МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**
Кафедра математической кибернетики и компьютерных наук

РАЗРАБОТКА АВТОТЕСТОВ ДЛЯ ТЕСТИРОВАНИЯ САЙТА

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 5 курса 551 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Грунина Евгения Яковлевича

Научный руководитель
доцент, к. ф.-м. н. _____ А. С. Иванова

Заведующий кафедрой
к. ф.-м. н., доцент _____ С. В. Миронов

Саратов 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Современный этап развития информационных технологий характеризуется бурным ростом спроса на качественное программное обеспечение, что делает разработку ПО одной из ключевых и наиболее динамичных отраслей IT-индустрии. Ежегодно появляются новые технологии, языки программирования и инструменты, что предъявляет высокие требования к методам и средствам тестирования. В условиях цифровой трансформации экономики и множества онлайн-сервисов качество ПО становится критическим фактором успеха бизнеса и удовлетворённости пользователей.

Тестирование является неотъемлемой частью процесса разработки и охватывает проверку функциональности, производительности, безопасности и удобства использования. Для повышения эффективности и снижения временных затрат всё шире внедряются автоматизированные методы, способные оперативно адаптироваться к усложняющимся системам и ускорять выпуск продуктов на рынок. Актуальность исследования заключается в необходимости постоянного обновления тестовых подходов, где автоматизация не только повышает качество ПО, но и оптимизирует весь процесс разработки, становясь важнейшим конкурентным преимуществом.

Целью работы: проведение анализа современных методов и инструментов тестирования программного обеспечения, а также их применение для тестирования web-приложения.

В ходе работы должны быть решены следующие задачи:

1. Исследование современных методов тестирования программного обеспечения, включая функциональное, нагрузочное, интеграционное и другие виды тестирования.
2. Анализ инструментов тестирования и выбор наиболее подходящих для решения поставленных задач.
3. Разработка тест-кейсов для тестирования web-приложения.
4. Реализация автоматизированного тестирования с использованием современных технологий, таких как Docker, Selenium и Allure.

1 Основы тестирования приложений

1.1 Жизненный цикл тестирования

Жизненный цикл тестирования веб-приложений начинается с планирования, когда на основе требований формируется тестовый план с определением целей, типов тестов, инструментов и критериев приёма. Далее аналитики изучают спецификации и пользовательские истории, выявляют критические функции и риски, создают матрицы трассировки, а затем проектируют тесты: позитивные и негативные сценарии, проверки граничных условий и подготовку данных.

После этого настраивается тестовая среда — сервера, базы данных, виртуальные машины и необходимое ПО — и начинается выполнение тестов: функциональных, производительных, безопасностных и UX-проверок как вручную, так и автоматически. По результатам фиксируются дефекты, составляются отчёты с рекомендациями, вносятся исправления, выполняется регрессионное тестирование и финальная оценка качества. После релиза осуществляется мониторинг работы приложения и периодическое тестирование обновлений, что обеспечивает стабильность и соответствие ПО ожиданиям пользователей.

1.2 Виды тестирования

Тестирование программного обеспечения включает два основных направления: функциональное и нефункциональное. Функциональное тестирование проверяет соответствие приложения требованиям на уровне отдельных функций (модульное тестирование), их взаимодействия (интеграционное тестирование) и работы системы в целом (системное тестирование), а приёмочное тестирование удостоверяется, что реализация удовлетворяет бизнес-критериям заказчика. Нефункциональное тестирование оценивает качественные характеристики системы: производительность (нагрузочное, стрессовое и объёмное тестирование), надёжность и безопасность (сканирование на уязвимости, тесты на проникновение), удобство использования (юзабилити) и совместимость (различные платформы, устройства и браузеры).

Регрессионное тестирование гарантирует, что внесённые изменения не нарушили существующую функциональность, а автоматизированные тесты ускоряют и упрощают проверку за счёт скриптов и специализированных инструментов. Каждый вид тестирования нацелен на обеспечение высокого качества ПО и удовлетворение потребностей пользователей.

1.3 Автоматизированное тестирование

Автоматизированное тестирование повышает эффективность QA-процессов, беря на себя рутинные и повторяющиеся проверки, что позволяет тестировщикам сосредоточиться на более сложных сценариях и снижает риск человеческих ошибок. За счёт быстрого и точного выполнения большого числа тестов оно ускоряет обнаружение дефектов, особенно в условиях сжатых сроков и непрерывных релизов, обеспечивая оперативную обратную связь при изменениях в коде и более полное покрытие функциональности.

