УДК 004.41

МЕТОДИКА АВТОМАТИЗИРОВАННОЙ ОЦЕНКИ КАЧЕСТВА ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА ВЕБ-ПРИЛОЖЕНИЙ НА ОСНОВЕ КОМПЛЕКСА ПРОГРАММНЫХ БИБЛИОТЕК

Кулагин А.В.

студент,

Поволжский государственный университет телекоммуникаций и информатики, Самара, Россия

Куваева Е.Н.

старший преподаватель кафедры информационных систем и технологий, Поволжский государственный университет телекоммуникаций и информатики, Самара, Россия

Аннотация. В статье рассматривается задача комплексной автоматизированной оценки качества пользовательского интерфейса современных веб-приложений. Предлагается методика, объединяющая функциональные, навигационные, визуальные и эксплуатационные показатели в единую модель оценки. Для реализации прототипа рассматривается комплекс из семи программных библиотек и инструментов: React, React Router, Axios, Chart.js, Playwright, Zod и Lighthouse. Сформирована архитектура сбора и агрегации показателей, определены группы критериев, предложена нормализация результатов и рассчитан интегральный показатель качества интерфейса. Практическая ценность подхода заключается в возможности применять одинаковую процедуру при разработке и регрессионном тестировании информационных систем. Показано, что разделение оценки на пользовательский сценарий, автоматизированные проверки и анализ производительности позволяет снизить субъективность при принятии решений о доработке интерфейса.

Ключевые слова: пользовательский интерфейс, веб-приложение, информационная система, автоматизированное тестирование, качество интерфейса, React, Playwright, Lighthouse, производительность, доступность.

A METHOD FOR AUTOMATED EVALUATION OF WEB APPLICATION USER INTERFACE QUALITY BASED ON A SET OF SOFTWARE LIBRARIES

Kulagin A.V.
Student,
Volga Region State University of Telecommunications and Informatics,
Samara, Russia
Kuvaeva E.N.
Senior Lecturer, Department of Information Systems and Technologies,
Volga Region State University of Telecommunications and Informatics,
Samara, Russia

Abstract. The paper considers the problem of comprehensive automated evaluation of the quality of user interfaces in modern web applications. A methodology is proposed that combines functional, navigation, visual and operational indicators into a unified assessment model. The prototype architecture uses seven software libraries and tools: React, React Router, Axios, Chart.js, Playwright, Zod and Lighthouse. The paper defines groups of quality criteria, describes the collection and aggregation of indicators, and proposes a normalized integral quality score. The practical value of the approach is the possibility of applying the same assessment procedure during information system development and regression testing. Separating user scenarios, automated checks and performance analysis can reduce subjectivity when selecting interface improvements.

Keywords: user interface, web application, information system, automated testing, interface quality, React, Playwright, Lighthouse, performance, accessibility.

ВВЕДЕНИЕ

Пользовательский интерфейс является одним из ключевых элементов современной информационной системы, поскольку именно через него пользователь воспринимает функциональные возможности программного продукта. Даже при корректной реализации серверной логики система может восприниматься как неудобная, если основные операции требуют большого количества действий, элементы управления имеют непредсказуемое поведение, страницы загружаются медленно или интерфейс недостаточно доступен для различных категорий пользователей. Поэтому качество интерфейса целесообразно рассматривать не только как результат экспертной оценки дизайна, но и как измеряемое свойство информационной системы.

Актуальность автоматизации такой оценки связана с распространением компонентных веб-фреймворков и необходимостью регулярно проверять интерфейс после внесения изменений. В приложениях на React интерфейс строится из повторно используемых компонентов, что упрощает развитие системы, но одновременно повышает значимость регрессионного контроля. React Router позволяет организовать маршрутизацию и переходы между экранами приложения, а Playwright предоставляет средства запуска автоматизированных сценариев в браузере [1; 2].

