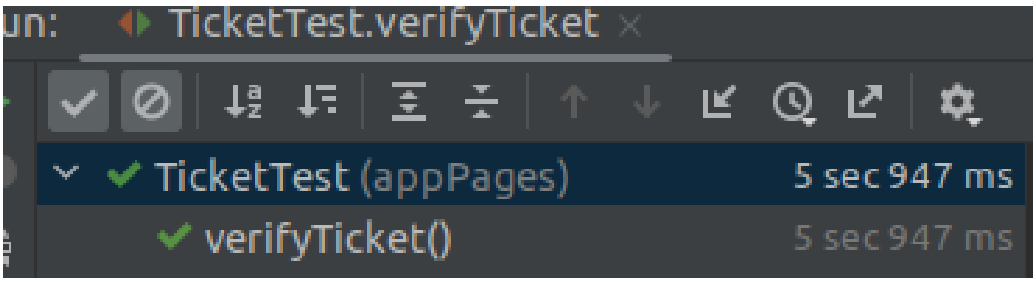

Рисунок 6 – Результат запуска автотеста

Были сделаны выводы по главе.

На практике были выявлены **преимущества автоматизированного тестирования** в сравнении с ручным:

— Скорость выполнения автотестов достаточна высока. Тест выполнился менее, чем за 6 секунд. По сравнению с прохождением этого сценария, где требуется заполнение многих полей, вручную, происходит существенная экономия времени.
— Входные данные для теста легко изменить.
— Можно использовать разные наборы входных параметров, писать параметризованные тесты, что увеличит тестовое покрытие.
— Тестировщику нужно только запустить скрипт, все остальные действия за него выполнит программа.
— Легкость добавления новых тестов при условии грамотного написания фреймворка и использования паттернов программирования в коде.

Были замечены следующие **недостатки автотестов**:

— Разработка, как и поддержка, автотестов требует большого количество времени и трудозатрат. Довольно высокий порог вхождения в написание автотестов.
— Чувствительность к среде, и часто проблемы с ее настройкой.
— Автоматизация не может полностью заменить ручное тестирование. Некоторые тест-кейсы в принципе невозможно автоматизировать.

ЗАКЛЮЧЕНИЕ

В настоящей работе рассмотрена проблема написания автотестов для веб-приложений с использованием Java и Selenium. Чтобы решить эту пробле-

му, был создан тест-план, который дает представление о том, что будет подлежать автоматизации, в какие сроки, дает информацию об имеющихся ресурсах для выполнения задачи, и многое другое. Составлением тест-плана часто занимаются тест-менеджеры или ведущие тестировщики. Владелец продукта и менеджер проекта могут также участвовать в процессе.

Для реализации автотестов веб-приложения был написан фреймворк автотестирования. Этот фреймворк удобен в использовании тем, что реализует паттер Page Object, и описывает каждую страницу в отдельном классе, благодаря чему находить сущности в коде проекта становится проще. Кроме того, такой фреймворк легко поддерживать, так как достаточно изменить локаторы в одном месте проекта, а не в каждом тестовом методе или классе. Чтобы продемонстрировать использование фреймворка, был написан автотест, проверяющий создание тикета в системе поддержки пользователей. Автотест успешно выполнился за довольно короткий промежуток времени — около 6 секунд. Код автотеста выглядит компактным и легким для понимания, благодаря использованию паттернов и фреймворка Selenium, предоставляющего удобные методы работы с драйвером браузера, а также фреймворку JUnit, реализующему проверки (так называемые assert-ы).

Применение автоматизации тестирования при грамотном подходе помогает снизить стоимость проведения тестирования, повысить качество продукта, дать оперативную информацию о состоянии продукта по запросу заинтересованных лиц, снизить нагрузку на членов команды, разработчиков и тестировщиков, освободив их от выполнения рутинных задач. Автотестирование — перспективное и относительно новое направление, которое уже набрало популярность и, несомненно, будет продолжать развиваться в будущем.

Основные источники информации:

1. Майерс Г. Искусство тестирования программ. // Москва: Диалектика, 2012. – 205 с.
2. Блэк Р. Ключевые процессы тестирования. // Москва: Лори, 2012. – 121 с.
3. Gundecha U. Selenium Testing Tools Cookbook. // Москва: Packt Publishing, 2015. – 231 с.
4. Эккель Б. Философия Java. // 4-е изд. СПб.: Питер , 2019. – 1168 с.
5. Романов А. А. Тестирование программного обеспечения. Лабораторный

практикум. // Ульяновск: УлГТУ, 2022. – 46 с.
6. Попова Ю. Б. Тестирование и отладка программного обеспечения. // Минск: БНТУ, 2020. – 66 с.
7. Лоре А. Проектирование веб-API. // Москва: ДМК Пресс, 2020. – 440 с.
8. Липаев В. В. Тестирование компонентов и комплексов программ. // Санкт-Петербург: Синтег, 2010. – 400 с.
9. Вольфсон Б. Гибкие методологии разработки. // Санкт-Петербург: Питер, 2012, 97 с.
10. Куликов С. С. Тестирование программного обеспечения. Базовый курс. // Минск: Четыре четверти, 2020. – 312 с.