Cycle 1 Report

Authors:

Tianyang Wang

Kate Moreland

DJ Oakman

Haoran Ding

Submitted in partial fulfillment of the requirements for COMP 4710 Senior Design to the Department of Computer Science and Software Engineering, Samuel Ginn College of Engineering, Auburn University

Auburn, Alabama

September 29, 2025

Comp 4710 Senior Design Project

[System Metaphor – DJ Oakman: 2](#_Toc233200611)

[Cycle Intent – DJ Oakman: 3](#_Toc971068619)

[User Stories – Kate Moreland/DJ Oakman: 3](#_Toc777570434)

[Design Documentation – DJ Oakman: 16](#_Toc2032877497)

[Architecture 17](#_Toc1941875764)

[Structure 18](#_Toc1715087109)

[Interfaces 19](#_Toc861684704)

[Justifications of Decisions 20](#_Toc2022231144)

[Assumptions & Dependencies 21](#_Toc2086406256)

[Lessons Learned - Tianyang Wang: 22](#_Toc1850454571)

[1. Acceptance Tests 24](#_Toc883553639)

[2. Unit Tests 26](#_Toc2121474082)

[3. Test Logs 27](#_Toc196400613)

[4. Test Evidence (Screenshots) 29](#_Toc446346923)

[Risk Management and Mitigation – DJ Oakman: 33](#_Toc1531653604)

[Meeting Minutes – Kate Moreland: 34](#_Toc948440432)

[Version Description – Hoaran Ding 36](#_Toc1598495195)

[Source Code - Kate Moreland: 37](#_Toc1651723706)

[Sponsor’s Approval: 38](#_Toc1182810600)

# System Metaphor – DJ Oakman:

To design and create a system which can fix and identify accessibility issues from an uploaded PDF, Word Document or PowerPoint.

# Cycle Intent – DJ Oakman:

In this cycle, our focus was on refining and finalizing the application rather than setting up basic functionality. We concentrated on improving the user interface and graphical design to ensure that WCAG results and accessibility messaging were displayed clearly within the app instead of requiring a download. We also created an About page to provide users with details on WCAG standards and the purpose of the tool. On the technical side, we strengthened security by enforcing branch protections, detailed pull requests, and secret management through .env files and GitHub’s scanning tools. Finally, we deployed the updated system on GitHub Pages and Vercel, making the application more polished, secure, and ready for user interaction.

# User Stories – Kate Moreland/DJ Oakman:

Functionality  
**Name:** Upload documents from a file selection

* **Summary:** Allow users to upload documents (PDF, Word, PowerPoint) through a file selection component so that the files can be programmatically stored and accessed later.
* **Description:** A user should be able to select and upload a document from their local device using a web-based file selection component. Supported formats include PDF, DOCX, and PPTX. Once uploaded, the file will be stored in a designated system location or database and made accessible to backend code for further processing, retrieval, or integration with other features.

|  |  |
| --- | --- |
| Planned Hours | 3 |
| Planned Hours this Cycle | 3 |
| Actual Hours | 3 |
| Hours this Cycle | 3 |
| Coder | DJ Oakman |
| Tester | Kate Moreland |
| Reviewer | Kate Moreland |
| Status | Completed |

**Name:** Send PDF to Back-End for Accessibility Processing

* **Summary:** After a user uploads a PDF and presses “Submit” on the front end, the document will be sent to the back end to be processed by the Adobe PDF Accessibility API. The result will include an accessibility report and optionally a remediated PDF.
* **Description:**  
   Users will be able to upload PDFs through the front-end interface. When the user submits the document, it will be transmitted securely to the back end. The back end will call the Adobe PDF Accessibility API to:
  + Check the PDF for WCAG accessibility compliance (tags, alt text, reading order, headings).
  + Generate an accessibility report.
  + Optionally provide a remediated version of the PDF. The system should handle errors gracefully (e.g., invalid PDFs, API failures) and return clear feedback to the user.

|  |  |
| --- | --- |
| Planned Hours | 10 |
| Planned Hours this Cycle | 5 |
| Actual Hours | 5 |
| Hours this Cycle | 5 |
| Coder | DJ Oakman |
| Tester | Kate Moreland |
| Reviewer | Kate Moreland |
| Status | Completed |

**Name:** Display Uploaded PDF and Accessibility Results

* **Summary:** After a PDF is processed by the back end via the Adobe PDF Accessibility API, the front end will display the uploaded PDF, the accessibility report, and optionally a remediated PDF for the user to review.
* **Description:**  
   Once the back-end finishes processing a PDF, the system will return:
  + **Original uploaded PDF** – displayed or downloadable.
  + **Accessibility report** – showing WCAG compliance results, highlighting missing tags, alt text issues, reading order problems, etc.
  + **Remediated PDF (if available)** – downloadable so the user can replace the original document with an accessible version.
  + The front end should present this information clearly, with sections for the report and file downloads, and handle any errors if processing fails.

|  |  |
| --- | --- |
| Planned Hours | 8 |
| Planned Hours this Cycle | 4 |
| Actual Hours | 4 |
| Hours this Cycle | 4 |
| Coder | DJ Oakman |
| Tester | Kate Moreland |
| Reviewer | Kate Moreland |
| Status | Completed |

**Name:** Alternative text for images and figures

* **Summary:** If there are pictures and/or figures in the file, then they will have to be substituted with accessible text. This will make sure that it is compliant with the WCAG standards and allows for screen readings of these files.
* **Description:**

Whenever a user uploads a file that contains an image or figure, the system will flag it and then provide a suitable substitute for accessible text. The interface will allow for the following:

* + **Display** - Present the uploaded file with the image and the text that has been generated as well.
  + **Review** - The user should be able to check the information and ensure that it is correct.
  + **Warning** - If an image is of low quality, corrupted, or cannot be processed for any other reason, it will flag it so the user is aware.
  + The back end should be able to scan the file, extract the images or figures, send them to a service for accessible text generation, and re-integrate that inside the new document.

|  |  |
| --- | --- |
| Planned Hours | 8 |
| Planned Hours this Cycle | 0 |
| Actual Hours | 0 |
| Hours this Cycle | 0 |
| Coder | N/A |
| Tester | N/A |
| Reviewer | N/A |
| Status | New |