Цель исследования — разработать методику комплексной автоматизированной оценки качества пользовательского интерфейса веб-приложения, пригодную для включения в жизненный цикл разработки информационной системы. Для достижения цели поставлены задачи: определить группы измеряемых характеристик; выбрать программные библиотеки для реализации прототипа; разработать архитектуру сбора данных; предложить способ нормализации разнородных метрик; сформировать интегральный показатель; определить порядок интерпретации результата.

ОБЗОР ЛИТЕРАТУРЫ

Понятие качества пользовательского интерфейса связано с удобством, эффективностью и удовлетворённостью пользователя при выполнении конкретных задач. В стандарте ISO 9241-11 качество взаимодействия связывается с результативностью, эффективностью и удовлетворённостью в заданном контексте использования [3]. Практические руководства по проектированию интерфейсов также подчёркивают необходимость ясной навигации, обратной связи от системы, согласованности элементов и предотвращения ошибок [4].

Автоматизированный контроль дополняет экспертный анализ. Lighthouse формирует отчёты по производительности, доступности, практикам разработки и поисковой оптимизации, что делает его подходящим источником объективных метрик [5]. Playwright, в свою очередь, позволяет выполнять пользовательские сценарии в браузере и проверять ожидаемое состояние страницы после последовательности действий [2].

Компонентный подход к разработке интерфейсов получил широкое распространение благодаря React. Документация React описывает интерфейс как совокупность компонентов, которые могут быть вложенными и повторно использоваться в разных частях приложения [1]. Для многоэкранных приложений важна маршрутизация: React Router предоставляет декларативную настройку маршрутов и поддерживает навигацию между состояниями интерфейса [6]. Axios может использоваться для стандартизации HTTP-взаимодействия клиентской части с сервером [7]. Для проверки входных и промежуточных данных в TypeScript-проекте целесообразно применять схемы валидации Zod [8]. Визуализация собранных показателей может быть выполнена средствами Chart.js [9].

Таким образом, существующие средства хорошо решают отдельные задачи: разработку компонентов, маршрутизацию, получение данных, проверку схем, браузерное тестирование или аудит производительности. Однако их раздельное

использование не формирует единой модели качества интерфейса. Это определяет исследовательскую задачу данной работы: связать показатели разных уровней в общий процесс оценки информационной системы.

ПОСТАНОВКА ЗАДАЧИ

Пусть веб-приложение рассматривается как информационная система S, содержащая набор пользовательских сценариев U = {u1, u2, ..., un}. Для каждого сценария необходимо определить набор наблюдаемых показателей. В работе выделены четыре группы: функциональная корректность F, навигационная эффективность N, визуальная и доступностная пригодность A, а также эксплуатационная производительность P. Такое разделение позволяет не смешивать причины неудовлетворительного результата.

Функциональная группа включает успешность выполнения основных операций, корректность переходов и обработку ошибочных действий. Навигационная группа характеризует количество шагов, наличие понятной структуры маршрутов и предсказуемость переходов. Группа A включает наличие доступных имён элементов управления, корректность структуры страницы, контрастные и семантические характеристики. Группа P включает показатели загрузки и отклика, получаемые инструментами автоматизированного аудита.

МЕТОДИКА ИССЛЕДОВАНИЯ

Предлагаемая методика строится по принципу последовательной обработки: пользовательский сценарий задаётся как тестовый набор; приложение запускается в контролируемом окружении; Playwright выполняет сценарии и фиксирует функциональные результаты; Lighthouse формирует аудит страницы; клиентский модуль получает необходимые данные через Axios; Zod проверяет структуру поступивших результатов; Chart.js используется для представления итогов; React и React Router образуют пользовательскую часть панели мониторинга.

Таким образом, в прототипе задействовано семь библиотек и инструментов: React, React Router, Axios, Chart.js, Playwright, Zod и Lighthouse. Их роли различаются: React отвечает за компонентную структуру панели; React Router за маршрутизацию; Axios за обмен данными; Zod за проверку схем; Playwright за воспроизведение сценариев; Lighthouse за аудит; Chart.js за визуализацию результатов. Такое разделение уменьшает связанность модулей и позволяет заменять отдельные инструменты без изменения общей модели оценки.

