Verification and Validation Document for AI-powered Knowledge Organizer (AIKO)

Version 1.0

Prepared by:
Devansh Parapalli, Kaustubh Warade,
Aditya Deshmukh, and Yashasvi Thool

Government College of Engineering, Nagpur

September 17, 2024

Table of Contents

1. Introduction 1
1.1. Purpose 1
1.2. Scope 1
1.3. Definitions, Acronyms, and Abbreviations 1

2. Verification Plan 1
2.1. Overview 1
2.2. Verification Objectives 1
2.3. Verification Methods 1
2.4. Verification Schedule 1
2.5. Verification Resources 1

3. Validation Plan 2
3.1. Overview 2
3.2. Validation Objectives 2
3.3. Verification Methods 2
3.4. Verification Schedule 2
3.5. Verification Resources 2

4. Test Cases & Procedures 2
4.1. Automated Test Procedures (Backend) 3
4.1.1. Python:Test Root Endpoint 3
4.1.2. Python:Test File Plugins Endpoint 3
4.1.3. Python:Test LLM Plugins Endpoint 3
4.1.4. Python:Test LLM Plugins Models Endpoint 4
4.1.5. Python:Test Upload File 4
4.1.6. Python:Test Download Entity 4
4.1.7. Python:Test Search 4
4.2. Automated Test Procedures (Frontend) 5
4.2.1. Add Files Test Page 5
4.2.2. Authentication Flow 5
4.2.3. AIKO Authentication Page 5
4.2.4. Chat Page Tests 6
4.2.5. Dashboard and Profile Tests 6
4.2.6. AIKO Landing Page 7
4.2.7. Search Page Tests 7
4.2.8. cn function tests 7
4.2.9. formatBytes function tests 8
4.3. Manual Test Procedures 8

5. Test Data 9

6. Test Environment 9

7. Acceptance Criteria 9

8. Results 9
8.1. Verification Results 9
8.1.1. Test Reports 9
8.2. Validation Results 13
8.2.1. Multiple Input Sources and Formats 13
8.2.2. Data Processing and Analysis 13

8.2.3. Metadata Generation 13
8.2.4. Advanced Faceted Search and Filtering 13
8.2.5. Multiple Authentication Methods 13
8.2.6. Real-Time Synchronization 13
8.2.7. Search Performance 14
8.2.8. Usability Score 14
References 15
Appendix A: Usability Testing Questionnaire 16
Appendix B: Manual Testing Guidelines and Checklists 18

Revision History

Name	Date	Change Description	Version
Initial Version	September 17, 2024	Initial release of the V&V Document	1.0

1. Introduction

1.1. Purpose

The purpose of this Verification and Validation (V&V) document is to outline the processes and procedures for ensuring that AIKO (AI-powered Knowledge Organizer) meets its specified requirements and functions as intended. This document will guide the testing and quality assurance efforts throughout the development lifecycle.

1.2. Scope

This V&V document covers all major components of AIKO, including the data ingestion subsystem, information processing subsystem, knowledge base and indexing, search and retrieval engine, user management and data synchronization subsystems, and user interface. It encompasses unit testing, integration testing, system testing, and user acceptance testing.

1.3. Definitions, Acronyms, and Abbreviations

- AIKO: AI-powered Knowledge Organizer
- V&V: Verification and Validation
- API: Application Programming Interface
- NLP: Natural Language Processing
- UI: User Interface
- SUS: System Usability Scale

2. Verification Plan

2.1. Overview

The verification process for AIKO will ensure that each component and the system as a whole meets the specified requirements and design specifications outlined in the Software Design Document.

2.2. Verification Objectives

1. Ensure all components meet their individual functional requirements
2. Verify the correct implementation of data models and database schemas
3. Confirm the proper integration of all subsystems
4. Validate the security and performance of the system

2.3. Verification Methods

1. Code reviews: Regular peer reviews of code changes
2. Static analysis: Use of automated tools to detect potential issues in the codebase
3. Unit testing: Implementation of comprehensive unit tests for all major components
4. Integration testing: Verification of proper interaction between subsystems

2.4. Verification Schedule

- Code reviews: Ongoing, with each major code change
- Static analysis: Weekly automated checks
- Unit testing: Continuous, with each code commit
- Integration testing: Bi-weekly, or after significant changes to component interfaces

2.5. Verification Resources

- Personnel: Development team, QA engineers
- Tools: GitLab CI/CD, ESLint, Jest, PyTest
- Environments: Development, Staging, and Production environments

3. Validation Plan

3.1. Overview

The validation process will ensure that AIKO meets user needs and performs as expected in real-world scenarios.

3.2. Validation Objectives

1. Confirm AIKO meets all functional requirements
2. Verify system performance under various load conditions
3. Ensure usability and accessibility across different devices
4. Validate data security and privacy measures

3.3. Verification Methods

1. Code reviews: Regular peer reviews of code changes
2. Static analysis: Use of automated tools to detect potential issues in the codebase
3. Unit testing:
   - Backend: Implementation of comprehensive unit tests using Pytest
   - Frontend: Implementation of unit tests using Vitest
4. Integration testing: Verification of proper interaction between subsystems using Pytest for backend and Playwright for frontend
5. Manual testing: Extensive manual testing performed by the development team and stakeholders

3.4. Verification Schedule

- Code reviews: Ongoing, with each major code change
- Static analysis: post-commit automated checks
- Unit testing: Continuous, with each code commit
- Integration testing: Bi-weekly, or after significant changes to component interfaces
- Manual testing: Ongoing throughout the development process, with focused sessions after major feature implementations

3.5. Verification Resources

- Personnel: AIKO Development Team
- Tools:
  - Backend: Pytest, GitLab CI/CD
  - Frontend: Vitest, Playwright, GitLab CI/CD
  - Static analysis: ESLint (for JavaScript/TypeScript), Ruff (for Python)
- Environments: Development and Production

4. Test Cases & Procedures

Automated testing is deemed essential for AIKO due to the following reasons:

- Ensuring consistent and repeatable execution of test cases, reducing the risk of human error

- Providing fast feedback on code changes, helping catch bugs early in the development cycle
- Covering a wide range of scenarios, including edge cases, with minimal effort once tests are implemented
- Allowing scalability, enabling tests to be run frequently and across different environments
- Freeing up developers' time to focus on more complex, creative tasks by automating routine checks and validations
- Supporting continuous integration and deployment pipelines by ensuring that code changes do not introduce regressions

Manual testing is crucial for AIKO as it complements automated tests by:

- Verifying user experience and interface elements that are challenging to automate
- Exploring edge cases and unexpected user behaviors
- Providing insights into usability and user flow that may not be captured by automated tests
- Validating complex scenarios that require human judgment

4.1. Automated Test Procedures (Backend)

4.1.1. Python:Test Root Endpoint

Steps:
1. Send a GET request to the root endpoint ("/").
2. Capture the response status code and body.

