Министерство науки и высшего образования Российской Федерации
ФГАОУ ВО «УрФУ имени первого Президента России Б.Н. Ельцина»
Кафедра «школа бакалавриата (школа)»

Оценка работы _____
Руководитель от УрФУ Спиричева Н.Р.

Тема задания на практику

Разработка и интеграция автоматизированной системы тестирования для веб-приложения

ОТЧЕТ

Вид практики Производственная практика
Тип практики Производственная практика, преддипломная

Руководитель практики от предприятия (организации)
к.т.н., доцент Соловьева Н.В. _____
ФИО руководителя **Подпись**

Студент Коновалов Т.А. _____
ФИО студента **Подпись**

Специальность (направление подготовки) 09.03.01 Информатика и вычислительная техника

Группа РИ-410910

Екатеринбург 2025

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.

В рамках данного отчёта будут рассмотрены основные моменты разработки данной системы, проведен анализ существующих решений, рассмотрены основные подходы и методы для автоматизации тестирования, а также определена архитектура системы и разработана большая часть её компонентов.

1 Описание работы

1.1 Используемые технологии

Для выполнения поставленных задач были использованы следующие средства разработки:

- язык программирования Python, так как он позволяет легко автоматизировать необходимые процессы;
- для автоматизации действий браузера и верхнеуровневого тестирования фреймворк Selenium, за счёт простой интеграции с Python;
- для формирования отчёта в формате docx была использована библиотека Python-docx;
- для написания модульных тестов был использован фреймворк Boost.Test;
- для автоматического запуска тестов была использована система Gitea Actions.

1.2 Календарный план

В таблице 1 представлен календарный план прохождения практики.

Таблица 1 – Календарный план

Этапы практики	Наименование работ студента	Срок	Примечание
Организационный	Усвоение правил внутреннего трудового распорядка Профильной организации; правил по охране труда и технике безопасности, санитарно-эпидемиологических правил, режима конфиденциальности, принятого в Профильной организации. Обсуждение с руководителем практики от Профильной организации темы индивидуального задания на практическую подготовку в организации, уточнение ее формулировки, составление конкретного последовательного перечня работ, необходимых для выполнения задания.	10.02.25 - 21.02.25	выполнено

Продолжение таблицы 1

основной	Выполнение индивидуального задания по следующему перечню работ: 1. Изучение аналогов, а также подходов для автоматизированного тестирования. 2. Написание основных сценариев для тестирования. 3. Написание верхнеуровневых и модульных тестов. 4. Разработка системы для автоматического запуска тестов и формирования отчёта.	21.02.25 - 30.03.25	выполнено
заключительный	Составление структуры и написание полного отчета о выполнении индивидуального задания на практическую подготовку в Профильной организации, в соответствии с требованиями шаблонов УрФУ. Согласование отчета с руководителем практики. Получение Отзыва на прохождение практической подготовки и документа к Отзыву от Профильной организации. Нормоконтроль отчета о практике. Подготовка презентации к защите практики.	31.03.25-06.04.25	выполнено

1.3 Анализ решений для тестирования

Тестирование программного обеспечения представляет собой систематический процесс верификации и валидации программного продукта, направленный на обеспечение его соответствия установленным требованиям и ожиданиям пользователей. В современной практике тестирования принято выделять следующие фундаментальные виды тестирования:

1) Модульное тестирование (Unit Testing)

Модульное тестирование — это разновидность тестирования в программной разработке, которое заключается в проверке работоспособности отдельных функциональных модулей, процессов или частей кода приложения [1]. Его целью является выявление дефектов на уровне отдельных модулей. Модульное тестирование представляет собой важнейший этап процесса верификации программного обеспечения, который выполняется разработчиками непосредственно в ходе реализации функциональности.

Данный вид тестирования требует обязательного использования специализированных фреймворков, которые предоставляют необходимый инструментарий для создания, запуска и анализа unit-тестов. Ключевым преимуществом модульного тестирования является возможность раннего обнаружения ошибок на этапе разработки отдельных компонентов системы, что позволяет выявлять и устранять дефекты до их интеграции в общую архитектуру приложения. Такой подход значительно сокращает стоимость исправления ошибок и повышает общую надежность программного кода, поскольку проблемы обнаруживаются и фиксируются в максимально локализованном контексте [2].

