International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Bridging the Digital Accessibility Gap: How Legacy-to-Modern Front-End Transformation Improves Usability, Compliance, and Inclusive Access in Enterprise Financial Platforms

Mohammed Sayerwala

Abstract: Enterprise financial applications have evolved from digital accessibility being an afterthought or a legal compliance issue to a purposeful design consideration. This evolution is spurred by compliance, litigation risk, and the growing recognition that inaccessible digital experiences create systemic barriers to useful involvement in financial systems for millions. This article examines the relationship between legacy front-end architectures and accessibility deficits. In particular, it explores the technical challenges that jQuery-based, server-rendered interfaces create for assistive technology (AT) users and people with adaptive needs. It additionally outlines new accessibility capabilities of the migration to component architectures based on React. These include intrinsic semantics, a convention for ARIA attributes, responsive reflowing, and conditional rendering. Additional topics discussed in the article beyond the interface are supported session handoffs and modularity for multi-user workflows. The authors also discuss regulations such as the Americans with Disabilities Act and the Web Content Accessibility Guidelines, including how shift-left testing and automation can be used for compliance testing. Towards the end of the paper, the authors discuss the implications of inclusive design for enterprise financial contexts.

Keywords: *Digital Accessibility, WCAG, Ada Compliance, React, Legacy Modernization, Inclusive Design, Assistive Technology, Enterprise Financial Platforms*

1. Introduction

As financial services have migrated online, the question of whether financial services are made accessible to all users has increasingly been asked. The financial services industry has historically had difficulty answering this question. For users with visual, auditory, motor, cognitive or other disabilities, however, this question is one of access, and one that cannot be decided on a convenience basis. The term is used to describe those who can manage, access, and use financial services, without involving another person in all steps of transactions. [1]

The technical roots of digital inaccessibility in enterprise financial applications can largely be traced to architectural choices made in the rush to digitize in the 2000s and early 2010s. Many JavaScript frameworks from the jQuery era execute DOM manipulation and server-side page reloads, and involve custom components that are poorly adapted to the semantic HTML elements that assistive technologies work with. Nonetheless, those patterns were rarely intentional or outright malicious, simply because accessibility was not considered an explicit constraint in that era of enterprise development. The unfortunate result is that many organizations now have front-end systems that are structurally resistant to the accessibility enhancements that more recent legislation requires. [2]

We analyze how legacy front-end architectures create accessibility barriers, and to what extent the use of React-based UI component frameworks can improve accessibility when migrating from a legacy architecture. We do not limit ourselves to interface design but also consider operations and regulation of accessibility in enterprise finance with its organizational learning dimensions for maintaining compliance [3][4].

2. Legacy Frontend Barriers to Inclusive Digital Access

2.1 Technical Constraints of Traditional User Interface Architectures

A common problem with legacy jQuery-based enterprise front-ends and server-side template engines is that there is not a consistent, predictable, semantic DOM structure that assistive technologies

Mphasis Corporation, USA

can understand. Screen readers are the most common assistive technology used by the blind and visually impaired. However, screen readers rely on the semantic meaning of HTML elements and ARIA attributes to convert the visual screen to an audible one. jQuery code that places HTML strings into the DOM, replaces a semantic element on a page with an HTML div styled using CSS, or alters a semantic element's attribute due to interaction from an user is often nonsensical to users of screen readers. [2][5]

Server-rendered scenarios such as multi-page apps that use full page load navigation may also suffer from this issue, as screen readers will always reset to the beginning of the document, forcing the screen reader user to navigate from the beginning through all previously seen content before arriving back at the content they originally interacted with. This navigation burden determines if a task is a trivial inconvenience, as in the loading of the financial form during multi-step transactions, or if it is effectively impossible, with research finding that 50.4% of issues experienced by blind web users are not covered by WCAG 2.0 success criteria, and that architecture remediation goes beyond guideline adherence. [7]

