УДК 004.415.533

АВТОМАТИЗАЦИЯ ТЕСТИРОВАНИЯ ДОСТУПНОСТИ КАК ЧАСТЬ ПРОЦЕССА РАЗРАБОТКИ ВЕБ-ПРИЛОЖЕНИЙ

Концевая В.Д., Толстенков И.А.

Белорусский государственный университет информатики и радиоэлектроники, г. Минск, Республика Беларусь

Научный руководитель: Коваленко И.В. – магистр экономических наук, ст. преподаватель кафедры ПИКС

Аннотация. В статье рассматривается практический подход к автоматизации тестирования доступности веб-приложений. Описана интеграция инструментов *eslint-plugin-jsx-a*11*y*, *jest-axe* и *GitHub Actions* в процесс разработки *React*-приложений. Проведён эксперимент с демонстрационным приложением, позволивший выявить типичные нарушения требований доступности.

Ключевые слова: доступность веб-приложений, *accessibility testing*, автоматизация тестирования, *React*, *CI*/*CD*.

Введение. Доступность веб-приложений является важным требованием при разработке современных цифровых сервисов. Она обеспечивает возможность использования информационных систем различными категориями пользователей, включая людей с ограниченными возможностями. В Республике Беларусь необходимость обеспечения доступности информационных ресурсов закреплена рядом нормативных документов, в частности Законом «О социальной защите инвалидов в Республике Беларусь» и Законом «Об информации, информатизации и защите информации» [1]. На практике проблемы доступности часто выявляются только на этапе тестирования интерфейса. Это усложняет исправление ошибок и увеличивает затраты на разработку. В связи с этим актуальной задачей является внедрение автоматизированных инструментов проверки доступности непосредственно в процесс разработки. Целью работы является демонстрация практического подхода к автоматизации тестирования доступности веб-приложений.

Основная часть. Автоматизация тестирования доступности предполагает использование инструментов, позволяющих анализировать интерфейс на разных этапах разработки. В рамках исследования использовались следующие инструменты:

– *eslint-plugin-jsx-a*11*y* – плагин для линтера *ESLint*, выполняющий статический анализ *JSX*-разметки и выявляющий нарушения требований доступности [2];

– *jest-axe* – библиотека для автоматического анализа *DOM*-структуры после рендеринга компонентов;

– *GitHub Actions* – инструмент *CI*/*CD* для автоматического запуска проверок.

Совместное использование этих инструментов позволяет обнаруживать ошибки доступности как на этапе написания кода, так и при выполнении тестов.

Процесс автоматизированного тестирования доступности пользовательского интерфейса включает последовательность этапов, представленных ниже:

1 Статический анализ исходного кода компонентов с использованием *ESLint* и плагина *eslint-plugin-jsx-a11y*.

2 Выявление типичных нарушений доступности (отсутствие *alt* у изображений, отсутствие *label* у элементов формы, использование несемантических *HTML*-элементов).

3 Рендеринг компонентов в тестовой среде при помощи библиотеки *React Testing Library*.

4 Анализ сформированного *DOM*-дерева с использованием библиотеки *jest-axe*, основанной на движке *axe-core* [3].

5 Формирование отчёта о найденных нарушениях доступности.

6 Автоматический запуск проверок в *CI*/*CD*-пайплайне при каждом изменении кода.

Для демонстрации работы инструментов было создано тестовое *React*-приложение, содержащее несколько компонентов пользовательского интерфейса.

Пример компонента, содержащего нарушения доступности:

```
export default function BadCard() {
  return (
    <div onClick={() => alert(«Clicked!»)}>
      <img src=«/image.png» />
      <h2>Card title</h2>
    </div>
  );
}
```

В данном примере изображение не содержит альтернативного текста, а элемент *div* используется как интерактивный элемент. Такие решения ухудшают доступность интерфейса. При запуске проверки на ошибки выводится отчет об ошибках (рисунок 1).

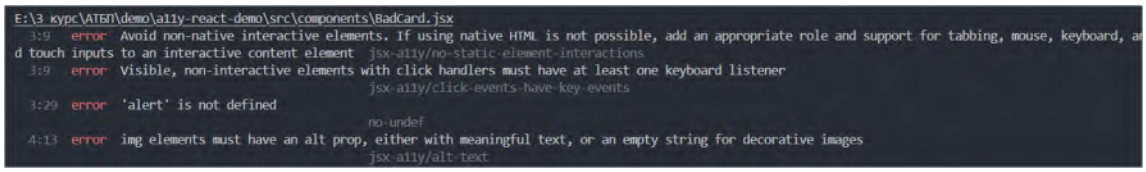

Рисунок 1 – Отчет об ошибках (*eslint-plugin-jsx-ally*)

Для автоматической проверки доступности используется библиотека *jest-axe*, которая анализирует *DOM* после рендеринга компонентов [3].

