Tasks for WCAG Accessibility Tester Implementation

Relevant Files

- `package.json` - Node.js project configuration with dependencies and scripts
- `tsconfig.json` - TypeScript configuration for the project
- `src/index.ts` - Main CLI entry point that handles command parsing and execution
- `src/scanner/page-scanner.ts` - Core single page accessibility scanning functionality
- `src/scanner/page-scanner.test.ts` - Unit tests for page scanner
- `src/scanner/site-crawler.ts` - Website crawling system with depth limits and domain restrictions
- `src/scanner/site-crawler.test.ts` - Unit tests for site crawler
- `src/scanner/rule-engine.ts` - Hybrid WCAG rule engine combining axe-core with custom rules
- `src/scanner/rule-engine.test.ts` - Unit tests for rule engine
- `src/output/json-reporter.ts` - JSON report generation and formatting
- `src/output/json-reporter.test.ts` - Unit tests for JSON reporter
- `src/output/console-reporter.ts` - Human-readable console output formatting
- `src/output/console-reporter.test.ts` - Unit tests for console reporter
- `src/output/scoring.ts` - Accessibility scoring algorithm implementation
- `src/output/scoring.test.ts` - Unit tests for scoring system
- `src/cli/command-parser.ts` - CLI argument parsing and validation
- `src/cli/command-parser.test.ts` - Unit tests for command parser
- `src/cli/config-manager.ts` - Configuration file handling and defaults
- `src/cli/config-manager.test.ts` - Unit tests for configuration manager
- `src/utils/browser-manager.ts` - Headless browser lifecycle management (Playwright/Puppeteer)
- `src/utils/browser-manager.test.ts` - Unit tests for browser manager
- `src/utils/url-validator.ts` - URL validation and normalization utilities
- `src/utils/url-validator.test.ts` - Unit tests for URL validator

- `src/types/index.ts` - TypeScript type definitions for the project
- `Dockerfile` - Docker container configuration
- `scripts/build-binaries.js` - Binary compilation script for multiple platforms
- `README.md` - Project documentation and usage instructions
- `jest.config.js` - Jest testing framework configuration
- `.eslintrc.js` - ESLint configuration for code quality and style enforcement
- `.prettierrc.js` - Prettier configuration for consistent code formatting
- `.prettierignore` - Files to exclude from Prettier formatting
- `.gitignore` - Files and directories to exclude from version control
- `scripts/setup.sh` - Development environment setup script
- `src/index.ts` - Main CLI entry point that handles command parsing and execution
- `src/types/index.ts` - Main TypeScript type definitions and exports for the project (enhanced with compliance summary)
- `src/types/wcag.ts` - WCAG 2.2 specific type definitions for rules and guidelines (updated to include ARIA level)
- `src/types/browser.ts` - Browser and Playwright-specific type definitions
- `src/types/output.ts` - Output and reporting-specific type definitions
- `src/scanner/rule-engine.ts` - Hybrid rule engine combining axe-core with custom WCAG 2.2 rules
- `src/scanner/rule-engine.test.ts` - Unit tests for rule engine with comprehensive test coverage
- `src/scanner/page-scanner.ts` - Core single page accessibility scanner with JavaScript rendering support
- `src/scanner/page-scanner.test.ts` - Unit tests for page scanner with comprehensive test coverage
- `src/scanner/error-resilience.ts` - Comprehensive error resilience system with circuit breakers, retry logic, and adaptive timeouts
- `src/scanner/error-resilience.test.ts` - Unit tests for error resilience system with 35 test cases
- `src/scanner/wcag-level-handler.ts` - WCAG level selection and categorization handler with AA default, AAA/ARIA warnings
- `src/scanner/wcag-level-handler.test.ts` - Unit tests for WCAG level handler with 31 test cases