2.2 Impact on Users with Disabilities and Regulatory Compliance Challenges

These limitations affect users differently depending on the disability, but both groups lack personalized exercise of economic agency. For example, blind users using JAWS or NVDA may encounter form fields that have no accessible label, drop-down menus that announce an incorrect item only, and modal dialogs that trap keyboard focus and can only be closed with a mouse. For example, keyboard users with mobility impairments may not be able to activate controls that trigger an action when focus is not visible, or there is no visual focus indicator. Participants with cognitive disabilities may not be able to complete multi-page workflows if there is no indicator of completion, they cannot save their partial completion, or there is no contextual help for any unclear label for form fields. [7][8]

The regulatory risk for these organizations may grow as courts expand the Americans with Disabilities Act to the digital space. The Web Content Accessibility Guidelines are maintained by the World Wide Web Consortium (W3C) and are the most commonly used technical standard for digital accessibility compliance testing. According to Vigo and Brown and Conway, automated assessment tools may cover at most half of the conformance criteria required by the WCAG, and actually only between 14% and 38%, with a correctness between 66% and 71%. Full WCAG compliance requires more effort beyond using an automated tool. [3]

3. React-Based Transformation for Enhanced Accessibility

3.1 Component-Driven Development for Inherently Accessible Interfaces

The main accessibility advantage of the React component architecture over jQuery is that an accessible React component can be built once, when the component is created, and all instances of the component in the application will inherit that accessibility. In a jQuery architecture, every custom dropdown, modal, and tab panel must have its own ARIA implementation. The consistency of ARIA implementations across the application is up to developer discipline and how much code review the project has. For each of the component types in the React component library, ARIA attributes, focus management, and keyboard interaction patterns are implemented once by the owners of each component type, tested once against screen reader software, and used each time that type is rendered. [9]

As accessibility specifications such as the WCAG progress through major versions, a React component library only needs to update a bounded number of base components, rather than auditing the entire codebase and refactoring each accessibility implementation when any single particular specification changes. The smaller the scope of an accessibility update for a library, the lower the risk of an audit gap forming. [4][10]

Table 1: Legacy vs. React Accessibility Feature Comparison

Accessibility Feature	Legacy jQuery/Server-Rendered	React Component Architecture
Semantic HTML structure	Often absent; divs replace semantic elements	Enforced at component level; consistent across application
ARIA attribute management	Manual; inconsistent across instances	Implemented once per component; inherited automatically
Focus management	Disrupted by full page reloads	Managed programmatically; preserved on async updates
Keyboard navigation	Often incomplete; mouse-dependent interactions common	Standardized per component type; tested against guidelines
Screen reader compatibility	Unpredictable; dynamic DOM changes not announced	Declarative state changes trigger appropriate ARIA live regions

Table 1. Accessibility feature availability in legacy jQuery/server-rendered systems versus React component architectures. (Author's own analysis)

Table 3: Assistive Technology Compatibility — Legacy vs. React Front-End

Assistive Technology	Legacy System Compatibility	React Component Architecture
JAWS screen reader	Poor; dynamic content changes not announced	Compatible; ARIA live regions surface state changes correctly
NVDA screen reader	Inconsistent; form labels frequently absent	Compatible; standardized label association across all inputs
Dragon NaturallySpeaking	Limited; unlabeled elements cannot be voice-targeted	Compatible; unique accessible names enable voice commands
Switch control	Incomplete; focus order disrupted by page reloads	Supported; logical focus sequence maintained in component tree
High contrast mode	Partial; custom styles may override OS contrast settings	Supported; design tokens respect system contrast preferences

Table 3. Assistive technology compatibility across legacy and modern front-end architectures. (Author's own analysis)

3

.2 Responsive Design and Adaptive User Experiences

Probably the most important accessibility specification, responsive design, is often improperly implemented in legacy enterprise systems. For people with low vision, at a 200% or higher browser zoom level, a layout that does not reflow (so that it can adjust to different widths without the user needing to scroll horizontally) can become inaccessible if it exceeds the screen viewport. These React component-based architectures utilize responsive design via CSS flexbox and grid to layout components in relation to the viewport such that magnified views do not cause horizontal scrolling. [11]

Beyond the window size, React frontends can adapt to other input modalities. For example, interactive elements require unique and meaningful accessible names, otherwise they may not work as voice commands for speech recognition software. For users relying on switch control or similar technologies, the focus order of the interactive elements should also be logical and predictable. These accessibility needs are likely to be easier to

reflect in the component structure of a React library than in a jQuery codebase in which each interactive component manages focus in its own way [12].

