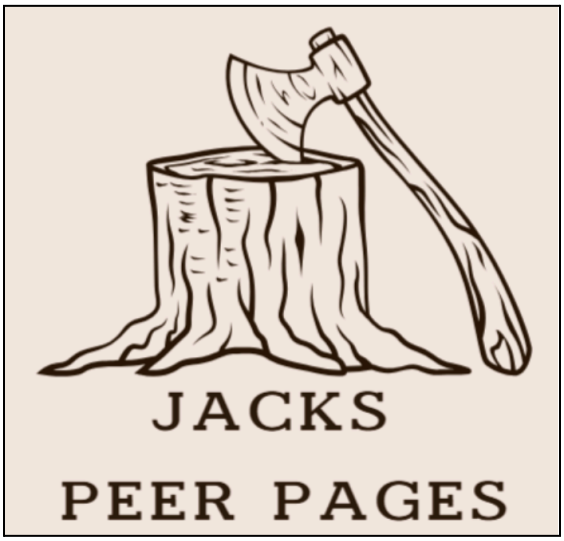
Software Testing Plan



Jack's Peer Pages

March 12th, 2026

Sponsor: Dr. Keith Nowicki
Team Mentor: Md Nazmul Hossain
Faculty Mentor: Dr. Ana Chaves

Team: Haley Berger (Team lead), Haley Kloss (member), Jeremiah Lopez (member), Tyler Austin (member)

Version 1.0

Table of Contents

Introduction

The website described in this document is a compendium of articles where users can share their peer reviews and read the peer reviews of other users. The nature of the website in terms of average usage includes uploading articles, searching for articles, posting peer reviews, reading peer reviews, and citing articles. There are both mentor accounts and student accounts, with student accounts being connected to their mentor so that mentor-level users can vet peer reviews made by student accounts. The primary interactions that users will have with the system are read and write actions. The intended audience for the website is researchers, from college-age students entering the ecosystem to veteran researchers. As a result, our primary quality goals are intuitiveness, usability, reliability, and ease of navigation.

The primary architectural components of the project are the Digital Ocean hosting, Django architecture, and a MySQL database. As a result, it is important that we test the website's ability to handle user concurrency, user navigation of the site, and user querying. Other aspects to be tested are uploading articles, searching for and accessing articles, writing and viewing peer reviews, flagging articles for AI, and account creation for students and mentors. All of the aforementioned features have been shown during Alpha Demonstrations I and II, but require further testing to ensure their functionality in all use cases.

This document describes the testing strategy that our team will use to validate our website. The types of testing that will be employed include unit testing, integration testing, and usability testing. Unit testing will be used to check for appropriate page navigation, integration testing will be used to validate database queries and usability testing will be used to ensure that users can navigate through the website comfortably with minimal friction. Unit testing and integration testing will occur continuously while the site is running and can be observed while viewing terminal output. User testing will be conducted beginning Monday, 3/30/2026.

Usability testing will be our focus during the testing process. This is because the project is heavily based on user interaction, and most functionality comes from how users interface with the website. The most important aspects required for our website to be successful are all in how comfortable the user is navigating the website, how quickly they can learn to use its features, and whether or not they run into bugs during interaction. Below is a more in-depth description of each of these facets of our testing strategy.

Unit Testing

For this project, unit testing will focus on verifying the correctness of specific pieces of the website. It will isolate specific application logic from the rest. Due to the project being Django based, this will primarily happen within the context of backend components such as model methods, form validation logic, and view functions that process user requests. These Django components are responsible for validating user input, managing article and peer review logic, and making sure the relationship between mentor and student accounts works properly.

The main goal of unit testing is to ensure that the main logic behind our primary features works as expected. This includes, but is not limited to, peer review submission, article uploading, and account validation. The function logic related to these features interacts with other parts of the system. This means that they must all individually work as expected to ensure that they can work together and not cause bugs through the system. Detecting defects early can allow our team to identify developmental issues and fix them before it becomes a bigger problem. Having these tests will also make sure new system components work as expected and don't cause unexpected bugs.

The unit tests will be implemented using Django's built-in testing framework, which is essentially an extension of Python's unittest library. These tests will be executed locally within the development environment and may also be run prior to demonstrations or major feature merges. These tests would be set to auto-run, where deployment of the new version only runs if the tests pass. Each and every test case will verify that a specific function or method behaves properly and as expected under both normal and abnormal conditions.

Because much of the system depends on user input and database interactions, the focus of unit testing will be in regards to validating:

- Form logic
- Model constraints and the relationship between them
- View logic (which processes user input)
- Access control logic between mentor and student accounts

Some parts of the system will intentionally not be unit tested. These include front-end HTML renders and third-party libraries like Google reCAPTCHA. Because these components are externally maintained and are better evaluated through actual integration testing rather than unit testing, we decided to go that route.

