System Verification and Validation Plan for CXR

Team 27, Neuralyzers
Ayman Akhras
Nathan Luong
Patrick Zhou
Kelly Deng
Reza Jodeiri

November 2, 2024

Revision History

Date	Version	Notes
Date 1	1.0	Notes
Date 2	1.1	Notes

[The intention of the VnV plan is to increase confidence in the software. However, this does not mean listing every verification and validation technique that has ever been devised. The VnV plan should also be a **feasible** plan. Execution of the plan should be possible with the time and team available. If the full plan cannot be completed during the time available, it can either be modified to "fake it", or a better solution is to add a section describing what work has been completed and what work is still planned for the future. —SS]

[The VnV plan is typically started after the requirements stage, but before the design stage. This means that the sections related to unit testing cannot initially be completed. The sections will be filled in after the design stage is complete. the final version of the VnV plan should have all sections filled in. —SS]

Contents

List of Tables

[Remove this section if it isn't needed —SS]

List of Figures

[Remove this section if it isn't needed —SS]

1 Symbols, Abbreviations, and Acronyms

symbol	description
T	Test

[symbols, abbreviations, or acronyms — you can simply reference the SRS (Author, 2019) tables, if appropriate —SS]

[Remove this section if it isn't needed —SS]

This document ... [provide an introductory blurb and roadmap of the Verification and Validation plan —SS]

2 General Information

2.1 Summary

There are multiple software components that are required for us to complete our end goal, as previously mentioned in SRS and Hazard Analysis, we need to test each of these components to ensure accuracy and reliability in our product. More of the specifics can be found below.

2.1.1 User Interface (UI)

Function: Ensure that the interface is easy to navigate for users like radiologists, doctors, or technicians. This includes testing for intuitive layouts, clear labels, and smooth workflows.

- **Functionality**: Check if every UI element this includes: buttons, drop downs, the menu and ensure the software front ended system responds correctly to user input.
- **Performance**: Verify the UI's responsiveness and loading times, particularly when displaying large medical images like X-rays.

2.1.2 Image Preprocessing and Enhancement

Function: Before analysis, the X-ray images undergo preprocessing steps to enhance image quality and ensure consistent inputs for the AI model.

- **Accuracy of Noise Reduction**: Removes artifacts and noise from X-rays, improving image clarity. This is usually done by making the image grey scale or removing color from the image to ensure further accuracy when reading the image.
- **Normalization**: Standardizes the pixel values across images to ensure uniformity, allowing the AI model to better detect subtle differences.

2.1.3 Processing The Image

Function: The AI model reads the image and looks at the features or patterns from the X-ray images that are indicative of various lung conditions.

- **Accuarcy of Texture and Shape Analysis**: System analyzes shape and density of lung tissues by looking at tensor values and identifying anomalies.

2.1.4 Disease Classification

Function: Once features are extracted, using AI classification system categorizes pattern into different diseases; returning the probabilities.

- **Accuracy and speed of Disease Models**: Model is trained on large datasets of labeled X-ray images specifically CheXRPT, aiding it to recognize the patterns associated with diseases, this includes: Pneumonia, Lung Cancer, Tuberculosis and more.
- **Accuracy of Probability Scores**: It provides a probability or confidence score for each disease, indicating the likelihood of the condition being present.

Function: Integrate our application hospital systems and hospital imaging software for seamless data flow.

- **Functionality of Automated Reporting**: Upon completing the analysis, the software can generate the report showing the probabilities of diseases and save that information onto the patients hospital data or user data of that hospital.

2.2 Objectives

2.2.1 Ensure Software Accuracy or Correctness (In scope)

- The primary objective of this project is to build confidence in the correctness of the AI-powered diagnostic tool for lung diseases.
- The software should be able to consistently and accurately predict the with a score of a minimum of 85% accuracy.

- This probability will be derived from the model estimating the odds of various lung conditions for example, pneumonia or lung cancer from X-ray images.
- Verifying the correctness of the AI's predictions is essential to ensure reliable clinical decision-making.