2) Интеграционное тестирование

Интеграционное тестирование — это процесс проверки взаимодействия между различными компонентами системы [3]. Данный вид тестирования применяется для выявления дефектов, возникающих при взаимодействии различных программных модулей. Важным преимуществом данного подхода является возможность раннего обнаружения ошибок взаимодействия между модулями, что существенно снижает стоимость их исправления по сравнению с выявлением на более поздних этапах тестирования. При этом разработчики получают возможность своевременно выявлять и устранять проблемы интерфейсов, несоответствия в передаче данных и другие дефекты, возникающие при интеграции компонентов системы [4].

3) Системное тестирование (System Testing)

Системное тестирование — это этап тестирования программного обеспечения, на котором тестируется полный и полностью интегрированный программный продукт на основе спецификации программного обеспечения [5]. Основной целью системного тестирования является подтверждение того, что разработанный программный продукт в полной мере соответствует всем заявленным функциональным и нефункциональным требованиям. В рамках системного тестирования выделяют несколько специализированных направлений, каждое из которых фокусируется на определенных аспектах

работы системы [6]. Функциональное тестирование обеспечивает проверку соответствия поведения системы заявленным бизнес-требованиям. Тестирование производительности позволяет оценить работоспособность системы при различных нагрузках и условиях эксплуатации. Тестирование безопасности направлено на выявление потенциальных уязвимостей и проверку механизмов защиты. Тестирование надежности подтверждает стабильность работы системы в течение продолжительного периода времени.

4) Регрессионное тестирование

Регрессионное тестирование – тестирование уже протестированной ранее программы. Проводится после модификации ПО для уверенности в том, что процесс модификации не внес или не активизировал ошибки в областях, не подвергавшихся изменениям. Проводится после изменений в коде программного продукта или его окружении [7].

Особое значение регрессионное тестирование приобретает в условиях активной разработки, когда продукт регулярно обновляется и дополняется новым функционалом. Оно позволяет выявлять так называемые "регрессионные ошибки" - дефекты, которые возникают в ранее стабильных частях системы из-за последних изменений.

1.4 Сравнительный анализ фреймворков для модульного тестирования

Для реализации автоматизированного модульного тестирования в языке программирования C++ разработан ряд специализированных фреймворков. Рассмотрим три наиболее распространенных решения:

1. Google Test Framework

Google Test Framework представляет собой мощный и надежный фреймворк для модульного тестирования C++ приложений. Разработанный компанией Google, он предлагает модульную структуру с поддержкой

иерархии тестов, что позволяет эффективно организовывать тестовые сценарии различной сложности.

Ключевые особенности:

поддержка параметризованных тестов для проверки различных входных данных;

- гибкая система фикстур для настройки тестового окружения;
- расширенный набор ассертов для детальной проверки условий.

Преимущества:

- отличная интеграция с системами CI/CD;
- подробные диагностические сообщения при обнаружении ошибок;
- активное сообщество разработчиков и регулярные обновления;

Недостатки:

- требует дополнительной настройки окружения;
- имеет достаточно высокий порог входа для новичков [8].

2. Boost.Test

Boost.Test - это универсальный фреймворк для модульного тестирования, входящий в состав популярной библиотеки Boost. Его компонентная архитектура поддерживает различные парадигмы тестирования, что делает его гибким инструментом для разных проектов [9].

Ключевые особенности:

- гибкая система отчетности о результатах тестирования;
- поддержка BDD (Behavior-Driven Development) стиля написания тестов;
- глубокая интеграция с другими компонентами Boost.

Преимущества:

- не требует дополнительных зависимостей;
- полная кроссплатформенность;
- поддержка различных стандартов C++.

Недостатки:

- менее информативные сообщения об ошибках по сравнению с Google Test;
- Относительно низкая производительность при работе с большими тестовыми наборами [10].

3. Catch2

Catch2 – это современный минималистичный фреймворк для модульного тестирования C++ кода. Его заголовочная реализация делает процесс интеграции в проект максимально простым и удобным [11].

Ключевые особенности:

- поддержка как TDD, так и BDD подходов;
- встроенные макросы для удобного написания тестов;
- автоматическая регистрация тестовых случаев.

Преимущества:

- простота установки и использования;
- минимальные накладные расходы;
- идеален для небольших и средних проектов.

Недостатки:

- ограниченная масштабируемость для крупных проектов;
- меньший набор функциональных возможностей по сравнению с конкурентами [12].