4. Operationalizing Inclusive Access in Enterprise Workflows

4.1 Bridging Digital Divides Through Assisted Session Handoffs

Not all users experiencing usability issues with a digital interface will have a disability. Older users, users with low digital skills and low-specification mobile devices with intermittent network connectivity may require user assistance to complete complex financial workflows online. The assisted session handoff pattern is one example of addressing this type of digital divide; the pattern permits a customer to passively allow a branch representative to take over their application session on an enterprise workstation without needing to disclose their disability or use a low-functionality alternative. [13]

As a result, assisted session handoffs require session persistence on the server, which enables the customer's authenticated session state to be transferred to a second device with no session loss or user re-authentication. In addition, as with all session handoffs, the session state needs to be protected from unauthorized access and tampering, while still ensuring that the handoff is reliably manageable by branch staff under time constraints. A well-designed implementation of this pattern avoids the restart-from-zero outcome that most commonly drives customer frustration in supervised completion scenarios. [14]

4.2 Streamlining Multi-User Workflows for Diverse User Needs

Multi-applicant financial workflows have unique accessibility challenges. In customary models of collecting applications for joint accounts, all applicants see the same flow and each applicant is forced to see each step of a flow even if only some of the steps are relevant to one applicant or another. This undifferentiated interface may be overwhelming for people with cognitive disabilities, and may impose an unreasonable burden of navigation for people who need assistive technologies. [15]

Because React has a component-based structure, the workflow can be divided by the role of the applicant. Each applicant sees only their own pathway, their own consent items, and their own role in an user interface that can be tailored for their modality. For example, a secondary applicant who is a screen reader user sees the same accessible interface for their role that they would see anywhere else in the system, and is not disadvantaged by the complexity of a different participant's workflow. This reduces the cognitive load on all participants and avoids a scenario where the accessibility needs of one participant conflict with another participant's interface needs. [6]

