Digital Declaration of Conformity Test Plan
Senior Project
Version 0.1 February 27th, 2023

By
Dominic Rohan
Sam Shadle
John Palladino
Mary Stempky

1. Introduction

Modern publishing is moving to a largely automated process. However, the Untied States Conference of Catholic Bishops (USCCB) still uses an primarily manual process which takes up to a year or more to complete depending on the document being reviewed for imprimatur approval; as such our team has chosen to assist in making an automated process available to the USCCB to streamline the imprimatur process.

This automated system will be a website from which publishers, reviewers, and Bishops may access documents for or under review. This website will allow documents to be uploaded (in various formats/only as one) by publishers and be downloadable by reviewers. Bishops will have the authority to assign submitted documents to reviewers. By automating this process, out team hopes to help create a digital imprimatur which helps the Catholic Church in the United States be able to spread the gospel more effectively.

Using this plan our team intends to test out each individual user interface. These interfaces include but are not limited to login screens for each unique user (publisher, reviewer, admin/bishop), the submit document page where publishers can upload documents for review, and the ability for a reviewer to open a submitted document to begin or continue a review. It will also cover the communication system with the database in use, where user data will be stored securely. Any and all third party applications and their applications services or components or any program interactions with the website will also be tested.

This test plan assumes the website runs on suitable hardware and software which is secure. Our team's testing will be limited by our access to any test data and resources such as personnel and testing tools. Time and budget limitations will affect testing.

2. Test Items

2.1 Description of the website

Our team has developed a website which allows publishers, USCCB reviewers, dioceses, and Bishops to make the imprimatur process more efficient and effective. Its main objective is to facilitate faster communication and reviewing so materials submitted to the USCCB may reach the wider Catholic Church as quickly as possible.

To achieve this, the website's frontend was developed using React and MUI, which are popular and widely used technologies for building dynamic user interfaces. The frontend communicates with the program's backend, which has been written in deno, through the use of REST calls. This ensures that the program operates seamlessly, with minimal latency or other operational glitches, thereby providing a superior user experience.

To maintain high levels of security, the website implements the use of PostgresSQL, a robust and scalable database, used to store and retrieve user data. The user data is encrypted and secured using appropriate security controls, ensuring that unauthorized access to the data is prevented. The encryption of the user data ensures that any sensitive information that the user inputs into the site remains safe from cyber-attacks or other forms of security breaches.

By prioritizing the security of user data, the website enables its users to confidently access the website and utilize its features, without concerns about the safety of their

data. This provides users with a seamless and highly secure experience, ensuring that the website meets and even exceeds the expectations of its users.

2.2 Testing approach and methodology

The testing approach will be a combination of manual and automated testing, using a range of testing techniques, including functional, usability, security, performance, compatibility, and regression testing. The testing methodology will follow the IEEE 829 standard for software testing.

2.3 Testing tools and equipment

The testing will use a range of testing tools and equipment, including web browsers, REST clients, load testing tools, security testing tools, and defect tracking tools. We will use the GitLab CI/CD pipeline to automatically run tests every time changes are merged into the ***develop*** branch. GitLab will provide components that will be used to stress test every component of the project. Testing will also be

2.4 Personnel requirements

The team will write all tests for all features. Each feature should have its own unique test written for it as it is added to the program. We will not have dedicated test quality engineers, but each developer will write tests for his code as it is being written.

Test Environment

3.1 Hardware and software requirements

3. The testing environment will require suitable hardware and software, including web servers, database servers, and testing tools. The environment should replicate the production environment as closely as possible. The test environment will be run using a cloud provider, possibly using Docker.

3.2 Test environment setup

The testing environment will be set up by the system administrators, following established procedures and guidelines. The testing environment should be isolated from the production environment to prevent data leaks or security breaches. The test environment will be run in a containerized environment, and can be setup and moved at any time

3.3 Test data requirements

The testing will require test data that replicates real-world scenarios, including user profiles, transactions, and data structures. The test data should be anonymized and secured using appropriate security controls. Tools will be written to get test data. Real life books will be used as test data for the approval process.

Test Categories

4.1 Functional testing

Functional testing will ensure that the program functions as intended, including validating inputs, displaying correct outputs, and handling errors appropriately. Functional testing will test the server and the client separately, and ensure for every input they give the correct output.

On the server end, functional testing will be split into two categories: REST and Database. Functional tests will be designed to ensure the server responds to REST requests correctly, and that the database stores and retrieves data from the database correctly.

When a pull request is opened on the *develop* branch, the server will be deployed to a testing environment, and all functional tests will be run. If any of the tests give a result different than is expected, the tests will be considered broken and the pull request will be denied.

4.2 Integration Testing

Integration testing will ensure the client, server, and database communicate and work together well. Integration testing will not be performed each pull request, as the components are not always expected to work together. Every two weeks, integration will be performed, and the components must be made to work together. Integration tests will be run until they all pass, and then the components will be considered integrated.

