System Verification and Validation Plan for Software Engineering

Team #11, OKKM Insights
Mathew Petronilho
Oleg Glotov
Kyle McMaster
Kartik Chaudhari

November 7, 2024

Revision History

Date	Version	Notes
11/04/2024	Oleg, Kyle, Kartik, Mathew	Revision 0

Contents

1 Symbols, Abbreviations, and Acronyms

Please refer to Section 4 of Software Requirements Specification found here.

This document is intended to provide a description of the specific validation and verification activites that will be completed throughout the development of the Orbit Watch System. The purpose of these activities is to ensure the system requirements agree with stakeholder needs, and to certify the implementation of the system satisifes the stakeholder requirements. This document will start with a review of the system, a description of the testing plan, and a list of tests to be completed.

2 2 General Information

2.1 2.1 Summary

[4]

Orbit Watch is an online platform developed by OKKM Insights to streamline and simplify satellite imagery data analysis. The software addresses the scarcity of high-quality, labeled satellite imagery datasets tailored for specific use cases across various industries such as disaster response, environmental monitoring, urban planning, and defense. Leveraging an AI-powered crowdsourcing model, Orbit Watch enables users to label commercially available satellite images. These labeled datasets are then utilized to train custom computer vision models, enhancing accuracy and efficiency in image analysis. The platform offers a paid service for identifying objects within satellite images and distributes earnings to users who contribute to the labeling efforts.

2.2 2.2 Objectives

[4]

The primary objectives of the Verification and Validation (V&V) plan for Orbit Watch are:

- **Ensure High Data Accuracy**: Build confidence in the software's correctness by verifying that the system achieves high classification accuracy for objects reported in the images. This is crucial for extracting useful information and providing value to stakeholders.
- **Validate System Reliability and Accessibility**: Confirm that the system is reliable with minimal downtime and is accessible remotely

for both purchasers and labelers, ensuring flexibility and efficiency in usage.

The following objectives are out of scope for this V&V plan:

- **Verification of External Components**: We will not verify external libraries or services used for image acquisition, payment processing, or AI algorithms. It is assumed that these components have been adequately verified by their respective development teams.
- **Stretch Goals**: Objectives related to automatic data labeling and multi-source integration are considered stretch goals and will not be included in the current V&V efforts due to time and resource constraints.

2.3 2.3 Challenge Level and Extras

[1] [4]

We aim to classify this project as an advanced level project.

This project can be classified at the **advanced** challenge level due to several factors:

- **Limited Domain Knowledge**: The team has minimal experience with satellite imagery and computer vision models, requiring additional research and learning to implement effective solutions.
- **Complex Implementation**: Developing a web application from scratch involves challenges in front-end development, integrating secure payment systems, handling parallel image labeling tasks, and designing consensus algorithms for data accuracy.
- **Integration of Multiple Components**: The project demands seamless integration of crowd-sourcing models, AI algorithms, payment processing, and user interfaces, increasing the complexity of development and testing.

Extras from the Problem Statement [4]

- **Usability Testing**: This phase involves selecting a group of users, ideally from the target demographic (e.g., professionals in satellite imagery analysis or end-users in agriculture or rescue services). These users interact with the application's interface, performing tasks relevant to the platform's purpose, like labeling images or reviewing AI-generated annotations. They then provide feedback via a structured questionnaire focusing on usability aspects—such as ease of use, intuitiveness, and clarity. This feedback helps refine the user interface to ensure it meets user expectations and requirements.

- **Demonstration Video**: The video will serve as an onboarding tool, guiding users on how to navigate and utilize the platform effectively. It will highlight key features, demonstrate a labeling task from start to finish, and showcase any unique functionalities, such as model-assisted annotations or accuracy checks. The video can be used in training materials or as a quick tutorial accessible directly from the platform to help new users become familiar with the tool's workflow.

- **Formal Proof of Convergence for Labeled Images**: This proof aims to establish that labeled images converge toward a standard of accuracy and consistency. This involves statistical or mathematical validation showing that repeated labeling by different users (or with the assistance of AI) leads to a stable and reliable dataset over time. Proving convergence is crucial in ensuring that the labeling quality remains high, even as new users interact with the platform. This might include metrics such as inter-annotator agreement or AI confidence scores, contributing to the credibility and reliability of the labeled dataset.

2.4 2.4 Relevant Documentation

Created BibTeX entries for our documents and within those entries we have included a hyperlink to the documents. As listed above. Here is a sample of the references for all of the documents. Future references are also included in this sample.

- Development Plan - [1]

- Module Guide - [2]
- Module Interface Specification - [3]
- Verification and Validation Plan - [6]
- Problem Statement - [4]
- Software Requirements Specification - [5]

By referencing these documents, we ensure that our V&V efforts are comprehensive and aligned with the project's goals and requirements. Each document provides specific insights that help us design effective verification and validation activities:

- **Problem Statement** ([4]): Outlines the core challenges and objectives of the Orbit Watch project, providing context and motivation for development. It helps align our V&V activities with the original goals and ensures that the software addresses the identified needs.
- **Development Plan** ([1]): Details the project timeline, milestones, and resource allocation. It assists in scheduling V&V activities appropriately within the development lifecycle, ensuring they are integrated at optimal points for maximum effectiveness.
- **Software Requirements Specification (SRS)** ([5]): Defines all functional and non-functional requirements of Orbit Watch. It serves as the foundation for developing test cases to verify that the software meets all specified requirements and performs as intended.
- **Module Guide (MG)** ([2]): Provides an overview of the system's modular architecture, including the decomposition into modules and their relationships. This information is vital for planning integration testing and ensuring that the modules interact correctly.
- **Module Interface Specification (MIS)** ([3]): Details the interfaces for each module, specifying input and output parameters. This is essential for verifying that modules communicate correctly and adhere to their specified interfaces, which is critical for system reliability.

- **Verification and Validation Plan (V&V Plan)** ([6]): Outlines the strategies, methodologies, and resources allocated for the V&V activities. It guides our testing efforts and ensures that all aspects of the system are adequately verified and validated according to the project's objectives.

By utilizing these documents, our V&V efforts are thorough and focused on delivering a reliable and accurate platform for satellite imagery analysis.

3 Plan

This section describes the team, verification plans for the system design and documentation, use of automated testing tools, and the validation plan of the software following implementation.

3.1 Verification and Validation Team

When not listed as lead, members of core team will support through team discussion and implementation of feedback.

Name	Role	Responsibilities
Mathew Petronilho	Core team	**SRS** Lead review of SRS **Implementation** Support review of implementation Review Implementation for coding standards, comment quality
Oleg Glotov	Core team	**VnV Plan** Support review of VnV Plan Review VnV Plan for formatting and grammar errors **Implementation** Lead review of Implementation
Kartik Chaudhari	Core team	**Design** Support review of Design Review diagrams and documents for correct notation, formatting and grammar **VnV Plan** Lead review of VnV Plan **Implementation** Support review of implementation
Kyle McMaster	Core team	**SRS** Check SRS for formatting and grammar errors Support review of SRS **Design** Lead review of design Ensure team is following standard design principles
Dr. Swati Mishra	Project Supervisor	Validate all team docs in structured review
Capstone Team 10	Primary Reviewers	Validate all team docs in async review through Git issues

3.2 SRS Verification Plan

The SRS will be verified through several channels. First, the document will be reviewed by another capstone team. This will help the team identify issues which are obscured to the team, due to the additional time they have spent thinking about the project. We expect this feedback to generally con-

sist of omitted definitions or unstated assumptions. Since the team is more familiar with the project, it is likely that we have some information which is obvious to the team, but is necessary to define for others. The team will collect feedback from our primary reviewers in the issue tracker.
The SRS will also be reviewed by our supervisor in a structured review meeting. The team will guide the review with the following checklist:

Constraints & Assumptions
The following sections are informed by the constraints & assumptions. Therefore, it is critical that the these are verified first.

- ☐ All constraints are correct and necessary
- ☐ All constraints are unambigious
- ☐ All constraints which should be present are present
- ☐ All constraints are verifiable
- ☐ All assumptions are correct and necessary
- ☐ All assumptions are unambigious
- ☐ All assumptions which should be present are present
- ☐ All assumptions are verifiable

Data Model
The data model affects how the system will be decomposed in the future design. This in turn affects the requirements, so it should be verified next.

- ☐ Data model is correct
- ☐ Data model is complete
- ☐ Each element of data dictionary is correctly described. There are no extra or missing attributes.
- ☐ Elements of data dictionary are unambigious

Functional Requirements
This section describes the functionality of the system. This will be useful for understanding the context of the NFRs.

- ☐ All requirements are correct and necessary
- ☐ All requirements are unambigious
- ☐ All requirements which should be present are present
- ☐ All requirements are verifiable
- ☐ All requirements are feasible
- ☐ All requirements are traceable

Non-Functional Requirements
The NFRs have now been properly introduced by the other sections and should now be assessed.

- ☐ All requirements are correct and necessary
- ☐ All requirements are unambigious
- ☐ All requirements which should be present are present
- ☐ All requirements are verifiable
- ☐ All requirements are feasible
- ☐ All requirements are traceable

General
Each of the sections reviewed should also be monitored for the following criteria.