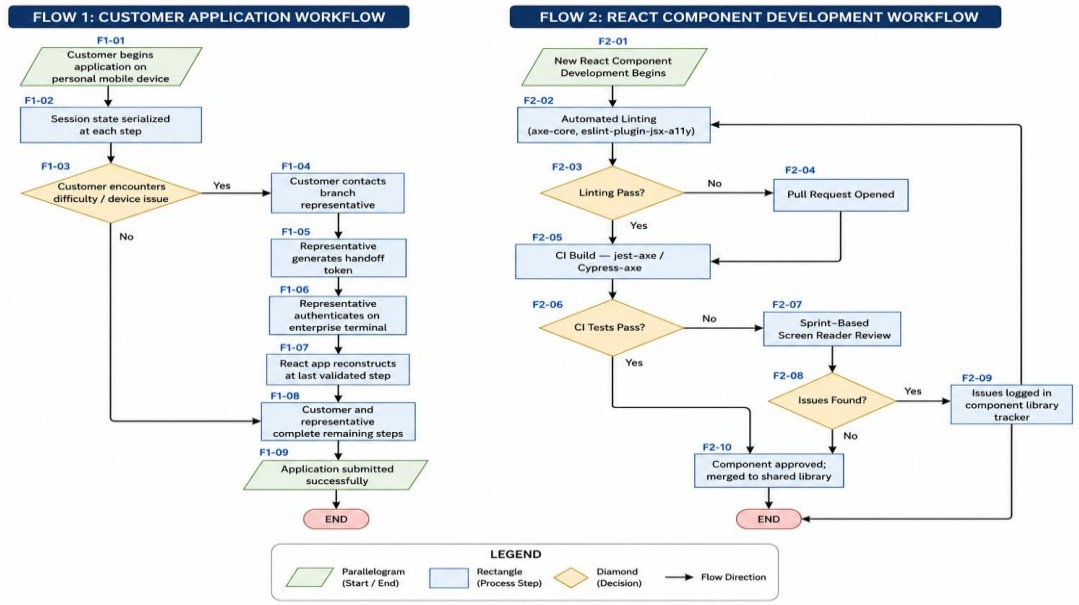

Figure 1. Customer and Development Workflow

Ensuring Regulatory Compliance and Ethical Design

5.1 Integrating ADA and WCAG Standards into Development Lifecycles

However, there's a tradeoff between accessibility compliance and creating technical debt to be reviewed later, such as through regular audits by third-party experts, as well as incurring technical costs that would have otherwise been avoided if accessibility-based remediation were performed earlier in the development lifecycle. That's why accessibility testing is done as early as possible. This is consistent with the goal of moving DevOps quality testing left, where testing occurs earlier in the development process (closer to the point of creation than the point of delivery). [8][9]

In a React development environment, it is operationally easier to adopt shift-left accessibility testing than in the context of maintaining a legacy codebase, as automated linting tools can detect ARIA usage pattern violations, missing accessible labels, and low color contrast ratios as soon as a component is created and before it is pushed to a shared branch. Automated accessibility test suites configured in the CI will at least test for 1) focus and focus management and 2) screen-reader behavior on every build. This is not a substitute for the periodic expert accessibility audit or user testing with disabled people but just another step towards fixing known-addressable violations before audits. [5]

Table 2: ADA/WCAG Compliance Requirements Mapped to React Implementation Patterns

Standard/Requirement	WCAG 2.1 Success Criterion	React Implementation Pattern
Accessible name for inputs	1.3.1 Info and Relationships	Component enforces label association via htmlFor/aria-labelledby
Keyboard operability	2.1.1 Keyboard	Focus management and keyboard handlers built into interactive components
Focus visibility	2.4.7 Focus Visible	Focus indicator styles standardized in design system CSS
Error identification	3.3.1 Error Identification	Validation error components include role=alert and field association
Status messages	4.1.3 Status Messages	Toast and notification components use aria-live regions
Reflow at 400% zoom	1.4.10 Reflow	Responsive CSS flexbox/grid ensures no horizontal scrolling at 320px width

Table 2. WCAG 2.1 success criteria and corresponding React component implementation patterns. (Author's own analysis)

5.2 Proactive Accessibility Testing and User-Centric Validation

Automated tools can catch a subset of accessibility problems, particularly those that can be detected with static analysis or through programmatic examination of the page structure. They cannot find all accessibility barriers. Research results suggest that a conformant automated tool can only find slightly more than half of the relevant WCAG success criteria [3]. Further, the issues blind people experience in practice go beyond the guidelines' success criteria [7]; some kinds of barriers, when a particular user interface meets a specific assistive technology, cannot be automatized. The interface may have cognitive accessibility barriers that cannot be detected using tests against the accessibility evaluation methods and tools based on the WCAG success criteria, and must instead be assessed with representative users testing the relevant assistive technologies and modalities.

User testing programs for accessibility can involve recruiting people with the relevant disabilities to use the product being tested and documenting the barriers to accessibility that they face while

performing realistic tasks. The results of such tests may inform design choices in ways that automated solutions cannot. Then, by feeding the results back into a component library's issue tracker and release cycle, accessibility issues can be fixed in a component and patched throughout the rest of the application. [3][4]

Table 4: Accessibility Testing Types, Tools, and CI/CD Integration Points

Testing Type	Tool/Method	Integration Point	What It Catches
Automated linting	axe-core, eslint-plugin-jsx-a11y	Pre-commit hook / CI build	Missing labels, invalid ARIA, colour contrast failures
Automated integration	jest-axe, Cypress-axe	CI test suite on every PR	Page-level WCAG violations; focus management regressions
Screen reader testing	NVDA + Firefox, JAWS + Chrome	Periodic sprint-based review	Announcement accuracy; navigation flow; live region behaviour
Expert audit	External specialist review	Quarterly or pre-major-release	Complex interaction patterns; cognitive accessibility; WCAG AA/AAA gaps
User testing	Participants with disabilities	Major feature releases	Real-world barriers not detected by automated or expert tools

Table 4. Accessibility testing types and their integration into the React development lifecycle. (Author's own analysis)