Таблица 1 – Программные библиотеки и инструменты прототипа

Инструмент	**Назначение**	**Оцениваемый аспект**
React	Компонентная реализация интерфейса	Структура и повторное использование UI
React Router	Маршрутизация	Навигация и переходы
Axios	HTTP-взаимодействие	Получение данных для сценариев
Zod	Валидация схем	Корректность входных результатов
Playwright	Браузерное тестирование	Функциональность и пользовательские сценарии
Lighthouse	Автоматизированный аудит	Производительность, доступность, практики
Chart.js	Визуализация	Представление итоговых показателей

Для устранения различий в шкалах показатели нормализуются. Для метрики, где большее значение соответствует лучшему качеству, используется преобразование $q = (x - x_{min})/(x_{max} - x_{min})$. Для метрики, где увеличение значения ухудшает результат, применяется $q = (x_{max} - x)/(x_{max} - x_{min})$. После нормализации каждая группа получает значение от 0 до 1. Интегральная оценка определяется как $I = w_F \cdot F + w_N \cdot N + w_A \cdot A + w_P \cdot P$, где сумма весов равна единице. При отсутствии специфических требований предметной области можно использовать равные веса.

Если для системы критична, например, скорость выполнения операций, вес P увеличивается.

Для интерпретации предлагается использовать четыре диапазона: I от 0 до 0,49 — неудовлетворительное качество; 0,50–0,69 — приемлемое; 0,70–0,84 — хорошее; 0,85–1,00 — высокое. Границы не являются универсальным стандартом и должны уточняться после накопления результатов по конкретному классу информационных систем. Их назначение — создать единый язык сравнения версий приложения.

АРХИТЕКТУРА ПРОТОТИПА

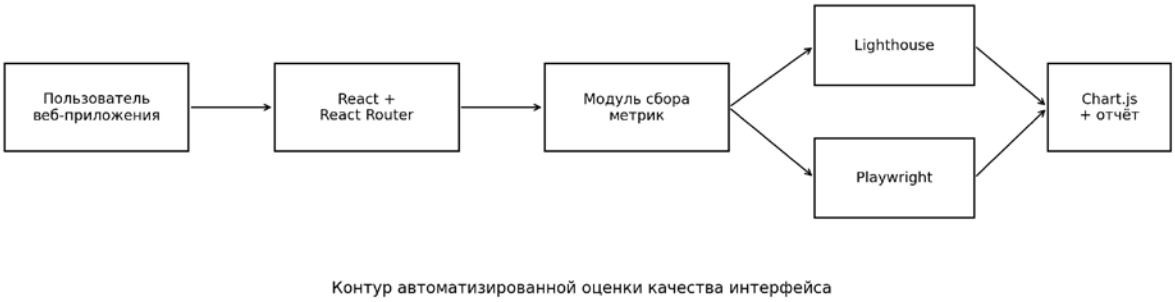


Рис. 1 – Архитектура автоматизированной оценки качества интерфейса

Источник: разработано автором.

На рисунке 1 пользовательский интерфейс и маршрутизация формируют объект проверки. Модуль сбора метрик инициирует два независимых контура: браузерные сценарии Playwright и аудит Lighthouse. После завершения проверок результаты приводятся к единой схеме. На этом этапе Zod позволяет отбрасывать некорректные или неполные записи. Далее показатели агрегируются и передаются в модуль визуализации.

Преимущество архитектуры состоит в разделении функционального и нефункционального тестирования. Например, успешное выполнение сценария

входа пользователя не означает, что страница обладает достаточной скоростью загрузки или доступностью. Поэтому итоговый показатель формируется только после получения результатов всех обязательных групп.