- ☐ Document contains properly formatted title, table of contents, references and all necessary sections
- ☐ Tables and figures are correctly formatted
- ☐ No grammar errors are present
- ☐ Each required section is present
- ☐ All requirements are feasible

3.3 Design Verification Plan

Upon completion of the System Design, the team will verify the design. This verification will involve reviews from the primary reviewers, in the same manner as described above. This again will provide fresh prospective on the design, and help identify any omitted or underexplained information. The verification will continue with a systematic review with The project supervisor using the following checklist:

Class Decomposition

- ☐ All functional requirements are covered by a class/subsystem
- ☐ All non-functional requirements are covered by a class/subsystem
- ☐ All classes are correctly sized. That is, each class should be concerned with one purpose
- ☐ When appropriate, interfaces and abstract classes are used
- ☐ All class attributes are complete and necessary
- ☐ When appropriate, design patterns are used to improve design clarity
- ☐ Function permissions are appropriate for all classes
- ☐ Classes cannot be simplified without degrading the understanability of the system

General

- ☐ The types of all attributes are listed
- ☐ The argument and return types of all functions are listed
- ☐ Correct notation is used to describe class relationships
- ☐ Class diagram is legible

3.4 Verification and Validation Plan Verification Plan

This document must be verified and validated, to ensure the validation and verification of other artifacts is correct. Like other documents, this will include review from our primary reviewers, as well as a review with the project supervisor. The review with the project supervisor will be guided by the folllowing checklist:

Testing Plan

- ☐ Testing plan includes a checklist for each artifact
- ☐ Checklist items are unambigious
- ☐ Checklist contains all recessary items
- ☐ Checklist does not contain unnecessary items
- ☐ Plan includes review from multiple parties
- ☐ Plan includes method for collecting feedback on artifacts
- ☐ Plan is verifiable

Tests

- ☐ All functional requirements are covered by one or more tests
- ☐ All non-functional requirements are covered by one or more tests
- ☐ All 'units' are sufficiently covered by one or more tests
- ☐ Tests are unambigious
- ☐ Tests are verifiable
- ☐ Tests are repeatable

3.5 Implementation Verification Plan

As part of the implementation verification plan, the team will conduct a series of static tests. One of which, will be a code walkthrough with members of the team. Before submitting Rev 0 and Rev 1, the members of the team will schedule time to walk through other members of the team through the code they have written. This will be an opportunity for the team to look for code quality and adherence to standards, correctness, and alignment to the system design. During this walkthrough, we expect that the lead developer of the code being displayed will explain the overall flow and control structures. In doing so, we expect they will find the errors in their own code.

The team will also use a suite of tests, which is described in section 4.

The team will used static analyzers to verify adherence to coding standards to perform type checking. Please refer to section 7.2.2 and 7.2.3 in the Development Plan found here.

3.6 Automated Testing and Verification Tools

Please refer to section 7.2.2 and 7.2.3 in the Development Plan found here.

3.7 Software Validation Plan

As discussed in section 6.2 of the Problem Statement and Goals, found here, we will validate our software through user testing. More specifically, we will conduct testing of the user interface with peers, who will fill in a survey containing both qualitative and quantitative questions. This survery will be used to track progress on the usability of the software by comparing results collecting during different iterations of the user interface. See the appendix for a sample of the user survey.

4 System Tests

4.1 Tests for Functional Requirements

FR0: Customer Account Creation Test

1. **Test ID**: T-FR0
2. **Control**: **Automated**
3. **Initial State**: The system is accessible, and no user account currently exists for the test user.
4. **Input**:
   - Customer provides valid personal information:
     - **Name**: "Alice Smith"
     - **Email**: `alice.smith@example.com`
     - **Password**: "StrongPassword!2021"
   - Customer agrees to the system's privacy policy.
5. **Output**:
   - **Expected Result**:
     - A new user account is created.
     - Account information is securely stored in the database.
     - The customer is redirected to the login page with a success message.
6. **Test Case Derivation**:
   - Based on the requirement that customers must have an account to access system services. This test verifies that with valid input and acceptance of terms, the account creation process is successfully executed and persists data appropriately.
7. **How Test Will Be Performed**:
   - **Automated Test Script Execution**:

(a) **Step 1**: Navigate to the account creation page using a testing tool like Selenium WebDriver.
(b) **Step 2**: Fill in the registration form fields with the provided valid data.
(c) **Step 3**: Check the agreement box for the privacy policy.
(d) **Step 4**: Submit the registration form.
(e) **Step 5**: Verify that the response indicates successful account creation (e.g., success message displayed).
(f) **Step 6**: Confirm redirection to the login page.
(g) **Step 7**: Query the database to verify that a new user record exists with the email `alice.smith@example.com`.
(h) **Step 8**: Ensure the password is hashed and not stored in plain text.

- **Cleanup**:
  - After verification, delete the test user account from the database to maintain a clean state for future tests.

FR1: Customer Authentication Test

1. **Test ID**: T-FR1
2. **Control**: **Automated**
3. **Initial State**: An existing user account with the following credentials:
   - **Email**: `user.test@example.com`
   - **Password**: "TestPass#123"
4. **Input**:
   - Customer enters the correct login credentials:
     - **Email**: `user.test@example.com`
     - **Password**: "TestPass#123"
5. **Output**:
   - **Expected Result**:

- Customer is successfully authenticated.
- Access to privileged information (e.g., user dashboard) is granted.
- A session token or cookie is established for the user session.

6. **Test Case Derivation**:
   - Ensures that only authenticated customers can access the system, satisfying security requirements.
7. **How Test Will Be Performed**:
   - **Automated Test Script Execution**:
     (a) **Step 1**: Navigate to the login page.
     (b) **Step 2**: Input the correct email and password.
     (c) **Step 3**: Submit the login form.
     (d) **Step 4**: Verify redirection to the user dashboard or home page.
     (e) **Step 5**: Check for the presence of privileged information on the page.
     (f) **Step 6**: Verify that a session token or authentication cookie has been set.
     (g) **Negative Test**:
       i. **Step 7**: Attempt to access the dashboard with incorrect credentials to ensure access is denied.
   - **Session Validation**:
     (a) **Step 8**: Use the session token to access a secure API endpoint to confirm authentication persistence.

FR2: Customer Account Modification Test

1. **Test ID**: T-FR2
2. **Control**: **Manual**
3. **Initial State**: Customer is logged in and has access to their account information page.

4. **Input**:

   - Customer updates personal information fields:
     - **Address**: "123 New Street, Cityville"
     - **Phone Number**: "555-1234"
     - **Profile Picture**: Uploads a new image file.

5. **Output**:

   - **Expected Result**:
     - Updated personal information is saved and reflected in the database.
     - The user receives a confirmation message indicating successful update.

6. **Test Case Derivation**:

   - Confirms that authenticated customers can modify their information, maintaining data accuracy and relevance.

7. **How Test Will Be Performed**:

   - **Manual Steps**:
     (a) **Step 1**: Log in to the customer account.
     (b) **Step 2**: Navigate to the account settings or profile page.
     (c) **Step 3**: Change the address and phone number to the new values.
     (d) **Step 4**: Upload a new profile picture.
     (e) **Step 5**: Save the changes.
     (f) **Step 6**: Verify that the changes are displayed correctly on the profile page.
     (g) **Step 7**: Check the database to ensure the new information is updated.
     (h) **Step 8**: Log out and log back in to confirm that the changes persist.

FR3: Payment Processing Test

1. **Test ID**: T-FR3
2. **Control**: **Automated**
3. **Initial State**: Customer is logged in and ready to make a payment for a service request.
4. **Input**:
   - Valid payment information:
     - **Credit Card Number**: "4111 1111 1111 1111" (Test Visa number)
     - **Expiry Date**: "12/25"
     - **CVV**: "123"
     - **Billing Address**: Matches the address on file.
5. **Output**:
   - **Expected Result**:
     - Payment is processed successfully.
     - A confirmation receipt is generated and emailed to the customer.
     - The service request status is updated to "Paid" or equivalent.
6. **Test Case Derivation**:
   - Verifies payment integration, ensuring that authenticated customers can complete transactions securely.
7. **How Test Will Be Performed**:
   - **Automated Test Script Execution**:
     (a) **Step 1**: Navigate to the payment page for the pending service request.
     (b) **Step 2**: Input the valid payment details.
     (c) **Step 3**: Submit the payment form.

(d) **Step 4**: Mock the payment gateway response if using a sandbox environment.

(e) **Step 5**: Verify that the system displays a payment success message.

(f) **Step 6**: Check that a confirmation receipt is generated and sent to the customer's email.

(g) **Step 7**: Verify that the service request status is updated appropriately in the database.

- **Security Verification**:

  (a) **Step 8**: Ensure that sensitive payment information is not stored in plain text and complies with PCI DSS standards.

FR4: Service Request Submission Test

1. **Test ID**: T-FR4
2. **Control**: **Manual**
3. **Initial State**: Customer is logged in and has a confirmed payment.
4. **Input**:
   - Customer fills out the service request form with necessary details:
     - **Service Type**: "Image Analysis"
     - **Description**: "Analysis of satellite images for deforestation."
     - **Preferred Completion Date**: "2023-12-31"
5. **Output**:
   - **Expected Result**:
     - Service request is accepted and logged in the system.
     - Customer receives a confirmation message and request ID.
6. **Test Case Derivation**:
   - Ensures that the system accepts valid requests from authenticated customers.

7. **How Test Will Be Performed**:

    - **Manual Steps**:
        (a) **Step 1**: Navigate to the new service request page.
        (b) **Step 2**: Fill in the form with the input data.
        (c) **Step 3**: Submit the form.
        (d) **Step 4**: Verify that a confirmation message with a unique request ID is displayed.
        (e) **Step 5**: Check the database to confirm that the service request is logged with correct details.
        (f) **Step 6**: Ensure that the service request appears in the customer's list of active requests.

FR5: Service Report Delivery Test

1. **Test ID**: T-FR5
2. **Control**: **Automated**
3. **Initial State**: Customer is logged in and has a completed service request.
4. **Input**:
    - Customer navigates to the "My Reports" section after being notified of service completion.
5. **Output**:
    - **Expected Result**:
        – The service report is available for viewing and download.
        – Report contents are accurate and correspond to the service request.
6. **Test Case Derivation**:
    - Confirms that customers receive reports on completed services, fulfilling the system's purpose.

7. **How Test Will Be Performed**:

    - **Automated Test Script Execution**:
        (a) **Step 1**: Simulate the completion of a service request in the system (can be mocked or set up in a test environment).
        (b) **Step 2**: Log in as the customer.
        (c) **Step 3**: Navigate to the "My Reports" or equivalent section.
        (d) **Step 4**: Verify that the completed service report is listed.
        (e) **Step 5**: Open the report and check that it loads correctly.
        (f) **Step 6**: Validate that the report content matches the expected output based on the service provided.
        (g) **Step 7**: Attempt to download the report and ensure the file is intact and accessible.