2.2.2 Demonstrate Adequate Usability (In scope)

- Given that medical professionals, such as radiologists, students, and doctors, will use the tool, the UI needs to be intuitive and efficient.
- The focus will be on ensuring the tool's interface is easy to navigate, with clear access to essential features such as uploading X-rays, viewing probabilities, and generating reports.
- This probability will be derived from the model estimating the odds of various lung conditions for example, pneumonia or lung cancer from X-ray images.
- The goal of the application is to ensure correct results and not create an inner challenge to use the software itself, so users can quickly perform key tasks.

2.2.3 Security of Patient Data (In scope)

- Since the software will handle sensitive patient information, it must comply with privacy regulations such as PIPEDA for Canada or HIPAA if used in the US.
- This objective will focus on verifying that patient data is securely stored and transferred, and that access control mechanisms are in place to protect against unauthorized use.

2.2.4 Verification of External Libraries and APIs (Out of scope)

- The system relies on pre-existing libraries and APIs for tasks such as image processing, patient data handling, and possibly pre-trained AI models.

- The quality of these external dependencies will not be verified in this project.
- Our team believes that the External libraries and APIs will be assumed to have been tested by their implementation teams.

2.2.5 Extensive Usability Testing (Out of scope)

- Ensuring we do adequate usability testing is a priority, however going further than that is a problem for the longer future as we have limited resources, only basic usability testing will be done to ensure the software meets general usability standards.

2.2.6 Non Canadian/American Regulatory Compliance (Out of scope)

- The project will not focus on ensuring compliance with other health data privacy regulation, for example, the General Data Protection Regulation in the European Union. As we are currently only looking to launch this application for North America more specifically Canada and the United States.

2.3 Challenge Level and Extras

2.3.1 Challenge Level

Our team believes the challenge level for this project to be advanced. This classification is based on the complexity of integrating AI for diagnostic purposes, the necessity for robust data handling and privacy compliance, and the need to develop a user-friendly interface for medical professionals. The project requires a solid understanding of machine learning principles, software development practices, and compliance with health data regulations. Our knowledge of Tensors, Linear Algebra, machine learning, and image processing will be essential for this application to function effectively.
In terms of feasibility, our approach includes specific strategies to handle data privacy and large-scale data management. We plan to adhere to compliance regulations (such as HIPAA guidelines) and use anonymized datasets where necessary. Additionally, our team has access to sufficient computational resources and relevant medical datasets, ensuring we can manage the data and

AI components of the project effectively. Links to some resources are below: https://youtu.be/lvXc3O2n56w?si=CdYoJH9WRrPKQU3i https://stanfordmlgroup.github.io/competitions/chexpert/
We have also identified several "extras" to enhance the application, such as incorporating a user feedback loop and additional image processing features that could benefit medical professionals and future development phases. Our System Verification and VAlidation plan will include cross-validation and real-world testing scenarios to ensure accuracy and reliability in a clinical context.

2.3.2 Extras

Extra Testing: As mentioned in SRS and in the previous sections: Basic usability testing will be conducted to ensure that the user interface is intuitive and easy for healthcare professionals to navigate. This will include gathering feedback from potential users to identify any usability issues and make necessary adjustments.

Feature one: One additional feature we plan to add, is a quick notification system to the people who are responsible patient. For instance, if the patient was to have a serious condition, for example, benign cancer in that region of his lungs, his physician would be immediately updated via SMS or other communication platforms.

Feature Two: Another feature we can look to add to our application is a checklist of the patients symptoms this would be used in a verification role. For example, if the patient facilitated his symptoms to be shortness of breath, the application would use this information and incorporate in the AI trained model, returning diseases which could have this potential symptom.

2.4 Relevant Documentation

The three main relevant documentation that helped guide us in System Verification and Validation Plan (VnV) was Software Requirements Specification (SRS), Hazard Analysis (HA), and development plan. SRS was crucial for the successful development of the AI-powered diagnostic tool for lung diseases, as

it clearly outlines the software's intended functionality, performance requirements, and design constraints, providing a road map for us developers and aiding us understand our stakeholders. The problem statement identifies the specific challenges to be addressed, while the development plan documented helped us by outlining what approach and resources were needed to achieve our project goals. Lastly, Hazard Analysis helped us identify security and safety requirements, with that in mind we were able to test the system more thoroughly. As HA helped us derive requirements for our implementation plan by creating by additional mitigation of health and safety risks. For example, if our server went down, we would not potentially fulfill the Service Level Agreement due to the downtime of the system. However, through HA we were able to create counter measures to ensure when things like that happen, those risks are mitigated.These documents are very relevant as they helped ensure that the SRS aligns with the project's objectives, guiding the VnV process by defining acceptance criteria, enabling the creation of test cases linked to specific requirements, and establishing a traceability matrix to ensure all functionalities are tested. Additionally, the SRS aids in managing changes, identifying risks, and ensuring compliance with regulations like PIPEDA or HIPAA, ultimately enhancing quality assurance practices and helping to deliver a reliable and effective software product.

