Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Уральский федеральный университет
имени первого Президента России Б.Н. Ельцина»
Институт радиоэлектроники и информационных технологий - РТФ
Школа бакалавриата

ДОПУСТИТЬ К ЗАЩИТЕ ПЕРЕД
ГЭК
РОП 09.03.01 Спиричева Н.Р.

_____
(подпись)

«_____»_____
2025 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
Пояснительная записка

Разработка и интеграция системы автоматизированного тестирования для веб-приложения

Руководитель: _____ Соловьёва Н.В.
подпись ФИО

Консультант _____
подпись ФИО

Нормоконтролёр _____
подпись ФИО

Студент группы РИ-410910 _____ Ф.И.О Коноваленко Т.А.
подпись

Екатеринбург
2025

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«Уральский федеральный университет имени первого Президента России Б.Н. Ельцина»
Институт радиоэлектроники и информационных технологий - РТФ
Школа бакалавриата
Направление подготовки 09.03.01 Информатика и вычислительная техника
Образовательная программа Информатика и вычислительная техника

УТВЕРЖДАЮ
РОП 09.03.01 Спиричева Н.Р.
«02» декабря 2024 г.

ЗАДАНИЕ

на выполнение ВКР

студента Коновалова Тимура Артёмовича **группы** *РИ-410910*
(фамилия, имя, отчество)

1. Тема ВКР Разработка и интеграция автоматизированной системы тестирования для веб-приложения
Утверждена распоряжением по институту ИРИТ-РТФ от «02» декабря 2024 г. № 33.02-05/334

2. Руководитель Соловьёва Наталья Владимировна зам.нач. отдела, доцент, к.т.н
(Ф.И.О., должность, ученое звание, ученая степень)

Консультант
_____
(Ф.И.О., должность,организация)

3. Исходные данные к работе Материалы, полученные в ходе преддипломной практики, стандарты предприятия, техническая документация

4 Содержание пояснительной записки
Реферат, Введение, Основная часть, Заключение

5. Перечень демонстрационных материалов
Презентация

6. Календарный план

№ п/п	Наименование этапов выполнения работы	Срок выполнения этапов работы	Отметка о выполнении
1.	*Изучение аналогов, а также подходов для автоматизированного тестирования.*	до 28.02.2025 г.	выполнено
2.	*Написание основных сценариев для тестирования*	до 07.03.2025 г.	выполнено
3.	*Написание верхнеуровневых и модульных тестов и разработка системы автомтизированного тестирования*	до 31.03.2025 г.	выполнено
4.	*Оформление пояснительной записки*	до 31.05.2025 г.	выполнено

Руководитель _____ Соловьева Н.В
(подпись) Ф.И.О.

Задание принял к исполнению 02.12.24
дата

7. ВКР закончена «31» 05 2025 г. считаю возможным допустить _____ к защите его ВКР в Государственной экзаменационной комиссии.

Консультант_____ _____
(подпись) Ф.И.О.

Руководитель_____ Соловьева Н.В.
(подпись) Ф.И.О.

8. Допустить _____ к защите ВКР в Государственной экзаменационной комиссии (Распоряжение ИРИТ-РТФ №_____ от «__» ___ 2025г.).

РОП 09.03.01 _____

_____
(подпись)

Ф.И.О.

РЕФЕРАТ

Выпускная квалификационная работа бакалавра 60 с., 2 рис., 26 источников, 4 приложения.

РАЗРАБОТКА И ИНТЕГРАЦИЯ АВТОМАТИЗИРОВАННОЙ СИСТЕМЫ ТЕСТИРОВАНИЯ ДЛЯ ВЕБ-ПРИЛОЖЕНИЯ.

Цель работы – создание и интеграция автоматизированной системы тестирования для веб-приложения

В ходе работы были подробно изучены механизмы тестирования веб-приложений, а также основные виды их архитектур. Написаны тестовые сценарии, проверяющие различные аспекты веб-приложения. Реализованы модульные и функциональные тесты. Разработана система формирования отчётов на основании результатов выполнения тестов. Автоматизирован процесс запуска тестов и формирования отчетов, после коммита в определенную ветку рабочего репозитория.

Результатом работы является автоматизированная система тестирования.

Выпускная квалификационная работа выполнена в текстовом редакторе Microsoft Word и представлена в электронном формате.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире цифровые технологии развиваются стремительными темпами, а веб-приложения становятся все более сложными и многофункциональными. Они включают в себя множество компонентов: динамические интерфейсы, распределенные серверные системы, интеграции со сторонними API, сложную бизнес-логику и высокие требования к безопасности и производительности. В таких условиях традиционные методы ручного тестирования оказываются недостаточно эффективными: они требуют значительных временных и трудовых затрат, а также не всегда способны обеспечить достаточное покрытие тестами.

Рост сложности веб-приложений приводит к увеличению числа потенциальных уязвимостей и ошибок, которые могут негативно сказаться на пользовательском опыте, безопасности и стабильности системы. Особенно критично это для проектов с частыми обновлениями, где изменения в одном модуле могут неожиданно повлиять на работу других компонентов. В таких условиях автоматизация тестирования становится не просто полезным инструментом, а необходимостью, позволяющей: сократить время проверки функциональности, повысить точность обнаружения ошибок, минимизировать человеческий фактор и снизить затраты на поддержку качества продукта.

Цель работы - разработать и интегрировать автоматизированную систему тестирования для веб-приложения, способную эффективно проверять его функциональность на разных уровнях.

Для достижения данной цели планируется выполнение следующих шагов:

1) Изучить существующие решения и подходы для автоматизации тестирования.

2) Спроектировать сценарии для тестирования пользовательского графического интерфейса.

3) Разработать программу для проверки работоспособности приложения с формированием отчета.

4) Настроить и развернуть систему автоматизированного тестирования.

5) Применить систему автоматизированного тестирования для тестирования веб-приложения на практике.

Объект исследования – процесс тестирования веб-приложений.

Предметом исследования являются методы и инструменты для автоматизированного тестирования.

1 Анализ предметной области

1.1 Основные виды архитектур web-приложений

За последние десятилетия веб-приложения очень сильно развились и прошли путь от простых веб-страниц до полноценных программных решений., обладающих сложной архитектурой и богатой функциональностью. Их ключевой особенностью является высокая степень интерактивности и возможность динамического обновления контента без необходимости полной перезагрузки страницы.

Это стало возможным благодаря развитию JavaScript-фреймворков и появлению новых веб-стандартов. Современные веб-приложения могут успешно конкурировать с их полноценными версиями по функциональности, при этом сохраняя все преимущества веб-платформы: кроссплатформенность, легкую доступность и простоту обновлений.

Архитектура веб-приложений существенно влияет на подходы к их тестированию. В современной разработке преобладают три основные модели.

1.1.1 Монолитная архитектура

Монолитная архитектура представляет собой традиционный подход к разработке веб-приложений, где все компоненты системы (интерфейс пользователя, бизнес-логика и доступ к данным) объединены в единое целое [1]. Основные характеристики:

– все части приложения разрабатываются, тестируются и развертываются как единое целое;

– простота начальной разработки и отладки;

– единая кодовая база упрощает процесс разработки на ранних этапах;

– проблемы с масштабированием при росте проекта;

– затрудненное внедрение новых технологий в устоявшуюся кодовую базу;

Типичные примеры: традиционные PHP-приложения (например, WordPress), Ruby on Rails приложения.

1.1.2 Архитектура Single Page Application

Архитектура SPA (Single Page Application) представляет собой современный подход к разработке веб-приложений, где загрузка страницы происходит один раз, а все последующие обновления контента выполняются динамически через API [2]. Основные особенности:

– клиентская часть полностью отделена от серверной;

– сервер выступает в роли поставщика данных через API (обычно REST или GraphQL);

– навигация между разделами приложения происходит без перезагрузки страницы;

– богатая интерактивность и плавность работы;

– первоначальная загрузка может занимать больше времени;

Примеры приложений: Gmail, Trello, Facebook.

1.1.3 Микросервисная архитектура

Микросервисная архитектура представляет собой подход, при котором приложение разбивается на множество небольших, слабо связанных между собой сервисов [3]. Ключевые характеристики:

– каждый сервис отвечает за определенную бизнес-функцию;

– сервисы общаются между собой через четко определенные API;

– возможность независимого масштабирования отдельных компонентов;

– разные сервисы могут использовать различные технологии;

– существенно усложняет процесс разработки и тестирования;

Примеры: Netflix, Uber, Amazon.

Тестирование веб-приложений требует комплексного подхода и включает несколько ключевых аспектов.