FR6: Image Upload Test

1. **Test ID**: T-FR6
2. **Control**: **Manual**
3. **Initial State**: Customer is logged in with an active service request requiring image uploads.
4. **Input**:
    - Customer uploads multiple image files:
        – `Image1.jpg`: 2 MB
        – `Image2.png`: 3 MB
        – `Image3.tif`: 5 MB
5. **Output**:
    - **Expected Result**:
        – All images are successfully uploaded and stored.
        – Images are correctly linked to the specific service request.
        – Customer receives an upload success message.

6. **Test Case Derivation**:

    - Ensures that customers can upload images for requested services, fulfilling the fit criterion.

7. **How Test Will Be Performed**:

    - **Manual Steps**:
        (a) **Step 1**: Navigate to the image upload section of the active service request.
        (b) **Step 2**: Select the image files for upload.
        (c) **Step 3**: Initiate the upload process.
        (d) **Step 4**: Monitor progress indicators for each file.
        (e) **Step 5**: Verify that a success message is displayed after upload completion.
        (f) **Step 6**: Check the service request details to ensure images are listed.
        (g) **Step 7**: Confirm that the images are stored in the correct directory or database location.

FR7: Satellite Image Request Test

1. **Test ID**: T-FR7

2. **Control**: **Automated**

3. **Initial State**: Customer is logged in with an active service request that requires satellite images.

4. **Input**:

    - Geographic coordinates:
        – **Latitude**: 37.7749° N
        – **Longitude**: 122.4194° W (San Francisco, CA)
        – **Date Range**: “2023-01-01” to “2023-01-31”

5. **Output**:

- **Expected Result**:
  - The system retrieves and stores satellite images corresponding to the provided coordinates and date range.
  - Customer is notified of successful retrieval.

6. **Test Case Derivation**:

   - Confirms the system can source images using specified geographical data, aiding in analysis.

7. **How Test Will Be Performed**:

   - **Automated Test Script Execution**:
     (a) **Step 1**: Input the geographic coordinates and date range into the request form.
     (b) **Step 2**: Submit the request.
     (c) **Step 3**: Mock the satellite data provider's API response if necessary.
     (d) **Step 4**: Verify that the system processes the input without errors.
     (e) **Step 5**: Check that the images are retrieved and stored in the system.
     (f) **Step 6**: Confirm that the images are linked to the correct service request.
     (g) **Step 7**: Validate that the customer receives a notification or confirmation message.

FR8: Service Request Failure Alert Test

1. **Test ID**: T-FR8

2. **Control**: **Manual**

3. **Initial State**: Customer has initiated a service request that cannot be fulfilled due to invalid parameters.

4. **Input**:

- Service request with unfulfillable criteria:
  - **Service Type**: "Image Analysis"
  - **Geographic Coordinates**: Invalid coordinates (e.g., Latitude: 95° N)

5. **Output**:
   - **Expected Result**:
     - Customer receives an alert indicating that the service request cannot be processed.
     - An explanation of the failure is provided.
6. **Test Case Derivation**:
   - Ensures customers are promptly notified when requests cannot be processed, enhancing user experience.
7. **How Test Will Be Performed**:
   - **Manual Steps**:
     (a) **Step 1**: Attempt to submit the service request with invalid coordinates.
     (b) **Step 2**: Observe the system's response.
     (c) **Step 3**: Verify that an alert or error message is displayed to the customer.
     (d) **Step 4**: Ensure the message clearly explains the reason for failure.
     (e) **Step 5**: Check that no service request is logged in the system for the invalid input.

FR9: Labeler Account Creation Test

1. **Test ID**: T-FR9
2. **Control**: **Automated**
3. **Initial State**: No labeler account exists for the test user in the system.

4. **Input**:
    - Labeler provides required account information:
        - **Name**: "Bob Labeler"
        - **Email**: `bob.labeler@example.com`
        - **Password**: "LabelerPass789!"
        - **Expertise Area**: "Satellite Image Annotation"
5. **Output**:
    - **Expected Result**:
        - A new labeler account is created and securely stored.
        - Labeler is prompted to complete any additional onboarding steps.
6. **Test Case Derivation**:
    - Ensures labelers can create accounts to access the system, which is essential for workflow.
7. **How Test Will Be Performed**:
    - **Automated Test Script Execution**:
        (a) **Step 1**: Navigate to the labeler registration page.
        (b) **Step 2**: Fill in the registration form with the input data.
        (c) **Step 3**: Submit the form.
        (d) **Step 4**: Verify that a success message is displayed.
        (e) **Step 5**: Check the database to ensure the new labeler account exists with the correct details.
        (f) **Step 6**: Confirm that the password is stored securely (hashed).
    - **Cleanup**:
        - After verification, delete the test labeler account from the database to maintain a clean state for future tests.

FR10: Labeler Authentication Test

1. **Test ID**: T-FR10
2. **Control**: **Automated**
3. **Initial State**: Labeler account exists with credentials:
   - **Email**: `labeler.test@example.com`
   - **Password**: "LabelerSecure!2022"
4. **Input**:
   - Labeler enters correct login credentials.
5. **Output**:
   - **Expected Result**:
     - Labeler is authenticated successfully.
     - Access to the labeler dashboard is granted.
6. **Test Case Derivation**:
   - Confirms authentication mechanisms for labelers, maintaining system security.
7. **How Test Will Be Performed**:
   - **Automated Test Script Execution**:
     (a) **Step 1**: Navigate to the labeler login page.
     (b) **Step 2**: Input the correct credentials.
     (c) **Step 3**: Submit the login form.
     (d) **Step 4**: Verify redirection to the labeler dashboard.
     (e) **Step 5**: Check for access to labeler-specific features and data.
     (f) **Negative Test**:
         i. **Step 6**: Attempt login with incorrect credentials to ensure authentication fails appropriately.

FR11: Labeler Account Modification Test

1. **Test ID**: T-FR11
2. **Control**: **Manual**
3. **Initial State**: Labeler is logged in and on the account settings page.
4. **Input**:
   - Update personal information:
     - **Expertise Area**: Add "Aerial Photography Annotation"
     - **Contact Number**: "555-6789"
5. **Output**:
   - **Expected Result**:
     - Personal information is updated and stored in the database.
     - Labeler receives a confirmation of successful update.
6. **Test Case Derivation**:
   - Ensures labelers can maintain current information, which is crucial for assignment matching.
7. **How Test Will Be Performed**:
   - **Manual Steps**:
     (a) **Step 1**: Navigate to account settings.
     (b) **Step 2**: Modify the expertise area and contact number.
     (c) **Step 3**: Save the changes.
     (d) **Step 4**: Verify that the updated information is displayed.
     (e) **Step 5**: Check the database for updated records.
     (f) **Step 6**: Log out and log back in to confirm persistence.

FR12: Labeler Earnings Transfer Test

1. **Test ID**: T-FR12
2. **Control**: **Automated**
3. **Initial State**: Labeler is logged in with available earnings exceeding the minimum transfer threshold.
4. **Input**:
   - Transfer request to linked banking platform:
     - **Amount**: Total available earnings.
     - **Bank Account Details**: Pre-verified and linked.
5. **Output**:
   - **Expected Result**:
     - Earnings are transferred successfully.
     - Transaction record is created.
     - Labeler receives confirmation and updated earnings balance.
6. **Test Case Derivation**:
   - Ensures labelers are compensated accurately and promptly, critical for system trust.
7. **How Test Will Be Performed**:
   - **Automated Test Script Execution**:
     (a) **Step 1**: Navigate to the earnings or wallet section.
     (b) **Step 2**: Initiate a transfer request for the total available amount.
     (c) **Step 3**: Mock the banking API response if necessary.
     (d) **Step 4**: Verify that the system processes the transfer without errors.
     (e) **Step 5**: Confirm that the earnings balance is updated to reflect the transfer.

(f) **Step 6**: Check that a transaction record is logged in the database.

(g) **Step 7**: Validate that a confirmation message or email is sent to the labeler.

FR13: Image Annotation Test

1. **Test ID**: T-FR13
2. **Control**: **Manual**
3. **Initial State**: Labeler is logged in with images assigned for annotation.
4. **Input**:
   - Labeler annotates an image using the provided tools:
     - Draws bounding boxes around objects.
     - Adds classification labels to each object.
     - Saves the annotation.
5. **Output**:
   - **Expected Result**:
     - Annotated image is stored in the system.
     - Annotation data is correctly linked to the image and service request.
     - Labeler receives confirmation of successful submission.
6. **Test Case Derivation**:
   - Confirms annotation capabilities, essential for image analysis services.
7. **How Test Will Be Performed**:
   - **Manual Steps**:

     (a) **Step 1**: Access the image annotation interface.

     (b) **Step 2**: Use annotation tools to mark objects.

(c) **Step 3**: Assign appropriate labels to each annotation.
(d) **Step 4**: Save and submit the annotations.
(e) **Step 5**: Verify that a confirmation is received.
(f) **Step 6**: Check the database to ensure annotations are stored correctly.
(g) **Step 7**: Attempt to re-access the annotated image to confirm annotations persist.

FR14: Consolidated Annotation Report Test

1. **Test ID**: T-FR14
2. **Control**: **Automated**
3. **Initial State**: All required labeler annotations are complete for a service request.
4. **Input**:
   - System triggers consolidation process for annotations.
5. **Output**:
   - **Expected Result**:
     - Consolidated report is generated if label accuracy meets the predefined threshold.
     - Report is stored and made accessible to the customer.
6. **Test Case Derivation**:
   - Verifies that the system consolidates annotations accurately, ensuring quality results for customers.
7. **How Test Will Be Performed**:
   - **Automated Test Script Execution**:
     (a) **Step 1**: Simulate completion of all annotations for a service request.

(b) **Step 2**: Initiate the consolidation process (this may be automatic upon annotation completion).