**Name:** Accessible headings and navigation

* **Summary:** The system needs to automatically analyze the structure of the heading in the files to make sure there is a logical hierarchy. This will create an accessible navigation for screen readers to correctly analyze the file.
* **Description:** Whenever a user loads a file, the system will need to automatically scan the content and analyze all of the headings and their types. Then the system will:
  + **Display a document outline** – Show a clear outline of the document’s structure with the headings included
  + **Identify issues** – Flag any issues that are discovered, like H1 followed by H3
  + **Automatic remediation** – The headings will automatically be resolved to fit with the WCAG rules
  + The back end has to be able to scan and analyze the file’s structure, apply the WCAG set of rules for heading hierarchy, and generate a new accessible file.

|  |  |
| --- | --- |
| Planned Hours | 6 |
| Planned Hours this Cycle | 0 |
| Actual Hours | 0 |
| Hours this Cycle | 0 |
| Code | N/A |
| Tester | N/A |
| Reviewer | N/A |
| Status | New |

Usability

**Name:** File upload and format handling

* **Summary:** The app must allow users to upload PDFs, PowerPoints, or Word documents, and provide clear feedback on what happens with each format.
* **Description:** Whenever a user uploads a file, then the system will:
  + Display accepted formats
  + Show upload progress
  + Provide clear message about what formats are supported
  + Notify the user if an unsupported file is uploaded

|  |  |
| --- | --- |
| Planned Hours | 3 |
| Planned Hours this Cycle | 3 |
| Actual Hours | 3 |
| Hours this Cycle | 3 |
| Code | Kate Moreland |
| Tester | DJ Oakman |
| Reviewer | DJ Oakman |
| Status | Completed |

**Name:** Secure backend processing

* **Summary:** The app must securely send files to the backend for processing without exposing sensitive information like API keys.
* **Description:**

When a file is uploaded, the system will:

* + Route the file to backend services via secure connections
  + Ensure API keys and processing logic remain hidden from the frontend
  + Return only results and processed files to the frontend

|  |  |
| --- | --- |
| Planned Hours | 3 |
| Planned Hours this Cycle | 3 |
| Actual Hours | 3 |
| Hours this Cycle | 3 |
| Code | Kate Moreland/DJ Oakman |
| Tester | Kate Moreland |
| Reviewer | DJ Oakman |
| Status | Completed |

**Name:** Results display on frontend as non downloadable format

* **Summary:** The app must clearly show results of accessibility checks in a simple, user-friendly format.
* **Description:**

After processing is complete, the system will:

* + Present results in a structured list or table
  + Highlight errors and warnings with clear labels
  + Offer tooltips or short explanations for each issue

|  |  |
| --- | --- |
| Planned Hours | 3 |
| Planned Hours this Cycle | 3 |
| Actual Hours | 3 |
| Hours this Cycle | 3 |
| Code | DJ Oakman |
| Tester | Kate Moreland |
| Reviewer | Kate Moreland |
| Status | Completed |

**Name:** Error handling and messaging

* **Summary:** The app must give clear, helpful error messages when something goes wrong.
* **Description:** If an error occurs, the system will:
  + Show a user-friendly error message (e.g., “This file type is not fully supported yet”)
  + Suggest next steps where possible (e.g., “Try converting to PDF first”)
  + Log technical details for developers while hiding them from end-users

|  |  |
| --- | --- |
| Planned Hours | 3 |
| Planned Hours this Cycle | 1.5 |
| Actual Hours | 1.5 |
| Hours this Cycle | 1.5 |
| Code | Kate Moreland |
| Tester | DJ Oakman |
| Reviewer | N/A |
| Status | In Development |

**Name:** Frontend hosted on GitHub pages

* **Summary:** The web app has to be available to the public, so it cannot be running locally.
* **Description: Taking use of Github pages to make the site public. Now it can be used by everyone and not just on the local machine.**

|  |  |
| --- | --- |
| Planned Hours | 5 |
| Planned Hours this Cycle | 5 |
| Actual Hours | 10 |
| Hours this Cycle | 10 |
| Code | Kate Moreland |
| Tester | Kate Moreland |
| Reviewer | Kate Moreland |
| Status | Completed |

**Name:** Getting the backend on Vercel

* **Summary:** The backend needs to be running on a https site so that the front end can access it for document analysis.
* **Description:**The backend has to have:
* Environmental Variables (Adobe API secrets)
* Running the backend code from Github

|  |  |
| --- | --- |
| Planned Hours | 4 |
| Planned Hours this Cycle | 4 |
| Actual Hours | 6 |
| Hours this Cycle | 6 |
| Code | Kate Moreland |
| Tester | Kate Moreland |
| Reviewer | Kate Moreland |
| Status | Completed |

**Name:** Adding the about page on the site

* **Summary:** The about page explains the need for the site and describes how it works.
* **Description:**The about page has the following:
* Link to Auburn site about accessibility standards
* Our reason for creating this
* How it works

|  |  |
| --- | --- |
| Planned Hours | 2 |
| Planned Hours this Cycle | 2 |
| Actual Hours | 1.5 |
| Hours this Cycle | 1.5 |
| Code | Kate Moreland |
| Tester | Kate Moreland |
| Reviewer | Kate Moreland |
| Status | Completed |