4.3 Unit Testing

Unit testing ensures each function works as it is meant to. Unit tests will be written for each function that performs a significant role in the program. A number of different cases should be written for each test, and the output of the function should be validated against what is expected. Unit tests are expected to pass when a pull request is made, and the CI/CD pipelines will fail if they do not.

4.4 Usability testing

Usability testing will occur when the program gets to a usable state. Usability testing should happen before the program is ready to be released. Usability testing is not automated, but is done by representative users. The pre-release program will be tested by representative users from each of the user classes, and they will give feedback on how usable it is. If the usability is rated a 4 out of 5 stars or better on average, the program will have passed usability testing.

4.5 Security testing

As our program deals with sensitive data, security is of the utmost concern. Security testing should be performed to ensure that all clients data remains safe and secure. Security testing will take the form of running some of the most common attacks in a development environment. We will test the security of our application through the following means:

- SQL Injection Attacks
- DDOS Attacks
- Attempt to access files we don’t have permission to access
- Attempt to change users password

Security testing will take place before a release version is created.

4.6 Performance testing

Performance is not the highest concern for our program, but it should be brought into consideration. Performance will be tested when a pre-release version is in testing. Performance testing will be done in a development containerized environment. Performance testing will

consist of endurance testing, load testing, and stress testing. The server will be bombarded with requests, and we will plot how responsive it is against the frequency of requests. If it is overloaded easily enough, then the test will be considered fails.

4.7 Compatibility testing

Our clients will most likely be using browsers as old as internet explorer, so it is very important to ensure compatibility with older browsers. Our compatibility tests will attempt to render the webpage on a number of different web environments, and will pass if it renders well.

4.8 Regression testing

Regression testing is a software testing technique that is used to verify whether modifications to an existing software application or system have caused any unintended changes or introduced any new defects. The goal of regression testing is to ensure that the changes made to the software do not negatively impact its existing functionality, stability, and performance.

We will perform regression testing with all of our tests. The output of the tests shouldn't change between commits.

Test Deliverables

5.1 Test plan document

This document outlines the approach, methodology, and scope of the testing effort.

5.2 Test cases document

This document provides detailed test cases and expected results for each testing category.

5.3 Test logs and reports

This document provides a record of test execution, including results, defects, and recommendations for improvements.

Testing Schedule

6.1 Test timeline

The testing timeline will be based on the project timeline and will be updated regularly to reflect any changes in the project schedule.

6.2 Test phases

The testing will be conducted in phases, starting with unit testing and progressing to system testing, integration testing, and acceptance testing.

Testing Risks and Contingencies

7.1 Risks

Potential risks include delays in test data availability, changes to project scope or requirements, and defects discovered late in the testing process.

7.2 Contingencies

Contingencies include having backup test data, incorporating changes to project scope or requirements into the testing plan, and allocating extra resources to address defects discovered late in the testing process.

Approvals

8.1 Test plan approval

This test plan requires approval by Prof. James Wessel and Mr. Jon Crumpacker

8.2 Test results approval

Test results will be reviewed and approved by Prof. James Wessel and Mr. Jon Crumpacker.

9. Test Cases

9.1 General Tests

- IF the server is in use,THEN Node.js should be automatically updating the database ✓
- Ensure server-client connections are operable ✓
- IF the server-database connection is initiated, THEN the system should encrypt ✓
- information sent from server to the database✓
- IF data is entered into the field, THEN the server should connect to the database to verify information is correct.✓
- IF there is an error in the data input, THEN the system should display the proper error code✓
- IF a new user attempts login, THEN the system should prompt account creation. ✓
- IF a new account creation is attempted, THEN the server should verify all information is valid (will require integration with email platforms to verify email exists). ✓
- IF a returning user logs in, THEN the system should verify all information is correct ✓
- If a publisher attempts to submit a document, THEN the system should verify the publisher has permissions to create a submission✓
- If the publisher does not yet have submission creation permissions, THEN the correct error code should appear ✓
- IF a user is logged in, THEN the system should show them the pertinent lists (documents, feedback, ect). ✓
- IF a user choose to read a piece of date (document, feedback ect), THEN the system should load the appropriate data ✓
- IF a user chooses to upload at data point (document, feedback, ect.) THEN the system should immediately begin the upload, If an error occurs, the proper error code should appear.✓
- IF a user chooses to retract a piece of data (document, feedback, ect), THEN the system should retract/delete/stop upload of said data point.✓

9.2 Publisher View Tests
9.2.1 Sequence

- If the users tries to upload a document,
    - Then the screen should prompt them to upload or drag a file ✓
- IF a user selects upload a file,
    - Then they should be prompted to select a file from their computer or online file storage service ✓
- IF the user selects a file,
    - Then the system should confirm this is the file the users wishes to upload✓
- IF the user selects a file,
    - Then the system should verify it's a valid file type (for this test determine the file types which will be accepted by the system) ✓
- IF the system verifies the document as legitimate,
    - THEN it should be begin uploading file ✓
- IF the system begins an upload,
    - THEN the system should show upload progress, with an option to cancel the upload attempt ✓
- IF the presses cancel,
    - the system should automatically stop the upload. ☐