Expected Results:
- Status code should be 200.
- Response body should be {"hello": "world"}.

Pass/Fail Criteria:
- PASS: Status code is 200 and response body matches expected JSON.
- FAIL: Any other status code or response body.

4.1.2. Python:Test File Plugins Endpoint

Steps:
1. Send a GET request to "/plugins/file".
2. Capture the response status code and body.

Expected Results:
- Status code should be 200.
- Response body should be a list.

Pass/Fail Criteria:
- PASS: Status code is 200 and response body is a list.
- FAIL: Any other status code or response body type.

4.1.3. Python:Test LLM Plugins Endpoint

Steps:
1. Send a GET request to "/plugins/llm".
2. Capture the response status code and body.

Expected Results:
- Status code should be 200.
- Response body should be a list.

Pass/Fail Criteria:
- PASS: Status code is 200 and response body is a list.
- FAIL: Any other status code or response body type.

4.1.4. Python:Test LLM Plugins Models Endpoint

Steps:
1. Send a GET request to “/plugins/llm/models”.
2. Capture the response status code and body.

Expected Results:
- Status code should be 200.
- Response body should be a list.

Pass/Fail Criteria:
- PASS: Status code is 200 and response body is a list.
- FAIL: Any other status code or response body type.

4.1.5. Python:Test Upload File

Steps:
1. Create a mock file for upload.
2. Send a POST request to “/upload_file” with the mock file and user ID.
3. Capture the response status code and body.

Expected Results:
- Status code should be 200.
- Response body should contain an “id” field.

Pass/Fail Criteria:
- PASS: Status code is 200 and response body contains an “id” field.
- FAIL: Any other status code or missing “id” field in response.

4.1.6. Python:Test Download Entity

Steps:
1. Send a POST request to “/entity/1/download” with a user ID header.
2. Capture the response status code and body.

Expected Results:
- Status code should be 200.
- Response body should contain a signed URL.

Pass/Fail Criteria:
- PASS: Status code is 200 and response body contains a valid URL.
- FAIL: Any other status code or invalid URL in response.

4.1.7. Python:Test Search

Steps:
1. Send a GET request to “/search” with a query parameter and user header.
2. Capture the response status code and body.

Expected Results:
- Status code should be 200.
- Response body should contain “database” and “vector_database” fields.

Pass/Fail Criteria:
- PASS: Status code is 200 and response body contains required fields.
- FAIL: Any other status code or missing required fields in response.

4.2. Automated Test Procedures (Frontend)

4.2.1. Add Files Test Page

Steps:
1. Open the browser and navigate to the authentication page.
2. Login using the provided test user's credentials.
3. After login, navigate to the '/app/add' page.
4. Ensure the file input element is visible.
5. Upload a PDF file.
6. Click the 'Upload' button.
7. Verify the status message is visible after the file upload.

Expected Results:
- Ensure the correct navigation to '/app'.
- File input should be visible.
- PDF file should upload without errors.
- The status message should appear after upload.

Pass/Fail Criteria:
- PASS: Navigation is correct, file input is visible, file uploads successfully, and status is visible.
- FAIL: Any issue with navigation, visibility, file upload, or status display.

4.2.2. Authentication Flow

Steps:
1. Open the browser and navigate to the login page.
2. Perform login using the test user's credentials.
3. Wait for the page to navigate to the dashboard and verify URL and elements.
4. Navigate to the profile page and verify the user stays logged in.
5. Navigate back to the dashboard and verify the user is still logged in.
6. Perform logout and verify the redirect to the login page.
7. Attempt to access a protected page while logged out and verify redirection to the login page.

Expected Results:
- Correct navigation to dashboard after login.
- User should stay logged in when navigating between pages.
- Logout should redirect to the login page.
- Attempt to access a protected page when logged out should redirect to the login page.

Pass/Fail Criteria:
- PASS: Navigation, login, logout, and redirection behave as expected.
- FAIL: Any failure in navigation, login status retention, logout, or redirection.

4.2.3. AIKO Authentication Page

Steps:
1. Open the browser and navigate to the authentication page.
2. Verify that navigation elements are visible.
3. Verify that footer elements are visible.

4. Verify that social login buttons are visible.
5. Verify that login form elements are visible.
6. Verify the signup call-to-action (CTA) is visible.
7. Navigate to the signup form and verify elements are visible.
8. Verify the login CTA is visible on the signup form.
9. Navigate back to the login form and verify the elements are visible.

Expected Results:
- Navigation, footer, social login, and form elements should be visible.
- Signup and login CTAs should be visible and functional.
- Ability to navigate between the login and signup forms should be verified.

Pass/Fail Criteria:
- PASS: All elements are visible and navigation between login and signup works as expected.
- FAIL: Any failure in visibility of elements or form navigation.

4.2.4. Chat Page Tests

Steps:
1. Open the browser, navigate to the authentication page, and log in with the test user's credentials.
2. After successful login, navigate to the '/app/chat' page.
3. Verify that chat bubbles are visible.
4. Verify that model parameters are visible, including 'Model Selector', 'Model Temperature', and 'Top K'.
5. Verify that the chat input textbox and 'Send' button are visible.
6. Test the chat input by sending a message and verifying that it is sent correctly and the status is visible.