ПРАКТИЧЕСКАЯ МОДЕЛЬ ПОЛЬЗОВАТЕЛЬСКОГО СЦЕНАРИЯ

В качестве базового сценария для информационной системы можно использовать последовательность «открытие главной страницы — переход к каталогу — поиск объекта — открытие карточки — выполнение целевого действия». Playwright позволяет представить такую последовательность в виде автоматизированного теста и проверять состояние элементов после каждого перехода. На каждом шаге фиксируются успешность операции, наличие целевого элемента и время выполнения. Если хотя бы один критический шаг не выполнен, функциональная оценка сценария снижается независимо от визуальных характеристик.

Для многоэкранного интерфейса отдельное значение имеет проверка маршрутов. Некорректная ссылка, потеря состояния при переходе или отсутствие понятного возврата к предыдущему экрану создают дополнительную когнитивную нагрузку. Поэтому навигационный показатель целесообразно рассчитывать как функцию числа лишних действий и доли успешно завершённых переходов. React Router предоставляет средства построения иерархии маршрутов, а тестовый слой проверяет, что каждый маршрут приводит пользователя к ожидаемому состоянию [6].

Производительность и доступность не следует сводить к одному показателю. Lighthouse формирует отдельные категории аудита, поэтому в предлагаемой модели они входят в группу P и A соответственно. Это позволяет увидеть ситуацию, при которой интерфейс быстро загружается, но содержит проблемы доступности, либо, наоборот, обладает хорошей доступностью при неудовлетворительной скорости.

РЕЗУЛЬТАТЫ ИССЛЕДОВАНИЯ

Результатом исследования является формализованная последовательность оценки интерфейса, которую можно включить в процесс разработки информационной системы: 1) формирование набора пользовательских сценариев; 2) автоматический запуск сценариев; 3) аудит страниц; 4) проверка структуры полученных данных; 5) нормализация показателей; 6) расчёт групповых оценок; 7) расчёт интегрального показателя; 8) визуализация проблем; 9) повторная проверка после внесения изменений.

Предложенная схема позволяет перейти от разрозненных сообщений инструментов разработки к единому отчёту. Например, если после изменения компонента время загрузки уменьшилось, но один из пользовательских сценариев стал завершаться с ошибкой, итоговая оценка не должна автоматически считаться улучшенной. Интегральный показатель в таком случае отражает конфликт между эксплуатационной и функциональной характеристиками. Это является важным отличием комплексной оценки от проверки одного показателя.

Отдельный практический эффект даёт сохранение результатов по версиям. Для каждой версии приложения можно хранить значения F, N, A, P и I, после чего Chart.js может отображать изменение показателей во времени. Такой подход позволяет выявлять регрессии даже в тех случаях, когда абсолютная ошибка отдельной метрики кажется небольшой. Для информационных систем, развиваемых итерационно, именно динамика показателей может быть более информативной, чем единичное измерение.

ОБСУЖДЕНИЕ

Главным ограничением предложенной модели является невозможность полностью автоматизировать субъективные аспекты удобства. Количество кликов или успешность сценария не отражают все особенности восприятия интерфейса. Например, два экрана могут требовать одинакового числа действий, но один из них

будет значительно понятнее пользователю. Поэтому автоматизированную оценку следует рассматривать как дополнение к экспертному анализу и пользовательскому тестированию, а не как их полную замену.

Второе ограничение связано с зависимостью результатов производительности от среды запуска. Сетевые задержки, характеристики устройства, версия браузера и параллельная нагрузка могут влиять на показатели. Следовательно, сравнение версий должно выполняться в максимально одинаковых условиях. В противном случае изменение оценки может быть вызвано не изменением интерфейса, а внешним фактором.

Третье ограничение связано с выбором весов интегрального показателя. Равные веса удобны для демонстрационного прототипа, но не всегда корректны для прикладной информационной системы. Для банковского интерфейса критичнее ошибки операций и доступность, для аналитической панели — скорость визуализации и читаемость, для каталога — навигация и поиск. Поэтому веса должны определяться требованиями конкретной системы.