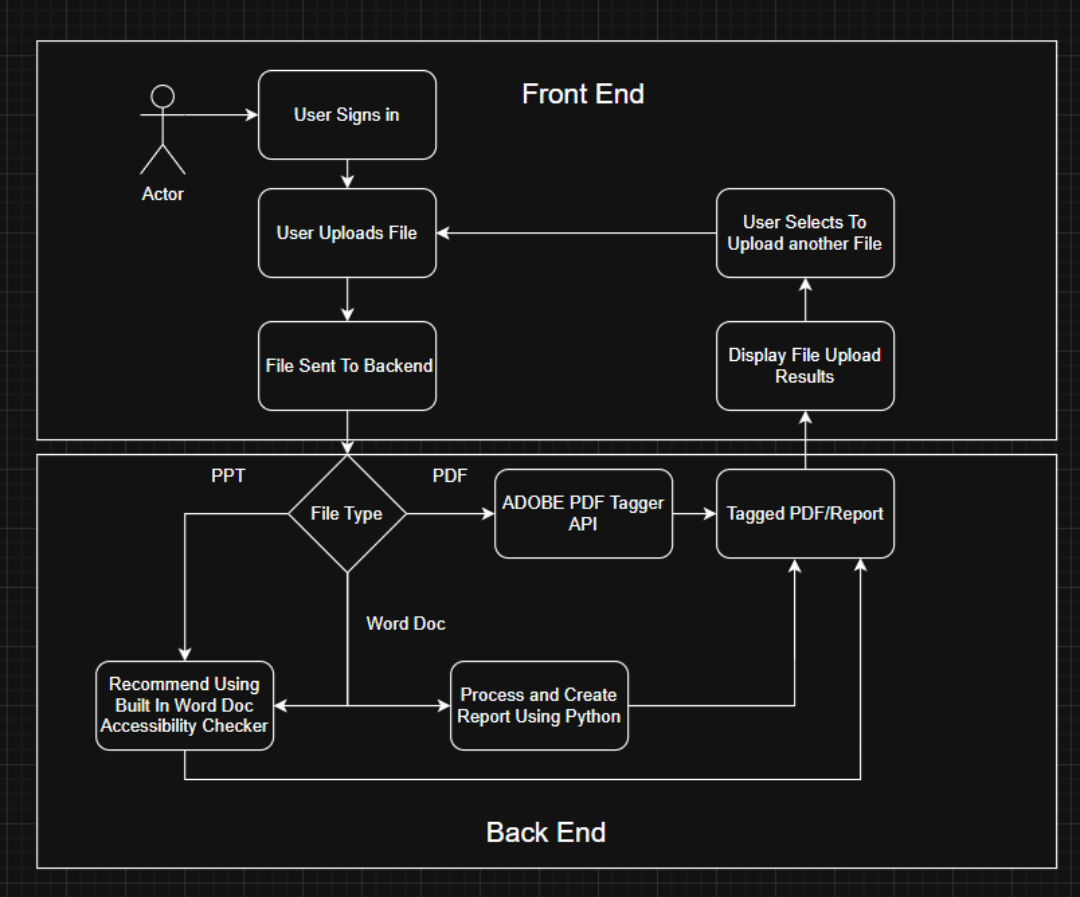
Performance

**Name:** App performance and responsiveness

* **Summary:** The app must process uploads and display results quickly so that users have a smooth experience and don’t feel delayed.
* **Description:** When a user uploads a file, the system will:
  + Ensure uploads complete within a reasonable time (under 10 seconds for average files)
  + Optimize backend requests so that large files or multiple users do not slow down performance
  + Provide clear feedback if processing takes longer than expected

|  |  |
| --- | --- |
| Planned Hours | 3 |
| Planned Hours this Cycle | 3 |
| Actual Hours | 3 |
| Hours this Cycle | 3 |
| Code | DJ Oakman |
| Tester | Kate Moreland |
| Reviewer | Kate Moreland |
| Status | Completed |

# Design Documentation – DJ Oakman:

All design decisions explained below

## **Architecture**

The Accessibility Checker is built on a client–server architecture. Users interact with the application through a secure, browser-based front end. After authenticating, they select one or more files using the upload component, which streams the data to the backend service. The backend serves as the system’s processing hub: it validates file type, size, and integrity before routing the file to the appropriate handler.

The processing pipeline branches depending on the file format. For Microsoft Word or PowerPoint files, the system currently returns guidance to use those applications’ built-in accessibility checkers. In upcoming iterations, the backend will incorporate Python-based analysis modules to extract structure, identify accessibility issues, and optionally generate detailed error reports or tagged outputs. For PDFs, the system integrates with Adobe’s PDF Tagger API and Accessibility Reporter to produce machine-readable reports and apply corrective tagging where possible.

Once processing is complete, the backend assembles a structured response that includes a status code, a user-friendly summary, detailed findings (with severity levels and recommended fixes), and metadata such as filename, number of pages scanned, and total processing time. The front end consumes this payload, presenting results in a clear, actionable format and maintaining a session history for repeat checks or report downloads.

Error handling is integral to the architecture: corrupted uploads, unsupported file types, or API failures are communicated through precise, retry-friendly error messages. Large files are supported via streaming uploads and asynchronous processing to preserve UI responsiveness.

## **Structure**

The system’s codebase is divided into front-end and back-end components.

On the front end, React-based components manage user authentication, file selection, and session history. These components interact with backend APIs via secure HTTP requests. The front end also interprets and renders response payloads, highlighting detected accessibility issues and offering contextual tips to guide user remediation.

On the backend, the structure centers on a routing and validation layer. This layer inspects incoming file metadata (MIME type, extension, and size) and directs the file to the appropriate processing path. File-specific modules handle Word, PowerPoint, and PDF content differently, calling either in-house Python analyzers or third-party APIs such as Adobe’s services. Additional submodules handle error reporting, metadata extraction, and response assembly.

Persistent state is minimized: the backend is largely stateless, with session information maintained on the client side to ensure scalability and simplify deployments.

## **Interfaces**

The primary interface between system components is a RESTful HTTP API. The front end submits file upload requests, and the backend responds with structured JSON payloads.

Each upload request includes:

* File stream or multipart form data
* Authentication tokens
* Metadata such as filename and size

Each response includes:

* Status code indicating success or failure
* Human-readable summary of findings
* Structured list of detected accessibility issues, with severity, location, and suggested remediations
* Metadata such as processing duration and pages scanned

External interfaces include the Adobe PDF Tagger API and Accessibility Reporter. These are accessed via HTTPS, and their responses are parsed and integrated into the unified output payload.

## **Justifications of Decisions**

Several key design decisions shape the Accessibility Checker.