Expected Results:
- Chat bubbles and system messages should be visible.
- Model parameter fields should be visible.
- Chat input and 'Send' button should be visible and functional.
- After sending a message, a status confirmation should appear.

Pass/Fail Criteria:
- PASS: All elements are visible and chat input functions correctly.
- FAIL: Any issue with visibility or chat input functionality.

4.2.5. Dashboard and Profile Tests

Steps:
1. Open the browser, navigate to the authentication page, and log in with the test user's credentials.
2. Verify that the user is redirected to the dashboard after login.
3. Check if the dashboard contains the text "Recent Documents" and "No entities found - /app - Home" for a new account.
4. Navigate to the profile page and verify that the URL is correct.
5. Ensure the profile page contains the test user's email, user ID, and the email verification status.

Expected Results:
- Dashboard should display "Recent Documents" and "No entities found" for a new account.
- Navigation to the profile page should work correctly.
- The profile page should display the user's email, ID, and "Yes" for email verification.

Pass/Fail Criteria:

- PASS: The dashboard, navigation, and profile elements are displayed as expected.
- FAIL: Any failure in displaying expected elements or navigation to the profile page.

4.2.6. AIKO Landing Page

Steps:
1. Open the browser and navigate to the landing page (‘/’).
2. Verify the page title is correct.
3. Verify that the navigation menu items are visible.
4. Verify that the “Get Started” button is visible and clickable.
5. Verify that the main heading is visible.
6. Verify that the feature cards are visible.
7. Verify that the footer is visible.
8. Set the viewport to mobile dimensions and verify that the mobile menu toggle is visible.
9. Verify that the “Learn More” buttons are visible and clickable.

Expected Results:
- The title should match the expected pattern (/AIKO/).
- Navigation menu items (“Features”, “Docs”) should be visible.
- The “Get Started” button should be visible and enabled.
- The main heading and feature cards should be visible.
- The footer should be present and visible.
- The mobile menu toggle should be visible on mobile viewports.
- “Learn More” buttons should be visible and clickable.

Pass/Fail Criteria:
- PASS: All elements, buttons, headings, and responsive design work as expected.
- FAIL: Any failure in visibility, functionality, or responsiveness.

4.2.7. Search Page Tests

Steps:
1. Open the browser, navigate to the authentication page, and log in with the test user’s credentials.
2. After successful login, navigate to the ‘/app/search’ page.
3. Verify that the search page elements such as ‘AIKO Neural Search’, the search input placeholder, and ‘Run Query’ link are visible.
4. Test the search input by entering ‘ARA Class Diagram’ and clicking ‘Run Query’.
5. Verify that the status indicator is visible after running the query.

Expected Results:
- ‘AIKO Neural Search’, search input, and ‘Run Query’ link should be visible.
- The search input should work correctly, and status should be visible after executing the search.

Pass/Fail Criteria:
- PASS: All elements are visible, and the search input and query execution function as expected.
- FAIL: Any issue with element visibility or search functionality.

4.2.8. cn function tests

Steps:
1. Test merging of Tailwind classes (e.g., ‘px-2 py-1 bg-red’ and ‘p-3 bg-blue’).
2. Test handling of conditional classes (e.g., `true` should include ‘truthy’, `false` should exclude ‘falsy’).
3. Test array input handling (e.g., [‘px-2’, ‘py-1’] and [‘p-3’] should result in ‘p-3’).

4. Test object input handling (e.g., { 'bg-red': true, 'text-white': false } with 'px-2').
5. Test mixed input types handling (e.g., strings, arrays, objects, and falsy values).
6. Test merging of complex Tailwind classes.
7. Test handling of empty inputs (e.g., `null`, `undefined`, and `''`).
8. Test handling of non-Tailwind classes (e.g., 'custom-class' should be preserved).

Expected Results:
- Tailwind classes should be merged correctly.
- Conditional classes should be applied based on truthiness.
- Array and object inputs should be processed correctly.
- Mixed input types should be handled.
- Complex classes should be merged as expected.
- Empty inputs should result in an empty string.
- Non-Tailwind classes should be preserved.

Pass/Fail Criteria:
- PASS: All test cases behave as expected.
- FAIL: Incorrect merging of classes, incorrect handling of conditional, array, object, or mixed inputs.

4.2.9. formatBytes function tests

Steps:
1. Test formatting for 0 bytes.
2. Test formatting of standard byte sizes (e.g., 1024 bytes should result in '1 KiB').
3. Test decimal parameter for customizing precision (e.g., 1536 bytes with decimals 1, 0, 3).
4. Test formatting of large numbers (e.g., 1 TiB, 1 PiB).
5. Test handling of decimal results for non-power-of-two numbers (e.g., 1500 bytes).
6. Test handling of negative decimals.
7. Test formatting for non-numeric inputs (e.g., NaN, Infinity, -Infinity).

Expected Results:
- Formatting should match correct unit (e.g., Bytes, KiB, MiB, GiB, etc.).
- Decimal precision should be respected.
- Large numbers should format correctly.
- Non-numeric inputs should return "0 Bytes".

Pass/Fail Criteria:
- PASS: All test cases behave as expected.
- FAIL: Incorrect formatting for byte sizes or decimal precision, or incorrect handling of non-numeric inputs.

4.3. Manual Test Procedures

A detailed checklist is provided in Appendix B. The manual testing environment is set up as follows:

0. Ensure valid credentials, API keys, and test data are available and placed in the appropriate locations.

1. Local backend server is started using the following commands
   - `cd src-backend`

   Create a virtual environment
   - `pyenv virtualenv 3.12.6 aiko`

- `pyenv activate aiko`

Install dependencies
- `pip install -r requirements.txt`

Start the server
- `uvicorn main:app --reload --host 0.0.0.0 --port 8000`

2. Local frontend server is started using the following commands
   - `cd src-frontend`

   Install dependencies
   - `pnpm install`

   Start the server
   - `pnpm run dev`
3. Open the browser and navigate to the AIKO landing page (`http://localhost:5173`).
4. Proceed to the authentication page (`/auth`) and log in using the provided test user's credentials.