(c) **Step 3**: Verify that the system calculates label accuracy and compares it against the threshold.

(d) **Step 4**: Confirm that if the accuracy meets or exceeds the threshold, the report is generated.

(e) **Step 5**: Check that the report contains consolidated data from all annotations.

(f) **Step 6**: Ensure the report is stored and accessible to the customer linked to the service request.

(g) **Step 7**: Validate that notifications are sent to the customer about the report availability.

- **Edge Case Testing**:

  (a) **Step 8**: Repeat the test with annotations that do not meet the accuracy threshold to verify that the report is not generated and appropriate actions are taken (e.g., re-annotation request).

4.2 Tests for Nonfunctional Requirements

4.2.1 Look and Feel Tests

Appearance Tests

1. Responsive Layout Validation: T-LF0

   Type: Manual, Usability

   Initial State: Users access the application on devices with screen resolutions ranging from 1024×768 pixels to 1920×1080 pixels.

   Input/Condition: The application is displayed on various screen sizes within the specified range to verify adaptability and layout consistency.

   How test will be performed:

   (a) Select a representative set of screen resolutions within the range 1024×768 to 1920×1080 pixels (e.g., 1024×768, 1366×768, 1440×900, 1600×900, 1920×1080).

(b) Use physical devices or browser developer tools to simulate each selected screen resolution.

(c) Verify that all visual elements (buttons, text, images, menus) are fully visible and do not exceed screen boundaries.

(d) Ensure that the layout remains uncluttered, with appropriate spacing and alignment across different screen sizes.

2. Interactive Elements Feedback Validation: T-LF1

   Type: Manual, Usability

   Initial State: Interactive elements (e.g., buttons, links) are present within the application interface.

   Input/Condition: Users interact with various interactive elements to verify that visual feedback is provided appropriately.

   How test will be performed:

   (a) Identify all interactive elements within the application, including buttons, links, and menu items.

   (b) For each interactive element, perform the following interactions: hover, click, and focus (if applicable).

   (c) Observe and document the visual feedback provided for each interaction, such as color changes, animations, or shadows.

   (d) Ensure that every interactive element provides at least one form of visual feedback as specified in the fit criterion.

Style Tests

1. Unified Visual Design Validation: T-LF2

   Type: Manual, Usability

   Initial State: The application interface displays all components (buttons, menus, text fields, images, etc.) with the unified visual design specifications.

Input/Condition: Users or testers navigate through the application to assess the consistency of font type, sizing, color, and background tones across all components.

How test will be performed:

(a) Review the application's style guide or design specifications to understand the expected visual standards.

(b) Navigate through each section of the application, observing the font type, size, color, and background tones used in various components.

(c) Compare each observed element against the style guide to ensure consistency.

(d) Utilize browser developer tools or design inspection tools to verify CSS properties related to fonts and colors.

4.2.2 Usability and Humanity Requirements

Ease of Use Tests

1. Navigation Menu Usability Test: T-UH0

   Type: Manual, Usability

   Initial State: Users have access to the platform's main interface with a well-organized navigation menu.

   Input/Condition: Users are asked to locate and access various main features (e.g., labeling tasks, settings, help, account details) using the navigation menu.

   How test will be performed:

   (a) Prepare a set of tasks where users must navigate to specific main features (e.g., "Find the labeling tasks section", "Access account details").

   (b) Instruct users to perform each task using the navigation menu, counting the number of clicks required to reach the target feature.

   (c) Record whether each task is completed within 3 clicks.

(d) Calculate the percentage of tasks where users were able to navigate to the target feature within 3 clicks.

(e) Ensure that at least MIN_USER_USABILITY_METRIC% of the tasks meet the success criterion.

(f) Analyze any tasks that exceed the 3-click limit to identify navigation issues.

2. Consistency of UI Elements Validation: T-UH1

   Type: Manual, Usability

   Initial State: The application interface displays all UI elements (buttons, forms, etc.) following the platform's visual style guidelines.

   Input/Condition: Users navigate through different pages of the application, interacting with various UI elements to assess consistency.

   How test will be performed:

   (a) Review the application's style guide or design specifications to identify the expected consistent styles for UI elements.

   (b) Identify all recurring interactive elements (e.g., buttons, forms, menus) across different pages of the application.

   (c) Instruct users to perform the tasks, observing whether they recognize and correctly use the UI elements based on their consistency.

   (d) Calculate the percentage of users who correctly recognized and used the recurring interface elements across different pages.

   (e) Ensure that at least MIN_USER_USABILITY_METRIC% of users meet the recognition and correct usage criterion.

Personalization and Internationalization Tests

1. Task Preference Matching Validation: T-UH2

   Type: Manual, Usability

   Initial State: Users have access to their profile settings where they can set preferences for specific types of labeling tasks (e.g., agricultural, urban).

Input/Condition: Users configure their task preferences in their profile settings and begin receiving assigned tasks over a defined period.

How test will be performed:

(a) Select a representative sample of users and have them set their task preferences in their profiles.

(b) Over a set period (e.g., two weeks), track the labeling tasks assigned to these users.

(c) Analyze the assigned tasks to determine the percentage that matches the users' specified preferences.

(d) Verify that at least MIN_PREF_MATCH% of the assigned tasks align with the users' preferences.

2. Localized Format Validation: T-UH3

Type: Manual, Dynamic

Initial State: Users have access to their profile settings where they can set or have their location automatically detected for date, time, and currency preferences.

Input/Condition: Users either allow the application to detect their location automatically or manually set their preferred locale settings for date, time, and currency.

How test will be performed:

(a) Select a diverse set of users from different geographical locations with varying locale settings.

(b) For each user, ensure that their location is either correctly detected by the application or manually set their preferred locale settings in their profile.

(c) Navigate to sections of the application where date, time, and currency are displayed (e.g., task deadlines, earnings reports).

(d) Verify that the date formats (e.g., DD/MM/YYYY vs. MM/DD/YYYY), time formats (e.g., 24-hour vs. 12-hour), and currency symbols and formats (e.g., $, €, ¥) align with the user's locale or manual settings.

(e) Repeat the process after changing the user's locale settings to different regions to ensure the formats update accordingly.

(f) Document any discrepancies where the formats do not match the expected locale settings.

(g) Aggregate the results to ensure that all tested instances correctly display localized formats based on user settings.

Learning Tests

1. Progress Tracker Effectiveness Test: T-UH4

   Type: Manual, Usability

   Initial State: The application includes a fully functional progress tracker that displays completed and pending tutorials and training tasks.

   Input/Condition: Users complete a series of tutorials and training tasks within the platform.

   How test will be performed:

   (a) Select a representative sample of users who will undergo the tutorials and training tasks.

   (b) Instruct users to navigate to the progress tracker before, during, and after completing the tutorials.

   (c) Ensure that the progress tracker accurately displays the status of each learning module, indicating which are completed and which are pending.

   (d) After users have completed the training, administer a post-training survey asking them to rate the helpfulness of the progress tracker.

   (e) Analyze the survey responses to determine the percentage of users who find the progress tracker helpful.

   (f) Verify that at least MIN_TRAINING_TRACKER_USEFULNESS% of the surveyed users rate the progress tracker as helpful.

   (g) Collect additional qualitative feedback to identify any issues or areas for improvement in the progress tracking feature.

2. Simulated Labeling Task Validation: T-UH5

   Type: Manual, Usability

   Initial State: The platform provides a dedicated section for simulated labeling tasks that do not affect actual datasets.

   Input/Condition: New users are prompted to complete at least one simulated labeling task before accessing real labeling tasks.

   How test will be performed:

   (a) Recruit a representative sample of new users and onboard them to the platform.

   (b) Instruct users to complete a simulated labeling task provided in the practice section.

   (c) Track the completion rate of simulated tasks to determine if at least MIN_PRACTICE_TASK_COMPLETION% of users complete a practice task before accessing real data.

   (d) After task completion, administer a satisfaction survey to users to gather feedback on their experience with the simulated tasks.

   (e) Analyze survey results to ensure that at least MIN_PRACTICE_TASK_SATISFACTION% of users report satisfaction with the practice tasks.

   (f) Document any issues or feedback received to identify areas for improvement in the simulated labeling experience.

Understandability and Politeness Tests

1. Contextual Help Pop-Up Effectiveness: T-UH6

   Type: Manual, Usability

   Initial State: The platform includes contextual help pop-ups integrated throughout various features and task interfaces.

   Input/Condition: Users interact with different features and tasks, triggering contextual help pop-ups as needed.

   How test will be performed:

(a) Instruct users to perform a set of typical tasks within the platform, ensuring they engage with areas where contextual help pop-ups are available.

(b) Monitor and record instances where contextual help pop-ups appear during user interactions.

(c) Analyze the survey responses to determine if at least MIN_POPUP_USEFULNESS% of users find the contextual help pop-ups clear and helpful.

(d) Collect qualitative feedback to identify any areas where help pop-ups could be improved or where additional help pop-ups may be needed.

2. Error Message Effectiveness Test: T-UH7

Type: Manual, Usability

Initial State: The platform is fully functional and capable of generating error messages in response to various user actions and system failures.

Input/Condition: Users perform actions that intentionally trigger different types of errors (e.g., invalid input, network issues, unauthorized access).

How test will be performed:

(a) Identify and document common error scenarios within the platform.

(b) Instruct users to perform tasks that will trigger each identified error scenario.

(c) Observe and record the error messages displayed, ensuring they include clear explanations of what went wrong.

(d) Verify that each error message provides actionable steps for resolution.

(e) After encountering errors, administer a survey asking users to rate their frustration level on a predefined scale and assess whether the error messages were helpful.

(f) Analyze the survey results to ensure that at least ERRORS_WITH_INSTRUCTIONS% of error messages include actionable instructions and that the reported frustration rate is below FRUSTRATION_RATE%.

Accessibility Requirements

1. Accessibility Options Validation: T-UH8

   Type: Manual, Usability

   Initial State: Users have access to the platform's settings or preferences section where they can adjust text size and select different color themes.