Для реализации данной работы был выбран фреймворк Boost.Test, так как в данном проекте уже были использованы другие компоненты Boost, что позволило сократить время для настройки инструментария, уменьшить количество зависимостей и избежать конфликтов при сборке проекта. Также Boost.Test обеспечивает поддержку сложных сценариев тестирования, что может быть особенно полезно при проверке компонентов, взаимодействующих с Boost-библиотеками.

1.5 Сравнительный анализ инструментов для автоматизированного тестирования веб-приложений

В настоящее время существует множество инструментов для автоматизированного тестирования веб-приложений. Рассмотрим наиболее популярные из них, их ключевые особенности, преимущества и недостатки.

1) Selenium WebDriver

Selenium WebDriver является одним из самых распространенных инструментов для автоматизированного тестирования веб-приложений. Разработанный в 2004 году, он поддерживает множество языков программирования, включая Java, Python, C# и JavaScript [13].

Основные преимущества:

- кроссплатформенность - поддерживает все основные браузеры (Chrome, Firefox, Safari, Edge);
- гибкость интеграции с различными фреймворками (TestNG, JUnit, pytest);
- возможность масштабирования для сложных проектов;
- большое сообщество пользователей и обширная документация;

Недостатки:

- тесты могут быть хрупкими при изменениях в интерфейсе;
- отсутствие встроенных инструментов для отладки.

2) Cypress

Cypress - современный фреймворк для end-to-end тестирования, появившийся в 2017 году. Работает на JavaScript/TypeScript [14].

Основные преимущества:

- простота настройки и использования;
- встроенные инструменты отладки;
- автоматические ожидания элементов;
- хорошая интеграция с современными фронтенд-фреймворками.

Недостатки:

- ограниченная поддержка браузеров (только Chromium-движки);
- не поддерживает работу с несколькими вкладками/доменами;
- меньшая гибкость по сравнению с Selenium.

3) Playwright

Playwright - относительно новый фреймворк (2020 год), разработанный Microsoft. Поддерживает несколько языков программирования [15].

Основные преимущества:

- полноценная кросс-браузерная поддержка;
- стабильность тестов благодаря встроенным механизмам retry;
- возможность эмуляции мобильных устройств;
- поддержка параллельного выполнения тестов.

Недостатки:

- меньшее сообщество по сравнению с Selenium;
- не поддерживает устаревшие версии браузеров.

Сравнительная таблица по всем перечисленным фреймворкам представлена ниже.

Таблица 2 – Сравнительная таблица

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

Выбор конкретного инструмента должен основываться на требованиях проекта, технологическом стеке команды и масштабах тестирования. Для сложных enterprise-решений по-прежнему оптимальным выбором остается Selenium, в то время как для более специализированных задач могут быть предпочтительнее Cypress или Playwright.

Для реализации данного проекта был выбран Selenium WebDriver.Выбор Selnium WebDriver был обусловлен рядом преимуществ, которые делают его оптимальным для решения поставленных задач.

Во-первых, данный фреймворк полностью поддерживает язык программирования Python, который был выбран основным в данном проекте.

Во-вторых, кросплатформенность и совместимость с большинством браузеров позволяет обеспечить возможность проверки приложения в различных окружениях.

Третьим фактором стала простота интеграции с тестовым фреймворком pytest, что позволило легко встроить автоматизированные тесты в процесс CI/CD и значительно ускорило процесс тестирования.

Также стоит отметить активное сообщество и обширную документацию Selenium WebDriver, упрощающие решение возникающих проблем.

Таким образом, выбор Selenium WebDriver был обусловлен его универсальностью, надежностью и широкими возможностями интеграции, что полностью соответствовало требованиям проекта к автоматизированному тестированию веб-интерфейсов.

2 Ход работы

2.1 Написание сценариев тестирования

Первым этапом работы являлось создание сценариев для тестирования приложения. Для этого потребовалось изучить функциональные возможности самого приложения, его архитектуру и взаимодействие между компонентами. Также были проанализированы технические требования и ожидаемое поведение системы, что позволило сформировать корректные и релевантные тестовые случаи.

На основе проведённого анализа были разработаны сценарии тестирования, охватывающие как позитивные (проверка работы приложения в штатных условиях), так и негативные (обработка ошибок и нестандартные ситуации) сценарии использования. Каждый тестовый сценарий включал:

- описание тестируемой функции;
- шаги для воспроизведения;
- ожидаемый результат;
- критерии успешного прохождения теста.