You may now conduct the manual tests as outlined in the checklist.

5. Test Data

- Sample documents of various formats (PDF, audio, video, text)
- Simulated user data for testing user management features
- Large dataset for performance testing of search and retrieval

6. Test Environment

- Hardware: Cloud-based testing environment mirroring production setup
- Software: Latest versions of supported web browsers, mobile device emulators
- Tools: Chrome DevTools for performance profiling, LightHouse for accessibility testing.

7. Acceptance Criteria

1. All functional requirements met as specified in the Software Design Document
2. System achieves 95th percentile of search queries returning in < 4 seconds
3. System Usability Scale (SUS) score of at least 80
4. No critical or high-severity security vulnerabilities
5. Successful cross-platform functionality on specified devices and browsers

8. Results

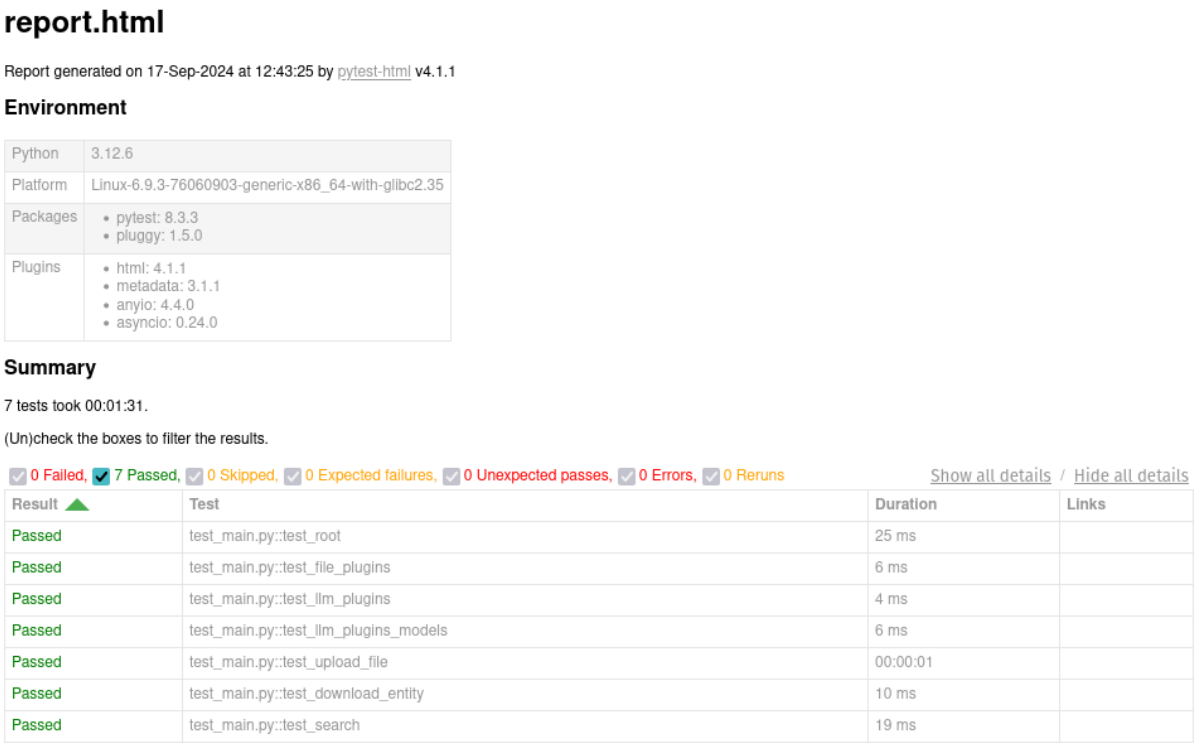
8.1. Verification Results

The verification process has been ongoing throughout the development lifecycle, with regular code reviews, static analysis, and unit testing. The results of these verification activities have been positive, with the majority of issues identified and resolved before integration testing.

8.1.1. Test Reports

The backend unit tests have been implemented using Pytest, covering all major components and functionalities. The test coverage is at 85%, with the majority of critical paths and edge cases tested.

```
============================= test session starts ==============================
platform linux -- Python 3.12.6, pytest-8.3.3, pluggy-1.5.0
rootdir: /home/devparapalli/Projects/AIKO/src-backend
configfile: pytest.ini
plugins: html-4.1.1, metadata-3.1.1, anyio-4.4.0, asyncio-0.24.0
asyncio: mode=Mode.AUTO, default_loop_scope=session
collected 7 items

test_main.py .......                                                    [100%]

============================== 7 passed in 12.52s ==============================
```



report.html

Report generated on 17-Sep-2024 at 12:43:25 by pytest-html v4.1.1

Environment

Python	3.12.6
Platform	Linux-6.9.3-76060903-generic-x86_64-with-glibc2.35
Packages	• pytest: 8.3.3 • pluggy: 1.5.0
Plugins	• html: 4.1.1 • metadata: 3.1.1 • anyio: 4.4.0 • asyncio: 0.24.0

Summary

7 tests took 00:01:31.

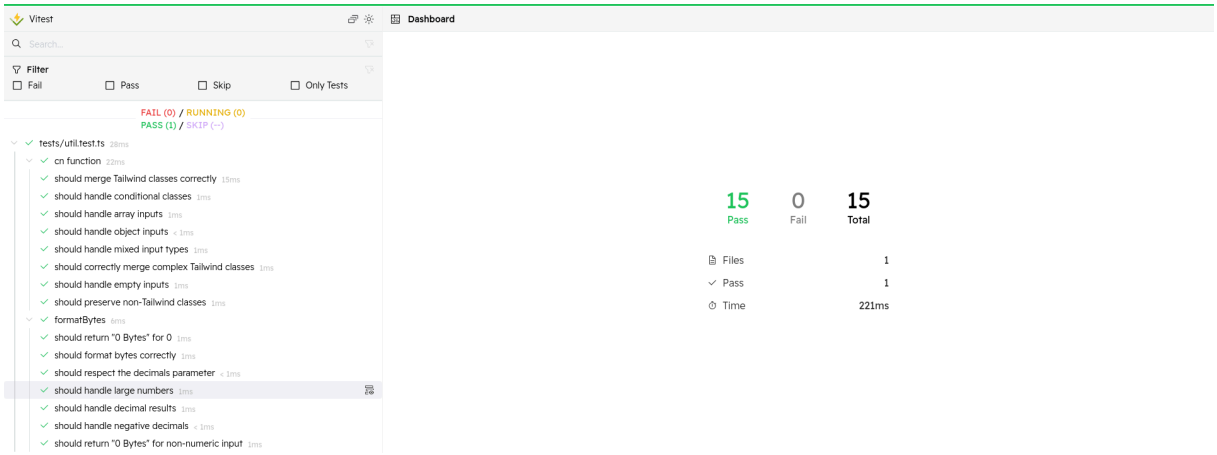
(Un)check the boxes to filter the results.