1.2 Виды тестирования web-приложений

1.2.1 Функциональное тестирование

Функциональное тестирование веб-приложений представляет собой систематическую проверку соответствия разрабатываемого продукта заявленным требованиям и спецификациям. Этот вид тестирования фокусируется на проверке бизнес-функций приложения и их соответствии ожиданиям конечных пользователей. Основная цель - убедиться, что все

функции работают так, как было задумано разработчиками [4]. Данный вид тестирования проходит в несколько этапов:

1) Модульное тестирование представляет собой фундаментальный уровень проверки качества приложения, фокусирующийся на изолированном тестировании отдельных компонентов системы. Основная цель этого подхода заключается в верификации корректности работы минимальных логических единиц кода - функций, методов или классов [5]. Особое внимание уделяется проверке обработки граничных условий и исключительных ситуаций, что позволяет выявить потенциальные проблемы на самых ранних этапах разработки. Для обеспечения чистоты экспериментов широко применяются mock-объекты, которые заменяют реальные зависимости тестируемого модуля. Характерной чертой модульного тестирования является крайне высокая степень автоматизации, что позволяет легко интегрировать его в процесс непрерывной интеграции.

2) Интеграционное тестирование, которое переводит фокус с отдельных компонентов на их взаимодействие. Этот уровень тестирования специально разработан для выявления проблем, возникающих при объединении модулей в единую систему. Ключевыми аспектами являются проверка API-интерфейсов, анализ корректности межмодульных взаимодействий и тестирование интеграции с внешними сервисами. Особое значение имеет исследование потоков данных между различными компонентами системы, что позволяет обнаружить несоответствия в форматах данных или протоколах обмена [6]. Важной задачей интеграционного тестирования становится выявление проблем совместимости, которые часто остаются незамеченными при изолированной проверке модулей.

3) Системное тестирование представляет собой комплексную проверку всего приложения как единого целого. На этом уровне

осуществляется тестирование сквозных бизнес-процессов, охватывающих всю систему от начала до конца. Основная цель - убедиться, что все компоненты, успешно прошедшие модульное и интеграционное тестирование, корректно работают в комплексе [7]. Тестирование проводится в условиях, максимально приближенных к промышленной эксплуатации, что позволяет оценить реальное поведение системы под нагрузкой. Важным аспектом является проверка соответствия системным требованиям, включая производительность, надежность и безопасность работы приложения в целом.

4) Приемочное тестирование, проводится непосредственно перед выпуском продукта в эксплуатацию. Этот уровень проверки отличается тем, что выполняется с непосредственным участием заказчика или представителей конечных пользователей. Основная задача - подтвердить, что разработанная система в полной мере соответствует всем бизнес-требованиям и ожиданиям заинтересованных сторон [7]. В ходе приемочного тестирования проводится комплексная оценка готовности решения к промышленной эксплуатации, включая анализ соответствия реального поведения системы заявленным в контракте условиям. Результаты этого тестирования становятся основанием для принятия решения о выпуске продукта или необходимости доработок.

1.2.2 Нефункциональное тестирования

Нефункциональное тестирование является критически важной частью процесса обеспечения качества веб-приложений. В отличие от функционального тестирования, которое проверяет "что делает система",

нефункциональное тестирование отвечает на вопрос "как система это делает" [8]. Основные направления нефункционального тестирования:

– Тестирование производительности включает несколько ключевых аспектов. Нагрузочное тестирование позволяет определить предельные возможности системы при различных уровнях нагрузки. Оно помогает выявить "узкие места" в архитектуре приложения. Стресс-тестирование проверяет поведение системы в экстремальных условиях, превышающих нормальные рабочие нагрузки. Это важно для оценки устойчивости системы в пиковые периоды [9].

– Тестирование стабильности (так называемое "тестирование на выносливость") проводится в течение длительного периода времени для выявления таких проблем как утечки памяти или постепенная деградация производительности. Бенчмаркинг представляет собой сравнительный анализ производительности системы с конкурентными решениями или предыдущими версиями продукта [9].

– Тестирование безопасности начинается с проверки на наличие уязвимостей из списка OWASP Top 10 [10], который включает наиболее распространенные и опасные уязвимости веб-приложений. Особое внимание уделяется тестированию механизмов аутентификации и авторизации, включая проверку стойкости паролей, работу сессий и систему прав доступа.

– Тестирование удобства использования (UX/UI) включает юзабилити-тестирование с привлечением реальных пользователей, что позволяет выявить проблемы в интерфейсе, незаметные разработчикам. Проверка соответствия гайдлайнам платформ обеспечивает единообразие интерфейса и привычное для пользователей поведение элементов [9].

– Кроссплатформенное тестирование гарантирует корректную работу приложения в различных средах исполнения. Кросс-браузерное тестирование охватывает проверку работы в основных браузерах, включая Chrome, Firefox, Safari и Edge, а также их различные версии. Особое внимание уделяется мобильным браузерам и особенностям рендеринга страниц в них [11].

1.2.3 Регрессионное тестирование

Регрессионное тестирование служит критически важным механизмом обеспечения стабильности веб-приложения при внесении изменений в код. Его основная задача заключается в своевременном выявлении "регрессий" — ситуаций, когда новые правки нарушают ранее работающий функционал [12]. Особую актуальность этот вид тестирования приобретает в agile-средах с частыми релизами, где риск непреднамеренного нарушения существующей функциональности особенно высок. Современные подходы к регрессионному тестированию предполагают формирование "защитного пояса" из автоматизированных тестов, охватывающих ключевые сценарии использования приложения. Оптимальная стратегия включает как полные прогоны тестов перед крупными релизами, так и селективное тестирование затронутых функциональных областей при небольших изменениях. Эффективная организация регрессионного тестирования требует тщательного анализа рисков, грамотного приоритизации тест-кейсов и интеграции в процесс непрерывной поставки (CI/CD), что в совокупности позволяет значительно снизить вероятность появления критических дефектов.

1.3 Обзор существующих решений для тестирования

1.3.1 Решения для тестирования пользовательских интерфейсов

В последние годы тестирование пользовательских интерфейсов (UI) веб-приложений стало важным направлением в обеспечении качества разработок. Современные технологии требуют сложных подходов к проверке функциональности, совместимости и устойчивости приложений к высоким нагрузкам и динамическим изменениям. Поэтому индустрия активно использует автоматизацию тестирования с помощью специальных инструментов, каждый из которых обладает своими характеристиками, преимуществами и ограничениями. На текущий момент к ключевым управляемым разработчиками и тестировщиками технологиям относят такие как Selenium WebDriver, Cypress, Playwright и TestCafe.

Selenium WebDriver можно рассматривать как универсальный и фундаментальный инструмент для автоматизации тестирования UI. Это мощный open-source API, который позволяет осуществлять программное управление браузерами, полностью имитируя действия реального пользователя. Selenium изначально проектировался для поддержания кроссбраузерного подхода, поэтому он совместим с различными операционными системами и поддерживает широкий спектр языков программирования, таких как Java, Python, C#, JavaScript и Ruby. Основой инфраструктуры Selenium является архитектура клиент-сервер. API взаимодействует с драйверами конкретных браузеров посредством стандартизированного JSON Wire Protocol, что обеспечивает стабильную работу даже на сложных проектах с многослойной системой проверки [12].

Главным преимуществом Selenium является его гибкость: он позволяет тестировать приложения разного масштаба и структуры, начиная с простых формовых интерфейсов и заканчивая многостраничными веб-приложениями. Однако, несмотря на свою универсальность, этот инструмент требует от команды глубокого понимания работы браузерного окружения и навыков управления ожиданиями асинхронных операций. Также нужно учитывать, что Selenium не предоставляет встроенных решений для автоматического ожидания загрузки контента или стабильности элементов, что может стать вызовом для новичков.

Cypress считается более современной альтернативой Selenium благодаря своей уникальной архитектуре и подходу к выполнению тестов. Этот инструмент разработан с учетом разработчиков, что делает его особенно полезным для тех, кто ищет мощные отладочные возможности. Cypress работает непосредственно в браузерной среде, что позволяет ему перехватывать и модифицировать команды до их выполнения [13]. Такой подход обеспечивает детальный контроль над процессом тестирования, что делает сценарии предсказуемыми и удобными для анализа.

Одной из выдающихся особенностей Cypress является его интеграция с функцией "time-travel". Она позволяет разработчикам буквально «перематывать» время выполнения действий теста и изучать изменения состояния веб-приложения пошагово. Это существенно упрощает диагностику ошибок и исправление багов. Также выделяется специальная интегрированная среда автоматического управления динамическими действиями браузера, которая делает Cypress особенно полезным для тестирования продуктов, созданных на современных JavaScript-фреймворках, таких как React, Vue или Angular.