Пример тестового сценария:

```
import { render } from «@testing-library/react»;
import { axe, toHaveNoViolations } from «jest-axe»;
import App from «../App»;
expect.extend(toHaveNoViolations);
test(«App accessibility test», async () => {
  const { container } = render(<App />);
  const results = await axe(container);
  expect(results).toHaveNoViolations();
});
```

При обнаружении нарушений тест завершается со статусом *failed* и выводит отчёт об ошибках (рисунок 2).

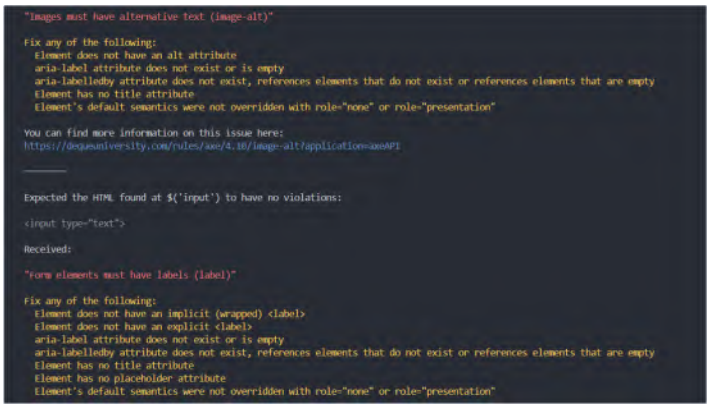

Рисунок 2 – Отчёт об ошибках (*jest-axe*)

Для автоматизации проверки был настроен *CI*/*CD*-пайплайн с использованием платформы *GitHub Actions* [4]. При каждом изменении кода выполняются установка зависимостей, запуск линтера и выполнение тестов доступности.

В ходе экспериментального применения предложенного подхода было протестировано демонстрационное *React*-приложение, содержащее несколько интерфейсных компонентов (карточки, формы ввода и кнопки взаимодействия). На начальном этапе анализа статический анализатор выявил несколько типичных нарушений доступности, включая отсутствие альтернативного текста у изображений и использование несемантических *HTML*-элементов для обработки пользовательских действий. Дополнительный анализ *DOM*-структуры при помощи библиотеки *jest-axe* обнаружил нарушения, связанные с отсутствием подписей у элементов формы. После исправления выявленных проблем повторный запуск тестов показал отсутствие нарушений доступности и успешное прохождение *CI*/*CD*-проверки.

Заключение. В работе продемонстрирован практический подход к автоматизации тестирования доступности веб-приложений. Использование инструментов *eslint-plugin-jsx-a*11*y*, *jest-axe* и *GitHub Actions* позволяет выявлять нарушения требований доступности на ранних этапах разработки. Результаты эксперимента показали, что интеграция автоматических проверок в процесс разработки повышает качество пользовательских интерфейсов и снижает вероятность появления регрессионных ошибок. Интеграция автоматизированных проверок доступности в процесс разработки позволяет разработчикам получать обратную связь о проблемах интерфейса непосредственно во время разработки. Это снижает вероятность появления ошибок доступности на поздних этапах проекта и уменьшает затраты на их исправление. Кроме того, автоматические проверки в *CI*/*CD*-пайплайне предотвращают повторное появление ранее исправленных дефектов, что повышает стабильность интерфейса и упрощает сопровождение проекта.

Список литературы

1. Закон Республики Беларусь «О социальной защите инвалидов в Республике Беларусь». – Минск, 1991.

2. eslint-plugin-jsx-a11y Documentation [Электронный ресурс]. – Режим доступа: https://github.com/jsx-eslint/eslint-plugin-jsx-a11y. – Дата доступа: 05.03.2026.

3. axe-core Documentation [Электронный ресурс]. – Режим доступа: https://github.com/dequelabs/axe-core. – Дата доступа: 05.03.2026.

4. GitHub Actions Documentation [Электронный ресурс]. – Режим доступа: https://docs.github.com/actions. – Дата доступа: 05.03.2026.

UDC 004.415.533

AUTOMATION OF ACCESSIBILITY TESTING AS PART OF THE WEB APPLICATION DEVELOPMENT PROCESS

Kontsevaya V.D., Tolstenkov I.A.

Belarusian State University of Informatics and Radioelectronics, Minsk, Republic of Belarus

Scientific supervisor: Kovalenko I.V. – Master of Economics, Senior Lecturer at the Department of ICSD

Annotation. The article describes a practical approach to automating accessibility testing during web application development. The integration of eslint-plugin-jsx-a11y, jest-axe and GitHub Actions into the development workflow of React applications is considered. An algorithm for automated accessibility testing is presented, including static code analysis and DOM testing. The experiment with a demo application revealed typical accessibility issues and demonstrated the effectiveness of automated checks.

Keywords: accessibility testing, web accessibility, test automation, React, CI/CD.