* **Front–back separation**: This was chosen to maximize scalability and allow the front end to evolve independently of backend logic.
* **REST API communication**: REST was selected for simplicity and broad compatibility, given the system’s need to handle diverse file types and integrate with external services.
* **Third-party API integration for PDFs**: Adobe’s APIs were chosen because of their maturity and comprehensive accessibility tagging features, accelerating our ability to deliver meaningful results without reinventing established tools.
* **Incremental Word and PowerPoint support**: Returning to built-in accessibility checkers was a pragmatic first step, providing useful feedback to users while Python-based analysis is under active development.
* **Streaming uploads and asynchronous processing**: These were essential for handling large files while preserving responsiveness in the user experience.

## **Assumptions & Dependencies**

The system assumes that:

* Users have valid authentication credentials.
* Uploaded files are in supported formats (Word, PowerPoint, or PDF).
* Third-party APIs such as Adobe PDF Tagger and Accessibility Reporter remain available and performant.

Dependencies include:

* Adobe APIs for PDF tagging and reporting.
* Python libraries for document parsing and analysis (in development for Word and PowerPoint).
* The underlying framework and hosting infrastructure for both frontend (React) and backend (Node.js/Python stack).

The system deliberately avoids deep reliance on proprietary client environments, focusing instead on standards-based interfaces and scalable backend services.

# Lessons Learned - Tianyang Wang:

From our project, we gained a wide range of important lessons that helped us grow both technically and as a team. One of the most significant lessons was learning how to create Angular components. At the beginning, most of us had little to no experience with Angular, so figuring out how to build components that were both functional and visually clear was a challenge. Over time, we realized how central components are to structuring a website, keeping the interface organized, and making the overall experience user-friendly. Another major area of learning involved hosting our application. We explored how to host the frontend on GitHub Pages and set up the backend with Vercel, which turned out to be a lengthy process of troubleshooting and testing before everything worked smoothly. This taught us not only the technical steps required for hosting but also the patience and persistence needed to resolve deployment issues.

We also developed a deeper understanding of CORS (Cross-Origin Resource Sharing). At first, it was confusing to figure out why certain endpoints could not be accessed, but we learned that CORS acts as a security measure, allowing or restricting requests depending on the URL making the call. This was especially relevant when we worked with external APIs, as we needed to ensure that our requests were properly authenticated and aligned with the allowed origins. On the design side, we placed a strong emphasis on creating a clear and effective graphical interface. We had to carefully plan how to display error messages, highlight which accessibility standards were met or missed, and present information in a way that users could easily understand. This focus on design reinforced how important accessibility and clarity are in software development, especially when working with standards like CAG.

Lastly, we expanded our skills in working with external APIs. This included understanding how to connect to them, properly handle their responses, and manage authentication through client IDs and secrets. At first, the authentication process was a hurdle, but as we experimented and researched, we became more comfortable with securely integrating third-party services into our system. Altogether, this project gave us a hands-on learning experience across development, design, deployment, and integration, preparing us to handle more complex challenges in future projects.

Test Documentation – Hoaran Ding/ Tianyang Wang

This section provides detailed test documentation for the Accessibility-Checker project. It includes Acceptance Tests, Unit Tests, and Test Logs as required by the development deliverables.

## 1. Acceptance Tests

The following acceptance tests validate critical user journeys and accessibility expectations aligned with WCAG 2.2 and PDF/UA.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | Feature / Name | Description | Required Files / Setup | Steps to Execute | Expected Result |
| AT-001 | Home loads & keyboard focus | Verify the Home page loads without console errors and is keyboard navigable. | Angular dev server running (ng serve). | 1) Open <http://localhost:4200/Accessibility-Checker> 2) Open DevTools Console – ensure no errors  3) Use Tab to move focus across navigation and controls | Page visible; no console errors; focus order is logical and visible. |
| AT-002 | Accept .pdf only | Non-PDF files are rejected with a helpful message. PDF files proceed to processing. | One non-PDF (e.g., foo.txt) and one PDF (demo.pdf). | 1) Choose 'foo.txt' → Process  2) Choose 'demo.pdf' → Process | Non-PDF rejected or warned; PDF accepted and processing starts. |
| AT-003 | Process accessible sample (positive) | Using a well-tagged PDF sample, the report shows few or no serious issues. | Accessible PDF Demo Document (Adobe). | Upload sample → Click Process → Observe results | Minimal violations; structure/alt text appear as compliant. |
| AT-004 | Process inaccessible sample (negative) | Using an intentionally inaccessible flyer, the report flags key problems. | Boise State 'Gardening 101' Inaccessible Flyer (PDF). | Upload sample → Click Process → Observe results | Multiple issues detected: no tags, missing alt text, reading order problems, etc. |
| AT-005 | Toggle dark mode (operable) | Theme toggles via button; keyboard can activate; contrast remains legible. | Running app; visual check for contrast; optional color contrast tool. | Click 'Toggle Dark Mode' or press Space/Enter when focused | Theme switches; focus visible; text remains readable (target AA contrast). |
| AT-006 | About page is reachable & structured | About page reachable by keyboard; headings/links are perceivable and operable. | Running app. | Tab to 'About' → Enter → Inspect headings & links | Headings present in logical order; links focusable and activatable. |
| AT-007 | File Upload Local Save | File Upload should temporarily save file locally until submit | Running app. | Upload Sample -> View Page | File is uploaded to local page for storage. |
| AT-008 | File Upload Send to back end | After clicking process button, the file should be sent to the BE for processing. | Running backend and front end. | Upload a file to send to BE -> Click Process -> View response on front end. | File is uploaded to backend and logs and after a small time, results are viewable on front end. |
| AT-009 | Viewable Results on front end | After clicking the process button, the file results should be viewable on front end. | Running backend and front end. | Upload a file to send to BE -> Click Process -> View response on front end | Number of successes, failures, and manual checking items are listed on front end page. |
| AT-010 | Secure backend processing | All frontend backend traffic is HTTPS and no secrets (API keys) appear in requests/responses or bundled code. | Deployed frontend (GitHub Pages) and backend (Vercel). | Start an upload and open DevTools → Network/Security  Inspect all requests and responses  Search loaded JS for key-like strings (“API\_KEY”, “Bearer ”) | Requests are HTTPS with valid certs; no keys/tokens/stack traces in headers/bodies or bundles. |
| AT-011 | Results display as non-downloadable format | Results render in an on-page table/list with severity labels; no “Download report/file” control is present. | Running app; any file that produces findings. | Upload → Process  Inspect results area and UI controls | Structured results with clear Error/Warning labels; no direct download/export of the processed report. |
| AT-012 | Error handling and messaging | Operational failures show friendly guidance while technical details are not exposed to users. | Running App | Upload → Process to trigger failure  Observe user message and console/Network | UI shows a helpful message; no stack traces in UI; request fails gracefully. |
| AT-013 | Frontend hosted on GitHub Pages | Public site loads and functions without console errors. | Public GitHub Pages URL available. | Open public URL  Navigate to Upload page  Open DevTools Console | Site reachable over HTTPS; UI visible/operable; no console errors. |
| AT-014 | Backend on Vercel | Backend is reachable over HTTPS and processing endpoint responds successfully (env vars active). | Deployed Vercel backend; test file. | From frontend, upload a supported file  Observe processing call/response in Network | 200 OK (or defined success); processing completes, indicating backend and env variables (e.g., Adobe API) work. |
| AT-015 | App performance and responsiveness | Average file uploads transition to processing within ≤10 seconds; long tasks show “still working” feedback. | Stable network; 5–10 MB test file. | Start upload and time until “Processing” state  If processing exceeds threshold, observe status text | Upload completes and enters processing ≤10 s; if longer, a visible status appears. |
| AT-016 | Alternative text for images and figures | Images are detected and candidate alt text is generated for review. | Sample document with 2–3 images. | Upload sample → open Images/Alt Text section  Inspect entries | Each image is listed with generated alt text for review/edit; low-quality images (if present) show a warning badge. |
| AT-017 | Accessible headings and navigation | Document outline is shown and invalid heading sequences are flagged. | Sample with H1/H2/H3 and at least one H1→H3 jump. | Upload sample → open Structure/Headings  Review outline and issues | Outline shows headings with levels; issues list includes the H1→H3 jump with location details. |