Тем не менее, Cypress не свободен от ограничений. Например, его работа ограничивается браузерами на базе Chromium, такими как Chrome и

Edge, что создает проблемы с тестированием кроссбраузерной совместимости. Несмотря на это, инструмент идеально подходит для создания и выполнения стабильных тестов в современных проектах.

Разработанный в Microsoft, Playwright сочетает в себе элементы традиционных решений вроде Selenium и современных подходов, характерных для Cypress. Инструмент позволяет работать с несколькими браузерными движками, такими как Chromium, WebKit и Firefox, что делает его особенно удобным в случае тестирования приложений, ориентированных на использование в различных операционных системах и браузерах. Инструмент поддерживает сложные сценарии тестирования и предоставляет уникальную возможность работы с множественными браузерными контекстами, что полезно для эмуляции действий нескольких пользователей или работы с изолированными сессиями.

Важной особенностью Playwright является функция Codegen — автоматической записи действий пользователя в формате тестового сценария. Это заметно ускоряет процесс создания тестов, что особенно полезно для внедрения автоматизации в быстро развивающихся проектах. Также Playwright предоставляет разработчикам детализированные инструменты для работы с API, позволяя прослушивать или изменять сетевые запросы, замещать данные для тестов или эмулировать нестабильные сценарии серверного отклика.

Однако Playwright может иметь ограниченное применение для устаревших проектов, где требуется поддержка таких браузеров, как Internet Explorer [14]. Несмотря на это, его использование особенно оправдано в условиях, где необходимо обеспечить высокую достоверность результатов и стабильность кроссбраузерного тестирования.

Особенностью TestCafe является архитектура, избавленная от внешних драйверов и зависимостей от WebDriver. Решение функционирует полностью независимо, что сокращает время и энергозатраты на настройку тестовой среды. Это также делает его удобным для использования командами с небольшим опытом автоматизации тестирования. Подход TestCafe включает встроенный прокси-сервер, который автоматически перехватывает и изменяет запросы и ответы между браузером и тестируемым приложением. Таким образом, инструмент показывает стабильные результаты даже в сложных динамических условиях.

TestCafe особенно хорошо работает с инновационными SPA (single-page application) благодаря встроенной системе ожидания загрузки динамического контента. Это позволяет тестам стабильно взаимодействовать с интерфейсами даже при интенсивных фоновых операциях. Также важно отметить, что TestCafe поддерживает браузеры без необходимости настройки дополнительных плагинов или внешних расширений.

Среди возможных ограничений TestCafe можно выделить сложности с кастомизацией тестов для особых нужд, что делает инструмент менее предпочтительным для сложных архитектур или высоконагруженных приложений [15]. Тем не менее, он остается идеальным выбором для быстрого выполнения тестов и проверки функциональности небольших продукта.

Основные особенности для каждого фреймворка приведены в таблице ниже.

Таблица 1 – сравнение фреймворков

Фреймворк	Языки программирования	Браузе-ры	Преимущества	Недостатки
Selenium	Java, Python, C#, JavaScript	Все	Гибкость, активное сообщество	Сложность освоения, хрупкость тестов
Cypress	JavaScript, TypeScript	Chrome, Edge	Простота использования встроенная отладка	Не поддерживает Firefox и Safari
Playwright	JavaScript, Python, Java, C#	Chrome, Firefox, WebKit	Стабильность, открытый API	Молодой проект
TestCafe	JavaScript, TypeScript	Все	Простая настройка, автоматически е ожидания	Ограниченная кастомизация сложных тестов

Правильный выбор инструмента для автоматизации тестирования UI зависит от множества факторов. На первых этапах следует учитывать конкретный набор технологий, использованный в проекте, так как это напрямую повлияет на совместимость с выбранным инструментом. Например, если задействуются фреймворки нового поколения вроде React или Vue, более предпочтительным может стать Cypress или TestCafe.

Другой важный аспект — требования к кроссбраузерной поддержке, которые определяют использование Selenium или Playwright, если приложения должны быть совместимы с различными платформами. Масштаб и размер проекта также играют значимую роль, фиксируя внимание на производительности инструмента, его скорости и

стабильности. Например, для крупномасштабных приложений с интенсивным пользовательским взаимодействием Playwright станет наиболее подходящим выбором, в то время как Cypress идеально подходит для тесной отладки и тестирования отдельного функционала.

Поскольку ни один инструмент не способен универсально удовлетворить все потребности, современные команды тестировщиков всё чаще комбинируют решения. Например, для одновременного охвата различных браузеров можно использовать Playwright. В то же время Cypress остается лучшим выбором для тестирования интерактивных интерфейсов и анализа сложных анимаций. Такой подход позволяет эффективно распределять ресурсы и достигать оптимальной гибкости процессов тестирования.

1.3.2 Решения для модульного тестирования

Функциональное тестирование обеспечивает проверку работы всей системы или её отдельных частей на уровне взаимодействия с пользователем или других системных модулей. Оно гарантирует, что каждый компонент выполняет свои основные функции в соответствии с техническими и пользовательскими требованиями. Однако, чтобы добиться высокого качества программного обеспечения, важно не только тестировать функциональность на уровне общего взаимодействия, но и проводить более точечные проверки. Здесь на помощь приходит модульное тестирование, главной задачей которого является проверка работы отдельных модулей или функций программы изолированно от всей системы.

В языках программирования, таких как C++, модульное тестирование играет особую роль, так как оно позволяет выявлять ошибки в базовой

логике работы на ранних этапах разработки. Учитывая сложность C++ и его применение в высокопроизводительных или критически важных системах, тестирование отдельных компонентов, таких как классы, функции или небольшие модули, становится ключевым шагом в создании надежного кода

Модульное тестирование на C++ требует использования специализированных инструментов, которые упрощают процесс написания, выполнения и анализа результатов тестов. Среди наиболее популярных фреймворков для модульного тестирования в C++ выделяют Google Test (GTest), Catch2, Boost.Test и некоторые другие. Каждый из них имеет свои особенности, преимущества и подходы к организации тестов, поэтому выбор инструмента зависит от требований проекта, навыков команды и целевого функционала. Ниже подробно рассмотрены ключевые фреймворки.

Google Test – это один из самых мощных и распространённых фреймворков для модульного тестирования в C++. Разработанный и поддерживаемый Google, он стал стандартным выбором для большинства проектов, благодаря своей функциональности, гибкости и активному сообществу. Ниже представлены ключевые особенности данного фреймворка.

– Мощный API для создания тестов: GTest предоставляет понятный синтаксис для организации тестов и их запуска. Для написания тестов используется макрос TEST (TestSuiteName, TestName), который упрощает организацию набора тестов [16].

– Поддержка настройки исходного состояния: GTest позволяет задавать общее состояние для группы тестов с помощью стандартных классов контекста (SetUp и TearDown). Это особенно полезно для тестирования кода, зависящего от одинакового начального состояния [16].

– Параметризация тестов: Вы можете писать тесты с параметрами, чтобы проверять одну и ту же функцию с различными входными данными. Это упрощает написание проверок для функций с множеством вариантов использования [16].

– Отчёты и интеграция: GTest поддерживает генерацию отчетов в формате XML, что удобно для анализа в системах непрерывной интеграции (CI). Кроме того, GTest легко интегрируется с популярными сборочными инструментами, такими как CMake [16].

– Экосистема: Google Test широко используется в реальных проектах, имеет обширную документацию и множество примеров.

Преимущества:

– простота организации тестов и полезные утилиты для проверки условий;

– активное развитие проекта и официальная поддержка со стороны Google;

– подробные отчёты об ошибках при выполнении тестов;

Недостатки:

– стандартная библиотека GTest довольно объемна, что может усложнить использование в проектах с ограниченными ресурсами;

– начальная настройка библиотеки может быть сложной для новичков;

Catch2 – это современный, интуитивно понятный фреймворк для модульного тестирования, который активно используется разработчиками C++ благодаря простоте написания тестов и лаконичному дизайну. Он был создан как альтернатива более сложным инструментам, таким как GTest, и

идеально подходит для небольших и средних проектов. Далее рассмотрены основные особенности этого фреймворка [17].

– Единственный заголовочный файл: Catch2 распространяется как заголовочная библиотека, что устраняет необходимость в сложной установке и минимизирует зависимость от внешних источников.

– Простой синтаксис для тестов: Тестовые кейсы создаются с помощью макроса TEST_CASE, где можно легко описать сценарий тестирования. Такой подход снижает порог вхождения.

– Встроенные проверки совпадений: Catch2 предоставляет инструменты для проверки условий и результатов выполнения кода, упрощая процесс написания проверок.

– Поддержка BDD: Catch2 поддерживает Behavior-Driven Development (BDD), что позволяет писать тесты, ориентированные на качественное описание поведения системы.

– Кроссплатформенность: Catch2 работает на всех основных операционных системах, включая Windows, Linux и macOS.