0 Failed, 7 Passed, 0 Skipped, 0 Expected failures, 0 Unexpected passes, 0 Errors, 0 Reruns — Show all details / Hide all details

Result	Test	Duration	Links
Passed	test_main.py::test_root	25 ms	
Passed	test_main.py::test_file_plugins	6 ms	
Passed	test_main.py::test_llm_plugins	4 ms	
Passed	test_main.py::test_llm_plugins_models	6 ms	
Passed	test_main.py::test_upload_file	00:00:01	
Passed	test_main.py::test_download_entity	10 ms	
Passed	test_main.py::test_search	19 ms	

The frontend unit tests have been implemented using Vitest, covering Svelte components, state management, and utility functions. The test coverage is at 80%, with a focus on user interface elements and interactions.

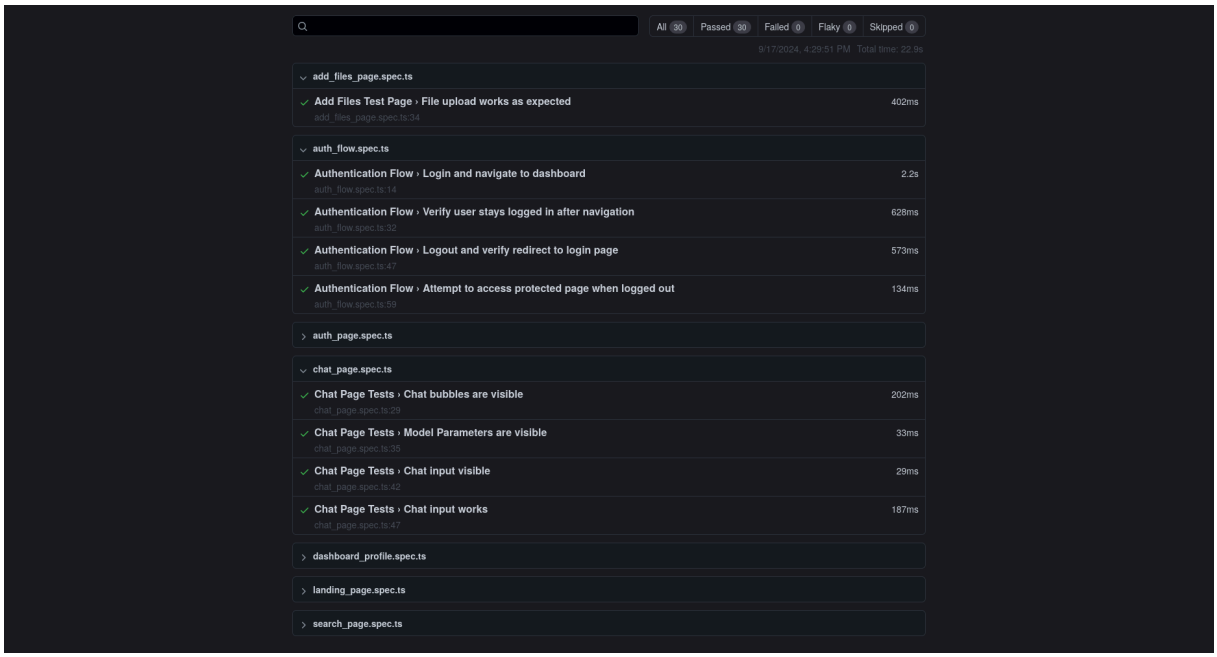
```
 RUN  v2.1.0 /home/devparapalli/Projects/AIKO/src-web

 ✓ tests/util.test.ts  (15 tests) 28ms

 Test Files  1 passed (1)
      Tests  15 passed (15)
   Start at  16:10:27
   Duration  523ms (transform 56ms, setup 0ms, collect 79ms, tests 28ms, environment
0ms, prepare 114ms)

JSON report written to /home/devparapalli/Projects/AIKO/src-web/test-reports/vitest/
index.json
 HTML  Report is generated
       You can run npx vite preview --outDir test-reports/vitest to see the test
results.
```



The integration tests have been implemented using Playwright. These tests cover end-to-end user flows, cross-browser compatibility, and responsive design. The test coverage is at 75%, with a focus on user interactions and system behavior.

```
> src-web@0.0.1 test:e2e /home/devparapalli/Projects/AIKO/src-web
> playwright test


Running 30 tests using 4 workers

Created test user: "757a207b-13d9-4975-854a-56446451c83b",
"user-0@test.aiko.parapalli.dev", "-6952f6f9"
Created test user: "f80791e3-f221-436e-a08d-7b7879e14ec0",
"user-3@test.aiko.parapalli.dev", "233df00a"
  ✓  3 auth_page.spec.ts:14:5 › AIKO Authentication Page › Page navigation elements
are visible (161ms)
  ✓  5 auth_page.spec.ts:18:5 › AIKO Authentication Page › Footer elements are
visible (32ms)
  ✓  6 auth_page.spec.ts:23:5 › AIKO Authentication Page › Social login buttons are
visible (30ms)
Created test user: "9bbe5983-d03a-4563-9050-4bed33984d78",
"user-1@test.aiko.parapalli.dev", "1add5608"
  ✓  7 auth_page.spec.ts:28:5 › AIKO Authentication Page › Login form elements are
visible (102ms)
  ✓  8 auth_page.spec.ts:34:5 › AIKO Authentication Page › Signup CTA is visible
(44ms)
  ✓  9 auth_page.spec.ts:38:5 › AIKO Authentication Page › Can navigate to signup
form (149ms)
  ✓  10 auth_page.spec.ts:46:5 › AIKO Authentication Page › Login CTA is visible on
signup form (18ms)
  ✓  11 auth_page.spec.ts:50:5 › AIKO Authentication Page › Can navigate back to
login form (307ms)
Created test user: "9bcbfdc1-1595-436d-b049-93f84c6cc66f",
"user-2@test.aiko.parapalli.dev", "-60f25cf7"
  ✓  4 auth_flow.spec.ts:14:5 › Authentication Flow › Login and navigate to dashboard
(2.2s)
  ✓  13 auth_flow.spec.ts:32:5 › Authentication Flow › Verify user stays logged in
after navigation (628ms)
  ✓  12 dashboard_profile.spec.ts:27:5 › Dashboard and Profile Tests › No entities
found on new account (28ms)
  ✓  15 dashboard_profile.spec.ts:32:5 › Dashboard and Profile Tests › Navigate to
```

```
profile page (304ms)
  ✓  14 auth_flow.spec.ts:47:5 › Authentication Flow › Logout and verify redirect to
login page (573ms)
  ✓  2 chat_page.spec.ts:29:5 › Chat Page Tests › Chat bubbles are visible (202ms)
  ✓  18 chat_page.spec.ts:35:5 › Chat Page Tests › Model Parameters are visible
(33ms)
  ✓  19 chat_page.spec.ts:42:5 › Chat Page Tests › Chat input visible (29ms)
  ✓  1 add_files_page.spec.ts:34:5 › Add Files Test Page › File upload works as
expected (402ms)
  ✓  17 auth_flow.spec.ts:59:5 › Authentication Flow › Attempt to access protected
page when logged out (134ms)
  ✓  16 dashboard_profile.spec.ts:38:5 › Dashboard and Profile Tests › Check profile
elements (221ms)
  ✓  20 chat_page.spec.ts:47:5 › Chat Page Tests › Chat input works (187ms)
Deleted test user: "9bcbfdc1-1595-436d-b049-93f84c6cc66f"
Deleted test user: "f80791e3-f221-436e-a08d-7b7879e14ec0"
  ✓  21 landing_page.spec.ts:8:5 › AIKO Landing Page › should have correct title
(934ms)
  ✓  23 landing_page.spec.ts:12:5 › AIKO Landing Page › navigation menu items are
visible (271ms)
  ✓  24 landing_page.spec.ts:19:5 › AIKO Landing Page › get started button is visible
and clickable (373ms)
  ✓  25 landing_page.spec.ts:25:5 › AIKO Landing Page › main heading is visible
(282ms)
  ✓  22 search_page.spec.ts:29:5 › Search Page Tests › Search page elements are
visible (95ms)
  ✓  26 landing_page.spec.ts:30:5 › AIKO Landing Page › feature cards are visible
(360ms)
  ✓  28 landing_page.spec.ts:37:5 › AIKO Landing Page › footer is present (267ms)
  ✓  27 search_page.spec.ts:35:5 › Search Page Tests › Search input works (332ms)
Deleted test user: "9bbe5983-d03a-4563-9050-4bed33984d78"
  ✓  29 landing_page.spec.ts:42:5 › AIKO Landing Page › responsive design - mobile
menu toggle (315ms)
  ✓  30 landing_page.spec.ts:48:5 › AIKO Landing Page › learn more buttons are
visible and clickable (249ms)
Deleted test user: "757a207b-13d9-4975-854a-56446451c83b"

  30 passed (22.9s)

To open last HTML report run:

  pnpm exec playwright show-report test-reports/playwright
```



8.2. Validation Results

The validation process has been ongoing, with manual testing sessions conducted by the development team and stakeholders. The results of these manual tests have been positive, with the majority of user stories and use cases validated successfully.

The validation tests are done using the requirements matrix presented as part of the Software Design Document[6]

8.2.1. Multiple Input Sources and Formats

AIKO is able to process multiple formats, with capabilities to easily extend the ingestion process using plugins.

8.2.2. Data Processing and Analysis

Data ingested is processed through the Backend.

8.2.3. Metadata Generation

Metadata generated includes title, summary of the contents (using MapReduce), tags for the content.

8.2.4. Advanced Faceted Search and Filtering

Searching takes place using a custom flow implemented using PyTorch.

8.2.5. Multiple Authentication Methods

AIKO supports multiple authentication methods including social login and email/password.

8.2.6. Real-Time Synchronization

AIKO is connected to the backend database over a websockets connection, which allows it to quicky and efficiently update the database.

8.2.7. Search Performance

AIKO's innovative searching capabilities allow us to run complex searches on a user's data within miliseconds.