Author (2019)

[Don't just list the other documents. You should explain why they are relevant and how they relate to your VnV efforts. —SS]

3 Plan

[Introduce this section. You can provide a roadmap of the sections to come. —SS]

3.1 Verification and Validation Team

[Your teammates. Maybe your supervisor. You should do more than list names. You should say what each person's role is for the project's verification. A table is a good way to summarize this information. —SS]

3.1.1 Understanding Medical Conditions

Assigned Team Members: All

For the verification process to ensure that the AI accurately detects lung diseases in chest X-rays, all team members must gain a deep understanding of the relevant medical conditions. This is essential for assessing whether the AI outputs align with real-world clinical expectations. To achieve this, the team needs to thoroughly review annotated chest X-ray datasets and medical literature. Without this foundational knowledge, the team cannot effectively validate the model's diagnostic accuracy. During verification, each member must be capable of recognizing potential errors in the model's predictions and correlating them with known medical indicators, ensuring the AI delivers clinically relevant results.

3.1.2 Learning PyTorch Fundamentals

Assigned Team Members: Patrick, Reza, and Kelly

To achieve high-quality verification of the AI model, the team must master PyTorch's core functionalities, particularly related to the data flow and training process. The team needs to understand how to verify that the model is not just functioning, but also learning and improving in a meaningful way. This includes using PyTorch's tensor operations and autograd features to test whether gradients are being calculated and applied correctly during backpropagation. Additionally, the team must verify that the preprocessing steps—such as resizing, normalizing, and augmenting X-ray images—are executed properly, as incorrect data handling could introduce significant errors during model training. High-quality verification requires confirming that all data transformations support the model's learning objectives and that the AI is robust across different image variations.

3.1.3 Data Visualization Analysis

Assigned Team Members: Ayman and Nathan

For effective verification, the team must ensure that data visualizations clearly reflect the AI model's performance, making it easier to spot anomalies or

trends that might indicate issues. Ayman needs to focus on developing visuals that accurately represent both the training progress and the diagnostic results from the AI model. The team must verify that these visualizations allow for easy comparison between predicted outcomes and the actual clinical diagnoses. Ensuring that the plots are interpretable and correctly display model metrics (e.g., accuracy, loss over time, false positives/negatives) is key to validating the AI's reliability. Without this verification, it would be difficult to pinpoint areas where the model may require refinement.

3.1.4 Web Creation

Assigned Team Members: Ayman and Nathan

To meet the high standards for user experience and functionality, the Nathan and Ayman must rigorously verify that the frontend interface is not only visually appealing but also fully functional and responsive. Nathan needs to verify that the UI design facilitates a seamless interaction with the AI model, ensuring that healthcare professionals can input data and interpret results with ease. It is critical to validate the integration between the frontend and backend, ensuring that data is accurately retrieved, processed, and displayed without errors. Usability testing must be a priority during verification, as any gaps in the user interface could hinder the adoption of the tool by medical professionals. Ensuring accessibility and performance across various devices is also essential to meet the project's quality standards.

3.1.5 Users of the Application

Assigned Team Members: All

Understanding user expectations is critical for verifying that the final product meets the needs of its primary users—healthcare professionals. The team must ensure that the AI model's outputs are not only accurate but also presented in a format that healthcare users find intuitive and actionable. Verification must include gathering feedback from potential users to assess whether the AI-generated diagnostics align with clinical workflows. Additionally, the team needs to verify that the interface provides relevant and precise information without overwhelming the user with unnecessary details. Ensuring that the tool is user-friendly and caters to both specialists and general prac-

titioners is key to validating its practicality and effectiveness in real-world scenarios.