   Input/Condition: Users adjust the text size up to 200% and choose among at least three different color themes (e.g., light, dark, high contrast) to improve readability based on their preferences.

   How test will be performed:

   (a) Verify that the settings or preferences section includes options for adjusting text size and selecting color themes.

   (b) Select each available color theme (e.g., light, dark, high contrast) and observe the changes in the application's appearance.

   (c) Adjust the text size incrementally up to 200%, ensuring that all text remains readable and that no content is cut off or overlaps.

   (d) Navigate through various sections and features of the application with each color theme and text size setting to ensure that functionality is not compromised.

   (e) Test the adjustments on different devices and screen resolutions to confirm consistency and responsiveness.

   (f) Document any instances where text size adjustments cause loss of content or functionality, or where color themes do not apply uniformly across the platform.

   (g) Compile the results to verify that all color themes are available and that text size can be increased up to 200% without any loss of content or functionality.

2. Keyboard Shortcuts Accessibility Validation: T-UH9

   Type: Manual, Usability

   Initial State: Users have access to the platform with all core actions available through both mouse and keyboard interactions.

Input/Condition: Users are instructed to perform essential actions using only keyboard shortcuts to assess accessibility and efficiency.

How test will be performed:

(a) Identify all core actions within the platform (e.g., navigating between tasks, submitting labels) and ensure each has an assigned keyboard shortcut.

(b) Create a comprehensive list of these keyboard shortcuts for reference during testing.

(c) Recruit a representative sample of users, including power users and those with limited mobility, for usability testing.

(d) Instruct users to perform a series of tasks using only keyboard navigation, utilizing the provided shortcuts.

(e) Track the completion rate of each action to determine if users can access and perform at least MIN_ACTIONS_WITH_KEYBOARD% of the core actions via keyboard.

(f) Observe and document any difficulties or barriers users encounter while using keyboard shortcuts.

4.2.3 Performance Requirements

Speed and Latency Tests

1. New User Account Processing Time Validation: T-PR0

   Type: Manual, Performance

   Initial State: The system is ready to accept new user account creation requests.

   Input/Condition: A set of new user account creation requests are submitted, and their processing times are tracked.

   How test will be performed:

   (a) Define a representative sample size of new user account creation requests to be submitted (e.g., 100 requests).

   (b) Submit each account creation request to the system, ensuring that the requests are made under typical operating conditions.

(c) Calculate the processing time for each request by determining the difference between the submission and completion timestamps.

(d) Analyze the processing times to calculate the percentage of requests processed within 15 minutes (0.25 hours).

(e) Verify that at least MIN_ACCOUNT_CREATION_SUCCESS% of the requests are processed within the 15-minute threshold.

(f) Additionally, confirm that all requests are processed within the 48-hour limit.

(g) Document any instances where processing times exceed the specified limits and investigate potential causes.

2. Service Request Completion Time Validation: T-PR1

Type: Manual, Performance

Initial State: The system is ready to accept service requests from Customers, with negotiated time limits defined for each request.

Input/Condition: A set of service requests are submitted with specific negotiated time limits, and their completion times are tracked.

How test will be performed:

(a) Define a representative sample size of service requests to be submitted (e.g., 100 requests), each with a predefined negotiated time limit, $t_{serviceRequestTimeLimit}$.

(b) Submit each service request to the system under typical operating conditions, ensuring that the negotiated time limits vary to reflect real-world scenarios.

(c) Calculate the completion time for each request by determining the difference between the submission and completion timestamps.

(d) Analyze the completion times to calculate the percentage of requests processed within the negotiated time limit, $t_{serviceRequestTime} < t_{serviceRequestTimeLimit}$.

(e) Verify that at least MIN_REPORT_RETURN% of the requests are completed within the negotiated time limit.

(f) Additionally, confirm that all requests are completed within the extended time limit of $t_{serviceRequestTimeLimit} + 48$ hours.

3. Next Image Serving Time Validation: T-PR2

   Type: Automated, Performance

   Initial State: Labelers are logged into the platform with a queue of images available for labeling.

   Input/Condition: Labelers request the next image to label while images are available in the queue.

   How test will be performed:

   (a) Ensure the labeling queue has a sufficient number of images available for testing.

   (b) Set up automated performance testing tools or scripts to simulate multiple labelers requesting the next image simultaneously.

   (c) For each image request, calculate the time difference between the request and image display for each instance.

   (d) Verify that for all instances where a next image is available, the serving time does not exceed MAX_IMAGE_DISPLAY_TIME seconds.

   (e) Identify and document any instances where the serving time exceeds the MAX_IMAGE_DISPLAY_TIME-second threshold.

   (f) Analyze the collected data to ensure that the system meets the fit criterion.

4. Payout Processing Time Validation: T-PR3

   Type: Manual, Performance

   Initial State: Labelers have earned payouts and have submitted payout requests through the system.

   Input/Condition: Payout requests are made by labelers, and the system processes these requests to deliver payments.

   How test will be performed:

   (a) Define a representative sample size of payout requests to be processed (e.g., 100 requests).

(b) Ensure that the system is configured to handle payout requests under typical operating conditions.

(c) Submit each payout request through the system, recording the timestamp when each request is made.

(d) Calculate the payout delay for each request.

(e) Analyze the payout delays to verify that all requests are processed within 7 business days.

(f) Identify and document any instances where payout delays exceed 7 business days to investigate potential bottlenecks or issues in the payout process.

Precision or Accuracy Tests

1. Label Accuracy Validation: T-PR4

   Type: Automated, Quality Assurance

   Initial State: The system has access to a dataset of objects O with known true classifications $LTrue$.

   Input/Condition: The system processes and labels each object in O, producing guessed classifications $LGuess$.

   How test will be performed:

   (a) Compile a representative sample of objects O with their true classifications $LTrue(o)$.

   (b) Input each object $o \in O$ into the system to obtain the system-generated labels $LGuess(o)$.

   (c) Compare each $LGuess(o)$ with the corresponding $LTrue(o)$ to determine correctness.

   (d) Calculate the label accuracy by dividing the number of correctly labeled objects by the total number of objects:

$$\text{Accuracy} = \frac{|\{o \in O \mid LGuess(o) = LTrue(o)\}|}{|O|}$$

   (e) Verify that the calculated accuracy is at least MIN_LABEL_ACCURACY%:

$$\text{Accuracy} \geq MIN\_LABEL\_ACCURACY\%$$

(f) Generate a report summarizing the accuracy results, including any discrepancies and potential areas for improvement.

(g) If the accuracy meets or exceeds MIN_LABEL_ACCURACY%, the system satisfies the fit criterion. If not, investigate and address factors contributing to lower accuracy.

Robustness or Fault-Tolerance Tests

1. System Uptime Validation: T-PR5

   Type: Automated, Performance

   Initial State: The system is fully operational and connected to uptime monitoring tools.

   Input/Condition: The system operates continuously under normal conditions over a defined monitoring period.

   How test will be performed:

   (a) Implement and configure uptime monitoring tools (e.g., Pingdom, Nagios, or AWS CloudWatch) to continuously track the system's availability.

   (b) Define the monitoring period (e.g., one calendar month) to collect sufficient uptime and downtime data.

   (c) Ensure that all critical system components and services are included in the monitoring setup.

   (d) Collect and log uptime and downtime events throughout the monitoring period.

   (e) Calculate the uptime percentage using the formula:

   $$\text{Uptime Percentage} = \frac{t_{\text{uptime}}}{t_{\text{uptime}} + t_{\text{downtime}}} \times 100\%$$

   (f) Verify that the calculated uptime percentage exceeds MIN_UPTIME%:

   $$\text{Uptime Percentage} > MIN\_UPTIME\%$$

   (g) Identify and document any instances of downtime, including their duration and underlying causes.

Capacity Tests

1. Labeler Capacity Validation: T-PR6

   Type: Automated, Performance

   Initial State: The system has a baseline number of active labelers and pending service requests.

   Input/Condition: Simulate varying numbers of labelers to ensure all service request deadlines are met as per NFR-PR2.

   How test will be performed:

   (a) Use load testing tools (e.g., JMeter, Locust) to simulate different numbers of active labelers.

   (b) For each load level, generate a set of service requests that need to be processed within the negotiated deadlines.

   (c) Monitor the percentage of service requests completed within the required timeframes.

   (d) Identify the maximum number of labelers the system can support while maintaining at least MIN_REPORT_RETURN% of requests meeting the deadlines.

2. Large Image File Handling Validation: T-PR7

   Type: Automated, Performance

   Initial State: The system is configured with sufficient storage capacity and is ready to accept image file uploads.

   Input/Condition: Image files up to 50 GB in size are uploaded and processed by the system.

   How test will be performed:

   (a) Prepare a set of image files, each up to 50 GB in size.

   (b) Use automated scripts to upload each large image file to the system.

   (c) Monitor the system for any crashes or failures during the upload and storage process.

(d) Verify that each image file is successfully stored without errors.

(e) Initiate processing tasks on the stored image files.

(f) Ensure that processing completes successfully without system crashes or failures.

(g) Compile the results to confirm that the system can handle image files up to 50 GB without crashing or failing.

Scalability or Extensibility Tests

1. Scalability Validation: T-PR8

   Type: Automated, Performance

   Initial State: The system is configured for scaling operations with monitoring tools in place.

   Input/Condition: Simulate the required labeler capacity as specified in NFR-PR7 to test the system's ability to scale accordingly.

   How test will be performed:

   (a) Utilize load testing tools (e.g., JMeter, Locust) to simulate the number of labelers required to meet the capacity defined in NFR-PR7.

   (b) Gradually increase the number of simulated labelers while monitoring system performance metrics such as response time, CPU usage, memory consumption, and network throughput.

   (c) Ensure that the system dynamically scales resources (e.g., adding more servers or allocating more memory) to handle the increased load without performance degradation.

   (d) Verify that all service request deadlines defined in NFR-PR7 are consistently met under the simulated load.