Преимущества:

– минимальная настройка: библиотека не требует сложного создания проекта – достаточно подключить заголовок;

– удобочитаемость тестов: благодаря интуитивно понятному синтаксису, написание тестов становится простым и быстрым;

– подходит для небольших проектов, где важна скорость настройки;

Недостатки:

– может быть менее производительным на крупных тестовых наборах по сравнению с Google Test;

– поддержка параметризированных тестов менее удобна и функциональна, чем в GTest;

Boost.Test является одним из зрелых и богатых функционалом инструментов для модульного тестирования. Это часть большой Boost-библиотеки, которая включает множество модулей, полезных для разработки на C++. Ниже показаны основные особенности Boost Test:

– Интеграция с библиотекой Boost: Boost.Test идеально подходит для крупных проектов, которые уже используют Boost. Фреймворк тесно интегрирован с другими модулями этой экосистемы и обеспечивает мощную совместимость [18].

– Гибкие настройки тестов: Boost.Test предоставляет несколько уровней тестирования – от простых проверок до продвинутой конфигурации тестовых наборов для сложных приложений [18].

– Генерация отчётов: Утилита Boost.Test поддерживает производительные механизмы логирования и предоставляет подробные отчеты [18].

– Управление исключениями: Boost.Test включает мощные средства для проверки работы кода, использующего исключения, поэтому идеально подходит для тестирования сценариев, в которых важно выявить ошибки обработки исключений [18].

Преимущества:

– глубокая интеграция с частью экосистемы Boost;

– поддержка сложного тестирования, включая многопоточность;

– поддержка старых версий стандарта C++ (до C++98);

Недостатки:

– высокая сложность настройки и использования;

– устаревший синтаксис, что делает его менее интуитивным по сравнению с Catch2;

– библиотека достаточно «тяжёлая», что может увеличивать размер исполняемого файла;

Другие инструменты для тестирования C++

Doctest: Этот инструмент вдохновлён концепцией Catch2. Его ключевыми преимуществами являются высокая производительность и минималистичный синтаксис. Doctest также отлично подходит для быстрого прототипирования [19].

CppUnit: это аналог JUnit для C++. Инструмент хорош для старых проектов, однако менее активно поддерживается [20].

Фреймворки для модульного тестирования на C++ имеют свои особенности и сферы применения. Google Test выделяется своей мощью, универсальностью и практически промышленным стандартом. Catch2 привлекает лёгкостью использования и лаконичностью, что делает его отличным выбором для небольших проектов. Boost.Test подойдёт для опытных разработчиков, работающих в крупных экосистемах Boost. Выбор конкретного инструмента зависит от задач проекта, требований к тестированию, а также квалификации команды.

2. Современные платформы для автоматизированного тестирования

Автоматизированное тестирование превратилось в ключевой элемент процесса разработки программного обеспечения, позволяя повысить качество, ускорить циклы релизов и снизить затраты на поддержку продукта. С развитием технологий и ростом требований к масштабируемости и гибкости тестовых процессов появились специальные платформы, которые значительно облегчают создание, выполнение и поддержку автотестов.

Сегодня на рынке присутствует множество решений, которые предлагают широкий спектр возможностей — от простых инструментов записи и воспроизведения действий до комплексных систем с поддержкой искусственного интеллекта и интеграцией в DevOps-экосистемы. Выбор подходящей платформы часто зависит от специфики проекта, используемых технологий и структуры команды.

Далее будут рассмотрены основные категории данных платформ.

2.1 Коммерческие платформы

2.1.1 Tricentis Tosca

Tricentis Tosca — это одна из ведущих коммерческих платформ для автоматизированного тестирования, широко используемая в крупных и средних организациях. Главной особенностью Tosca является технология model-based testing, которая позволяет создавать цифровую модель

приложения. Вместо написания традиционных скриптов тестировщики конструируют абстрактные модели бизнес-процессов и пользовательских сценариев, а сама платформа на их основе автоматически генерирует тесты [21].

Это предоставляет значительные преимущества:

– Быстрая генерация тестовых сценариев для широкого спектра функциональных требований.

– Повышенная устойчивость тестов к изменениям интерфейса, поскольку модели отражают логику работы приложения, а не детали реализации.

– Инструменты для визуального моделирования и управления тестовыми процессами, что облегчает работу нетехнических специалистов.

– Поддержка end-to-end тестирования, охватывающего пользовательские интерфейсы, API, базу данных и интеграции.

– Встроенная функциональность управления тестовыми данными и тест-кейсами, что облегчает поддержание и анализ качества тестового покрытия.

Tosca также интегрируется с популярными системами управления проектами и CI/CD, что позволяет организовать автоматизацию тестирования в рамках полного цикла разработки и доставки программного продукта. Одним из ключевых преимуществ является возможность масштабировать тестовые сценарии для крупных проектов с большим количеством функциональных модулей.

2.1.2 Micro Focus UFT One

Micro Focus UFT One (Unified Functional Testing) — платформа, изначально разработанная для профессионалов. Она предназначена для автоматизации тестирования сложных корпоративных приложений, включая веб, десктоп, мобильные и API-интерфейсы [22].

Основные возможности UFT One включают:

– Мощный механизм распознавания объектов, способный точно идентифицировать элементы интерфейса даже в динамичных и сложных приложениях.

– Поддержка разнообразных технологий и платформ, включая HTML5, Java, .NET, SAP, Oracle, а также тестирование баз данных.

– Встроенные инструменты для автоматизации тестирования API, баз данных и пользовательских интерфейсов в единой интегрированной среде.

– Гибкий редактор скриптов на VBScript, который позволяет создавать кастомные и детализированные тестовые сценарии с возможностью расширения и интеграции с другими инструментами.

– Интеграция с инструментами управления тестированием, такими как Micro Focus ALM (Application Lifecycle Management), что повышает эффективность организации процессов обеспечения качества.

– Поддержка автоматической регрессионной проверки, позволяющая быстро выявлять и локализовать ошибки после изменений в коде приложения.

Особенность UFT One — возможность работы с корпоративными системами со сложной бизнес-логикой, где требуется глубинное

тестирование разных слоев программного продукта. Платформа широко применяется в банковской сфере, страховании, телекоммуникациях и других отраслях с высокими требованиями к качеству ПО.

2.2 Облачные платформы для автоматизированного тестирования

В отличие от традиционных коммерческих платформ, которые обычно требуют установки и настройки на локальной инфраструктуре компании, облачные решения предоставляют тестовые инструменты как сервис — полностью в облаке. Такой подход значительно упрощает запуск и масштабирование автоматизации тестирования, снижает потребность в самостоятельном администрировании и поддержке среды.

Облачные платформы позволяют пользователям создавать, запускать и анализировать тесты через веб-интерфейс, а управление инфраструктурой и обновлениями берёт на себя провайдер сервиса. Благодаря этому снижены затраты на начальное развёртывание и эксплуатацию, а также сокращено время выхода проекта на рынок.

Ниже рассмотрим два ярких представителя современных облачных решений — Mabl и Testim, которые активно используют технологии машинного обучения и искусственного интеллекта для повышения эффективности автотестирования.

2.2.1 Mabl

Mabl — это облачная платформа, предоставляющая инструмент для создания и поддержки автоматизированных тестов с помощью машинного обучения. Одна из ключевых особенностей Mabl — способность

автоматически адаптировать тесты при изменениях пользовательского интерфейса. Это позволяет существенно снизить трудоемкость обслуживания тестовой базы и ускоряет процесс обновления сценариев.

Платформа объединяет функциональное, регрессионное и тестирование производительности в едином цикле, обеспечивая всесторонний подход к обеспечению качества. Mabl легко интегрируется с популярными инструментами DevOps, такими как Jenkins, Bamboo, GitHub Actions, что делает возможной их бесшовную интеграцию в CI/CD [23].

Кроме того, Mabl предлагает удобный интерфейс с возможностью записи действий пользователя, что подходит даже для инженеров с минимальным опытом программирования. Аналитические возможности платформы включают детальные отчёты с рекомендациями по устранению выявленных дефектов и мониторингом стабильности тестов.

2.2.2 Testim

Testim также представляет собой облачное решение, активно использующее искусственный интеллект для оптимизации процесса автоматизации тестирования. Главной особенностью Testim является интеллектуальная стабилизация тестов: система автоматически подстраивает локаторы элементов пользовательского интерфейса при изменениях в структуре страниц, что предотвращает часто встречающиеся поломки тестов.

Для Agile-команд Testim предлагает быстрый старт через запись действий пользователей, позволяя быстро создавать первоначальные тесты. По мере повышения квалификации инженеры могут расширять и совершенствовать тесты, используя встроенный редактор кода. Такое

комбинирование упрощает переход от базовой автоматизации к более продвинутому уровню.