3.2 SRS Verification Plan

3.2.1 Peer Review

- **Ad Hoc Feedback:** We will conduct peer reviews where classmates and our primary reviewer provide feedback.
- **Creating Issues:** Each team member will be responsible for reading the SRS, evaluating its clarity, and checking for missing requirements and for anything missing that team member will required to make a github issue entailing that we need to look to add or fix a section of SRS.
- **Peer reviews:** Supervisor, stakeholders, and other peers will read our the document and give use feedback by doing this we can ensure its understandable to all stakeholders, especially those unfamiliar with the project.
- **Pre-Meeting Preparation:** Before meeting with the supervisor, the team will ensure that the SRS is updated and ready for review. We would this by going through the rubric and creating a checklist. This checklist would include: Questions like are all the functional and non-functional requirements included? Are the constraints in scope or out-side scope constraints are they truly constraints. Making a checklist with questions similar to these will help verify that the project is co-herent and comprehensive.
- **Post-Meeting Notes:** we will take notes on what the supervisor or TA told us and ensure that we look to add or fix whatever was men-tioned during the meeting. This will be another checklist of things we must fix not like the previous checklist that is primarily focused on asking questions.
- **Questions for the Supervisor** & **TA's:** We will ask key questions that would focus on verifying the scope, clarity, and feasibility of our SRS document. An example, could be asking if the NFRs or non-functional requirements align with the project's quality objectives.

3.3 Design Verification Plan

The design verification plan outlines the strategies and procedures that the team will use to verify the correctness and reliability of the design of our X-ray analysis and disease prediction application. This plan will serve as a guideline during the testing phase to ensure that the design meets the intended requirements and can mitigate potential risks identified by the team. The following procedures will be undertaken by the testing team during the verification process:

3.3.1 Document Review

The system's design documentation and related materials will be reviewed by each member of the testing team after the initial draft is produced. During the document review process, the testing team will:

- Ensure that the design of the system aligns with all functional and non-functional requirements, particularly those related to medical accuracy, data security, and regulatory compliance.
- Assess if the documentation accurately describes the intended functionalities and behavior of the system, including its ability to analyze X-rays, classify diseases, and provide accurate predictions. Any discrepancies between the design and requirements should be recorded and discussed further.

3.3.2 Prototype Testing

After the scheduled Proof of Concept (POC) demonstration, a prototype for each major system component should be completed. The prototypes will be assembled for system testing. During this phase, the testing team will:

- Verify the application's ability to process X-ray images accurately and consistently. Assess system performance under different load conditions, such as batch processing multiple images simultaneously.
- Test the robustness of disease prediction algorithms under various scenarios, including rare and complex cases.
- Evaluate the application's user interface for healthcare professionals, ensuring it is intuitive, provides clear predictions, and integrates well

with clinical workflows. Simulate failure conditions to verify error-handling mechanisms, such as incomplete data, corrupted files, or system downtime.

3.3.3 Design Verification Checklist

Document Review

- Ensure design aligns with all functional and non-functional requirements.
- Confirm documentation accuracy for image processing, classification, and prediction.
- Record any discrepancies from requirements.

Prototype Testing

- **System Performance**
  - Verify accurate X-ray image processing.
  - Test performance under various load conditions.
  - Check response time for predictions.
  - Ensure scalability for future usage.
- **Disease Prediction Accuracy**
  - Test accuracy with a benchmark dataset.
  - Evaluate algorithm on rare and complex cases.
  - Check consistency of predictions across runs.
- **User Interface (UI)**
  - Confirm UI is intuitive for healthcare professionals.
  - Ensure clear display of disease predictions.
  - Verify data visualization tools are easy to interpret.
  - Check seamless integration with clinical systems.
- **Failure Handling**

  - Test response to incomplete or corrupted data.
  - Verify clarity of error messages.
  - Simulate network failures for recovery and integrity.
  - Check logging for error tracking.

- **Security and Compliance**
  - Confirm encryption and patient privacy protections.
  - Verify compliance with healthcare data regulations.
  - Ensure access control mechanisms are in place.