   (e) Identify and document any performance bottlenecks or scaling issues encountered during the test.

   (f) Compile the results to confirm that the system can scale to meet the required capacity as specified in NFR-PR7.

4.2.4 Operational and Environmental

Wider Environmental Tests

1. Energy Efficiency Validation: T-OE0

   Type: Automated, Performance

   Initial State: The system is running on cloud infrastructure with existing server management configurations.

   Input/Condition: Implement energy-efficient practices in cloud usage and server management, then measure energy consumption before and after optimization.

   How test will be performed:

   (a) Establish baseline energy consumption by monitoring current cloud infrastructure and server operations using energy monitoring tools.

   (b) Identify and implement energy-efficient practices such as optimizing server utilization, enabling power-saving modes, and selecting energy-efficient instance types.

   (c) Collect energy consumption data for a defined period (e.g., one month) after implementing the optimizations.

   (d) Compare the post-optimization energy consumption data against the baseline measurements.

   (e) Calculate the percentage reduction in energy usage to verify that it meets or exceeds the ENERGY_REDUCTION_TARGET% target.

   (f) Identify any deviations or unexpected increases in energy consumption and investigate potential causes.

Adjacent System Interfacing Tests

1. API and Data Format Integration Validation: T-OE1

   Type: Automated, Integration

   Initial State: The system is configured with API access credentials for at least two major satellite data providers.

Input/Condition: The system attempts to automatically acquire and integrate satellite images from the specified providers using their standardized APIs and data formats.

How test will be performed:

(a) Identify two major satellite data providers and obtain their API documentation and access credentials.

(b) Configure the system to connect to each provider's API, ensuring support for their standardized data formats.

(c) Develop and execute automated scripts to initiate data acquisition from each provider.

(d) Monitor the system for successful ingestion of satellite images without manual intervention.

(e) Verify that the acquired data is correctly formatted, stored, and integrated into the platform's datasets.

2. Payment Processor Integration Validation: T-OE2

Type: Automated, Integration

Initial State: The system is configured with API access credentials for reliable and secure payment processors (e.g., Stripe, PayPal).

Input/Condition: Users and clients perform financial transactions through the integrated payment gateways.

How test will be performed:

(a) Identify and obtain API documentation and access credentials for at least two major payment processors.

(b) Develop automated scripts to simulate various types of transactions, including user compensations and client payments.

(c) Monitor and log each transaction to verify successful processing without errors or delays.

(d) Calculate the transaction success rate by dividing the number of successful transactions by the total number of attempted transactions.

(e) Ensure that the transaction success rate meets or exceeds MIN_TRANSACTION_SUCCE

3. Multiple Currency Support Validation: T-OE3

   Type: Automated, Integration

   Initial State: The system is configured to support multiple currencies, including USD, EUR, GBP, and INR.

   Input/Condition: Users perform transactions in each supported currency to verify correct processing.

   How test will be performed:

   (a) Configure the system with exchange rates for USD, EUR, GBP, and INR.

   (b) Develop automated scripts to simulate transactions in each currency.

   (c) Execute the scripts and verify:

   - Accurate currency conversion.
   - Correct display of currency symbols and values.
   - Proper recording of transactions with the correct currency.

   (d) Ensure all transactions are processed without errors.

4. Machine Learning Framework Compatibility Validation: T-OE4

   Type: Automated, Integration

   Initial State: The system is configured with machine learning frameworks such as TensorFlow, PyTorch, and scikit-learn installed.

   Input/Condition: Users train and deploy models using each framework to verify compatibility.

   How test will be performed:

   (a) Install TensorFlow, PyTorch, and scikit-learn on the system.

   (b) Develop or use existing sample models compatible with each framework.

   (c) Train each sample model using the respective framework.

(d) Deploy the trained models to the production environment.

(e) Verify that training and deployment processes complete without errors.

(f) Ensure that the deployed models function correctly within the platform.

5. Data Pipeline Efficiency Validation: T-OE5

   Type: Automated, Performance

   Initial State: The system has established data pipelines for transferring labeled datasets between the platform and ML models.

   Input/Condition: Large labeled datasets (e.g., 10,000 images) are transferred through the data pipelines.

   How test will be performed:

   (a) Prepare a large labeled dataset consisting of 10,000 images.

   (b) Initiate the data transfer from the platform to the ML models using the established pipelines.

   (c) Monitor the transfer process to measure the time taken to complete the transfer.

   (d) Verify that the data transfer completes within the specified timeframe.

   (e) Execute batch processing of the large dataset through the pipelines and ensure no errors occur during the process.

   (f) Repeat the transfer and processing under different load conditions to assess pipeline robustness.

Productization Tests

1. Web Browser Accessibility Validation: T-OE6

   Type: Manual, Usability

   Initial State: Users have access to the platform's web URL.

Input/Condition: Users attempt to access and use the platform via various supported web browsers without installing any software.

How test will be performed:

(a) Identify supported web browsers (e.g., Chrome, Firefox, Safari, Edge).

(b) Using each browser, navigate to the platform's web URL.

(c) Verify that the platform loads correctly without prompting for any software installations.

(d) Perform key tasks (e.g., login, label images, access settings) to ensure full functionality is available through the browser.

(e) Confirm that all users can access and use the platform solely through their web browsers without additional installations.

Release Tests

1. Road Map Consistency: T-OE7

   Type: Manual, Static

   Initial State: Application has a release road map that is publicly accessible.

   Input/Condition: Team member conducts a review.

   Output/Result: At least MIN_ON_TIME_MILESTONE% of the listed milestones have been met on time.

   How test will be performed: The team member looks over the road map and cross references the completion date of milestones to the dates listed in the road map.

2. Beta Testing: T-OE8

   Type: Dynamic, Exploratory

   Initial State: Beta version of application is deployed and accessible

   Input/Condition: At least BETA_TESTERS beta testers are provided access to use the application.

Output/Result: Feedback on any bugs, navigation issues, or aesthetic problems is provided. Less than MAX_BUGS_FOUND bugs are found.

How test will be performed: Testers will be recruited and identified. They will be from fields of interest that include scientists, labelers, and domain experts. Then, the beta testing environment will be set up and the url will be distributed to the testers along with any other set up resources. Specific tasks are provided for testers to complete that focus on the annotation tools, sign up process and project creation. Feedback will be collected through direct comments from the tester.

3. Regression Testing: T-OE9

   Type: Dynamic, Automated

   Initial State: Application is deployed.

   Input/Condition: Run regression test suite, consisting of unit tests.

   Output/Result: All regression tests are passed.

   How test will be performed: An automated script with regression tests will run when updates are made to the production build.

4.2.5 Maintainability and Support

Maintenance Tests

1. Ease of Change: T-MS0

   Type: Manual, Static

   Initial State: Application's source repository contains complete documentation.

   Input/Condition: Competent software developer who has not previously worked on the app reviews documentation and attempts to perform tasks.

   Output/Result: The developer can easily make a minor update to a specified part of the application.

   How test will be performed: Give the developer time to read through the documentation. Give them a maintenance task, such as updating

the size of the title font to 20px. Observe them and document how long it takes them and if they encountered any troubles.

4.2.6 Security

Access Tests

1. Logged Out Permissions: T-SE0

   Type: Manual, Dynamic, White-box

   Initial State: Application is deployed.

   Input/Condition: Tester who is not signed in tries to access application paths for project creation and image labeling (Ex. /projects or /label).

   Output/Result: The tester is denied access to these paths and is told to sign in.

   How test will be performed: On the deployed application, the tester will visit all possible paths as a logged out user.

2. Labeler Permissions: T-SE1

   Type: Manual, Dynamic, White-box

   Initial State: Application is deployed.

   Input/Condition: Tester who is signed in as a labeler tries to access application paths for project creation.

   Output/Result: The tester is denied access to these paths. However, the tester has access to paths related to image labeling.

   How test will be performed: On the deployed application, the tester will visit all possible paths as a labeler.

3. Invalid Email Format: T-SE2

   Type: Automatic, Dynamic

   Initial State: Front-end registration page is created and integrated with the database.

Input/Condition: Email with invalid format, such as an empty string or a string missing '@', is entered.

Output/Result: Application rejects email and tells the user that the email format is wrong.

How test will be performed: A unit test will be performed where the input is entered into the email section of the registration form.

4. Duplicate Email: T-SE3

   Type: Automatic, Dynamic

   Initial State: Front-end registration page is created and integrated with the database.

   Input/Condition: Email that is already in database is entered.

   Output/Result: Application rejects email and tells the user that the email is in use.

   How test will be performed: A unit test will be performed where the input is entered into the email section of the registration form.

5. Invalid Password Format: T-SE4

   Type: Automatic, Dynamic

   Initial State: Front-end registration page is created and integrated with the database.

   Input/Condition: Password with invalid format, such as an empty string or a string with no numbers, is entered.

   Output/Result: Application rejects password and tells the user what requirements they have not met.

   How test will be performed: A unit test will be performed where the input is entered into the password section of the registration form.

6. System Error: T-SE5

   Type: Manual, Dynamic

   Initial State: Application is deployed.

Input/Condition: Purposely invoke a system failure, and attempt to perform an action such as a label submission.

Output/Result: Application provides an error message on the user interface. The database has not changed in anyway.

How test will be performed: Go on to the application, start a labeling task, purposely disconnect from the internet, and try to submit a labeled image.

Integrity Tests

1. Duplicate Entries: T-SE6

   Type: Manual, Dynamic

   Initial State: Database is deployed.

   Input/Condition: Duplicate database entry is inserted into the database.

   Output/Result: Database has only one of the inputted entry and the duplicate has been removed.

   How test will be performed: Attempt to insert the same entry twice into the database through the database UI.

Privacy Tests

1. Encrypted User Data: T-SE7

   Type: Manual, Dynamic

   Initial State: Application is deployed.

   Input/Condition: Tester registers an account.

   Output/Result: All sensitive user data that is stored in the database is encrypted.

   How test will be performed: Tester will create a new account, then check the corresponding user entry in the database and see if the sensitive information is encrypted.