Перспективным направлением является накопление исторических данных и автоматическое выявление аномальных изменений показателей. В этом случае система оценки может перейти от разового отчёта к постоянному контролю качества интерфейса. Дополнительно можно расширить модель данными реальных пользовательских сессий, если соблюдены требования безопасности и конфиденциальности.

ЗАКЛЮЧЕНИЕ

В работе предложена методика автоматизированной комплексной оценки качества пользовательского интерфейса веб-приложений, рассматриваемого как составная часть современной информационной системы. В отличие от проверки одного параметра методика объединяет функциональные, навигационные, доступностные и эксплуатационные показатели.

Разработана архитектура прототипа, использующая семь программных библиотек и инструментов: React, React Router, Axios, Zod, Playwright, Lighthouse и Chart.js. Для разнородных результатов предложена нормализация и расчёт интегрального показателя с настраиваемыми весами. Сформирована последовательность автоматизированного контроля, пригодная для повторного запуска после изменения программного кода.

Практическая значимость работы заключается в возможности применять предложенную модель как часть процесса регрессионного тестирования информационной системы. При этом полученная оценка не заменяет экспертную проверку, а обеспечивает объективную основу для сравнения версий и приоритизации доработок. Дальнейшее развитие методики связано с проведением экспериментального исследования на реальном наборе веб-приложений, калибровкой весовых коэффициентов и сопоставлением автоматических оценок с результатами пользовательских исследований.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. React Documentation. Learn React [Электронный ресурс]. – URL: https://react.dev/learn (дата обращения: 14.09.2026).

2. Playwright Documentation. Running and debugging tests [Электронный ресурс]. – URL: https://playwright.dev/docs/running-tests (дата обращения: 14.09.2026).

3. ISO 9241-11:2018. Ergonomics of human-system interaction – Part 11: Usability: Definitions and concepts. – Geneva: International Organization for Standardization, 2018.

4. Nielsen J. Usability Engineering. – San Francisco: Morgan Kaufmann, 1994. – 362 p.

5. Chrome for Developers. Lighthouse: Optimize website speed [Электронный ресурс]. – URL: https://developer.chrome.com/docs/devtools/lighthouse/ (дата обращения: 14.09.2026).

6. React Router Documentation. React Router Home [Электронный ресурс]. – URL: https://reactrouter.com/home (дата обращения: 14.09.2026).

7. Axios Documentation. Introduction [Электронный ресурс]. – URL: https://axios-http.com/docs/intro (дата обращения: 14.09.2026).

8. Zod Documentation. Basic usage [Электронный ресурс]. – URL: https://zod.dev/basics (дата обращения: 14.09.2026).

9. Chart.js Documentation. Getting Started [Электронный ресурс]. – URL: https://www.chartjs.org/docs/latest/getting-started/ (дата обращения: 14.09.2026).

10. W3C. Web Content Accessibility Guidelines (WCAG) 2.2 [Электронный ресурс]. – URL: https://www.w3.org/TR/WCAG22/ (дата обращения: 14.09.2026).

11. Krug S. Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability. – Berkeley: New Riders, 2014. – 216 p.

12. Sommerville I. Software Engineering. – 10th ed. – Boston: Pearson, 2016. – 816 p.

13. Pressman R.S., Maxim B.R. Software Engineering: A Practitioner's Approach. – 9th ed. – New York: McGraw-Hill Education, 2019. – 768 p.

14. Shneiderman B., Plaisant C., Cohen M., Jacobs S., Elmqvist N., Diakopoulos N. Designing the User Interface: Strategies for Effective Human-Computer Interaction. – 6th ed. – Pearson, 2016. – 672 p.

15. Nielsen J., Budiu R. Mobile Usability. – Berkeley: New Riders, 2013. – 280 p.