Этот этап позволил систематизировать процесс тестирования и обеспечить максимальное покрытие требований, что в дальнейшем способствовало выявлению потенциальных уязвимостей и ошибок в работе приложения.

Пример тестового сценария приведен в приложении А.

2.2 Автоматизация тестовых сценариев

Для реализации данной задачи был выбран комплекс инструментов, включающий:

- pytest - как основной фреймворк для организации и выполнения тестов;

- Selenium WebDriver - для автоматизации взаимодействия с пользовательским интерфейсом тестируемого приложения.

Процедура автоматизации включала последовательное выполнение следующих работ:

1. Подготовка тестового окружения:

- установка и конфигурация программного обеспечения;

- настройка драйверов для взаимодействия с веб-браузером;

- создание структуры проекта с выделением модулей для тестов, вспомогательных функций и конфигурационных файлов.

2. Разработка автоматизированных тестов:

- трансформация ручных тестовых сценариев в программный код с использованием возможностей pytest;

- реализация фикстур для управления жизненным циклом тестовых данных и состояний системы;

- организация модульной структуры тестов в соответствии с принципами поддержки и повторного использования кода.

3. Интеграция Selenium WebDriver:

- разработка скриптов для автоматизированного взаимодействия с элементами пользовательского интерфейса,

- реализация механизмов явных и неявных ожиданий для обеспечения стабильности выполнения тестов,

- внедрение системы проверок (assert) для верификации соответствия фактических результатов ожидаемым.

Пример реализации автотеста приведён в приложении Б.

Внедрение автоматизированного тестирования позволило достичь следующих результатов:

- сокращение временных затрат на проведение регрессионного тестирования;

- повышение точности проверок за счёт исключения человеческого фактора;

- обеспечение возможности интеграции в процессы непрерывной поставки.

2.3 Разработка модульных тестов

Следующим этапом работы стала разработка модульных тестов для проверки корректности работы отдельных компонентов приложения. Для реализации данной задачи был применен комплекс инструментов, включающий:

- Boost.Test - как основной фреймворк для организации модульного тестирования на C++,

- FakeIt - библиотеку для создания mock-объектов и заглушек (stubs),

- дополнительные инструменты CMake для интеграции тестов в процесс сборки.

Работа по внедрению модульного тестирования включала следующие этапы:

1) Проектирование тестовой инфраструктуры.

2) Настройка тестового окружения.

3) Создание заглушек и mock-объектов.

4) Реализация тестовых случаев.

Пример тестового случая приведён в приложении В.

Реализация модульных тестов позволила достичь следующих результатов:

- повышение надежности отдельных компонентов приложения,

- раннее выявление ошибок на этапе разработки,

- упрощение рефакторинга за счет наличия тестовой базы.

2.4 Разработка автоматической системы для запуска тестов и формирования отчетов

Следующим этапом работы стала разработка автоматизированной системы запуска тестов и формирования отчетов**.** Данная система была реализована в виде специализированного скрипта на языке Python и обеспечивает последовательное выполнение следующих операций:

1) Запуск модульных тестов:

- автоматический запуск тестовых наборов, реализованных на Boost.Test;

- обработка кодов возврата для определения успешности выполнения.

2) Парсинг результатов модульного тестирования:

- анализ сгенерированного XML-файла с использованием библиотеки xml.etree.ElementTree;

- извлечение ключевых параметров.

3) Формирование отчета:

- создание документа в формате DOCX с использованием библиотеки python-docx;

- структурирование информации.

4) Запуск системных тестов и дополнение отчета:

- автоматический запуск тестов Selenium через pytest;

- захват и обработка выходных данных;

- дополнение отчета результатами каждого теста.

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

ЗАКЛЮЧЕНИЕ

В рамках выполнения практической работы была реализована комплексная система тестирования программного обеспечения, включающая несколько взаимосвязанных этапов, направленных на обеспечение высокого качества разрабатываемого продукта.

Основные достигнутые результаты:

1) Разработана методологии тестирования.

2) Проведен детальный анализ функциональных требований к приложению.

3) Разработана система тестовых сценариев, охватывающая все ключевые аспекты работы приложения.

4) Реализована многоуровневая система тестирования:

- внедрено модульное тестирование с использованием фреймворка Boost.Test;

- реализована система автоматизированного UI-тестирования на базе Selenium WebDriver.

