WCAG Accessibility Tester - Product Requirements Document

Introduction/Overview

The WCAG Accessibility Tester is a command-line utility designed for software developers, designers, and QA engineers to evaluate web pages and websites against WCAG 2.2 AA accessibility standards. The tool provides immediate, actionable feedback through detailed JSON reports and human-readable summaries, enabling teams to identify and fix accessibility issues early in the development workflow.

Problem Statement: Manual accessibility audits are time-consuming, expensive, and often occur too late in the development cycle. Teams need an automated, developer-friendly tool that provides immediate feedback on accessibility compliance while integrating seamlessly into existing workflows.

Solution: A local CLI tool that scans individual pages or crawls entire sites, generating comprehensive accessibility reports with clear remediation guidance and an overall accessibility score.

Goals

Business Goals

- Reduce time and cost associated with manual accessibility audits by 50% within six months
- Improve measurable site accessibility scores and compliance rates by at least 25% within the first quarter post-launch
- Position the product as an essential utility for development and QA teams at technology-driven organizations
- Decrease the number of critical accessibility issues found in production by 60%

User Goals

- Provide immediate, actionable feedback on accessibility issues adhering to WCAG 2.2 AA standards

- Enable both targeted single-page audits and comprehensive website crawls
- Generate machine-readable JSON results for integration and further analysis
- Offer a clear, aggregate accessibility score for benchmarking progress across audits
- Seamlessly fit into existing development workflows without requiring significant setup

Technical Goals

- Start with small to medium site support, with architecture designed for future scalability
- Focus on standalone functionality first, with CI/CD integration as a secondary priority
- Provide multiple installation options to accommodate different team preferences

User Stories

Developer

- As a developer, I want to test the web page I'm building for accessibility so that I can fix issues before code review
- As a developer, I want to run an audit with a single CLI command so that it fits seamlessly into my workflow
- As a developer, I want to export results in JSON format so I can integrate with my existing tools and CI pipelines
- As a developer, I want the tool to continue scanning even when it encounters errors so I get a complete picture of issues

Designer

- As a designer, I want to understand which elements are causing accessibility failures so I can iterate on design decisions for better user outcomes
- As a designer, I want to see an overall accessibility score after a crawl so I can quickly assess the state of my designs across multiple pages
- As a designer, I want clear, jargon-free explanations of issues so I can understand how to fix them

QA Engineer

- As a QA engineer, I want to crawl an entire staging site to identify accessibility issues before release so that we meet compliance requirements
- As a QA engineer, I want the tool to highlight both AA violations and AAA/ARIA warnings so I can prioritize remediation by severity
- As a QA engineer, I want detailed reports that I can share with development teams for remediation

Functional Requirements

Core Scanning Functionality

1. **Single Page Scan**: Execute accessibility tests on a single URL with full WCAG 2.2 AA coverage
2. **Site Crawling**: Crawl entire sites following in-domain links with configurable depth limits
3. **JavaScript Rendering**: Support modern web applications with JavaScript-enabled page rendering
4. **Error Resilience**: Continue scanning when encountering errors, reporting issues in final output
5. **Rule Engine**: Hybrid approach combining existing accessibility testing libraries (like axe-core) with custom WCAG 2.2 rules

Output and Reporting

6. **JSON Primary Output**: Generate structured, machine-readable JSON reports as the primary output format
7. **Human-Readable Summary**: Provide optional console summary with key metrics and issue counts
8. **Detailed Issue Reporting**: Include WCAG/ARIA reference, element information, severity level, and suggested remediation for each issue
9. **Accessibility Scoring**: Calculate and output weighted, normalized accessibility scores for individual pages and aggregate sites
10. **Error Logging**: Report scanning errors, timeouts, and technical issues within the output structure

Installation and Distribution

11. **Package Manager Support**: Distribute via standard package managers (pip for Python, npm for Node.js)
12. **Standalone Binary**: Provide self-contained binary downloads for major operating systems
13. **Docker Container**: Offer containerized version for consistent environments and CI/CD integration
14. **Cross-Platform Compatibility**: Support Windows, macOS, and Linux environments

Configuration System

15. **Progressive Configuration**: Start with minimal configuration options and sensible defaults
16. **Expandable Settings**: Design architecture to support increasing configuration complexity over time
17. **WCAG Level Selection**: Default to WCAG 2.2 AA with warnings for AAA and ARIA violations
18. **Crawl Configuration**: Configurable depth limits, domain restrictions, and page limits

Performance and Scalability

19. **Small Site Focus**: Initially target small to medium sites (up to 100 pages)
20. **Scalability Architecture**: Design with future scalability in mind for larger site support
21. **Timeout Handling**: Implement reasonable timeouts for page loading and scanning operations
22. **Resource Management**: Efficient memory and CPU usage during crawling operations

Non-Goals (Out of Scope)

Initial Release Exclusions

- **Graphical User Interface (GUI)**: CLI-only interface for initial release
- **Cloud Services**: No web-based hosting, cloud storage, or remote processing
- **User Account Management**: No authentication, user profiles, or persistent user data
- **Browser Extensions**: No browser plugin or extension functionality
- **Real-time Monitoring**: No continuous monitoring or alerting capabilities

Authentication and Advanced Features

- **Authenticated Content**: No support for login-required pages or gated content in initial release

- **Complex Site Interactions**: No form filling, multi-step user flows, or complex user interactions
- **Visual Regression Testing**: Focus on accessibility, not visual or functional testing
- **Performance Testing**: Accessibility-focused, not general performance metrics