6. Societal Impact and Future Directions

6.1 Expanding Equitable Access to Essential Financial Services

The potential user population for such accessible digital financial services goes well beyond those with disabilities. For example, populations are aging in many of the world's most advanced economies. Older users may encounter age-related impairments to vision, hearing and fine motor skills which present barriers to independent use of online channels. Another consideration is that in practice many low-income users are likely using financial services on small screen mobile devices in low-bandwidth environments, where the best design is one that is accessible and responsive to the conditions of use. The case for accessible finance therefore extends beyond disability access to the full range of users whose access is compromised by design decisions that favor narrow, able-bodied, device-privileged usage contexts. [13][14]

While additional data are required to quantify the relationship between front-end modernization and improving financial inclusion, some companies have reported that migrating legacy financial services platforms to the more accessible component-based React architecture was correlated with higher completion rates of digital self-service solutions among client segments that had customarily relied on in-branch financial services. This is a demanding exercise in causal attribution, but the data across settings are uniformly in the same direction, suggesting that accessible design may be a powerful way of extending the reach of digital finance. [15]

6.2 Cultivating Trust and Engagement Through Inclusive Digital Platforms

Beyond compliance, the reputational dimensions of digital accessibility should not be underestimated. When customers encounter an inaccessible digital experience, not only has it made their life more difficult, it says to them that the organization does not value the effort to make it usable by everyone. Although difficult to quantify, the perception contributes to measures of brands' trust, which is increasingly considered by financial institutions to affect long-term customer retention. [6][7]

By contrast, websites that invest in inclusive design signal to all their users that they believe usability is not a special accommodation, but a fundamental value offered to everyone. Many accessibility improvements for people with disabilities, such as clearer labels of form elements, improved keyboard navigation and better error messages, also improve usability for people without disabilities. And even when the law does not compel a site, the business case for accessible design extends beyond the size of the population with disabilities. All users benefit from a clearer, more predictable and more forgiving interface when the design is more accessible. [8]

Conclusion

However, legacy front-end architectures may create digital accessibility barriers structurally rather than incidentally: not using semantic HTML, unpredictable DOM content, and breaking the user experience by performing full page reloads. Barriers created this way may not be easily remediated. Instead they require updates to the architecture of applications. Migrating to component-based architectures for React, for instance, in order to build accessible applications may be possible as accessibility attributes only need to be provided once on a component and can be used throughout the application.

The operational and compliance aspects of accessibility, such as assisted session handoffs, modular multi-user workflow design, shift-left compliance testing and user testing with people with disabilities, show that technical measures alone do not lead to accessible digital services. Instead, the context is provided for inclusive design to happen consistently, to be transparently audited, and to be iteratively improved over time. These platforms must be made available not merely to comply with regulations, but to ensure that the digitization of services does not reproduce in the digital domain the exclusionary characteristics of physical services.

References

[1] W3C Web Accessibility Initiative, "Web Content Accessibility Guidelines (WCAG) 2.1," W3C Recommendation, 2018. [Online]. Available: https://www.w3.org/TR/WCAG21/

[2] G. Brajnik, Y. Yesilada, and S. Harper, "The expertise effect on web accessibility evaluation methods," Hum.-Comput. Interact., vol. 26, no. 3, pp. 246-283, 2011. DOI: 10.1080/07370024.2011.601670. [Online]. Available: https://www.tandfonline.com/doi/full/10.1080/07370024.2011.601670

[3] M. Vigo, J. Brown, and V. Conway, "Benchmarking web accessibility evaluation tools: Measuring the harm of sole reliance on automated tests," in Proc. 10th Int. Cross-Disciplinary Conf. Web Accessibility (W4A), 2013, art. 1. DOI: 10.1145/2461121.2461124. [Online]. Available: https://dl.acm.org/doi/10.1145/2461121.2461124

[4] Y. Yesilada, G. Brajnik, M. Vigo, and S. Harper, "Understanding web accessibility and its drivers," in Proc. 9th Int. Cross-Disciplinary Conf. Web Accessibility (W4A), 2012, art. 1. DOI: 10.1145/2207016.2207027. [Online]. Available: https://dl.acm.org/doi/10.1145/2207016.2207027