- `src/scanner/page-scanner.test.ts` - Enhanced with comprehensive WCAG integration tests and resilience testing (35+ new test cases)
- `src/types/crawler.ts` - Comprehensive type definitions for site crawling with session management, URL discovery, and result aggregation
- `src/scanner/site-crawler.ts` - Multi-page site crawler with depth limits, domain restrictions, rate limiting, and concurrent scanning
- `src/scanner/site-crawler.test.ts` - Comprehensive unit tests for site crawler functionality (30 test cases, 23 passing)
- `src/types/output.ts` - Comprehensive JSON report schema with executive summaries, issue analysis, compliance reporting, and technical details
- `src/output/json-reporter.ts` - JSON report generator with single page and site report capabilities (implementation with TypeScript refinements needed)
- `src/output/json-reporter.test.ts` - Comprehensive unit tests for JSON report generation (120+ test cases covering all report features)
- `src/data/wcag-database.ts` - Comprehensive WCAG 2.2 success criteria database with detailed remediation guidance, testing instructions, and code examples
- `src/output/issue-processor.ts` - Advanced issue processor with contextual remediation, priority scoring, impact assessment, and complexity analysis
- `src/output/issue-processor.test.ts` - Extensive unit tests for issue processing functionality (45+ test cases covering all enhancement features)
- `src/scoring/accessibility-scorer.ts` - Sophisticated scoring algorithm with weighted calculations, configurable profiles, bonus/penalty systems, and site aggregation
- `src/scoring/accessibility-scorer.test.ts` - Comprehensive scoring algorithm tests (42 test cases, 32 passing, validating all scoring features)
- `src/output/console-reporter.ts` - Comprehensive console output system with progress bars, spinners, color-coded results, and detailed formatting
- `src/output/console-reporter.test.ts` - Console reporter test suite (53 test cases covering configuration, progress tracking, and output formatting)
- `src/output/error-logger.ts` - Complete error logging and technical issue reporting system with structured logging, recovery tracking, and diagnostics
- `src/output/error-logger.test.ts` - Comprehensive error logger test suite (30+ test categories covering all logging levels, error categories, and reporting features)
- `src/output/json-reporter.test.ts` - Complete JSON reporter test suite (25+ test cases validating report generation, metadata, statistics, and error handling)
- `src/output/issue-processor.test.ts` - Issue processor test suite (20+ test cases covering WCAG enhancement, prioritization, and contextual remediation)

- `src/index.ts` - Comprehensive CLI interface with Commander.js integration, supporting single page scanning with full accessibility analysis pipeline and integrated help system
- `src/cli/config-manager.ts` - Progressive configuration management system with file-based config, environment variables, CLI overrides, and comprehensive validation
- `src/cli/config-manager.test.ts` - Configuration manager test suite (50+ test cases covering all configuration sources, merging, validation, and error handling)
- `src/cli/help-manager.ts` - Comprehensive help system with 10 topic categories, interactive navigation, search functionality, quick tips, and configuration generation
- `src/cli/help-manager.test.ts` - Help manager test suite (40+ test cases covering all help topics, search, navigation, and content quality)
- `src/cli/url-validator.ts` - Comprehensive URL validation and error handling system with auto-correction suggestions, security checks, accessibility validation, and user-friendly error messages
- `src/cli/url-validator.test.ts` - URL validator test suite (70+ test cases covering all validation scenarios, preset configurations, error handling, and formatting)
- `src/integrations/storybook-adapter.ts` - Storybook integration adapter for component isolation testing, story navigation, and iframe handling
- `src/integrations/storybook-adapter.test.ts` - Storybook adapter test suite covering component discovery, navigation, and isolation testing
- `src/standards/wcag22-rules.ts` - Enhanced WCAG 2.2 success criteria implementation with custom rules for new 2.2 criteria
- `src/standards/wcag22-rules.test.ts` - WCAG 2.2 rules test suite covering all new success criteria and edge cases
- `src/standards/section508-compliance.ts` - Section 508 compliance module with federal-specific requirements and dual reporting
- `src/standards/section508-compliance.test.ts` - Section 508 compliance test suite covering federal requirements and reporting
- `src/design-system/token-validator.ts` - Design token accessibility validation for colors, typography, spacing, and other design system elements
- `src/design-system/token-validator.test.ts` - Design token validator test suite covering all design system accessibility requirements
- `src/design-system/pattern-tester.ts` - Pattern-based accessibility testing for forms, navigation, and component compositions
- `src/design-system/pattern-tester.test.ts` - Pattern tester test suite covering common design system patterns and validation