## 2. Unit Tests

Run unit tests (Jasmine/Karma) on front end using command:

ng test

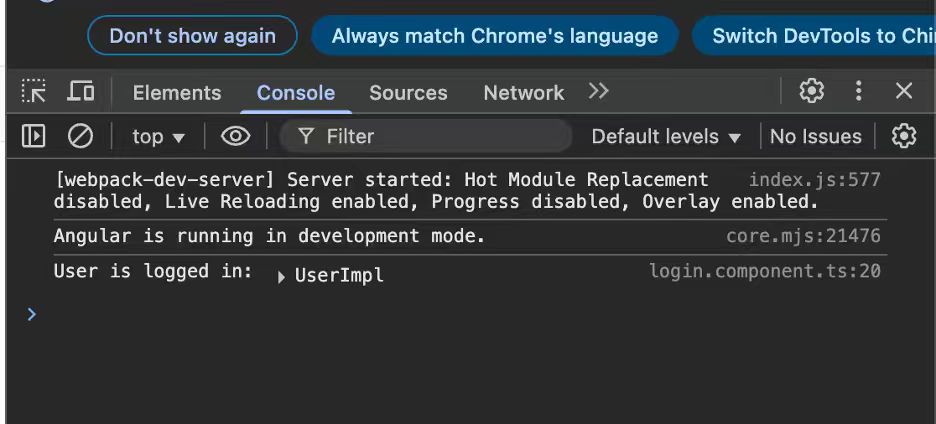
## 3. Test Logs

Record outcomes for each execution:

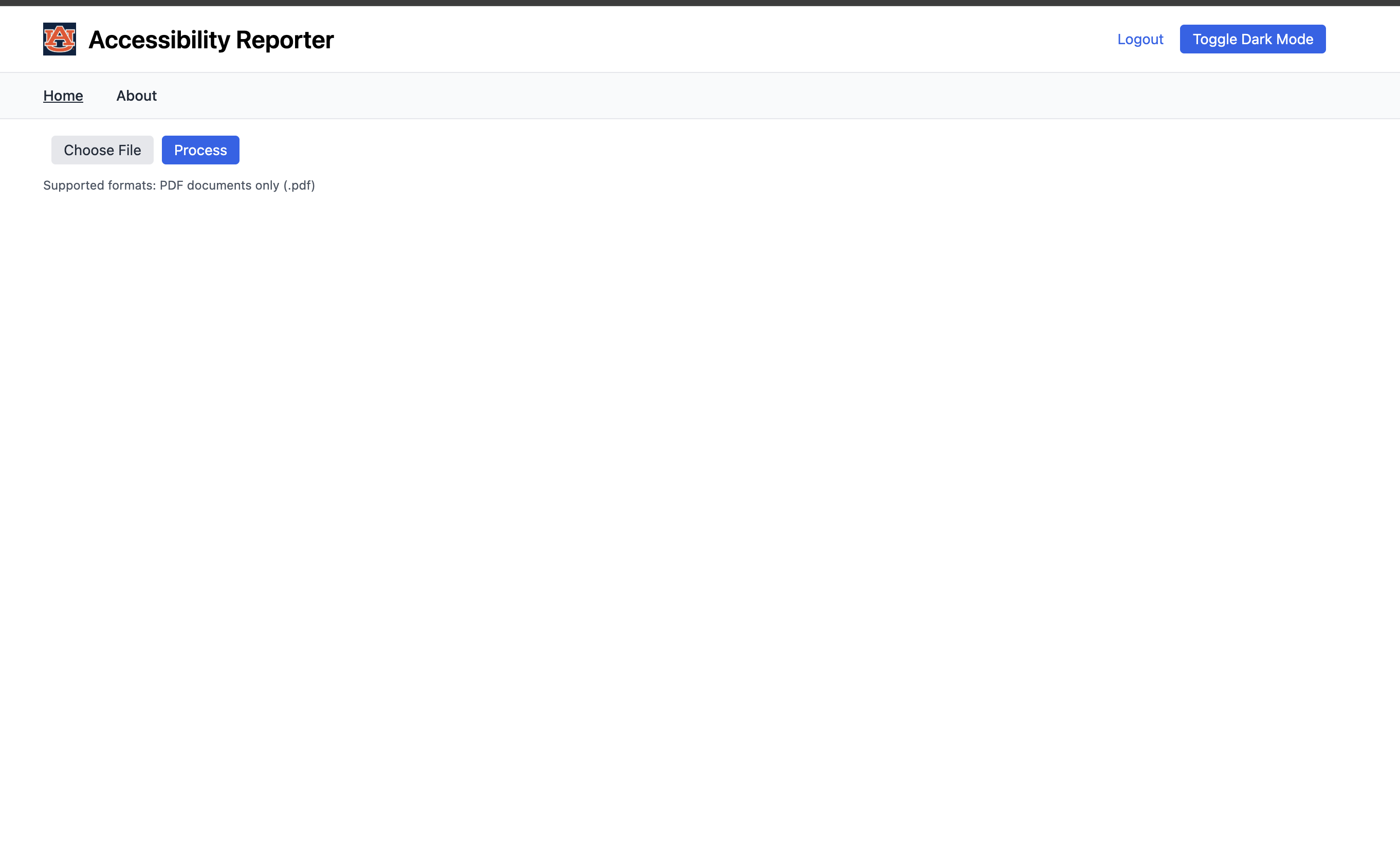
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Feature | Test ID | Tester | Date | Result (Pass/Fail) | Comments |
| Home loads | AT-001 | Tianyang Wang | 2025-10-01 | Pass | Console clean; focus order OK |
| Accept PDF only | AT-002 | Tianyang Wang | 2025-10-01 | Pass | foo.txt rejected; demo.pdf accepted |
| Accessible sample | AT-003 | Tianyang Wang | 2025-10-01 | Pass | 0–1 minor issues |
| Inaccessible sample | AT-004 | Tianyang Wang | 2025-10-01 | Pass | Multiple violations as expected |
| Dark mode | AT-005 | Tianyang Wang | 2025-10-01 | Pass | Contrast readable; focus visible |
| About page | AT-006 | Tianyang Wang | 2025-10-01 | Pass | Headings/links operable |
| File Upload | AT-007 | Tiangyang Wang | 2025-10-01 | Pass | File uploads properly and is visible to user |
| File Upload Send to back end | AT-008 | Tiangyang Wang | 2025-10-01 | Pass | File successfully sent; results returned after short delay. |
| Viewable Results on front end | AT-009 | Tiangyang Wang | 2025-10-01 | Pass | Results visible with counts of successes/failures/manual checks. |
| Secure backend processing | AT-010 | Tiangyang Wang | 2025-10-01 | Pass | All requests over HTTPS; no secrets exposed. |
| Results display as non-downloadable format | AT-011 | Tiangyang Wang | 2025-10-01 | Pass | Results structured in table; no download link present. |
| Error handling and messaging | AT-012 | Tiangyang Wang | 2025-10-01 | Pass | Helpful error displayed; no stack traces in UI. |
| Frontend hosted on GitHub Pages | AT-013 | Tiangyang Wang | 2025-10-01 | Pass | Public URL reachable; no console errors. |
| Backend on Vercel | AT-014 | Tiangyang Wang | 2025-10-01 | Pass | Backend responds with 200 OK; processing completes successfully. |
| App performance and responsiveness | AT-015 | Tiangyang Wang | 2025-10-01 | Pass | Upload completed in under 10s; long processing showed feedback. |
| Alternative text for images and figures | AT-016 | Tiangyang Wang | 2025-10-01 | Fail (Feature New) | Images detected but alt text generation not implemented yet. |
| Accessible headings and navigation | AT-017 | Tiangyang Wang | 2025-10-01 | Fail (Feature New) | Heading outline/auto-fix not available; flagged as pending. |

## 4. Test Evidence (Screenshots)

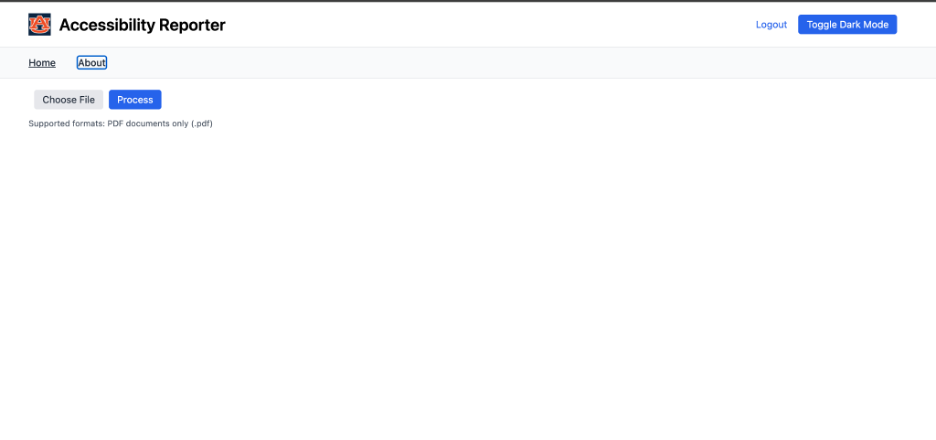
This section provides visual evidence for acceptance tests. Each figure is labeled with its corresponding Test ID.