[5] M. Cooper, D. Sloan, B. Kelly, and S. Lewthwaite, "A challenge to web accessibility metrics and guidelines: Putting people and processes first," in Proc. 9th Int. Cross-Disciplinary Conf. Web Accessibility (W4A), 2012, art. 20. DOI: 10.1145/2207016.2207028. [Online]. Available: https://researchportal.bath.ac.uk/en/publications/a-challenge-to-web-accessibility-metrics-and-guidelines-putting-p/

[6] C. Stephanidis, G. Salvendy et al., "Seven HCI grand challenges," Int. J. Hum.-Comput. Interact., vol. 35, no. 14, pp. 1229-1269, 2019. DOI: 10.1080/10447318.2019.1619259. [Online]. Available: https://www.tandfonline.com/doi/full/10.1080/10447318.2019.1619259

[7] C. Power, A. Freire, H. Petrie, and D. Swallow, "Guidelines are only half of the story: Accessibility

problems encountered by blind users on the web," in Proc. SIGCHI Conf. Human Factors in Computing Systems (CHI), 2012, pp. 433-442. DOI: 10.1145/2207676.2207736. [Online]. Available: https://dl.acm.org/doi/10.1145/2207676.2207736

[8] J. Abascal, M. Arrue, and X. Valencia, "Tools for web accessibility evaluation," in Web Accessibility, 2nd ed., Y. Yesilada and S. Harper, Eds. Springer, 2019, pp. 479-503. DOI: 10.1007/978-1-4471-7440-0_26. [Online]. Available: https://link.springer.com/chapter/10.1007/978-1-4471-7440-0_26

[9] A. Vollenwyder, G. H. Iten, F. Bruhlmann, K. Opwis, and E. D. Mekler, "Salient beliefs influencing the intention to consider web accessibility," Comput. Hum. Behav., vol. 92, pp. 352-360, 2019. DOI: 10.1016/j.chb.2018.11.016. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S074756321830551X

[10] C. Pahl and P. Jamshidi, "Microservices: A systematic mapping study," in Proc. 6th Int. Conf. Cloud Computing and Services Science (CLOSER), 2016, pp. 137-146. [Online]. Available: https://dl.acm.org/doi/10.5220/0005785501370146

[11] A. Balalaie, A. Heydarnoori, and P. Jamshidi, "Microservices architecture enables DevOps: Migration to a cloud-native architecture," IEEE Software, vol. 33, no. 3, pp. 42-52, 2016. DOI: 10.1109/MS.2016.64. [Online]. Available: https://ieeexplore.ieee.org/document/7436659

[12] D. Taibi, V. Lenarduzzi, and C. Pahl, "Architectural patterns for microservices: A systematic mapping study," in Proc. 8th Int. Conf. CLOSER, 2018, pp. 221-232. [Online]. Available: https://www.scitepress.org/Papers/2018/67983/

[13] M. Waseem, P. Liang, M. Shahin, A. Ahmad, and A. R. Nasab, "On the nature of issues in five open source microservices systems," in Proc. 25th Int. Conf. EASE, 2021, pp. 201-210. DOI: 10.1145/3463274.3463337. [Online]. Available: https://dl.acm.org/doi/10.1145/3463274.3463337

[14] N. A. Ernst, S. Bellomo, I. Ozkaya, R. L. Nord, and I. Gorton, "Measure it? Manage it? Ignore it? Software practitioners and technical debt," in Proc. 10th Joint Meeting ESEC/FSE, 2015, pp. 50-60. DOI: 10.1145/2786805.2786848. [Online]. Available: https://dl.acm.org/doi/10.1145/2786805.2786848

[15] T. Cerny, M. J. Donahoo, and M. Trnka, "Contextual understanding of microservice architecture: Current and future directions," ACM SIGAPP Appl. Comput. Rev., vol. 17, no. 4, pp. 29-45, 2018. DOI: 10.1145/3183628.3183631. [Online]. Available: https://dl.acm.org/doi/10.1145/3183628.3183631