Быстрая регрессия и частое прогон тестов в автоматическом режиме улучшают качество продукта и сокращают время выпуска новых версий. Автоматизация также упрощает интеграцию тестов в CI/CD-конвейеры, обеспечивая стабильность сборок и своевременное выявление проблем на ранних этапах разработки.

1.3.1 Методы автоматизированного тестирования

Автоматизированное тестирование включает несколько методов, каждый из которых решает свою задачу. Модульное тестирование проверяет отдельные функции в изоляции, быстро выявляя ошибки на ранних этапах разработки; интеграционное тестирование проверяет взаимодействие между модулями, а системное — работу всей системы в целом, включая и нефункциональные аспекты. Приёмочное тестирование удостоверяется в соответствии системы бизнес-требованиям, а регрессионное тестирование гарантирует, что новые изменения не нарушили существующую функциональность. Автоматизация этих видов тестирования обеспечивает высокую скорость выполнения, стабильность результатов и более полное покрытие тестами, хотя требует поддержания тестовых сценариев актуальными по мере эволюции кода.

Нефункциональное тестирование фокусируется на производительности, надёжности, безопасности, юзабилити и совместимости. Нагрузочное и стрессовое тестирование симулируют пиковые и экстремальные нагрузки, выявляя узкие места, тогда как безопасность проверяется с помощью сканирования уязвимостей и тестов на проникновение. Автоматизированное UI-тестирование (например, с Selenium) имитирует действия пользователя и проверяет отображение интерфейса на разных платформах. Хотя такие тесты могут быть медленнее и требовать сложной настройки, их регулярное применение в CI/CD-конвейере

существенно повышает качество продукта и ускоряет выпуск новых версий.

1.3.2 Инструменты автоматизированного тестирования

Инструменты автоматизированного тестирования позволяют существенно повысить качество и скорость проверки приложений, снижая нагрузку на команду QA и минимизируя человеческие ошибки. Среди наиболее популярных решений стоит выделить Selenium — универсальный фреймворк для автоматизации веб-приложений с поддержкой множества языков (Java, C#, Python, Ruby) и браузеров (Chrome, Firefox, Safari), а также Appium — аналогичный инструмент для мобильных платформ iOS и Android. Для модульного тестирования на Java широко применяются JUnit и его расширение TestNG, предлагающие гибкую конфигурацию, аннотации и параметризацию тестов. При тестировании веб-сервисов и API на первый план выходит SoapUI, поддерживающий как SOAP, так и REST, с мощными средствами создания, выполнения и анализа запросов.

Для организации «поведенческого» тестирования популярностью пользуется Cucumber, который позволяет описывать сценарии на естественном языке с помощью Gherkin-синтаксиса и интегрируется с различными языками программирования и фреймворками. Вместе эти инструменты охватывают все уровни тестирования — от низкоуровневых unit-тестов и интеграционных проверок до end-to-end сценариев и проверки API, что делает их неотъемлемой частью современных CI/CD-конвейеров.

2 Автоматизированное тестирование Web-приложения

2.1 Стратегия тестирования

В рамках данной работы проводилось тестирование веб-приложения расположенного по адресу https://киннор.рф/. Тестирование сайта включает в себя ряд этапов, начиная от планирования и заканчивая выполнением и анализом результатов.

Объектами тестирования является:

— Веб-интерфейс пользователя.
— Функциональность сайта, включая все пользовательские сценарии.
— Совместимость с различными браузерами и устройствами.
— Производительность сайта.
— Безопасность сайта.

2.2 Используемые инструменты автоматизированного тестирования

В данной работе для реализации автоматизированных тестов выбран язык Java и среда IntelliJ IDEA Community Edition, которая, несмотря на бесплатный статус, предоставляет все необходимые инструменты разработки на разных платформах (Windows, macOS, Linux). Java остаётся одним из самых популярных и надёжных языков благодаря своей мультиплатформенности, объектно-ориентированной модели и масштабируемости, что делает её удобным выбором для написания автотестов.

В качестве фреймворка для модульного тестирования используется JUnit, позволяющий легко организовать и выполнять проверки корректности отдельных блоков кода с высокой степенью повторяемости. Для автоматизации веб-интерфейса применяется Selenium WebDriver, обеспечивающий кроссбраузерное управление браузером и выполнение функциональных UI-тестов. Такое сочетание Java, JUnit и Selenium в IntelliJ IDEA обеспечивает эффективную, надёжную и масштабируемую платформу для автоматизированного тестирования веб-приложений.