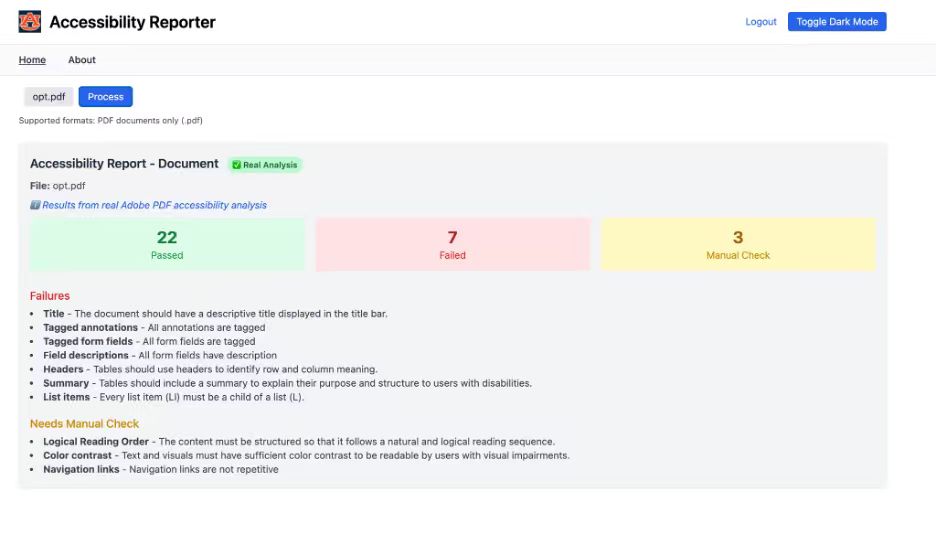
**Figure E-1. AT-001 Console — No errors shown (DevServer + Angular dev mode).**



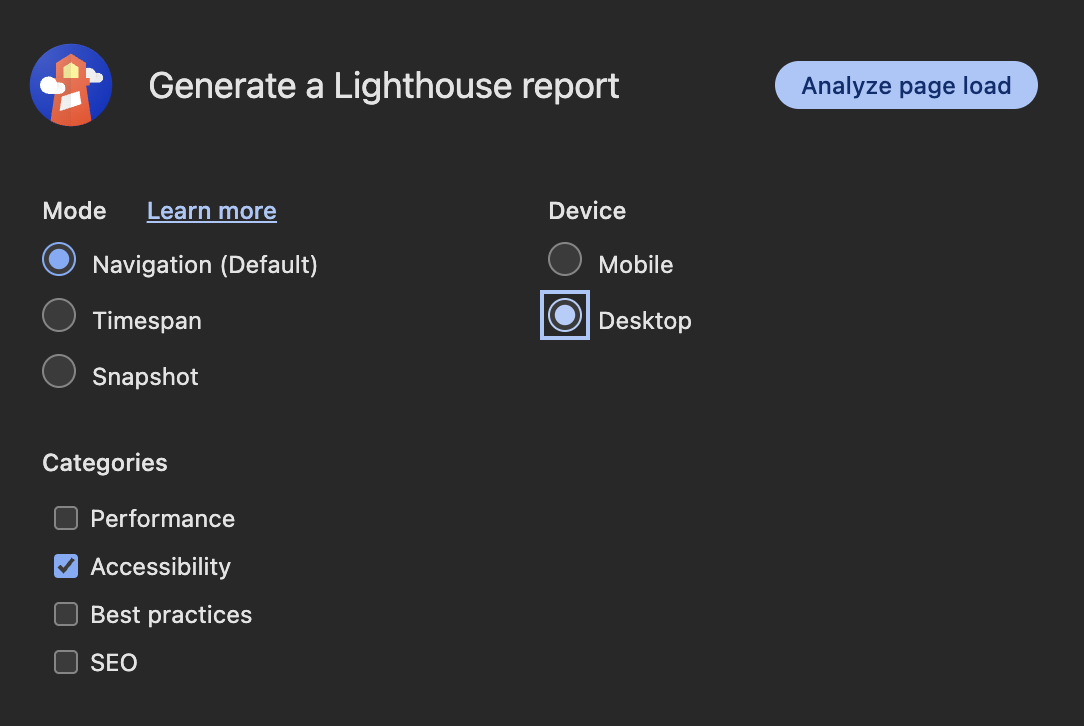
**Figure E-2. AT-001 Home — Application home page rendering.**



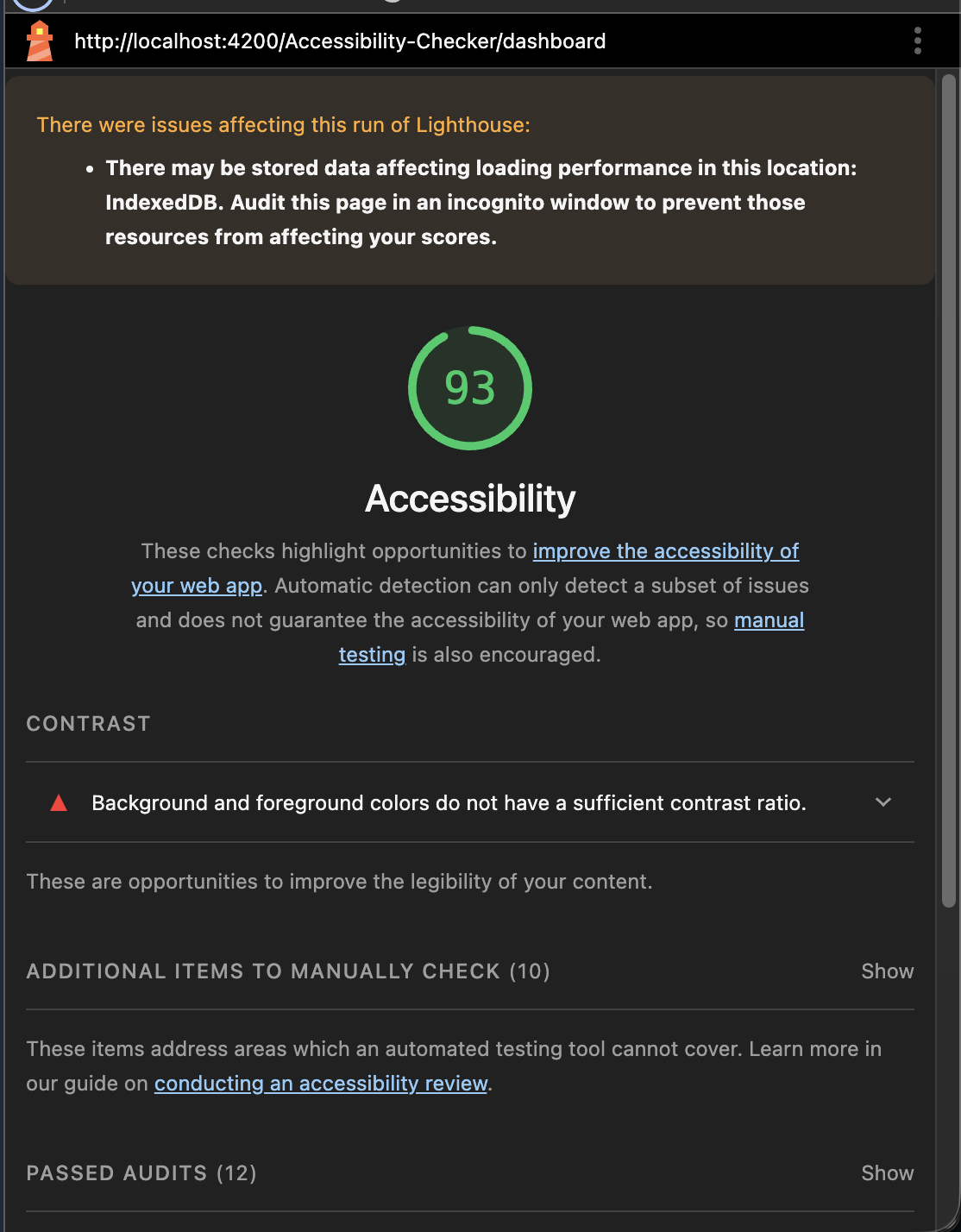
**Figure E-3. AT-006 About — Keyboard focus visible on 'About' link.**