Кроме того, Testim поддерживает выполнение регрессионного и функционального тестирования, что обеспечивает гибкость в покрытии различных аспектов качества ПО [24]. Система интегрируется с основными системами управления проектами и CI/CD платформами, обеспечивая автоматическое выполнение тестов с анализом результатов и уведомлениями.

2.3 Решения с открытым исходным кодом

В корпоративной сфере всё большую популярность приобретают решения с открытым исходным кодом, которые сочетают гибкость кастомизации с экономической эффективностью. Такие инструменты позволяют компаниям создавать масштабируемые и настраиваемые тестовые окружения без необходимости больших затрат на лицензии, при этом обеспечивая профессиональный уровень автоматизации. Далее рассмотрим два хорошо зарекомендовавших себя представителя — Apache JMeter и Galen Framework.

2.3.1 Apache JMeter

Apache JMeter — это многофункциональный инструмент, первоначально разработанный для нагрузочного тестирования, но со временем получивший богатые возможности для функционального и API-

тестирования. Его популярность в корпоративной среде объясняется расширяемостью и масштабируемостью.

Ключевые возможности JMeter включают:

– Нагрузочное тестирование: моделирование большого количества виртуальных пользователей для проверки производительности и устойчивости приложений.

– Функциональное тестирование: проверка бизнес-логики через сценарии для веб-приложений, API, баз данных и других сервисов.

– Тестирование API: поддержка HTTP, REST, SOAP и других протоколов с возможностью создания сложных сценариев запросов и обработки ответов.

– Широкий набор плагинов: расширяют функционал и позволяют интегрировать JMeter практически с любым инструментом DevOps и управления тестами.

– Гибкие средства создания отчетности и анализа: визуализация результатов помогает быстро выявлять узкие места и сбои.

JMeter запускается на Java, что гарантирует совместимость с разнообразными платформами. Благодаря открытой архитектуре корпоративные команды могут адаптировать инструмент под свои задачи и создавать собственные расширения [25].

2.3.2 Galen Framework

Galen Framework — специализированный инструмент для тестирования адаптивного и кросс-браузерного пользовательского

интерфейса с акцентом на качество верстки. Он особенно полезен для проектов с жёсткими требованиями к внешнему виду интерфейса и точному соответствию дизайн-макетам [26].

Основные особенности Galen:

– Проверка расположения элементов и визуальных стилей: позволяет определить отклонения от заданных правил и макетов на различных разрешениях экранов.

– Использование простого языка описания спецификаций (.spec): тестировщики описывают расположение, размеры и другие характеристики элементов, включая условия адаптивности.

– Поддержка разнообразных браузеров и платформ: позволяет выполнять тесты на реальных и виртуальных устройствах, интегрируется с Selenium для реализации автоматических сценариев.

– Возможность интеграции с CI/CD: обеспечивает автоматическую проверку верстки при каждом коммите или сборке.

– Отчетность с визуальными доказательствами: включают скриншоты, показывающие найденные несоответствия и ошибки верстки.

Galen отлично подходит для QA-команд, которые уделяют повышенное внимание визуальному качеству и пользовательскому опыту, обеспечивая уверенность в корректном отображении интерфейсов на разных устройствах.

2.4 Обоснование создания собственной системы тестирования

Одной из главных причин, по которой использование коммерческих и облачных платформ оказалось невозможным, стала внутренняя политика компании, ограничивающая применение внешних сервисов из соображений безопасности и конфиденциальности данных. Такие ограничения существенно сужают выбор доступных инструментов и делают невозможным интеграцию облачных решений или коммерческого ПО без значительных изменений в инфраструктуре.

Второй, не менее важной причиной, является необходимость гибкого и глубокого контроля над процессами автоматизации, а также возможность адаптировать инструменты под специфические требования проекта и интегрировать их с уже существующими системами разработки и релиза. Многие готовые решения не предоставляют достаточной свободы модификаций, что осложняет внедрение нестандартных подходов и процессов.

Выходом из этой ситуации стала разработка собственной системы автоматизированного тестирования, которая полностью удовлетворяет требованиям безопасности и функциональным потребностям проекта. Поскольку в проекте уже использовалась платформа Gitea для управления исходным кодом, её встроенный инструмент Gitea Actions был логичным выбором для организации CI/CD процессов и автоматического запуска тестов.

Для функционального тестирования пользовательских интерфейсов и интеграций был выбран Selenium — проверенный временем и гибкий инструмент, обеспечивающий возможность эмуляции взаимодействия с браузером. Для модульного тестирования применён Boost Test, который актуален благодаря описанным ранее преимуществам: простоте

использования, мощным возможностям и хорошей интеграции с кодовой базой.

Таким образом, созданная платформа сочетает в себе надёжность, масштабируемость и высокую адаптивность, отвечая как корпоративным требованиям безопасности, так и высоким стандартам качества программного обеспечения.

3 Разработка и интеграция автоматизированной системы тестирования

3.1 Написание сценариев тестирования

Первым, основополагающим этапом работы по автоматизации тестирования стало создание подробных и хорошо структурированных сценариев, предназначенных для всесторонней проверки приложения. Для этого был проведён глубокий анализ функциональных возможностей самого программного продукта: изучена его логика работы, архитектурные особенности и механизмы взаимодействия между основными компонентами и модулями системы. Такой всесторонний подход был необходим для того, чтобы тесты максимально полно отражали реальную работу приложения и охватывали все ключевые бизнес-процессы.

Помимо анализа функционала, особое внимание было уделено изучению технических требований и спецификаций, где детально описано ожидаемое поведение системы в различных ситуациях – от стандартных рабочих сценариев до крайних случаев. Это позволило сформировать тестовые случаи, которые не только соответствуют поставленным задачам,

но и учитывают возможные проблемы и исключительные ситуации, что в свою очередь повышает надёжность тестирования.

Исходя из результатов анализа, были разработаны комплексные сценарии тестирования, которые условно можно разделить на две категории. Первые — позитивные сценарии, проверяющие корректную работу приложения в типичных и штатных условиях использования, гарантируя, что все функции выполняются согласно требованиям. Вторые — негативные сценарии, направленные на проверку поведения системы в ситуациях обработки ошибок, нестандартных и непредвиденных ситуациях, что важно для выявления уязвимостей и повышения устойчивости продукта.

Каждый тестовый сценарий был составлен с чётким соблюдением структуры, включающей четыре обязательных элемента:

1) Описание тестируемой функции: подробное и понятное объяснение функционального блока или бизнес-процесса, который проверяется данным тестом. Здесь отражается суть тестируемой возможности и условия её использования.

2) Шаги для воспроизведения: последовательное и детализированное описание действий, необходимых для выполнения теста. Это позволяет гарантировать, что тест можно воспроизвести и выполнить однозначно в любой момент, а также упрощает последующую автоматизацию.

3) Ожидаемый результат: конкретное описание того, что должно произойти после выполнения каждого шага, включая все визуальные и функциональные изменения в системе. Это помогает определить правильное поведение приложения и служит основой для оценки успешности теста.

4) Критерии успешного прохождения теста: чётко обозначенные условия, при выполнении которых тест считается пройденным, что исключает субъективность в оценке результатов и повышает объективность тестирования.

Строгое соблюдение этой структуры позволило на системном уровне организовать процесс тестирования, обеспечив максимальное покрытие функциональных требований проекта. Это, в свою очередь, значительно повысило качество продукта и способствовало своевременному обнаружению потенциальных ошибок, ещё на ранних этапах жизненного цикла разработки.

Пример тестового сценария приведен в приложении А.

3.2 Автоматизация тестовых сценариев

Для реализации задачи автоматизации тестирования был выбран мощный комплекс инструментов, способный эффективно организовать процесс создания, выполнения и сопровождения автоматизированных тестов. В состав выбранного стека вошли следующие компоненты:

pytest — современный и гибкий фреймворк на языке Python, предназначенный для организации и выполнения тестов различного уровня сложности. Благодаря расширяемости, поддержке тестового контекста и простой интеграции с другими инструментами, pytest стал основой для написания устойчивых и удобных в сопровождении автоматизированных тестов.

Selenium WebDriver — признанный отраслевой стандарт для автоматизации взаимодействия с пользовательским интерфейсом веб-приложений. Возможность управлять браузером программно, эмулировать действия пользователя и работать с динамическими веб-элементами делает

этот инструмент незаменимым при реализации функционального автоматизированного тестирования.

Процедура автоматизации была разбита на несколько логических этапов, которые обеспечили поэтапное и качественное внедрение автоматизации:

1) Подготовка тестового окружения

– Выполнена установка всех необходимых программных компонентов и библиотек, включая Python, pytest и веб-драйверы для поддерживаемых браузеров.