Integration Complexity

- **Deep CI/CD Integration**: Advanced CI/CD features postponed in favor of standalone functionality
- **IDE Plugins**: No editor or IDE integration in initial scope
- **Custom Reporting Formats**: Additional output formats beyond JSON and console summary

Design Considerations

Command-Line Interface Design

- **Accessibility-First CLI**: Design CLI output to be screen reader compatible with plain text and high-contrast elements
- **Progressive Disclosure**: Simple commands for basic use, with advanced flags available for power users
- **Clear Error Messages**: Provide actionable error messages with suggested solutions
- **Help System**: Comprehensive `--help` documentation with examples and common use cases

Output Format Design

- **JSON Schema**: Well-defined, consistent JSON structure for programmatic consumption
- **Human-Readable Elements**: Clear issue descriptions using plain language, avoiding technical jargon
- **Severity Indicators**: Visual indicators for different severity levels in console output
- **Progress Indicators**: Real-time progress feedback during scanning operations

Technical Considerations

Architecture and Technology Stack

- **Hybrid Rule Engine**: Combine proven accessibility testing libraries (axe-core) with custom WCAG 2.2 rule implementations
- **Platform Agnostic**: Choose technology stack that supports multiple installation methods (likely Node.js for broad compatibility)
- **Headless Browser Integration**: Use Playwright or Puppeteer for JavaScript rendering and modern web app support
- **Modular Design**: Separate scanning engine, rules engine, and output formatting for maintainability

Performance Requirements

- **Scanning Speed**: Target processing time of 10 seconds or less per page including JavaScript rendering
- **Memory Efficiency**: Optimize for standard development laptop specifications
- **Concurrent Processing**: Support parallel page processing during site crawls
- **Graceful Degradation**: Handle slow networks and heavy JavaScript sites with appropriate timeouts

Data Handling and Privacy

- **Local Processing**: All data processing occurs locally with no external service dependencies
- **No Data Retention**: Tool does not store or cache user data between runs
- **Result File Management**: Clear documentation on handling and privacy of generated report files
- **Configurable Output**: User control over output file locations and formats

Integration Considerations

- **JSON Schema Compatibility**: Design output format to align with common CI/CD tools and accessibility testing workflows
- **Exit Codes**: Implement standard exit codes for success, warnings, and failures
- **Logging Standards**: Use standard logging levels and formats for operational visibility
- **Future Extension Points**: Design APIs and interfaces to support future CI/CD integration plugins

Success Metrics

User Adoption Metrics

- Tool installation and active usage rates across target user segments
- Frequency of tool usage per developer/QA engineer per week
- User retention and recurring usage patterns
- Net Promoter Score (NPS) from user surveys

Business Impact Metrics

- 50% reduction in manual accessibility audit time and costs within 6 months
- 25% improvement in pre-release accessibility compliance rates within first quarter
- 60% decrease in critical accessibility issues found in production
- Reduction in external accessibility audit costs

Technical Performance Metrics

- Tool reliability with less than 1% failed scans on supported websites
- Average processing time per page under 10 seconds
- Compatibility rate of 95%+ with public websites (non-authenticated)
- User-reported accuracy of accessibility issue detection

Usage Pattern Metrics

- Number of pages scanned per session
- Distribution of single-page vs. site crawl usage
- Most common configuration options and flags used
- Integration adoption rate with CI/CD pipelines (future metric)

Open Questions

Technical Implementation Questions

1. **Technology Stack Selection**: Which platform (Node.js vs. Python vs. Go) provides the best balance of performance, ecosystem support, and installation flexibility?

2. **Rule Engine Integration**: What's the optimal balance between leveraging existing libraries (axe-core) and implementing custom WCAG 2.2 rules?
3. **Docker Image Size**: How can we minimize Docker image size while maintaining full functionality?

Product Strategy Questions

4. **Configuration Evolution**: What specific configuration options should be prioritized in the second iteration?
5. **Scoring Algorithm**: How should the accessibility score be weighted to provide meaningful, actionable insights?
6. **CI/CD Integration Timeline**: When should advanced CI/CD integration features be prioritized based on user feedback?

User Experience Questions

7. **Output Verbosity**: What level of detail strikes the right balance between comprehensive reporting and usability?
8. **Error Handling**: How should the tool handle edge cases like infinite redirects, very slow sites, or malformed HTML?
9. **Documentation Requirements**: What additional documentation or examples would most benefit junior developers using the tool?

Business and Compliance Questions

10. **WCAG Updates**: How should the tool handle future WCAG standard updates and rule changes?
11. **Accuracy Validation**: What process should be established for validating the accuracy of accessibility rule implementations?
12. **Community Feedback**: How can user feedback be systematically collected and incorporated into future iterations?

Recent Improvements (2025)

- **Axe-core rules array bug fixed:** Robust validation ensures axe-core is always configured with a valid rules object.

- **Configurable page ready wait timeout:** Users can set the timeout for waiting for the page to be ready via CLI/config.
- **Improved scoring:** Score is reduced if there are warnings, not just errors, for more accurate results.
- **Enhanced reporting:** Console and report output now clearly distinguish between issues and warnings, with a summary and grouped warnings.
- **Remediation tips:** For common warnings (e.g., tab order, missing roles/names), remediation guidance is provided in the output.
- **Better output formatting:** Color coding and clear separation for errors, warnings, and passes. A summary section is included at the end of each scan.
- **Documentation:** All new options and behaviors are documented in the relevant sections (output, configuration, CLI).