2. Encrypted Payments: T-SE8

   Type: Manual, Dynamic

   Initial State: Application is deployed.

   Input/Condition: Tester enters sample payment details to pay for a labeling project that has been created.

   Output/Result: These details are encrypted and can not be read through packet analyzers. The amount in the request can not be modified by an adversary.

   How test will be performed: Tester will enter sample payment details, and submit their payment. Using a packet analyzer (such as Wireshark), packets from this request will be looked at to ensure all information is encrypted.

Immunity Tests

1. SQL Injection: T-SE9

   Type: Manual, Dynamic

   Initial State: Application is deployed.

   Input/Condition: A malicious SQL statement is entered into a text field.

   Output/Result: The system raises an error telling the user that it is invalid.

   How test will be performed: Tester will enter a SQL statement such as "{valid email}'–" into an input such as the email input. This example has the potential to bypass a password check by commenting out the rest of the SQL query. The tester will check that when this statement is entered, the system gives feedback that it is invalid.

4.2.7 Cultural

Language Tests

1. Support of Different Languages: T-CU0

   Type: Manual, Dynamic

   Initial State: Application is deployed.

   Input/Condition: Tester selects a language from a list of available languages.

   Output/Result: All text on the website is translated and displayed in the selected language.

   How test will be performed: Tester will check that the language selection list is accessible, and that the most popular languages are included. When a language is selected, the tester will check that the translation has been applied and there is no untranslated or gibberish text. This can be checked for each language.

4.2.8 Compliance

Financial Tests

1. Compliant Payment Process: T-CO0

   Type: Manual, Static

   Initial State: Application is deployed.

   Input/Condition: Qualified Security Assessor (QSA) assesses the application.

   Output/Result: They determine that it meets the PCI-DSS standard.

   How test will be performed: A QSA will be found and contacted to perform an assessment. The QSA will be shown all parts of the application that deal will financial transactions and will be able to make a determination on if it meets the standard.

Legal Tests

1. System Availability: T-CO1

   Type: Manual, Dynamic

Initial State: Application is deployed.

Input/Condition: Tester changes country they are accessing the application from using a tool such as a VPN.

Output/Result: Application is blocked in countries facing economic sanctions by the Government of Canada.

How test will be performed: A list of the countries facing economic sanctions by Canada will be compiled. Then, the tester will simulate that they are accessing the application from these countries, and ensure it is unreachable.

2. Taxes: T-CO2

   Type: Manual, Dynamic

   Initial State: Application is deployed.

   Input/Condition: Tester redeems a cash balance.

   Output/Result: If the cash balance exceeds a threshold, a tax form will be issued.

   How test will be performed: Tester creates a test account with an account balance over the threshold. When they withdraw, they check that a tax form has been emailed to the email associated with the account. The tax form should reflect the withdrawal balance.

3. Project Availability: T-CO3

   Type: Manual, Dynamic

   Initial State: Application is deployed.

   Input/Condition: Tester changes country they are accessing the application from using a tool such as a VPN.

   Output/Result: Specific project is not shown.

   How test will be performed: A project will be specified to only be distributed in a specific country. Then, the tester will simulate that they are accessing the application from other countries, and ensure the project does not show up.

4.2.9 User Documentation and Training

1. Helpfulness of User Aids: T-UDT0

   Type: Manual, Dynamic

   Initial State: Application is deployed with all help features. Tutorials and user documentation have been created.

   Input/Condition: Users attempt to complete a basic labeling task using only the platform's built-in help resources (help system, quick start guide, tutorials and contextual tooltips).

   Output/Result: At least MIN_USER_HELP_SATISFACTION% of users who used a help feature found that feature helpful. With the assistance of the help tools, the user was able to perform the task within MAX_TASK_TIME minutes.

   How test will be performed: The purpose of the platform will be explained to the users and the built-in help features will be shown. Then, the labeling task will be given to them. Time to complete task is observed and the help tools they use are recorded. Participants will then fill out a usability survey, which can be viewed in the appendix.

2. Usefulness of Sandbox: T-UDT1

   Type: Automatic, Dynamic

   Initial State: Application is deployed with all help features. Tutorials and user documentation has been created.

   Input/Condition: A new user has accessed the platform.

   Output/Result: At least MIN_PRACTICE_USAGE% of new users utilize the practice environment, with self-assessment scores indicating an average improvement of IMPROVE_IN_ACC% in labeling accuracy over their first three attempts.

   How test will be performed: Practice environment utilization will be tracked by the application. Improvement in accuracy will also be tracked. If the metrics meet or succeed our thresholds, then we can conclude the sandbox is useful.

4.3 Traceability Between Test Cases and Requirements

FR #	Test Case														
	T-FR0	T-FR1	T-FR2	T-FR3	T-FR4	T-FR5	T-FR6	T-FR7	T-FR8	T-FR9	T-FR10	T-FR11	T-FR12	T-FR13	T-FR14
FR0	X														
FR1		X													
FR2			X												
FR3				X											
FR4					X										
FR5						X									
FR6							X								
FR7								X							
FR8									X						
FR9										X					
FR10											X				
FR11												X			
FR12													X		
FR13														X	
FR14															X

NFR #	Test Case														
	T-LF0	T-LF1	T-LF2	T-UH0	T-UH1	T-UH2	T-UH3	T-UH4	T-UH5	T-UH6	T-UH7	T-UH8	T-UH9	T-PR0	T-PR1
LF0	X														
LF1		X													
LF2			X												
UH0				X											
UH1					X										
UH2						X									
UH3							X								
UH4								X							
UH5									X						
UH6										X					
UH7											X				
UH8												X			
UH9													X		
PR0														X	
PR1														X	
PR2															X

NFR #	Test Case														
	T-PR2	T-PR3	T-PR4	T-PR5	T-PR6	T-PR7	T-PR8	T-OE0	T-OE1	T-OE2	T-OE3	T-OE4	T-OE5	T-OE6	T-OE7
PR3	X														
PR4		X													
PR5			X												
PR6				X											
PR7					X										
PR8						X									
PR9							X								
OE0								X							
OE1									X						
OE2										X					
OE3											X				
OE4												X			
OE5													X		
OE6														X	
OE7															X

NFR #	Test Case														
	T-OE8	T-OE9	T-MS0	T-SE0	T-SE1	T-SE2	T-SE3	T-SE4	T-SE5	T-SE6	T-SE7	T-SE8	T-SE9	T-CU0	T-CO0
OE8	X														
OE9		X													
MR0			X												
SE0				X											
SE1					X										
SE2						X	X								
SE3								X							
SE4									X						
SE5						X	X	X							
SE6									X						
SE7										X					
SE8											X				
SE9												X			X
SE10													X		
CU0														X	

NFR #	Test Case				
	T-C01	T-C02	T-C03	T-UDT0	T-UDT1
CO0	X				
CO1		X			
CO2			X		
UD0				X	
UD1				X	
UD2				X	
TR0				X	
TR1					X

5 Unit Test Description

[This section should not be filled in until after the MIS (detailed design document) has been completed. —SS]

[Reference your MIS (detailed design document) and explain your overall philosophy for test case selection. —SS]

[To save space and time, it may be an option to provide less detail in this section. For the unit tests you can potentially layout your testing strategy here. That is, you can explain how tests will be selected for each module. For instance, your test building approach could be test cases for each access program, including one test for normal behaviour and as many tests as needed for edge cases. Rather than create the details of the input and output here, you could point to the unit testing code. For this to work, you code needs to be well-documented, with meaningful names for all of the tests. —SS]

5.1 Unit Testing Scope

[What modules are outside of the scope. If there are modules that are developed by someone else, then you would say here if you aren't planning on verifying them. There may also be modules that are part of your software, but have a lower priority for verification than others. If this is the case, explain your rationale for the ranking of module importance. —SS]

5.2 Tests for Functional Requirements

[Most of the verification will be through automated unit testing. If appropriate specific modules can be verified by a non-testing based technique. That can also be documented in this section. —SS]

5.2.1 Module 1

[Include a blurb here to explain why the subsections below cover the module. References to the MIS would be good. You will want tests from a black box perspective and from a white box perspective. Explain to the reader how the tests were selected. —SS]

1. test-id1

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input:

   Output: [The expected result for the given inputs —SS]

   Test Case Derivation: [Justify the expected value given in the Output field —SS]

   How test will be performed:

2. test-id2

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input:

   Output: [The expected result for the given inputs —SS]

   Test Case Derivation: [Justify the expected value given in the Output field —SS]

   How test will be performed:

3. ...

5.2.2 Module 2

...

5.3 Tests for Nonfunctional Requirements

[If there is a module that needs to be independently assessed for performance, those test cases can go here. In some projects, planning for nonfunctional tests of units will not be that relevant. —SS]

[These tests may involve collecting performance data from previously mentioned functional tests. —SS]

5.3.1 Module ?

1. test-id1

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input/Condition:

   Output/Result:

   How test will be performed:

2. test-id2

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input:

   Output:

   How test will be performed:

5.3.2 Module ?

...

5.4 Traceability Between Test Cases and Modules

[Provide evidence that all of the modules have been considered. —SS]

References

[1] OrbitWatch. *Development Plan*. Tech. rep. https://github.com/OKKM-insights/OKKM.insights/blob/main/docs/DevelopmentPlan/DevelopmentPlan.pdf. McMaster University, 2024.

[2] OrbitWatch. *Module Guide (MG)*. Tech. rep. https://example.com/mg. McMaster University, 2024.

[3] OrbitWatch. *Module Interface Specification (MIS)*. Tech. rep. https://example.com/mis. McMaster University, 2024.

[4] OrbitWatch. *Problem Statement*. Tech. rep. https://github.com/OKKM-insights/OKKM.insights/blob/main/docs/ProblemStatementAndGoals/ProblemStatement.pdf. McMaster University, 2024.

[5] OrbitWatch. *Software Requirements Specification*. Tech. rep. https://github.com/OKKM-insights/OKKM.insights/blob/main/docs/SRS/SRS.pdf. McMaster University, 2024.