– Проведена конфигурация драйверов, позволяющая обеспечить стабильное взаимодействие с конкретным браузером в рамках тестирования — настройка путей, параметров запуска и совместимости.

– Создана структурированная архитектура проекта, в которой выделены отдельные модули для хранения тестовых сценариев, вспомогательных функций и конфигурационных файлов. Такая организация позволяет обеспечить удобство навигации, поддержки и масштабирования тестового кода.

2) Разработка автоматизированных тестов

– На основании ранее разработанных ручных тестовых сценариев выполнена их трансформация в программный код с использованием возможностей pytest. Это обеспечило автоматическое выполнение, параметризацию и удобный вывод результатов.

– Для каждого теста разработан контекст — специальные структуры для управления подготовкой и очисткой тестового окружения, а также манипуляциями с тестовыми данными и состояниями системы, обеспечивая независимость и повторяемость тестов.

– Организована модульная структура тестовых программ, что соответствует современным практикам поддержки и повторного использования кода, облегчая сопровождение и развитие тестовой базы в дальнейшем.

3) Интеграция с Selenium WebDriver

– Разработаны специализированные программы, реализующие автоматическое взаимодействие с элементами интерфейса.

– Применены эффективные методы ожиданий, гарантирующие стабильность выполнения тестов при динамически изменяющемся интерфейсе и асинхронных процессах в приложении.

– Внедрена система проверок, которая позволяет однозначно верифицировать соответствие полученных фактических результатов теста ожидаемым, что способствует обнаружению дефектов и отклонений.

Пример реализации автоматизированного теста приведён в приложении Б.

В итоге внедрение автоматизированного тестирования дало следующие ключевые преимущества:

– Существенное сокращение временных затрат на проведение регрессионного тестирования благодаря быстрому и автоматическому повторному прогону тестов без участия человека.

– Повышение точности и надёжности проверок, исключающее ошибки, связанные с человеческим фактором, а также многократное и последовательное воспроизведение тестов в одинаковых условиях.

3.3 Разработка модульных тестов

Следующим значимым этапом работы стала разработка модульных тестов, направленных на проверку корректности и стабильности работы отдельных компонентов приложения. Модульное тестирование играет ключевую роль в обеспечении качества кода, позволяя выявлять ошибки на самых ранних стадиях разработки и минимизировать риски при внесении изменений.

Для реализации поставленной задачи был выбран комплекс инструментов, способствующий эффективной организации и управлению процессом модульного тестирования, а также удобной интеграции тестов в существующий цикл сборки и разработки. В состав выбранного набора вошли:

– Boost.Test — основной фреймворк для модульного тестирования на языке C++, обладающий широкими возможностями для структурирования тестов, управления их выполнением и генерации подробных отчётов. Его стабильность и расширяемость делают его предпочтительным выбором для тестирования высоконагруженных или комплексных модулей.

– FakeIt — современная библиотека для создания mock-объектов и заглушек (stubs), позволяющая имитировать поведение зависимостей при тестировании отдельных компонентов. Использование FakeIt обеспечивает высокую изоляцию тестируемых модулей и делает возможным управление внешними эффектами, тем самым повышая точность и надёжность тестов.

– Инструменты CMake — для интеграции модульных тестов в процесс сборки проекта. Конфигурация CMake позволила автоматизировать этапы компиляции и запуска тестов, обеспечивая единый удобный рабочий процесс для разработчиков.

Работа по внедрению модуля модульного тестирования была организована в несколько этапов:

1) Проектирование тестовой инфраструктуры

На этом этапе были определены основные компоненты и модули приложения, которые подлежат тестированию, а также разработана архитектура тестовой системы. Были выработаны стандарты написания тестов, выбраны подходы к организации тестовых данных и параметризации тестовых случаев.

2) Настройка тестового окружения

Создано и сконфигурировано тестовое окружение, включающее все необходимые библиотеки, зависимости и инструменты сборки. Особое внимание уделялось совместимости версий и настройке автоматического запуска тестов в рамках процесса сборки.

3) Создание заглушек и mock-объектов

С целью изоляции тестируемых компонентов от внешних зависимостей, были реализованы mock-объекты и заглушки с помощью библиотеки FakeIt. Это позволило эмулировать поведение различных интерфейсов и системных ресурсов, исключая влияние внешних факторов при выполнении тестов.

4) Реализация тестовых случаев

На основе спецификаций и требований к функционалу приложения были разработаны конкретные тестовые случаи с покрытием важных аспектов работы каждого модуля. Тесты были построены таким образом, чтобы обеспечить максимальное выявление ошибок при различных сценариях использования.

Пример тестового случая приведён в приложении В.

Реализация модульных тестов позволила достичь значимых результатов, существенно повысив качество проекта:

– Существенное повышение надёжности и стабильности работы отдельных компонентов приложения за счёт своевременного выявления и устранения дефектов.

– Раннее обнаружение ошибок непосредственно на этапе разработки, что сокращает затраты на их исправление и минимизирует влияние на общий жизненный цикл проекта.

3.4 Разработка автоматизированной системы для запуска тестов и формирования отчетов

Следующим этапом работы стала разработка автоматизированной системы для запуска тестов и формирования отчетов о результатах их выполнения. Целью было создать универсальный инструмент, который обеспечит координированное выполнение как модульных, так и системных тестов, а также позволит автоматически формировать отчеты, удобные для анализа и принятия решений по качеству продукта.

Данная система была реализована на языке Python в виде специализированного скрипта, который последовательно выполняет ряд ключевых операций, необходимых для полного цикла автоматизированного тестирования:

1) Запуск модульных тестов

– Скрипт автоматически инициирует выполнение всех наборов тестов, разработанных с использованием фреймворка Boost.Test. Это

позволяет полноценно проверить корректность работы отдельных компонентов приложения.

– После завершения тестов производится анализ кодов возврата, что позволяет однозначно определить успешность выполнения тестовых наборов и зафиксировать наличие или отсутствие ошибок.

2) Парсинг результатов модульного тестирования

– Для извлечения подробной информации о ходе тестирования скрипт осуществляет разбор (парсинг) сгенерированных Boost.Test отчетов в формате XML.

– Для этого используется стандартная библиотека Python — xml.etree.ElementTree, что гарантирует надёжную и эффективную обработку структурированных данных.

– В ходе парсинга скрипт извлекает ключевые параметры, такие как количество пройденных и проваленных тестов, сообщения об ошибках, а также временные метки, что позволяет предоставить объективную и детальную картину состояния тестов.

3) Формирование отчета

На основе собранных данных создаётся отчет в формате DOCX с помощью библиотеки python-docx, позволяющей структурировать информацию в удобочитаемом виде, включающем заголовки, таблицы и списки.

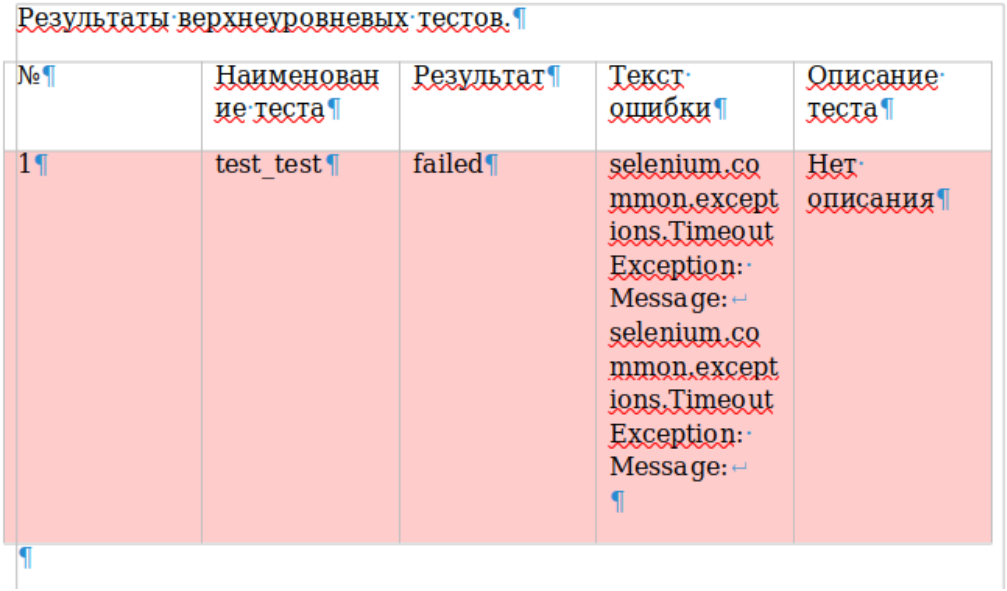
4) Запуск системных тестов и дополнение отчета

– После успешного выполнения модульных тестов скрипт переходит к автоматическому запуску системных тестов, реализованных с помощью

Selenium и запускаемых через pytest. Это покрывает проверку взаимодействия компонентов на уровне пользовательского интерфейса и бизнес-логики.