3.4 Verification and Validation Plan Verification Plan

This section provides guideline for Quality Assurance team to validate the Verification and Validation Plan document. The following steps will be used:

- Cross checking with Functional Requirements from SRS document: VnV documentation must objectively cover all business use-cases and functional requirements verification plan.
- Review uncertenties with stakeholders: VnV plan must be reviewed by the stakeholders to ensure that the plan is feasible, realistic, and complete.
- Each test case mentioned will have to state the following:
  - What is the expected input and output of the test case.
  - How the test case is going to be implemented (Using what tools, techniques).
  - Which functional requirements, or non-functional requirements (from SRS document) does this test case belongs to.
  - Which system is being tested (Front-end, Back-end, ML model, etc.).
  - Which team is responsible for the test case. For example, a end-to-end test case might require everyone's effort

3.5 Implementation Verification Plan

The following testing techniques will be used to verify the implementation our system:

- **Static Code and Documentation walkthroughs**
  - Documentation walkthroughs will be performed by the team members, as a group, to verify that business use-cases and functional requirements are all covered.
  - Code walkthroughs/ Peer reviews will be performed by the team to ensure that the code is written as per the design documents (VNV Plan, SRS, and Development Plan).
- **Unit Testing**
  - Individual units of the software are tested in isolation from the rest of a particular system. This is done to ensure that each unit of the software is working as designed in the business use-cases.
  - Examples of unit-test includes: testing the input validation, testing the output of a function, testing button clicks, testing authenticated web pages, etc.
  - All unit tests must cover the happy path and edge cases (boundary conditions, invalid input, etc.).
- **Integration Testing**
  - Two Individual Systems's Integration are tested. This is done to ensure that the system is working together as designed in the business use-cases.
  - In the context of Neuralanalyzer, required integrations to test are: the front-end and the back-end, back-end with AWS Cognito, back-end with Neuralanalyzer ML model, back-end with TorchXRayVision etc.
  - Examples of system tests include: Testing back-end responses with different input from front-end , Testing ML prediction with different images input onto backend, etc.

  - All Integration tests are ideally to be tested in isolation, for example: front-end to back-end integration test should mock the ML model's responses.
  - All system tests must cover the happy path and edge cases (boundary conditions, invalid input, etc.).

- **System/ End-to-End Testing**
  - System testing is done to ensure that all systems are working together as designed in the business use-cases.
  - Example of 1 System test cases: Sign up on the front-end and validating the existing user on the back-end, etc.
  - All end-to-end tests must cover all business use-cases (Log-in, sign-up, image-upload, prediction, etc.), and Functional Requirements.

- **Periodic Health Checks**
  - The front-end, back-end, and ML model will be checked periodically to measure system's uptime. Measured uptime must satisfy the agrred-upon SLA (95%).
  - Periodic Health Checks will be performed regardless of the system' software version and state.

- **Manual Testing**
  - Manual testing will be performed by the team to ensure intuitive and simple UI/UX to the end-user.
  - Manual testing will only include interaction with the front-end on different devices (mobile, tablet, desktop).
  - All manual tests must cover all business use-cases (Log-in, sign-up, image-upload, prediction, etc.).

- **Machine Learning Model Validation and Testing**
  - All the Chest X-Ray data available will be split into 3 diffrent sets: Training, Validation, and Testing.
  - The ML model will be trained on the Training set, and validated on the Validation set.

  - The ML model will be tested on the Testing set, and the accuracy will be measured. The accuracy of the models will be aggreed upon by the team, and the stakeholders.

- **User Acceptance Testing**
  - After the MVP is ready, the team will invite selected stakeholders to test out the system.
  - Based on the feedback, the team will make necessary changes to the system via GitHub Issues.

3.6 Automated Testing and Verification Tools

With many testing techniques in place, the following tools will be used to automate the testing process:

- **Static Code/ Document Review via Github Pull Requests**
  - During development phase, every line of code ideally would have to be reviewd, per **Github Pull Requests**, by the team. As mentioned, pull requests onto main branch must be approved by all 4 team members.
  - If a pull request is way too long (ie: more than 500 lines of code), the pull request owner will have to break down the pull request into smaller pull requests, or the team will have a meeting to review the code.
  - Since the documentation is also version controlled, the team will review documents similar to code review via **Github Pull Requests**.

- **Static Code Analysis Tools**
  - Utilizing **Pylint** for Python code, and **ESLint** for JavaScript code To ensure that the code is written as per the PEP8 standards.
  - Pylint and ESLint will be integrated into the development environment, and will be run on every commit to the main branch via **Gtihub Actions**.