[6] OrbitWatch. *Verification and Validation (VnV) Plan*. Tech. rep. https://github.com/OKKM-insights/OKKM.insights/blob/main/docs/VnVPlan/VnVPlan.pdf. McMaster University, 2024.

6 Appendix

6.1 Symbolic Parameters

The definition of the test cases will call for SYMBOLIC_CONSTANTS. Their values are defined in this section for easy maintenance.

Parameter	Value	Unit	Description
MIN_USER_USABILITY_METRIC	90	%	Minimum percent of users that satisfy usability requirements
MIN_PREF_MATCH	80	%	Minimum percent of label task preference match
MIN_TRAINING_TRACKER_USEFULNESS	80	%	Minimum percent of trackers usefulness
MIN_PRACTICE_TASK_COMPLETION	85	%	Minimum percent of user to complete practice task
MIN_POPUP_USEFULNESS	80	%	Minimum percent of users to find popups useful
ERRORS_WITH_INSTRUCTIONS	80	%	Percent of errors with follow up instructions
FRUSTRATION_RATE	10	%	Frustration rate when encountering errors
MIN_ACTIONS_WITH_KEYBOARD	95	%	Minimum percent of actions available with keyboard inputs
MIN_ACCOUNT_CREATION_SUCCESS	90	%	Minimum percent of accounts created successfully within 48 hours
MIN_REPORT_RETURN	90	%	Minimum percent of reports returned to customer within time frame

Parameter	Value	Unit	Description
MAX_IMAGE_DISPLAY_TIME	10	Minutes	Maximum time to display image to labeller
MIN_LABEL_ACCURACY	75	%	Minimum percent of label accuracy
MIN_UPTIME	80	%	Minimum percent of system uptime
ENERGY_REDUCTION_TARGET	20	%	Minimum percent of achieved energy reduction
MIN_TRANSACTION_SUCCESS_RATE	99.5	%	Minimum percent of successful transactions
MIN_ON_TIME_MILESTONE	80	%	Minimum percent of milestones that have been met on time
BETA_TESTERS	50	People	Number of beta testers
MAX_BUGS_FOUND	10	Bugs	Number of software bugs found
MIN_USER_HELP_SATISFACTION	80	%	Minimum percent of users satisfied with help feature
MAX_TASK_TIME	15	Minutes	Maximum time it takes a user to complete a task
MIN_PRACTICE_USAGE	80	%	Minimum percent of new users who have used the practice sandbox
IMPROVE_IN_ACC	20	%	Improvement in accuracy of a user after practicing

6.2 Usability Survey Questions

1. On a scale of 1 to 5, how would you rate the readability of text at this screen size?
2. On a scale of 1 to 5, how clear was the visual feedback when interacting with buttons and other interactive elements?
3. On a scale of 1 to 5, how consistent do you find the visual design across different sections of the application?
4. On a scale of 1 to 5, how intuitive did you find the navigation menu?
5. On a scale of 1 to 5, how consistent did you find the style of buttons and forms across different pages of the application?
6. On a scale of 1 to 5, how satisfied are you with the alignment of your assigned tasks to your preferences?
7. On a scale of 1 to 5, how helpful do you find the progress tracker in monitoring your training progress?
8. On a scale of 1 to 5, how satisfied are you with the simulated labeling tasks provided for practice?
9. On a scale of 1 to 5, how clear and understandable did you find the contextual help pop-ups?
10. On a scale of 1 to 5, how clear and actionable were the error messages you encountered?
11. On a scale of 1 to 5, how easy was it to adjust the text size to your preference?
12. On a scale of 1 to 5, how satisfied are you with the available color themes for improving readability?
13. On a scale of 1 to 5, how easy was it to access the platform through your web browser?
14. Did you use the help system to aid in completing your task? Yes/No

15. If you answered yes, please rate how useful it was in helping you accomplish your task: 1 (Not Useful) - 5 (Very Useful)

16. Did you use the quick start guide to aid in completing your task? Yes/No

17. If you answered yes, please rate the clarity and helpfulness of it: 1 (Not Helpful) - 5 (Very Helpful)

18. Did you notice the tool-tips or pop-ups providing contextual help as you worked? Yes/No

19. If you answered yes, please rate the clarity and usefulness of the in-app help indicators: 1 (Not Helpful) - 5 (Very Helpful)

Appendix — Reflection

[This section is not required for CAS 741 —SS]

The information in this section will be used to evaluate the team members on the graduate attribute of Lifelong Learning.

The purpose of reflection questions is to give you a chance to assess your own learning and that of your group as a whole, and to find ways to improve in the future. Reflection is an important part of the learning process. Reflection is also an essential component of a successful software development process.

Reflections are most interesting and useful when they're honest, even if the stories they tell are imperfect. You will be marked based on your depth of thought and analysis, and not based on the content of the reflections themselves. Thus, for full marks we encourage you to answer openly and honestly and to avoid simply writing "what you think the evaluator wants to hear."

Please answer the following questions. Some questions can be answered on the team level, but where appropriate, each team member should write their own response:

1. What went well while writing this deliverable?

   While working on this deliverable, we found that several things went particularly well for our team. From the very beginning, we laid out a clear plan and divided the tasks based on each person's strengths, which made the workload manageable and played to our individual skills. Getting an early start was a game-changer; it allowed us ample time to really delve into each aspect without feeling rushed. We also benefited from having some spare time due to lighter workloads in our other courses, which we used to do deeper research on each topic. Revisiting our SRS and applying it to our VnV plan challenged us to think critically and solidified our understanding of the project. Our communication was smooth throughout—we checked in regularly and supported each other whenever questions or issues came up. Overall, the combination of good planning, effective collaboration, and the extra time we invested made the process both successful and satisfying.

2. What pain points did you experience during this deliverable, and how did you resolve them?

While working on this deliverable, we did face a few pain points that tested our teamwork and project management skills. One of the main challenges was investigating topics related to our project and tests that we hadn't openly discussed before; this led to some confusion and extra time spent getting everyone on the same page. Maintaining consistency across the entire document was also tricky—different writing styles and interpretations made parts of the doc feel uneven. We encountered build issues as well; some of us had to wait for others' pull requests to merge before we could proceed with our own work, which slowed down our progress. To tackle these problems, we discussed it during our meeting on Monday to openly discuss and clarify the unclear topics, ensuring everyone understood the direction we were heading. We also discussed that why working on documents is extremely important since it's worth a lot of our grade. Ensuring the deliverable met high standards despite any time pressures or setbacks is a key thing we discussed. By addressing these issues head-on, we were able to smooth out the rough patches and keep the project moving forward.

3. What knowledge and skills will the team collectively need to acquire to successfully complete the verification and validation of your project? Examples of possible knowledge and skills include dynamic testing knowledge, static testing knowledge, specific tool usage, Valgrind etc. You should look to identify at least one item for each team member.

   To successfully verify and validate our project of processing high-definition satellite images for segmentation, our team needs to acquire a few specific skills. One team member plans to deepen his understanding of dynamic testing so he can effectively test his segmentation algorithms under various conditions. One of us will focus on static testing techniques, learning to use tools like Valgrind to identify and fix any code issues or inefficiencies. Another team member is going to get up to speed with image processing tools like OpenCV, which will be crucial for validating our segmentation results. The fourth member will work on performance profiling to ensure our application can handle large HD images smoothly without bogging down. We'll all need to become more familiar with handling and processing large satellite datasets, and possibly learn about machine learning model validation since we will be

incorporating AI techniques for most part. By each of us focusing on these areas, we'll collectively cover all the bases needed for thorough verification and validation of our project.

4. For each of the knowledge areas and skills identified in the previous question, what are at least two approaches to acquiring the knowledge or mastering the skill? Of the identified approaches, which will each team member pursue, and why did they make this choice?

   This is the potential list of what each team member might be doing (subject to changes):

   - Kartik – Dynamic Testing
   - Mathew – Static Testing
   - Kyle – OpenCV
   - Oleg – Performance Profiling

   For dynamic testing, one of us considered enrolling in an online course or jumping straight into writing and running test cases on our code. We chose the hands-on route because some of us learn best by doing, and it lets us contribute directly to the project while improving our skills. The team member focusing on static testing and tools like Valgrind thought about attending a workshop or studying the documentation and applying it themselves. They decided to dive into the documentation and start using Valgrind on our code, preferring self-paced learning and immediate application. Our teammate working with OpenCV weighed taking a formal course versus following online tutorials and building small projects. They opted for the practical approach, believing that working on example projects would help them understand how to apply OpenCV to our needs more effectively. The fourth member, handling performance profiling, considered seeking guidance from a mentor or learning from online resources and applying them directly. They chose to start with online resources to quickly begin identifying and addressing performance issues in our application. Each of us picked the approach that aligns with our learning styles and allows us to make meaningful contributions to the project.

References

[1] OrbitWatch. *Development Plan*. Tech. rep. https://github.com/OKKM-insights/OKKM.insights/blob/main/docs/DevelopmentPlan/DevelopmentPlan.pdf. McMaster University, 2024.

[2] OrbitWatch. *Module Guide (MG)*. Tech. rep. https://example.com/mg. McMaster University, 2024.

[3] OrbitWatch. *Module Interface Specification (MIS)*. Tech. rep. https://example.com/mis. McMaster University, 2024.

[4] OrbitWatch. *Problem Statement*. Tech. rep. https://github.com/OKKM-insights/OKKM.insights/blob/main/docs/ProblemStatementAndGoals/ProblemStatement.pdf. McMaster University, 2024.

[5] OrbitWatch. *Software Requirements Specification*. Tech. rep. https://github.com/OKKM-insights/OKKM.insights/blob/main/docs/SRS/SRS.pdf. McMaster University, 2024.

[6] OrbitWatch. *Verification and Validation (VnV) Plan*. Tech. rep. https://github.com/OKKM-insights/OKKM.insights/blob/main/docs/VnVPlan/VnVPlan.pdf. McMaster University, 2024.