5) Проведена автоматизация процесса тестирования приложения:

- создана инфраструктура для автоматического выполнения тестовых сценариев;

- реализована система формирования отчетной документации;

- внедрены механизмы обработки и анализа результатов тестирования.

Проведенная работа позволила создать эффективную систему контроля качества программного обеспечения, соответствующую современным стандартам разработки. Реализованные решения демонстрируют высокую эффективность и могут быть успешно применены в последующих проектах.

Полученные результаты подтверждают целесообразность выбранного подхода и открывают перспективы для дальнейшего совершенствования процессов тестирования.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1) Unit-тестирование (модульное): что это и для чего используется URL: https://blog.skillfactory.ru/glossary/unit-testirovanie/ (дата обращения: 12.02.2025 - 25.03.2025)

2) Савин, Р. О. Тестирование dot com / Р. О. Савин. — Санкт-Петербург : Питер, 2017. — 272 с.

3) Что такое интеграционное тестирование URL: https://sky.pro/media/chto-takoe-integraczionnoe-testirovanie/ (дата обращения: 15.02.2025 - 20.03.2025)

4) Куликов, С. В. Тестирование программного обеспечения. Базовый курс: электронное учебное пособие / С. В. Куликов. — 2021. — URL: https://svyatoslav.biz/software-testing-book/ (дата обращения: 15.03.2025)

5) Уиттакер, Дж. Как тестируют в Google / Дж. Уиттакер, Дж. Арбон, Дж. Каролло ; пер. с англ. А. В. Славянского. — Москва : Альпина Паблишер, 2013. — 414 с.

6) Клейн, Т. Дневник охотника за ошибками / Т. Клейн ; пер. с англ. И. В. Борисова. — Москва: ДМК Пресс, 2012. — 352 с.

7) ISTQB Glossary of Terms in Software Testing [Электронный ресурс]. — URL: https://istqb-glossary.page/ (дата обращения: 20.02.2025).

8) Современные подходы к модульному тестированию на C++ URL: https://habr.com/ru/articles/807813/

9) Лоспинозо, Дж. C++ для профи / Дж. Лоспинозо ; [пер. с англ.]. — Санкт-Петербург : Питер, 2021. — 816 с. (дата обращения: 18.02.2025 - 22.03.2025)

10) Основы модульного тестирования: теория и практика
URL: https://habr.com/ru/articles/119090/
(дата обращения: 10.02.2025 - 15.03.2025)

11) Boost.Test: документация
URL: https://www.boost.org/doc/libs/1_82_0/libs/test/doc/html/index.html
(дата обращения: 14.02.2025 - 28.03.2025)

12) Разработка модульных тестов на C++ с использованием Catch2, JUnit и GitLab CI
URL: https://about.gitlab.com/blog/2024/07/02/develop-c-unit-testing-with-catch2-junit-and-gitlab-ci/
(дата обращения: 20.02.2025 - 31.03.2025)

13) Selenium WebDriver: документация
URL: https://www.selenium.dev/documentation/webdriver/
(дата обращения: 11.02.2025 - 18.03.2025)

14) Cypress: документация
URL: https://docs.cypress.io/
(дата обращения: 16.02.2025 - 24.03.2025)

15) Playwright: документация
URL: https://playwright.dev/docs/intro
(дата обращения: 10.02.2025 - 31.03.2025)

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:

- 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 модульного теста

```
BOOST_AUTO_TEST_CASE(TestCheckResistPositiveAnswer){
    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<RequestStateTask*>(&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();
    Json::Value* value = (*state_request).get_result();
    Mock<Request> resist_mock;
    When(Method(resist_mock, get_result)).Return(&resist);
    When(Method(resist_mock, run)).Return();
```

Продолжение ПРИЛОЖЕНИЯ Б

```
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_resist));

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";

input_message["list"][2] = "2";

input_message["list"][3] = "3";

std::cout << "\n 4 \n";

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();
```

Продолжение ПРИЛОЖЕНИЯ Б

```
    std::cout << "1 \n";
        std::cout << ((*json_ptr)["array"][i]["module"]["res_task"].asString());
    BOOST_CHECK_EQUAL((*json_ptr)["array"][i]["module"]["res_task"].as
String(), "positive");
        iteration++;
      }
    }
```

ПРИЛОЖЕНИЕ В

(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")
    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()='Начать проверку']").click()
```

Продолжение ПРИЛОЖЕНИЯ В

```
        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"

        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"
```