- **Tools for Unit Testing**

  - **Front-end**: Utilizing **Jest** for testing React components, and front-end data fetching functions. Jest is capable of mocking API calls and other front-end modules, isolating front-end units at test-execution time, making it a perfect tool for React unit testing.
  - **Back-end and ML plugins**: Utilizing **Pytest** for testing individual functions and layers. Pytest is capable of mocking front-end input, external service calls, and other back-end modules.

- **Tools for Integration Testing**
  - Front-end and back-end integration will be tested with **Cypress**. Certain function that requires ML model's responses can be mocked with Cypress at the api layer.
  - Back-end and ML Plugins integration will be tested with **Pytest**. Certain function that requires Front-end input can be mocked at the API layer.
  - Back-end and AWS Cognito integration will be tested with **Pytest**. Certain functions that requires Front-end input and ML models output can be mocked.

- **Tools for End-to-End Testing**
  - End-to-end testing can be done with **Cypress**. Without mocking any functions, end-to-end testing can be performed in a fashion that mimics the user's interaction with the system.
  - For example Cypress can create an account on the front-end, and validate if the account is created on AWS Cognito.

- **Tools for Periodic Health Checks**
  - **Github Actions Schedules** will be used to run health checks on running front-end and back-end.
  - Since AWS Cognito's uptime is fully mamanged by AWS, perioid health check for this service is unnecessary.

- **Test Automation Framework**
  - **GitHub Actions** will be utilized to automate the testing process.

  - GitHub Actions will be configured to run all tests (All unit tests, integration tests, and system tests) on every Pull Request, and on every commit to the main branch.

- **Machine Learning Model Validation and Testing**

  - **Pytorch** will be used to test the ML model's accuracy on the Testing set.

[What tools are you using for automated testing. Likely a unit testing framework and maybe a profiling tool, like ValGrind. Other possible tools include a static analyzer, make, continuous integration tools, test coverage tools, etc. Explain your plans for summarizing code coverage metrics. Linters are another important class of tools. For the programming language you select, you should look at the available linters. There may also be tools that verify that coding standards have been respected, like flake9 for Python. —SS]

[If you have already done this in the development plan, you can point to that document. —SS]

[The details of this section will likely evolve as you get closer to the implementation. —SS]

3.7 Software Validation Plan

[If there is any external data that can be used for validation, you should point to it here. If there are no plans for validation, you should state that here. —SS]

[You might want to use review sessions with the stakeholder to check that the requirements document captures the right requirements. Maybe task based inspection? —SS]

[For those capstone teams with an external supervisor, the Rev 0 demo should be used as an opportunity to validate the requirements. You should plan on demonstrating your project to your supervisor shortly after the scheduled Rev 0 demo. The feedback from your supervisor will be very useful for improving your project. —SS]

[For teams without an external supervisor, user testing can serve the same purpose as a Rev 0 demo for the supervisor. —SS]

[This section might reference back to the SRS verification section. —SS]

4 System Tests

4.1 Tests for Functional Requirements

This section contains the tests for the Functional Requirements. The subsections for these tests were created based on the Functional Requirements listed in the SRS document. Each test was created according to the Fit Criterion of the requirements they were covering. Traceability for these requirements and tests can be found in the traceability matrix.

4.1.1 X-ray Image Input Tests

This subsection covers Requirement #1 of the SRS document by testing that the system is able to accept chest X-ray images as input from authorized users, including healthcare professionals and patients.

1. **test-FR1-1**

   Control: Automatic

   Initial State: The system is operational, and the user is logged in as an authorized healthcare professional.

   Input: Uploading a valid chest X-ray image in DICOM format (`sample_image.dcm`).

   Output: The system accepts the image without errors and displays a confirmation message indicating successful upload.

   Test Case Derivation: Authorized users should be able to upload images in supported formats without issues.

   How the test will be performed: The test will be conducted automatically using the Cypress testing framework.

   - *Step 1*: The automated test script will simulate a user logging into the system as an authorized healthcare professional using valid credentials (`username: doctor_user`, `password: SecurePass123`).
   - *Step 2*: Upon successful authentication, the script will navigate to the image upload section located at `/upload`.
   - *Step 3*: The script will simulate the user clicking the "Upload Image" button and selecting the file `sample_image.dcm` from the directory `./test_data/`.