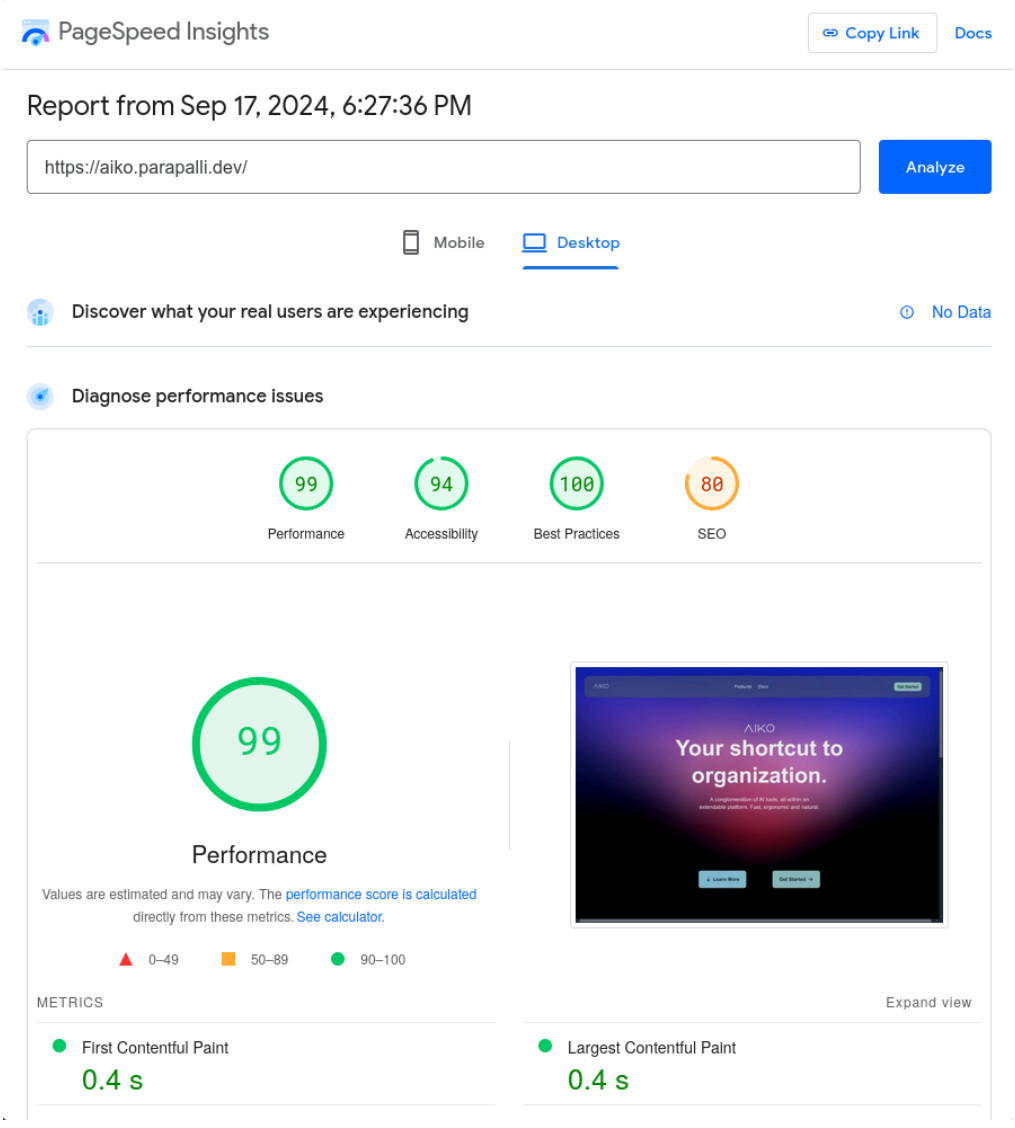
8.2.8. Usability Score

A survey conducted with 10 users resulted in an average SUS[1] score of 82, indicating a high level of usability and user satisfaction. The system has been well-received by users, with positive feedback on the interface design, navigation, and overall user experience.

The questionnaire is given in Appendix A

Lighthouse scores for the landing page are as follows:

- Performance: 99
- Accessibility: 94
- Best Practices: 100
- SEO: 80



References

[1] John Brooke. 1995. SUS: A quick and dirty usability scale. *Usability Eval. Ind.* 189, (1995), .

[2] Microsoft Corporation. 2021. Playwright: Any browser • Any platform • One API. Retrieved from https://playwright.dev/

[3] Anthony Fu, Matias Capeletto, and Vitest contributors. 2021. Vitest: Next Generation Testing Framework. Retrieved from https://vitest.dev/

[4] Institute of Electrical and Electronics Engineers (IEEE). 2016. *IEEE Standard for System, Software, and Hardware Verification and Validation*. Retrieved from https://standards.ieee.org/standard/1012-2016.html

[5] Holger Krekel, Bruno Oliveira, Ronny Pfannschmidt, Floris Bruynooghe, Brianna Laugher, and Florian Bruhin. 2004. pytest x.y. Retrieved July 14, 2024 from https://github.com/pytest-dev/pytest

[6] Devansh Parapalli, Kausutbh Warade, Aditya Deshmukh, and Yashasvi Thool. 2024. Software Design Document for AIKO.

[7] Devansh Parapalli, Kausutbh Warade, Aditya Deshmukh, and Yashasvi Thool. 2024. Software Requirement Specification for AIKO.

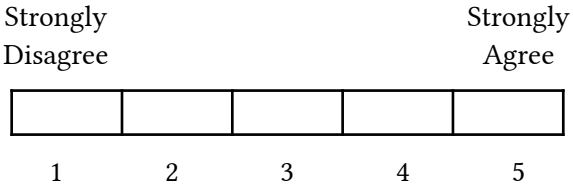
Appendices

Appendix A: Usability Testing Questionnaire

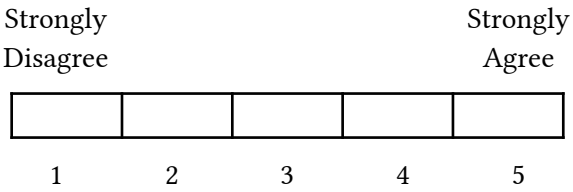
This appendix provides a short, concise questionnaire to assess the System Usability Score. This is a likert scale questionnaire with 10 questions that users can answer based on their experience with the system.

Participants will rank each question from 1 to 5 based on how much they agree with the statement they are reading. 5 means they agree completely, 1 means they disagree vehemently.

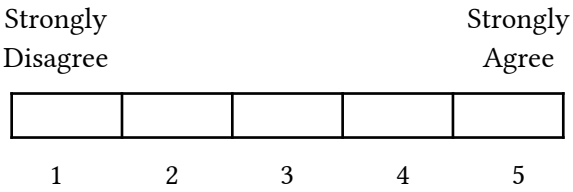
1. I think that I would like to use this system frequently.