– Полученные данные системных тестов автоматически добавляются в тот же DOCX отчет, обеспечивая целостное и всестороннее представление о состоянии качества программного продукта.

Пример сформированного отчета представлен на рис.1 и 2



Результаты верхнеуровневых тестов.

№	Наименование теста	Результат	Текст ошибки	Описание теста
1	test_test	failed	selenium.common.exceptions.TimeoutException: Message: selenium.common.exceptions.TimeoutException: Message:	Нет описания

Рисунок 1 – Пример отчета для верхнеуровневых тестов

Результаты юнит-тестов.

№	Наименование теста	Результат	Текст ошибки	Описание теста
1	CheckCorrectResult	passed	Тест выполнился без ошибок	-----Проверка ответа валидации на верных данных----- Верные параметры 1
2	CheckCorrectResult	passed	Тест выполнился без ошибок	-----Проверка ответа валидации на верных данных----- Верные параметры 2

Рисунок 2 – Пример отчета для юнит-тестов

3.5 Автоматизация тестирования

Последним этапом работы стала автоматизация запуска тестов, что значительно улучшило процессы проверки качества приложения и сократило время на их выполнение. Для реализации данной задачи был использован инструмент Gitea Actions, который позволяет автоматизировать рабочие процессы в рамках CI/CD.

Для настройки автоматизации был написан конфигурационный файл, который можно найти в приложении Г.

Основные функции, выполняемые конфигурацией, включают следующие шаги:

1) Клонирование репозиториев: сначала происходит извлечение кода из репозитория серверной части приложения, а затем дополнительно из репозитория графической части приложения. Это гарантирует, что у нас есть последняя версия приложения при выполнении тестов.

2) Создание каталога сборки и копирование файлов: после установки зависимостей создаётся директория для сборки, в которую копируется содержимое графической части. Это позволяет объединить серверную и графическую часть приложения.

3) Настройка и сборка проекта: выполняются команды, необходимые для конфигурации проекта с помощью CMake и его последующей сборки.

4) Запуск модульных тестов: после успешной сборки проекта запускаются модульные тесты, которые проверяют корректность работы отдельных компонентов.

5) Передача отчета модульных тестов: по окончании выполнения тестов результаты в формате XML передаются на локальную машину для дальнейшего анализа.

6) Уведомление о запуске приложения: заключительным этапом является запуск приложения и отправка HTTP-запроса, который оповещает компьютер о начале системных тестов.

Для получения этого запроса используется специальная программа. Как только она его получает, сразу же запускает системные тесты и генерирует отчёт.

ЗАКЛЮЧЕНИЕ

В ходе выполнения работы была разработана и реализована автоматизированная система тестирования программного обеспечения, охватывающая весь жизненный цикл проверки качества разрабатываемого приложения. Особое внимание было уделено созданию комплексного подхода, объединившего модульное и функциональное тестирование, автоматизацию запуска тестов и формирование структурированных отчетов.

Первым этапом внедрения автоматизации стало проектирование и реализация набора модульных тестов. Было разработано 20 тестовых сценариев, направленных на проверку трех основных режимов работы приложения, что обеспечило высокий охват кода и дало возможность оперативно выделять и устранять ошибки на ранних стадиях разработки. Модульные тесты позволили изолированно проверить корректность выполнения ключевых компонентов, выявлять некорректную логику и регрессионные баги при дальнейших доработках кода.

Далее акцент был смещен на создание 15 функциональных, или системных тестов, также охватывающих три ключевых режима работы программы. В отличие от модульных, они ориентированы на проверку работы всей системы целиком, тестируя взаимодействие всех компонентов через интерфейсы, максимально приближённые к пользовательским сценариям. Это дало возможность убедиться, что приложение работает корректно не только на уровне отдельных функций, но и как целостный продукт, удовлетворяя предъявляемым требованиям пользователей и заказчиков.

Важным элементом всей системы стала разработанная с нуля программа для автоматического формирования отчета о результатах

тестирования. Данный инструмент объединяет результаты модульных и функциональных тестов. Такой отчет содержит все необходимые сведения об успешных и проваленных тестах, выявленных ошибках и их локализации, что облегчает анализ результатов не только разработчикам, но и менеджменту проекта. Кроме того, автоматизация формирования отчета минимизирует человеческий фактор и сокращает время подготовки итоговой документации.

Завершающим этапом явилось интегрирование всей системы тестирования в процессы CI/CD посредством использования инструмента Gitea Actions. Для этого был написан и адаптирован конфигурационный файл, определяющий полноту и порядок автоматических процедур. Подобная интеграция обеспечила прозрачность процессов тестирования, их воспроизводимость и независимость от человеческого вмешательства. Любое изменение в коде — будь то исправление ошибки или внедрение новой функции — сопровождается автоматическим запуском всех тестов и обновлением отчета.

Таким образом, реализованная система охватывает полный цикл проверки релевантных свойств программного обеспечения. Она гарантирует своевременное обнаружение и устранение ошибок, ускоряет выпуск новых версий и выступает эффективным инструментом для обеспечения качества и надежности программного продукта.

Проведенная работа продемонстрировала, что внедрение автоматизированного тестирования в связке с современными CI/CD инструментами позволяет не только повысить качество программного продукта, но и существенно повысить эффективность командной работы, прозрачность процессов и скорость выпуска обновлений. Такое решение стало важной основой надежного и стабильного развития информационной системы.

СПИСОК ИСТОЧНИКОВ

1) Купер А. Об интерфейсе. Основы проектирования взаимодействия. – 4-е изд. – Санкт-Петербург : Символ-Плюс, 2019. – 688 с.

2) Scott E. Jr. SPA Design and Architecture: Understanding single-page web applications. – Simon and Schuster, 2015. – 256 p.

3) Мычко С. И. Микросервисная архитектура // Информационные технологии. – 2019. – С. 166–168.

4) Beizer B. Black-box testing: techniques for functional testing of software and systems. – John Wiley & Sons, Inc., 1995. – 320 p.

5) Hamill P. Unit test frameworks: tools for high-quality software development. – O'Reilly Media, Inc., 2004. – 232 p.

6) Orso A. Integration testing of object-oriented software // Dottorato di Ricerca in Ingegneria Informatica e Automatica, Politecnico di Milano. – 1998. – P. 1–119.

7) Di Lucca G. A., Fasolino A. R., Tramontana P. Testing web applications // International Conference on Software Maintenance, 2002. Proceedings. – IEEE, 2002. – P. 310–319.

8) Сеидова И. Э., Абдуллаев Э. М. Нефункциональное тестирование программного обеспечения: особенности и применение // In The World Of Science and Education. – 2025. – № 15 (март). – С. 60–63.

9) OWASP Foundation. OWASP Top Ten: Critical Web Application Security Risks. – 2021. – URL: https://owasp.org/www-project-top-ten/ (дата обращения: 10.04.2025).

10) Roy Choudhary S. Cross-platform testing and maintenance of web and mobile applications // Companion Proceedings of the 36th International Conference on Software Engineering. – 2014. – P. 642–645.

11) Котляров В. П. Основы тестирования программного обеспечения. – Москва, 2016. – 320 с.

12) BrowserStack. Selenium Testing Tool. – URL: https://www.browserstack.com/selenium (дата обращения: 20.04.2025).

13) Cypress: Fast, easy and reliable testing for anything that runs in a browser. – URL: https://www.cypress.io/ (дата обращения: 22.04.2025).

14) Microsoft. Playwright: Reliable end-to-end testing for modern web apps. – URL: https://learn.microsoft.com/en-us/microsoft-edge/playwright/ (дата обращения: 25.04.2025).

15) BrowserStack. TestCafe Framework Tutorial. – URL: https://www.browserstack.com/guide/testcafe-framework-tutorial (дата обращения: 26.04.2025).

16) Google. GoogleTest: C++ Testing Framework. – URL: https://google.github.io/googletest/ (дата обращения: 27.04.2025).

17) GitLab. Develop C++ Unit Testing with Catch2, JUnit, and GitLab CI. – 2024. – URL: https://about.gitlab.com/blog/2024/07/02/develop-c-unit-testing-with-catch2-junit-and-gitlab-ci/ (дата обращения: 28.04.2025).

18) Boost C++ Libraries. Boost.Test Documentation. – URL: https://www.boost.org/doc/libs/1_82_0/libs/test/doc/html/index.html (дата обращения: 28.04.2025).

19) JetBrains. Better Ways of Testing with doctest. – 2019. – URL: https://blog.jetbrains.com/rscpp/2019/07/10/better-ways-testing-with-doctest/ (дата обращения: 28.04.2025).