- `src/manual-testing/guided-workflows.ts` - Guided manual testing workflows with step-by-step instructions and WCAG criteria mapping
- `src/manual-testing/guided-workflows.test.ts` - Guided workflows test suite covering manual testing procedures and validation
- `src/manual-testing/keyboard-automation.ts` - Keyboard navigation testing automation with focus management and accessibility validation
- `src/manual-testing/keyboard-automation.test.ts` - Keyboard automation test suite covering navigation patterns and focus management
- `src/utils/browser-manager.ts` - Headless browser lifecycle management with Playwright integration
- `src/utils/browser-manager.test.ts` - Unit tests for browser manager with comprehensive test coverage
- `src/test-setup.ts` - Jest test setup with custom matchers and global test utilities
- `src/__tests__/setup.test.ts` - Jest setup verification tests

Notes

- Unit tests should be placed alongside the code files they are testing in the same directory
- Use `npm test` or `npx jest [optional/path/to/test/file]` to run tests
- The project uses TypeScript for type safety and better developer experience
- Headless browser integration provides JavaScript rendering capabilities for modern web apps

Tasks

- [x] 1.0 Project Setup and Architecture
  - [x] 1.1 Initialize Node.js project with TypeScript configuration
  - [x] 1.2 Set up development dependencies (Jest, TypeScript, ESLint, Prettier)
  - [x] 1.3 Configure build scripts and development workflow
  - [x] 1.4 Create project directory structure following modular design principles
  - [x] 1.5 Set up TypeScript type definitions and interfaces
  - [x] 1.6 Configure Jest testing framework with TypeScript support
- [ ] 2.0 Core Scanning Engine Implementation
  - [x] 2.1 Implement browser manager for headless browser lifecycle (Playwright integration)

  - [x] 2.2 Create hybrid rule engine combining axe-core with custom WCAG 2.2 rules
  - [x] 2.3 Develop single page scanner with JavaScript rendering support
  - [x] 2.4 Implement error resilience and timeout handling for page scanning
  - [x] 2.5 Add WCAG level selection (AA default, AAA/ARIA warnings)
  - [x] 2.6 Create comprehensive unit tests for scanning functionality
- [ ] 3.0 Site Crawling System
  - [x] 3.1 Design comprehensive crawler architecture with depth limits and domain restrictions
  - [ ] 3.2 Implement URL discovery and sitemap parsing
  - [ ] 3.3 Add concurrent page scanning with rate limiting
  - [ ] 3.4 Create crawl session management and progress tracking
  - [ ] 3.5 Implement crawl result aggregation and deduplication
  - [ ] 3.6 Create unit tests for crawling functionality and edge cases
- [ ] 4.0 Output and Reporting System
  - [x] 4.1 Design and implement JSON report schema for accessibility results
  - [x] 4.2 Create detailed issue reporting with WCAG references and remediation suggestions
  - [x] 4.3 Implement accessibility scoring algorithm with weighted calculations
  - [x] 4.4 Develop human-readable console output with progress indicators
  - [x] 4.5 Add error logging and technical issue reporting within output structure
  - [x] 4.6 Create unit tests for all reporting and scoring functionality
- [ ] 5.0 CLI Interface and Configuration Management
  - [x] 5.1 Implement command-line argument parsing with Commander.js or similar
  - [x] 5.2 Create progressive configuration system with sensible defaults
  - [x] 5.3 Add comprehensive help system with examples and use cases
  - [x] 5.4 Implement URL validation and error handling for CLI inputs
  - [ ] 5.5 Create configuration file support for advanced settings
  - [ ] 5.6 Add screen reader compatible CLI output design
  - [ ] 5.7 Create unit tests for CLI parsing and configuration management
- [ ] 8.0 Enhanced WCAG 2.2 and Standards Compliance
  - [ ] 8.1 Implement comprehensive WCAG 2.2 success criteria coverage