2.3 Разработка тест-кейсов

В рамках работы были разработаны тест-кейсы для проверки ключевых пользовательских сценариев приложения. Во-первых, проверяется функция поиска: при переходе на главную страницу пользователь кликает по иконке поиска,

вводит запрос «Сукка Проекта Кешер» и подтверждает его нажатием Enter, после чего система должна вернуть релевантные результаты, соответствующие введённому запросу. Во-вторых, тестируется переход к странице оплаты пожертвований: с главной страницы пользователь нажимает кнопку «Поддержать проект», выбирает «Другая сумма», вводит 500 рублей, заполняет имя и адрес электронной почты, после чего система должна перенаправить его на страницу оплаты YooMoney с указанной суммой.

Кроме того, был создан тест-кейс для проверки навигации на страницу «Контакты»: при клике на соответствующую ссылку в главном меню приложение должно перейти на страницу с контактной информацией. Каждый тест-кейс содержит описание предпосылок, последовательность действий и ожидаемый результат, что обеспечивает систематическую и воспроизводимую проверку основных функций интерфейса.

2.4 Автоматизированное тестированное с использованием Selenium

Для автоматизации тестирования использован паттерн «Page Object», при котором все локаторы и методы взаимодействия вынесены в класс `MainPage`. Это позволяет инкапсулировать логику работы с элементами, повторно использовать код и быстро адаптировать тесты при изменениях интерфейса. Например, в тесте `SearchFilterTest` на Java, Selenium и JUnit в методе `@BeforeAll` настраивается ChromeOptions и WebDriver, в `@BeforeEach` — открывается главная страница, а сам тест создаёт объект `MainPage`, вызывает методы для открытия поля поиска, ввода текста и отправки запроса, после чего проверяет результат через `assertEquals`.

Аналогично реализованы автотесты для проверки перехода на страницу пожертвований и страницы «Контакты»: в каждом классе в `@BeforeEach` происходит инициализация браузера и переход на главную страницу, в `@Test` — выполняются шаги пользовательского сценария (клик по кнопкам, ввод данных, ожидание перенаправления), а в `@AfterEach` — закрытие браузера. Такой подход обеспечивает структурированность, читаемость и надёжность проверок ключевых функций веб-приложения.

2.5 Интеграция результатов тестирования в Allure

Для визуализации результатов тестирования в дипломной работе используется Allure, который собирает подробные отчёты с эпиками, фичами, история-

ми, степами и степени важности, а также включает вложения — скриншоты, логи запросов и ответов. Интеграция Allure происходит через добавление зависимостей в `pom.xml` и аннотаций (`@Epic` , `@Feature` , `@Story` , `@Step` , `@Severity` , `@Description` и др.) в тестах на JUnit 5. Это позволяет группировать и документировать тесты, добавлять понятные описания шагов и автоматически фиксировать, на каком шаге произошёл сбой.

Для удобства обмена отчётами используется одностраничный формат Allure (`--single-file`), который упаковывает все ресурсы (HTML, CSS, JS, изображения) в один файл. Такой отчёт легко хранить, передавать и интегрировать в CI/CD (Jenkins, GitLab CI) или загружать в облачные хранилища (например, Yandex.Cloud Storage) без развёртывания веб-сервера. Хотя интерактивность и производительность одностраничного отчёта ниже, чем у традиционного, его компактность и простота использования делают его оптимальным решением для регулярной доставки результатов тестирования.

2.6 Интеграция с Yandex.Cloud Storage для сохранения результатов тестирования

В дипломной работе интеграция с Yandex.Cloud Storage была реализована для централизованного хранения отчётов Allure, что значительно упрощает их анализ и совместный доступ. Yandex.Cloud Storage, совместимое с S3 API, обеспечивает масштабируемость, высокую доступность и безопасность за счёт шифрования и гибких прав доступа. С помощью утилиты s3cmd, установленной в Docker-контейнере, отчёты автоматически загружаются в указанный бакет после генерации: сначала в Dockerfile устанавливается s3cmd и создаётся конфигурационный файл с учётными данными, а затем скрипт `upload_results.sh` копирует все файлы из директории с отчётом в облако.

Процесс выгрузки отчётов организован следующим образом: после запуска тестов Allure генерирует HTML-отчёт в локальной папке, далее скрипт `upload_results.sh` запускается в контейнере, подключенном к Yandex.Cloud Storage, и с помощью s3cmd загружает все файлы отчёта в нужный бакет. По завершении файлы становятся доступны по URL из консоли Yandex.Cloud или через s3cmd, что делает процесс хранения и распространения отчётов простым, безопасным и воспроизводимым.

2.7 Автоматизация UI-тестов с использованием Docker

В рамках дипломной работы был предложен современный подход к автоматизации UI-тестов, объединяющий контейнеризацию, браузерную автоматизацию и облачные сервисы. Все компоненты тестовой среды (Java 11, Google Chrome, ChromeDriver, Selenium и Allure) упакованы в единый Docker-контейнер, что обеспечивает воспроизводимость и изоляцию окружения. После выполнения тестов Allure генерирует подробные отчёты, которые автоматически выгружаются в Yandex.Cloud Storage, позволяя команде централизованно хранить и анализировать результаты.

Этот метод отражает актуальные тенденции CI/CD-автоматизации и демонстрирует практическую пользу современных технологий. Использование паттерна Page Object упрощает поддержку и расширение тестов, а аннотации Allure обеспечивают информативную визуализацию шагов и метаданных. Внедрение Docker и облачного хранилища говорит о комплексном подходе к качеству ПО и готовности решения к применению в реальных проектах.

2.7.1 Технология контейнеризации автоматизированных тестов

Контейнеризация UI-тестов в Docker начинается с создания Dockerfile, который на базе образа `openjdk:11` устанавливает все необходимые компоненты: Maven, Google Chrome с ChromeDriver, утилиты s3cmd и yc для работы с Yandex.Cloud, а также Allure для генерации отчётов. В Dockerfile задаётся рабочая директория, копируются исходники, устанавливаются локализация, зависимости и браузерные инструменты, после чего происходит сборка проекта, генерация отчёта и автоматическая загрузка результата в облако через скрипт `upload_results.sh`. Контейнер запускается одной командой, выполняет тесты, создаёт отчёт и выгружает его в указанный бакет Yandex.Cloud Storage, доступный по единому URL.

Такой подход обеспечивает полную воспроизводимость и изоляцию тестовой среды, устраняя «это работает только на моей машине». Автоматизация развёртывания через Docker упрощает интеграцию в CI/CD (Jenkins, GitLab CI, GitHub Actions) и позволяет параллельно запускать несколько контейнеров для ускорения тестов. К преимуществам относятся переносимость окружения, единая точка конфигурации и централизованное хранение отчётов, а к недостаткам — первоначальная сложность настройки и зависимость от наличия ин-

тернета для загрузки образов и отчётов.

2.7.2 Будущие направления развития автоматизации тестирования с использованием Docker

Автоматизация тестирования на базе Docker продолжит эволюционировать в направлении полной оркестрации и интеллектуализации. Интеграция с Kubernetes позволит автоматически масштабировать и распределять нагрузку тестовых контейнеров, обеспечивая высокую доступность и упрощённое управление окружениями. Serverless-архитектуры (AWS Lambda, Google Cloud Functions) дадут возможность запускать тесты по требованию с оплатой только за фактическое время выполнения, а облачные функции начнут поддерживать контейнеры для изолированных и лёгковесных прогонов.

Параллельно развивается внедрение AI и ML-инструментов, способных анализировать результаты тестов, предсказывать проблемные зоны в коде и автоматически генерировать новые сценарии на основе поведения пользователей. Новые фреймворки — Cypress, Playwright, TestCafe — уже оптимизированы для контейнеризированных и распределённых сред, а их тесная интеграция с CI/CD и облачными платформами ускоряет непрерывную проверку. В совокупности эти тенденции формируют будущее тестирования как полностью автоматизированного, адаптивного и легко масштабируемого процесса.

ЗАКЛЮЧЕНИЕ

В ходе выполнения дипломной работы были успешно решены поставленные задачи, связанные с исследованием современных методов и инструментов автоматизированного тестирования веб-приложений. В работе были рассмотрены ключевые аспекты тестирования, включая жизненный цикл тестирования, виды и методы тестирования, а также инструменты автоматизации, такие как Selenium, JUnit и Allure.

Для достижения поставленной цели были решены следующие задачи:

1. Исследование современных методов тестирования программного обеспечения, включая функциональное, нагрузочное, интеграционное и другие виды тестирования.
2. Анализ инструментов тестирования и выбор наиболее подходящих для решения поставленных задач.
3. Разработка тест-кейсов для тестирования web-приложения.
4. Реализация автоматизированного тестирования с использованием современных технологий, таких как Docker, Selenium и Allure.

В заключение можно отметить, что предложенный подход к автоматизации тестирования с использованием Docker, Selenium и Allure является эффективным решением для обеспечения высокого качества веб-приложений и может быть рекомендован для внедрения в реальные проекты.