20) Freedesktop.org. CppUnit: C++ Unit Testing Framework. – URL: https://www.freedesktop.org/wiki/Software/cppunit/ (дата обращения: 28.04.2025).

21) Tested & Failed by Tricentis. – URL: https://testedfailed.tricentis.com/ (дата обращения: 28.04.2025).

22) Micro Focus. UFT One: Data Sheet [PDF]. – URL: https://www.microfocus.com/ru-ru/media/data-sheet/uft-one-ds-ru.pdf?utm_source=OSPRU (дата обращения: 28.04.2025).

23) Mabl: Low-code test automation. – URL: https://www.mabl.com/ (дата обращения: 29.04.2025).

24) testIT. Integrations with Testim. – URL: https://docs.testit.software/user-guide/integrations/automation/testim.html (дата обращения: 29.04.2025).

25) The Apache Software Foundation. Apache JMeter. – URL: https://jmeter.apache.org/ (дата обращения: 30.04.2025).

26) Хабрахабр. Тестирование производительности: JMeter vs Gatling vs Tsung. – 2015. – URL: https://habr.com/ru/articles/272213/ (дата обращения: 30.04.2025).

ПРИЛОЖЕНИЕ А

(справочное)

Пример тестового сценария

Описание функции: режим замера сопротивления для соединителей.

1) Перейти во вкладку "Работа".

2) Перейти во вкладку "Режимы испытаний".

3) Перейти во вкладку с необходимым режимом.

4) Выбрать все элементы для проверки.

5) Нажать кнопку “Начать проверку”.

6) Подтвердить начало проверки.

7) Проверить, что кнопка "Начать проверку" заблокирована.

8) Проверить, что кнопка "выход" изменилась на кнопку "отмена".

9) Дождаться окончания проверки.

10) Проверить правильность сформированного протокола.

11) Проверить, что по всем элементам получен результат.

12) Выйти из режима.

13) Проверить, что во вкладке диагностика, есть запись об окончании проверки.

Ожидаемый результат: режим корректно отрабатывает, по всем выбранным модулям приходит ответ.

Критерии успешного прохождения теста:

Продолжение ПРИЛОЖЕНИЯ А

– Во время проверки все необходимые кнопки блокируются

– По всем модулям был получен ответ

– После окончания проверки протокол правильно сформирован

– Во вкладке “Диагностика” есть запись с результатом проверки.

ПРИЛОЖЕНИЕ Б

(справочное)

Пример системного теста

```
    def check_mode(web_driver_mode, mode):
        web_driver_mode.find_element(By.ID,
mode).click()
        try:
            WebDriverWait(web_driver_mode,
3).until(lambda          d:d.find_element(By.TAG_NAME,
"span").is_displayed())
            circuits_background_color                =
web_driver_mode.find_element(By.TAG_NAME,
"span").value_of_css_property("background-color")
        except Exception:

web_driver_mode.execute_script("arguments[0].click()"
,               web_driver_mode.find_element(By.XPATH,
"//button[text()='выход']"))
            time.sleep(1)
            web_driver_mode.find_element(By.ID,
mode).click()
            WebDriverWait(web_driver_mode,
3).until(lambda          d:d.find_element(By.TAG_NAME,
"span").is_displayed())
            circuits_background_color                =
web_driver_mode.find_element(By.TAG_NAME,
"span").value_of_css_property("background-color")
```

Продолжение ПРИЛОЖЕНИЯ Б

```
        check_boxes                                    =
web_driver_mode.find_elements(By.TAG_NAME, "label")
        for i in range(len(check_boxes) - 1):
            check_boxes[i].click()


    web_driver_mode.find_element(By.XPATH,
"//button[text()='Да']").click()
        WebDriverWait(web_driver_mode,
100).until(lambda d:d.find_element(By.XPATH,
"//button[text()='Ok']").is_displayed())
        report =
web_driver_mode.find_element(By.TAG_NAME, "p").text
        result = report[report.find("Результат")+10:]
== 'отрицательный'
        web_driver_mode.find_element(By.XPATH,
"//button[text()='Ok']").click()


        web_driver_mode.find_element(By.XPATH,
"//button[text()='Выход']").click()
        circuits_states =
web_driver_mode.find_elements(By.TAG_NAME, "span")
        is_broken =
any([state.value_of_css_property("background-color")
== 'rgb(250, 0, 33)' for state in circuits_states])


        assert result == is_broken, "Неправильно
сформирован протокол"
```

Продолжение ПРИЛОЖЕНИЯ Б

```
    assert
all([state.value_of_css_property("background-color")
!= circuits_background_color for state in
circuits_states]), "По одному или нескольким модулям
не было получено ответа"
```

ПРИЛОЖЕНИЕ В

(справочное)

Пример модульного теста

```
    BOOST_AUTO_TEST_CASE(TestCheckResistPositiveAnswe
r){

        BOOST_TEST_MESSAGE("Проверка режима");

        Json::Value state;

        state["Operable"] = 15;

        state["Available"] = 0;

        Mock<Request> state_mock;

        When(Method(state_mock,
get_result)).Return(&state);

        When(Method(state_mock, run)).Return();

        Mock<RequestStateFactory> state_factory_mock;

        When(Method(state_factory_mock,
createRequestState)).Return(dynamic_cast<RequestState
Task*>(&state_mock.get()));

        IRequestStateFactory* state_factory =
&state_factory_mock.get();


        RequestStateTask* state_ptr  =
dynamic_cast<RequestStateTask*>(&state_mock.get());;

        RequestStateTask* state_request = state_ptr;

        state_ptr -> run();
```

Продолжение ПРИЛОЖЕНИЯ В

```
        Json::Value* value =
(*state_request).get_result();

        Mock<Request> resist_mock;

        When(Method(resist_mock,
get_result)).Return(&resist);

    Mock<IRequestResistFactory> resist_factory_mock;

    std::unique_ptr<RequestResistTask>
unique_ptr_resist((RequestResistTask
*)&resist_mock.get());

        When(Method(resist_factory_mock,
createRequestResist)).Return(std::move(unique_ptr_res
ist));

        RequestResistFactory* resist_factory =
(RequestResistFactory* )&resist_factory_mock.get();

        std::string path_file =
"../storage/mode.csv";

        iteration = 0;

        Json::Value json;

        Json::Value* json_ptr = &json;

        CsvParsTask parser = CsvParsTask(&iteration,
&path_file, &data, json_ptr);

        parser.run();

        iteration = 0;

        Json::Value input_message;

        input_message["list"][0] = "0";

        input_message["list"][1] = "1";
```

Продолжение ПРИЛОЖЕНИЯ В

```
        input_message["list"][2] = "2";
    input_message["list"][3] = "3";
        for (int i=0; i<4; i++){
            ConnectorsChek chek_object =
ConnectorsChek(&input_message, &iteration, &data,
json_ptr,
    (RequestResistFactory*)&resist_factory_mock.get()
, (RequestStateFactory*)state_factory);
            chek_object.run();
            std::cout <<
((*json_ptr)["array"][i]["module"]["res_task"].asStri
ng());
    BOOST_CHECK_EQUAL((*json_ptr)["array"][i]["module
"]["res_task"].asString(), "positive");
            iteration++;
        }
    }
```

ПРИЛОЖЕНИЕ Г

(справочное)

Конфигурационный файл

```
name: Build and Test

on:
  push:
    branches: ["develop"]
  pull_request:
    branches: ["develop"]
jobs:
  build_and_test:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout backend repository
        uses: actions/checkout@v4
      - name: Checkout frontend repository
        uses: actions/checkout@v4
        with:
          repository: konovalovta/frontend
          path: frontend
```

Продолжение ПРИЛОЖЕНИЯ Г

```
      - name: Install dependencies (Ubuntu)
        run: |
    sudo apt-get update
          sudo apt-get install -y cmake build-
essential
      - name: Create build directory
        run: mkdir -p build
      - name: Clean cache
        run: cmake --build ./ --target clean
      - name: Copy frontend dist to backend build
        run: |
          cp -r frontend/dist/* build/
      - name: Configure CMake
        working-directory: ./build
        run: cmake ..
      - name: Build project
        working-directory: ./build
        run: cmake --build ./
      - name: Run tests
        working-directory: ./build
```

Продолжение ПРИЛОЖЕНИЯ Г

```
            run: ./test --

          - name: Transfer XML report to local
machine

    run: |

              scp -o StrictHostKeyChecking=no
build/test_report.xml
konovalovta@<127.1.40.1>:<home/konovalovta/tests>


      - name: Notify local machine

            run: |

              curl -X POST
http://<127.1.40.1:500>/run_system_tests
```