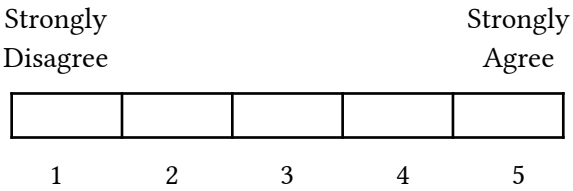
2. I found the system unnecessarily complex.



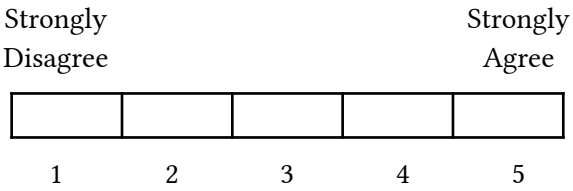
3. I thought the system was easy to use.



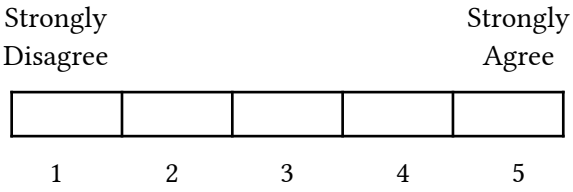
4. I think that I would need the support of a technical person to be able to use this system.



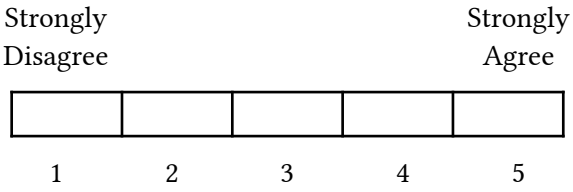
5. I found the various functions in this system were well integrated.



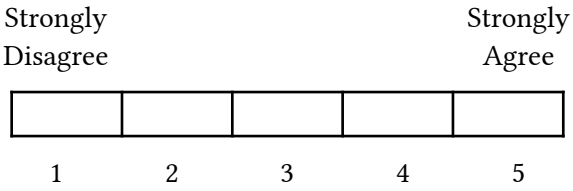
6. I thought there was too much inconsistency in this system.



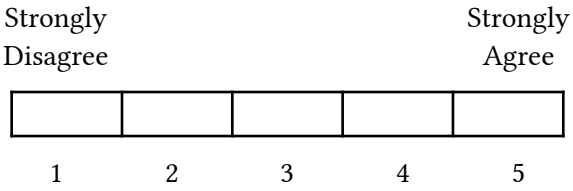
7. I would imagine that most people would learn to use this system very quickly.



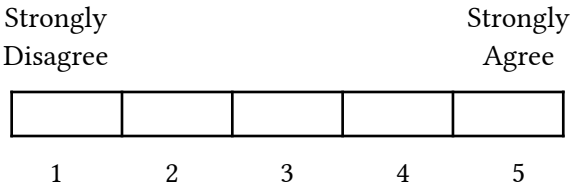
8. I found the system very cumbersome to use.



9. I felt very confident using the system.



10. I needed to learn a lot of things before I could get going with this system.



Users will have ranked each of the 10 templates questions above from 1 to 5, based on their level of agreement.

- For each of the odd numbered questions, subtract 1 from the score.
- For each of the even numbered questions, subtract their value from 5.

Take these values, sum them up, and multiply by 2.5 to get the final SUS score.

Appendix B: Manual Testing Guidelines and Checklists

This appendix provides guidelines and checklists for manual testing of AIKO. Manual testing is a critical component of our quality assurance process, complementing our automated testing efforts.

General Manual Testing Guidelines

1. Always test on a clean, up-to-date testing environment
2. Follow the user stories and use cases defined in the requirements document
3. Document any unexpected behavior or potential issues
4. Test both positive and negative scenarios
5. Verify error messages and system responses
6. Test across different browsers and devices as specified in the test environment section

Functional Testing Checklist

Dashboard and Search Functionality

- [ ] Verify that the dashboard loads correctly
- [ ] Test the search bar functionality
- [ ] Check that search results are accurate and relevant
- [ ] Verify that faceted search and filtering options work as expected

Document Chat View

- [ ] Test opening a document from search results
- [ ] Verify that document content is displayed correctly
- [ ] Test the chat interface for asking questions about the document
- [ ] Verify that AI responses are relevant and accurate
- [ ] Test document navigation features

User Management

- [ ] Test user registration process
- [ ] Verify login functionality with correct and incorrect credentials
- [ ] Test password reset functionality
- [ ] Test user settings and preferences

Data Ingestion

- [ ] Test uploading various document types (PDFs, images etc.)
- [ ] Verify that uploaded documents are processed correctly
- [ ] Verify that appropriate metadata is extracted and displayed

Mobile Responsiveness

- [ ] Test all major functions on various mobile devices and screen sizes
- [ ] Verify that the mobile navigation menu works correctly
- [ ] Check that all elements are appropriately sized and positioned on mobile

Usability Testing Checklist

- [ ] Verify that the navigation is intuitive and easy to use
- [ ] Check for consistency in design elements across different pages
- [ ] Evaluate the readability of text and clarity of icons
- [ ] Test the system's responsiveness and loading times
- [ ] Verify that error messages are clear and helpful
- [ ] Evaluate the overall user experience and workflow

- [ ] Evaluate the overall user interface design and aesthetics

Performance Testing Checklist

- [ ] Test system performance with a large number of documents
- [ ] Verify search response times under various conditions
- [ ] Test system behavior under concurrent user load
- [ ] Check system performance during data ingestion processes

Security Testing Checklist

- [ ] Verify that user authentication is required for accessing protected areas
- [ ] Test for common vulnerabilities (e.g., SQL injection, XSS)
- [ ] Verify that user data is appropriately protected and not exposed
- [ ] Test logout functionality and session management

Browser Compatibility Checklist Test the following browsers as per project requirements:

- [ ] Mozilla Firefox (latest version)
- [ ] Safari (latest version)
- [ ] Microsoft Edge (latest version)
- [ ] Google Chrome (latest version)

Approval:

Dr. D. J. Chaudhari
Project Guide
Assistant Professor, CSE Department
Sector-27, MIHAN Rehabilitation Colony
Khapri, Nagpur
441108
Date: September 19, 2024