Scenarios (Units Under Test)

Article Model and Upload Logic

Purpose: Represents an uploaded research article and stores associated metadata like title, author, and abstract. Unit testing ensures that article creation, validation, and storage behave correctly.

Test Case Categories:

- Valid Inputs: articles with a valid title and author; articles with a link to the original paper
- Boundary Case: articles with long/short titles; articles with and without links; articles with missing information
- Invalid Inputs: empty submissions; missing title; unsupported text field types
- Sample Tests:
    - Uploading an article with valid information successfully creates a database record
    - Submitting an article without required fields (like title) triggers a validation failure and does not upload the article to the database
    - Adding improperly triggers an input type error

Peer Review Submission Form

Purpose: Validates and processes peer reviews submitted by users. Unit tests verify that review data is validated and stored correctly.

Test Case Categories:

- Valid Inputs: review text submitted for an existing article
- Boundary Case: short or long review submissions; submissions with different set status' (draft, submitted, etc.)
- Invalid Inputs: empty review submissions; review submitted for a non-existent article
- Sample Tests:
    - Submitting a valid peer review, which saves the review to the database
    - Submitting an empty review, which results in a validation error
    - Submitting a review for a deleted (or non-existent) article, which returns an error

Mentor Approval Logic

Purpose: Ensures that student-submitted peer reviews can be reviewed or approved by their associated mentor accounts.

Test Case Categories:

- Valid Inputs: student account linked to a valid mentor account; mentor reviewing an associated student's submission
- Boundary Case: student accounts with multiple pending peer reviews
- Invalid Inputs: student without mentor attempting submission; mentor attempting to approve the peer review of a student that is not theirs
- Sample Tests:
    - Mentor successfully approves a student's peer review
    - Student without a mentor relationship cannot submit a review
    - Unauthorized mentors cannot approve a student's review

Article Search Function

Purpose: Processes article search queries and retrieves matching results from the database.

Test Case Categories:

- Valid Inputs: searches using keywords present in article titles and/or tags
- Boundary Case: empty search queries; queries with a large number of results; queries with no results
- Invalid Inputs: query strings with unknown symbols
- Sample Tests:
    - Searching for an existing keyword returns the correct articles
    - Searching with an empty query returns all results
    - Searching with uncommon keywords returns an empty list (no matches)
    - Searches with many matches only display the first 25 and then have an option to go to the next page

Bibliography Management

Purpose: Maintains a personal bibliography that allows them to store references to research articles. Articles can be added either from within the system or from external sources. If an article already exists within the system, the bibliography entry should maintain a link back to the internal article record. If the article originates from an external source, the entry should be stored as a standalone reference.

Test Case Categories:

- Valid Inputs: Adding an internally hosted article to the bibliography; adding an externally sourced article reference
- Boundary Case: duplicate article entries; large numbers of bibliography entries
- Invalid Inputs: missing required article information
- Sample Tests:
    - Adding an internal article correctly creates a linked bibliography entry
    - Adding an external reference stores the entry without a system link
    - Attempting to link a bibliography entry to an invalid article triggers an error

Article / Peer Review Editing

Purpose: Users may edit articles and peer reviews previously submitted (if they have the proper permissions). Unit tests verify that edits correctly update the stored review while maintaining the relationship to the original article and reviewer.

Test Case Categories:

- Valid Inputs: users editing their own peer reviews; mentors editing student peer reviews; admins editing articles
- Boundary Case: editing only some of the fields
- Invalid Inputs: unauthorized user attempts to edit peer review/article; submitting empty text during editing
- Sample Tests:
    - Author successfully updates their peer review review
    - Edited review text replaces the previous review content
    - Editing article metadata updates the correct database fields
    - Unauthorized users cannot modify articles/peer reviews

Integration Testing

For our project, integration testing is tied to the project's backend services, which are used to support read and write operations, as well as other UI navigation. Our project is very dependent on reliably getting and posting data, and as a result, we will look for integration points in which users need to retrieve data and upload data. We will also be looking for areas of the app where users interact with API supported features.

For our project, the main integration that we will be focusing on is querying the MySQL database. The MySQL database is the primary external component of our website, and holds user data, article data, and peer review data. As a result, the website does not work without reliable integration with MySQL. We will also briefly look at the reCAPTCHA authentication on our sign-in page, which is another external aspect of our website that is integrated using Django Allauth. If this integration fails, it could lead to users losing access to their accounts or bots gaining access to users' accounts, making it essential that this feature works correctly. These two integrations connect with the project very differently, and as a result, our approach to each will be unique.

Our testing environment will be on the live deployment environment for the most recent pull from our GitHub repository. It will use Python's virtual environment module (venv) to create a Python 3.11 environment. From here, a local version of the site can be launched in mysite with the run python manage.py runserver command. From here, the tester can monitor their terminal for all test-relevant output. The tester will have to connect to their own MySQL account in the .env file, and will have to make their own student and mentor accounts for testing via the sign-in page of the site.

Scenarios

Integration Point: Frontend – Django Backend -- MySQL Database

Feature: Create New Article

Scenario Description: A user uploads an article to the website

Integration Steps:

- User submits the upload_article form in the frontend UI
- The frontend sends a POST request to the Django backend with the article data
- The Django backend (views.py) validates the article form data
- The Django ORM does an SQL insert query to save the article to the database
- Frontend redirects the user to the article details page for the new article

Expected Results:

- The SQL query is accepted
- The article is saved to the database
- The UI shows the new article's completed details page without errors

Failure Handling:

- Invalid input results in a clear error message

Integration Point: Frontend -- Django Allauth -- Google reCAPTCHA

Feature: Serve User a reCAPTCHA Check on Sign In

Scenario Description: A user signs into their account and is monitored for bot activity by reCAPTCHA

Integration Steps:

- User submits the sign-in form in the frontend UI
- The reCAPTCHA widget evaluates the user behavior and generates a verification token
- The frontend sends a POST request to the Django backend with the reCAPTCHA token and login credentials
- Django Allauth processes the login request and forwards the reCAPTCHA token to the Google reCAPTCHA verification API
- Google reCAPTCHA analyzes the token and returns a success or failure response
- If verification succeeds, the user is authenticated and redirected to the home page
- If verification fails, the sign-in request is rejected, and the user stays on the sign-in screen

Expected Results:

- The user is logged in and redirected to the home page if the reCAPTCHA returns a success response
- The sign-in request is rejected, and the user stays on the sign-in screen if the reCAPTCHA returns a failure response, with an error message

Failure Handling:

- The sign-in request is rejected, and the user stays on the sign-in screen if the reCAPTCHA returns a failure response, with an error message

Usability Testing

Usability testing evaluates how effectively users can interact with the website to do what they wish to do (achieve their intended tasks). Because our application is designed primarily as an interactive platform where researchers can upload articles, read and post peer reviews, and contribute their own feedback, usability plays a huge role in making sure our system is successful.

The goal of usability testing is to identify areas of the interface that may confuse users, slow down their ability to complete their task, or cause users to make mistakes. By observing how actual users interact with the system, our team can refine how our navigation works, improve user instructions, and make sure that important actions (uploading articles, submitting peer reviews, etc.) are straightforward. In other words, we can improve navigation, layout, and workflow.

Our target users are expected to include:

- Student Researchers (college students or new researchers)
- Mentors (faculty supervising the student researchers, or experienced researchers who want to submit their own reviews/browse articles)

These users may have varying levels of familiarity with academic publishing (from none to very experienced), so the interface needs to be super intuitive to accommodate all levels of experience.

Context and Assumptions

Our usability testing approach is primarily based on the expected characteristics of the system's target users. The primary users of the system are student researchers and academic mentors who participate in peer review and article sharing. These users tend to have moderate to high levels of computer literacy, but they may have varying levels of familiarity with research management tools or academic publishing platforms. Some users may already be familiar with citation managers or research repositories, while others may be encountering a system like this for the very first time. We need to be able to account for all levels of experience.

This platform will need to support academic works like article reviews and bibliography management, therefore, we need to have very clear navigation and a relatively low learning curve so that they can use it quickly and efficiently. Users should be able to quickly understand how to search for articles, submit peer reviews, and maintain their

bibliography without requiring extensive training. Poor usability could lead to several issues like:

- Users becoming confused when attempting to upload review articles
- Difficulty locating articles or peer reviews
- Mistakes when managing bibliography entries
- Frustration that discourages users from contributing to the platform

For these reasons, usability testing needs to focus (and will focus) heavily on evaluating how easy the website is to use and navigate. This will allow our team to make necessary changes to the overall UI for the sake of usability before we enter the final acceptance demo.

Usability Testing Methods

Usability testing will be conducted using a combination of task-based usability sessions, and informal feedback from project mentors. Task-based usability testing allows participants to attempt realistic tasks using the website while observers record their actions, difficulties, and feedback. This approach allows the development team to identify usability problems that may not be apparent during development.

Each usability session will involve participants completing a series of representative tasks while interacting with the system. The development team will observe whether participants can complete each task successfully and will record the time required to complete each task, any errors that occur, and any confusion or hesitation observed during interaction. Participants will also be encouraged to provide feedback about any parts of the interface that they find unclear or difficult to use.

In addition to these structured usability sessions, informal feedback will also be collected from project mentors and other reviewers during demonstrations. This feedback helps supplement the formal usability testing sessions and allows the team to identify additional usability concerns that may arise during normal use of the system.