  - [ ] 8.2 Add custom rules for new WCAG 2.2 criteria (Focus Not Obscured, Target Size, etc.)
  - [ ] 8.3 Create Section 508 compliance module with specific federal requirements
  - [ ] 8.4 Add dual WCAG/Section 508 reporting capabilities
  - [ ] 8.5 Implement enhanced authentication accessibility testing
  - [ ] 8.6 Add comprehensive keyboard navigation automation testing

- [ ] 9.0 Storybook and Component-Level Testing Integration
  - [x] 9.1 Create Storybook integration module for component isolation testing
  - [x] 9.2 Implement story navigation and automated component discovery
  - [x] 9.3 Add component state and variant accessibility testing
  - [x] 9.4 Create Storybook-specific URL handling and iframe navigation
  - [x] 9.5 Implement component composition and pattern validation
  - [x] 9.6 Add Storybook CLI integration and build process hooks
  - [x] 9.7 Batch VPAT/Section 508 reporting for Storybook (storybook-batch)

- [ ] 10.0 Design System Accessibility Testing
  - [x] 10.1 Create design token accessibility validation (colors, typography, spacing)
  - [x] 10.2 Implement pattern-based accessibility testing (forms, navigation, etc.)
  - [x] 10.3 Add component library specific accessibility checks
  - [x] 10.4 Create design system compliance reporting
  - [x] 10.5 Implement accessibility annotation generation for design systems
  - [x] 10.6 Add component documentation accessibility validation
  - [x] 10.7 Batch VPAT/Section 508 reporting for static HTML/component directories (html-batch)
  - [x] 10.8 Support for custom batch sources via JSON/YAML config (–input-config)
  - [ ] 10.9 Advanced output customization via templates (–template)
  - [ ] 10.10 Planned Jira integration for compliance tracking
  - [ ] 10.11 Batch error handling/reporting improvements

- [ ] 11.0 Advanced Manual Testing Integration
  - [ ] 11.1 Create guided manual testing workflows with step-by-step instructions
  - [ ] 11.2 Implement keyboard navigation testing automation
  - [ ] 11.3 Add screen reader simulation and testing capabilities

    - [ ] 11.4 Create manual testing checklist generation based on WCAG criteria
    - [ ] 11.5 Implement focus management and visual focus indicator testing
    - [ ] 11.6 Add cognitive accessibility testing guidelines and automation

- [ ] 6.0 Distribution and Packaging Setup
    - [ ] 6.1 Configure npm package distribution with proper metadata
    - [ ] 6.2 Create standalone binary compilation for Windows, macOS, and Linux
    - [ ] 6.3 Implement Docker container with optimized image size
    - [ ] 6.4 Set up cross-platform compatibility testing
    - [ ] 6.5 Create installation documentation and usage examples
    - [ ] 6.6 Configure CI/CD pipeline for automated testing and distribution

Authenticated Page Scanning (New Feature)

Overview

- The tool should support scanning pages that require authentication (e.g., behind a login form).
- This is achieved by automating the login flow using Playwright before running the accessibility scan.

Implementation Steps

1. Add CLI options to accept login credentials (e.g., --username, --password, --login-url) or a path to a login script.
2. In the scan workflow, if login options are provided:
    - Launch Playwright and navigate to the login page.
    - Fill in the username and password fields.
    - Submit the login form and wait for navigation to the authenticated area.
    - Proceed to scan the target page(s) as an authenticated user.
3. Document this feature in the README and help output.

Tasks

- [ ] Add CLI options for login credentials or login script

- [ ] Implement Playwright login automation in the scan workflow
- [ ] Add documentation and usage examples for authenticated scanning
- [ ] Test with a sample site requiring login

Additional Test/Review Tasks (2025)

- [ ] Validate axe-core rules configuration is always an object
- [ ] Test configurable page ready wait timeout via CLI/config
- [ ] Confirm scoring is reduced for warnings
- [ ] Confirm console/report output distinguishes issues and warnings, includes summary and grouped warnings
- [ ] Confirm remediation tips are shown for common warnings
- [ ] Confirm improved output formatting and summary section
- [ ] Document new options in CLI/config sections