9.2.2 Requirement tests

- IF the user presses 'begin document upload",
    - THEN screen should prompt them to select a file for upload from their computer or cloud service or drag and drop one from one of these two locations ☐
- IF the file is dragged and dropped,
    - THEN screen should recognize the actions taking place and confirm the drop of the file. ☐
- If the file is selected or dragged and dropped,
    - THEN the system should verify it is a valid file type.☐
- IF the file is a valid type,
    - THEN upload should begin ☐
- IF the file is not a valid file type,
    - THEN an error message should appear which states this ☐
- IF the file begins,
    - THEN upload progress should be visible ☐
- IF the upload progress is visible,
    - THEN the user should also be able to view a "cancel upload" option ☐
- IF the user chooses the cancel upload option,
    - THEN the system should immediately stop the upload progress. ☐

9.3 Reviewer: Download Submitted Documents Tests

9.3.1 Sequence

- IF reviewer chooses to see incoming documents,
  - THEN they should be able to view them ☐
- IF reviewer selects download option,
  - THEN the system should prompt them to select download format (PDF or Word Doc) ☐
- IF the reviewer selects the download document as a pdf,
  - THEN the document should open in another tab in the browser. ☐
- If the reviewer decides to download as a doc file,
  - THEN the system should automatically open the document in word (this may be a down the road test and not in this stage of development) ☐

9.3.2 Functional Requirements

- IF the reviewer selects to see incoming documents for review,
  - THEN the system should load all relevant document files ☐
- IF reviewer chooses to download the document,
  - THEN it should download in the upload format ☐
- If the uploaded document is a pdf and reviewer selects to download it,
  - THEN the document should appear in a new tab as an editable pdf ☐
- If the uploaded document is a word doc or similar doc file,
  - THEN it should automatically open this external application for editing ☐
- If the reviewer selects to convert the file into another file type,
  - THEN the system should begin document conversion. ☐

9.4 Reviewer view: View Submitted Document on the Web

9.4.1 Sequence tests

- If the reviewer selects to view incoming documents,
  - THEN all incoming documents should load on the screen ☐
- IF a reviewer selects to view an incoming document from the list,
  - THEN system should navigate to the document viewer (if it does not an error message should appear to indicate and issue with the viewer) ☐
- IF the document viewer does open,
  - THEN it ;should load the selected document. An error message should appear to indicate inability to load if it does not happen. ☐
- IF the document loads,
  - THEN the viewer should be operational for use (each tool in the viewer should be able to be used by the reviewer) ☐

9.4.2 Functional requirements

- IF the documents are loaded,
    - THEN they should be able to be displayed in different formats (pdf, doc. ect.) ☐
- If the document is loaded in a specific format (i.e. pdf, word doc, ect.),
    - THEN the system should provide easy navigation between sections of the particular document ☐
- IF the reviewer closes the document before reaching its end,
    - THEN the system should save the reviewer's location in the document for the next session of edits. ☐

9.5 Reviewer view: Leave Comments on the document

- If the reviewer opens a document in the viewer,
    - THEN the system should load the document for the reviewer. ☐
- If the reviewer chooses a piece of text to annotate,
    - THEN the system should provide a text comment box for the reviewer to use. ☐
- If the reviewer selects to publish the comment,
    - THEN the system should save the comment on the document to be viewable by the publisher/author. ☐

9.6 Publisher: Contact reviewer(s)

9.6.1 Stimulus/Response Sequence Tests

- If a publisher selects a document,
    - THEN the system should load the document. ☐
- If the publisher selects to see reviewers contact information,
    - THEN the system should load the reviewer's contact information. ☐

9.7 Reviewer: Track where the document is in the review process

9.7.1 Stimulus/Response Sequence tests

- IF the user selects an open document,
    - THEN the systems should load the status of the document on screen ☐
- If the User views the status of the document,
    - THEN the system should show the details of when each step was completed. ☐

9.7.2 Functional Requirements Tests

- If the user chooses to view the list of all documents in progress,
    - THEN the system should load the list ☐
- If a publisher/reviewer loads a list of documents in progress,
    - THEN the system should allow the publisher/reviewer to view next steps for each document in progress. ☐
- If a publisher/reviewer loads a list of documents in progress,

    - THEN the system system should allow him or her to view a list of only approved/denied documents. ☐
- If a publisher/reviewer selects to see a list of only approved/denied documents,
    - THEN the system should load the selected list of approved/denied documents ☐

10. Summary

TBD

Glossary

9. This section defines key terms used in this document.

References

This section lists any references used in the creation of this test plan.

This IEEE test plan provides a comprehensive approach to testing a secure website written in React, MUI, and nodejs. By following this plan, the testing team can ensure that the website functions as intended, is secure, performs well under load, is usable, and is compatible with a range of devices and platforms.