**Figure E-4. AT-004 Inaccessible sample — Result panel with 22 Passed / 7 Failed / 3 Manual Check.**



**AT-005 – Lighthouse Accessibility Audit (Evidence)**



**FigureL-2.** **Lighthouse report with Accessibility score=93 and contrast warning.**

# Risk Management and Mitigation – DJ Oakman:

For risk management in our project, we took several steps to ensure stability, security, and accountability throughout development. To begin with, we required that every contributor create a separate branch named with their initials followed by a description of the feature or fix (for example, *djo/branch-description*). This naming system allowed us to clearly identify who was working on what and to keep changes isolated until they were fully reviewed. We also enforced a strict pull request policy so that no one could push directly to the main branch, reducing the chance of errors or unreviewed code being introduced into production.

In terms of security, we implemented secret management practices through GitHub, ensuring that sensitive information was never exposed in the repository. We made use of a .env file for deployment and included it in .gitignore so that it would never be pushed to version control. On top of that, GitHub’s secret scanning system was enabled to automatically detect and validate that no secrets were leaked during the development process.

Finally, we placed a strong emphasis on documentation and review. Every pull request was required to have a detailed description of the changes being made, and we carefully reviewed each file before approving the merge. This thorough review process, combined with secret scanning and branch protections, provided a multi-layered approach to mitigating risks in both our codebase and workflow.

# Meeting Minutes – Kate Moreland:

9/17/25

* + Met with Tyler Hamilton (on our sponsor’s team for the project)
  + Showed our Demo with a locally hosted web app (before we got it working publicly)
  + Tyler suggested that we add a bulk option for multiple files
  + Also, it is suggested that we add links to our site for other resources
  + Add the following goals for the next couple of weeks:
    - Start talking about the bulk option
    - Start the SSO process
    - Ensure that it is available to employees as well as students

9/19/25

* + Talked as a team about testing the new frontend site that is public and making sure it is correct
  + Take use of postman to ensure that it is talking to the correct backend endpoint
  + Modified backend endpoint to return a regular HTTP response rather than a downloadable file.

9/25/25

* + Discussed what should be on our about page
  + Created the about page and made sure that the links worked and that it was readable in both light and dark mode
  + Modified the front end to show new display format rather than downloadable.

10/1/25

* + Meet with our sponsor
  + They want us to focus on only .docx files and have a remediated file ready for download

# Version Description – Hoaran Ding

|  |  |
| --- | --- |
| **Version Number** | **v0.1.0 (Dev)** |
| **Description of Application** | Angular front-end that uploads and analyzes PDF accessibility and displays results. |
| **Key Features** | PDF upload validation; processing & results view; dark/light theme toggle; routing to Home/About. |
| **Known Issues / Limits** | Scanned (image-only) PDFs lack real text and often fail checks; large PDFs may affect performance; ruleset may require tuning. |

**References**

[W3C – Web Content Accessibility Guidelines (WCAG) 2.2](https://www.w3.org/TR/WCAG22/)

[W3C WAI – What's new in WCAG 2.2](https://www.w3.org/WAI/standards-guidelines/wcag/new-in-22/)

[W3C – PDF Techniques for WCAG](https://www.w3.org/TR/WCAG20-TECHS/pdf)

[W3C WAI – Techniques for WCAG (PDF-specific items)](https://www.w3.org/WAI/WCAG21/Techniques/)

[PDF Association – ISO 14289 (PDF/UA) overview](https://pdfa.org/resource/iso-14289-pdfua/)

[PDF Association – PDF/UA Flyer](https://pdfa.org/resource/pdfua-flyer/)

[Adobe – Accessible PDF Demo Document](https://developer.adobe.com/document-services/docs/assets/b3b8ca5aad09665ff577cffb35ad98cf/Adobe_Accessibility_Auto_Tag_API_Sample.pdf)

[Boise State – Inaccessible flyer sample (Gardening 101)](https://www.boisestate.edu/webguide/wp-content/uploads/sites/16/2021/05/example-inaccessible-flyer.pdf)

[Deque – axe-core documentation](https://www.deque.com/axe/core-documentation/)

[Deque – axe API documentation](https://www.deque.com/axe/core-documentation/api-documentation/)

# Source Code - Kate Moreland:

<https://kmoreland126.github.io/Accessibility-Checker> (Web app)

<https://github.com/kmoreland126/Accessibility-Checker> (Frontend code)

<https://github.com/TiantangWangAUBURN/Architectural-Spike> (Backend code)

<https://accessibility-checker-2sj3m9idk-kates-projects-e59a7a1b.vercel.app/> (Backend)

# Sponsor’s Approval:

Asim told Tyler Hamilton to approve for us and Tyler Hamilton said he sent an email confirming status report.  
Screenshots of this below:

