Загидуллин Рустам Мубаризович, бакалавр
Уфимский университет науки и технологий
Zagidullin Rustam Mubarizovich
Ufa University of Science and Technology

АВТОМАТИЗИРОВАННОЕ ТЕСТИРОВАНИЕ СОВРЕМЕННЫХ ВЕБ-ПРИЛОЖЕНИЙ: УРОВНИ, ИНСТРУМЕНТЫ И ОРГАНИЗАЦИЯ ПРОЦЕССА
AUTOMATED TESTING OF MODERN WEB APPLICATIONS: LEVELS, TOOLS, AND PROCESS ORGANIZATION

Аннотация. Систематизированы уровни автоматизированного тестирования современных веб-приложений. Рассмотрены модульные, компонентные, интеграционные и сквозные проверки, устойчивые способы взаимодействия с интерфейсом и выполнение тестов в конвейере поставки. Предложен риск-ориентированный процесс, объединяющий функциональный, доступностный и безопасностный контроль.

Abstract. The levels of automated testing for modern web applications are systematized. Unit, component, integration, and end-to-end checks, stable user interface interaction, and continuous delivery execution are reviewed. A risk-based process combining functional, accessibility, and security controls is proposed.

Ключевые слова: Веб-приложение, автоматизированное тестирование, интеграционное тестирование, сквозной тест, WebDriver, качество.

Keywords: Web application, automated testing, integration testing, end-to-end test, WebDriver, quality.

Введение

Современное веб-приложение состоит из интерфейсных компонентов, клиентского состояния, API, серверной логики, хранилищ и внешних сервисов. Ошибка может возникнуть внутри функции, на границе модулей или только в реальном браузерном сценарии. Один тип тестов не способен одновременно обеспечить быструю локализацию дефекта и полное подтверждение пользовательского маршрута. Поэтому автоматизация строится как система уровней с различной стоимостью, скоростью и областью уверенности.

Цель статьи – определить рациональную структуру автоматизированного контроля веб-приложения. Анализируются модульные, компонентные, интеграционные и сквозные проверки, стабильные локаторы и ожидания, изоляция данных, обработка нестабильных тестов и размещение проверок в конвейере поставки. Подход опирается на риск: глубина контроля определяется последствиями отказа и вероятностью изменения, а не стремлением получить максимальное число тестов.

Многоуровневая модель автоматизации

ISTQB разделяет уровни тестирования по объекту и целям, включая компонентный, компонентно-интеграционный, системный, системно-интеграционный и приемочный уровни [1]. В веб-проекте границы адаптируются к архитектуре. Модульная проверка изолирует функцию, преобразователь, правило валидации или редуктор состояния. Она быстро выполняется и точно указывает место отказа, но не подтверждает корректность браузерного DOM, сети и серверного контракта.

Компонентный тест отображает элемент интерфейса вместе с ближайшим поведением: вводом, событиями, состояниями загрузки и ошибками. Предпочтительны запросы по роли, доступному имени и подписи, поскольку они отражают способ восприятия интерфейса пользователем. Testing Library прямо ориентирует тесты на пользовательское взаимодействие и рекомендует избегать проверки внутренних методов и состояния компонента [4]. Такой тест устойчивее к рефакторингу разметки, если внешний контракт не изменился.

Интеграционная проверка охватывает границу между клиентом и API, сервером и базой данных, обработчиком и внешним сервисом. Здесь выявляются расхождения форматов, сериализации, авторизации, транзакций и обработки ошибок. Зависимость можно заменить управляемым двойником, если тестируется клиентская реакция, но критический контракт отдельно проверяется против реальной реализации. Иначе набор зеленых тестов подтверждает согласованность только с имитацией.

Сквозной тест запускает приложение в браузере и проходит пользовательский маршрут через развернутые компоненты. WebDriver стандартизует удаленное управление браузером, навигацию, поиск элементов, действия и получение их состояния [2]. Этот уровень дает высокую уверенность в соединении частей, но медленнее и чувствительнее к данным и окружению. Поэтому им покрываются критические маршруты: вход, ключевая транзакция, изменение данных, восстановление после ошибки и завершение сеанса.

Устойчивость тестов веб-интерфейса

Локатор является частью тестового контракта. Селектор, привязанный к цепочке CSS-классов или позиции элемента, ломается при косметическом изменении. Поиск по роли и доступному имени ближе к реальному действию пользователя и одновременно стимулирует семантическую разметку [4]. Специальный test-id оправдан, когда устойчивого пользовательского признака нет, но не должен заменять осмысленные подписи и роли во всем интерфейсе.

Асинхронность требует ожидать наблюдаемое состояние, а не фиксированное время. Playwright выполняет проверки применимости перед действием, а веб-ориентированные утверждения повторяются до выполнения условия или истечения тайм-аута [3]. Фиксированная задержка либо замедляет быстрый запуск, либо остается недостаточной на нагруженной машине. Ожидание формулируется через видимость, доступность, текст, URL или сетевой результат, который действительно означает готовность сценария.

Каждый тест получает контролируемое исходное состояние. Данные создаются через API или фикстуру, идентификаторы не разделяются между параллельными запусками, время и случайность фиксируются там, где влияют на результат. Тест не зависит от порядка выполнения и очищает созданные сущности. Внешний сервис заменяется двойником только на явно определенной границе; отдельный контрактный набор подтверждает, что имитация соответствует реальному протоколу.

Нестабильный тест рассматривается как дефект средства контроля. Для расследования сохраняются шаги, снимок страницы, трассировка, консоль, сетевые обращения и версия окружения. Повторный запуск может собрать диагностические данные, но не превращает случайный результат в успешный выпуск. Полезные показатели включают долю нестабильных запусков, длительность по уровням, частоту дефектов после выпуска и среднее время локализации причины. Простое количество тестов не измеряет защищенность критических рисков.

Интеграция контроля в процесс разработки

Конвейер организуется от дешевых проверок к дорогим. При каждом изменении выполняются статический анализ, модульные и компонентные тесты; после сборки – интеграционные и небольшой набор критических сквозных сценариев. Расширенная матрица браузеров и длительные проверки запускаются по расписанию или перед выпуском. Риск-ориентированный подход выбирает и приоритизирует активности на основе вероятности и воздействия отказа [1]. Поэтому платежная операция получает более глубокое покрытие, чем декоративная настройка.

Функциональная автоматизация не заменяет проверку безопасности. OWASP WSTG предлагает структурированный набор сценариев для сбора информации, конфигурации, управления идентификацией и сессиями, авторизации, валидации входа и других областей веб-приложения [5]. Часть проверок включается в конвейер, а анализ бизнес-логики и проникновение выполняются отдельно. Обнаруженная уязвимость приводит не только к исправлению, но и к регрессионному тесту применимого уровня.

Доступность также требует сочетания автоматического и ручного контроля. ACT Rules Format 1.1 задает воспроизводимый формат правил, пригодных для инструментов и ручных методик, и явно учитывает возможность ложноположительных и ложноотрицательных результатов [6]. Автоматизация выявляет формализуемые нарушения, но не подтверждает логичность фокуса, понятность сообщения и удобство полного маршрута. Эти свойства проверяются клавиатурой, вспомогательными технологиями и пользовательской оценкой.

Тестируемость закладывается в требования и архитектуру: у операции определяются наблюдаемый результат, допустимые ошибки, границы времени и способ подготовки данных. NIST SSDF рекомендует встраивать безопасные практики в конкретную реализацию жизненного цикла разработки, чтобы снижать число уязвимостей и устранять причины повторения [7]. Аналогично функциональный контроль становится постоянной частью разработки: критерии приемки связываются с тестами, набор пересматривается после изменений, а исключение из обязательного запуска имеет владельца и срок.

Заключение

Эффективная автоматизация веб-приложения представляет собой многоуровневую систему. Модульные тесты быстро проверяют правила, компонентные подтверждают поведение интерфейса, интеграционные контролируют границы, а сквозные воспроизводят критические пользовательские маршруты в браузере. Устойчивость обеспечивают пользовательские локаторы, ожидание состояния, изоляция данных и обязательное расследование нестабильности.

Процесс строится по риску и объединяет функциональные, безопасностные и доступностные проверки. Быстрые наборы защищают каждое изменение, дорогие проверки применяются к наиболее значимым сценариям, а найденный дефект превращается в воспроизводимую регрессию. Такой подход оценивает не количество тестов, а управляемую уверенность в свойствах продукта и его готовности к выпуску

Список литературы:

1. International Software Testing Qualifications Board. Certified Tester Foundation Level Syllabus v4.0.1. 2024. URL: https://istqb.org/wp-content/uploads/2024/11/ISTQB_CTFL_Syllabus_v4.0.1.pdf (дата обращения: 16.08.2026).

2. W3C. WebDriver. Working Draft, 02.07.2026. URL: https://www.w3.org/TR/webdriver2/ (дата обращения: 16.08.2026).

3. Microsoft. Playwright Documentation: Writing Tests and Assertions. URL: https://playwright.dev/docs/writing-tests (дата обращения: 16.08.2026).

4. Testing Library. Introduction and Guiding Principles. Updated 22.01.2026. URL: https://testing-library.com/docs/ (дата обращения: 16.08.2026).

5. OWASP Foundation. Web Security Testing Guide. Version 4.2. 2020. URL: https://owasp.org/www-project-web-security-testing-guide/v42/ (дата обращения: 16.08.2026).

6. W3C. Accessibility Conformance Testing Rules Format 1.1. W3C Recommendation, 05.02.2026. URL: https://www.w3.org/TR/act-rules-format/ (дата обращения: 16.08.2026).

7. Souppaya M., Scarfone K., Dodson D. Secure Software Development Framework Version 1.1. NIST SP 800-218. 2022. DOI: 10.6028/NIST.SP.800-218