Integration with Development Cycles

Usability testing will be integrated into the development process rather than performed as a singular activity. Initial usability testing will occur following the Alpha II demonstration, when the majority of the system’s features are available to be tested. At this stage, users will be able to interact with the system in a way that closely resembles the final product.

Findings from usability testing sessions will be documented and reviewed by the development team. Identified usability issues will be categorized based on their impact

on user experience. High-impact problems (navigation issues, unclear workflows, or a confusing interface, etc.) will be prioritized for resolution during the next development cycle. Lower-impact improvements may be added to a document, which lists future improvements.

Usability feedback may also continue to be collected during later demonstrations and prior to the final acceptance stage. By having usability testing as something ongoing, we can find more problems that are not immediately noticed and continuously improve the project as we move into completion.

Usability Test Summary

Users

- Student researchers who upload articles, write peer reviews, and manage personal bibliographies.
- Mentor who supervises student activity and reviews peer reviews.

Goals - Ensure that the system's core workflows are efficient, understandable, and easy to use. Usability testing focuses on verifying that users can:

- Navigate the website without confusion
- Search for and access articles easily
- Submit and edit peer reviews without errors
- Upload new articles successfully
- Add articles to their personal bibliography
- Add external references to their bibliography
- Locate previously submitted articles and reviews

Method - Short task-based usability testing sessions combined with informal feedback from project mentors and expert interface reviews.

Sessions - Planned sessions include:

- 10 student researchers
- 1 mentor

Tasks - Participants will attempt several representative tasks:

- Log in to the website and navigate to the homepage
- Search for an article using the article search feature
- Open and read an article
- Submit a peer review for an article
- Edit an existing peer review

- Upload a new article
- Add an internal article to a personal bibliography
- Add an external reference to a personal bibliography
- Navigate to previously submitted articles or reviews

Measures - During testing sessions, the following metrics will be collected:

- Task completion success rate
- Time required to complete each task
- Navigation errors or confusion points
- User comments and qualitative feedback

Follow-up: After each testing session, findings will be documented and reviewed by the development team. Usability issues will be prioritized based on severity and impact on user experience. High-priority usability issues will be addressed during the next development cycle, ensuring that usability testing contributes to continuous improvement of the system before the final acceptance demonstration.

Testing Workflow and Quality Controls

In order to maintain consistent software quality throughout the project, our team will follow a very clear process for identifying, reporting, prioritizing, and resolving defects discovered during testing. Because testing occurs continuously during development, it is important that issues discovered during unit testing, integration testing, and usability testing are all documented and very clear so that they can be both reproduced (for verification) and resolved.

Bug Reporting / Severity Levels

When a defect is discovered during testing, it will be documented using the team's GitHub issue tracker. Each bug report must contain enough information for all team members to be able to reproduce and diagnose the issue. Bug reports should include a clear description of the issue, steps required to reproduce the bug, the expected behavior of the system, the actual behavior (with the bug), any relevant screenshots/terminal output, and the steps the user takes that results in the problem. By documenting issues this way, the bugs can be reliably reproduced and fixed quickly.

Additionally, we will have four severity levels by which we will classify the bugs. Those being: critical (which includes things like system crashes, data loss, security problems, and users being unable to login or access the system), major (where features do not work), moderate (where features work, but not as intended), and minor (which includes things like UI changes and stuff that does not affect functionality).

Critical and major defects will be addressed immediately before further development continues, while moderate and minor issues may be scheduled for later fixes depending on development priorities.

Completion Criteria

Before the acceptance demo, all mentioned unit tests must pass successfully, integration tests must confirm that database interactions function correctly, the primary user features (uploading articles, etc.) function properly, and that there are no critical problems.

Minor user interface issues or cosmetic problems may remain if they do not affect feature functionality, but hopefully will be resolved. All bigger problems will be prioritized first. No critical or major defects can remain and everything must operate reliably.

Conclusion

Overall, this test plan outlines the strategy our team will use to make sure that the platform meets the functional and usability requirements defined in prior design and requirement documents. Because the system supports several features (like uploading articles, submitting peer review, searching for research content, and maintaining bibliographies), it is important that these features operate and are easy to interact with. Our testing strategy uses unit testing, integration testing, and usability testing. Unit testing focuses on verifying the functionality of individual components within our website. Integration testing verifies that the major components of our website (that being our backend, MySQL database, and the authentication services) all work together correctly when used in conjunction with actual user actions. Usability testing evaluates how effectively users can navigate the website and complete actual tasks, which will help us identify areas where the interface can be improved.

By using these three methods of testing, we cover multiple levels of the system (from individual components to full use processes). Identifying issues early and throughout development can help us reduce defects and overall improve the usability of the system. It also makes sure users can interact with the platform comfortably. Overall, this testing plan helps us ensure that the final system will be reliable and functional for both students and mentors.