**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) Соловьева Н.В. И

**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

# 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, способной 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. 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, засчёт 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 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. | 10.02.2025– 21.02.2025 | выполнено |

|  |  |  |  |
| --- | --- | --- | --- |
| 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. | 21.02.2025 - 30.03.2025 | 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. | 31.03.2025- 06.04.2025 | 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 три фундаментальных вида:

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.Интеграционное тестирование**

Интеграционное тестирование - это процесс проверки взаимодействия между различными компонентами системы. [2] 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.

**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. [3]. 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. 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.

**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 [4]

1. **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. [5]

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

**3. Catch2**

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

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 [6]

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. [7]

**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

1. **Cypress**

Cypress - современный фреймворк для end-to-end тестирования, появившийся в 2017 году. Работает на JavaScript/TypeScript. [8]

**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. [9]

**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

Таблица 2 – Сравнительная таблица

| **Фреймворк** | **Языки** | **Браузеры** | **Преимущества** | **Недостатки** |
| --- | --- | --- | --- | --- |
| **Selenium** | Javа, Python, C#, JS | Все | Гибкость, активное сообщество | Сложность, хрупкость |
| **Cypress** | JS/TS | Chrome, Edge | Простота использования, отладка | Не поддерживает Firefox/Safari |
| **Playwright** | JS, 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.

# 2. Ход работы

## 2.1 Написание сценариев тестирования

**Первым этапом работы являлось создание сценариев для тестирования приложения.** Для этого потребовалось тщательно 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.

Ниже вы можете увидеть пример тестового сценария:

## 2.2 Автоматизация тестового сценария

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;
* **Дополнительные библиотеки Python** - для обработки данных, управления ожиданиями и генерации отчётов.

**Процесс автоматизации тестирования**

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 (декоратор @pytest.fixture) 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**

[При необходимости здесь может быть приведён фрагмент кода с примером автотеста]

**Результаты автоматизации**

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.

Процесс разработки модульных тестов

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**

[Здесь может быть приведен пример реализации unit-теста с использованием Boost.Test и FakeIt]

**Результаты внедрения модульного тестирования**

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:

**Архитектура решения**

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 кода, пример итогового отчета.

# 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. **Разработка методологии тестирования**
   * Проведен детальный анализ функциональных требований к приложению
   * Разработана система тестовых сценариев, охватывающая все ключевые аспекты работы приложения
   * Сформированы критерии оценки качества тестирования
2. **Реализация многоуровневой системы тестирования**
   * Внедрено модульное тестирование с использованием фреймворка Boost.Test
   * Реализована система автоматизированного UI-тестирования на базе Selenium WebDriver
   * Разработаны механизмы интеграционного тестирования
3. **Автоматизация процессов тестирования**
   * 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 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. Что такое интеграционное тестирование  
     URL: <https://sky.pro/media/chto-takoe-integraczionnoe-testirovanie/>  
     (дата обращения: 15.02.2025 - 20.03.2025)
  3. Современные подходы к модульному тестированию на C++  
     URL: <https://habr.com/ru/articles/807813/>  
     (дата обращения: 18.02.2025 - 22.03.2025)
  4. Основы модульного тестирования: теория и практика  
     URL: <https://habr.com/ru/articles/119090/>  
     (дата обращения: 10.02.2025 - 15.03.2025)
  5. 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)
  6. 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)
  7. Selenium WebDriver: документация  
     URL: <https://www.selenium.dev/documentation/webdriver/>  
     (дата обращения: 11.02.2025 - 18.03.2025)
  8. Cypress: документация  
     URL: <https://docs.cypress.io/>  
     (дата обращения: 16.02.2025 - 24.03.2025)
  9. Playwright: документация  
     URL: <https://playwright.dev/docs/intro>  
     (дата обращения: 10.02.2025 - 31.03.2025)