- *Step 4*: The script will simulate the user confirming the upload by clicking the "Submit" button.
- *Step 5*: After the upload action, the script will verify that the system displays a confirmation message on the user interface, such as "Image uploaded successfully."

2. **test-FR1-2**

   Control: Automatic

   Initial State: The system is operational, and the user is logged in as an authorized healthcare professional.

   Input: Uploading a valid chest X-ray image in DICOM format (`sample_image.dcm`).

   Output: The system stores the image correctly on the server and the API endpoint returns a success status code.

   Test Case Derivation: The backend must accurately store uploaded images and respond appropriately to API requests.

   How the test will be performed:

   - *Step 1*: The automated test script will perform the same steps as in **test-FR1-1** to upload `sample_image.dcm`.
   - *Step 2*: The network traffic will be monitored to capture the API request sent to the server during the image upload.
   - *Step 3*: The script will verify that the API request includes the correct payload, ensuring that the file `sample_image.dcm` is included under the correct parameter name (e.g., `image_file`).
   - *Step 4*: The server's response to the API request will be checked by the automated test, confirming it returns a success status code (HTTP 200 OK) and a response body with a success message (e.g., {`"message": "Upload successful", "image_id": 12345`}).
   - *Step 5*: The script will access the server's storage directory (e.g., `.../images/uploads/`) to verify that `sample_image.dcm` is stored correctly. It will check:
     - The file exists in the expected directory.
     - The file name matches the expected naming convention (e.g., `image_{image_id}.dcm`).

– The file size and checksum match those of the original file to ensure integrity.

4.1.2 Patient Symptom for diagnosis verification

This subsection covers Requirement #2 of the SRS document by testing that the system enables users to input additional patient symptoms, such as cough, chest pain, or fever.

1. **test-FR2-1**

   Control: Manual

   Initial State: The user has successfully uploaded a chest X-ray image and is on the patient information page.

   Input: Entering patient symptoms into the symptom input fields.

   Output: The system accepts the symptoms and associates them correctly with the uploaded X-ray image.

   Test Case Derivation: The system should allow input of symptoms and link them to the corresponding image.

   How the test will be performed:

   - *Step 1*: The tester uploads `patient_xray_01.dcm` and waits for the AI model to complete analysis.
   - *Step 2*: On the patient information page, the tester enters symptoms such as "cough," "chest pain," and "fever" into the symptom fields.
   - *Step 3*: The tester submits the symptoms by clicking the "Save" button.
   - *Step 4*: The system processes the symptoms and updates the patient's record.
   - *Step 5*: The tester views the generated report to verify that:
     – The symptoms are listed and associated with the diagnosis.
     – The report highlights the consistency between the symptoms and the diagnosis.

4.1.3 AI Model Inferece Tests

This subsection covers Requirement #3 of the SRS document by testing that the system analyzes chest X-ray images using a convolutional neural network (CNN)-based AI model to detect the presence or absence of specific diseases with an accuracy of 85% or higher.

1. **test-FR3-1**

   Type: Functional, Dynamic, Automatic

   Initial State: The AI model is deployed and integrated into the system, ready for analysis.

   Input: A test dataset of 1,000 chest X-ray images with known diagnoses (500 positive cases, 500 negative cases).

   Output: The AI model achieves an accuracy of at least 85%, correctly identifying at least 850 out of 1,000 cases.

   Test Case Derivation: The AI model must meet or exceed the specified accuracy threshold to be considered effective.

   How the test will be performed:

   - *Step 1*: The automated test script will load the test dataset from `./test_data/test_set/`.
   - *Step 2*: Preprocess each image according to FR17 requirements (e.g., resize to 224x224 pixels, normalize pixel values).
   - *Step 3*: Feed the preprocessed images into the AI model in batches (e.g., batch size of 32).
   - *Step 4*: Record the AI model's predictions for each image.
   - *Step 5*: Compare the predictions with the ground truth labels from `fr3_test_set_labels.csv`.
   - *Step 6*: Calculate performance metrics including accuracy, precision, recall, and ROC curves.
   - *Step 7*: Verify that the accuracy is at least 85%, using assertions.

4.2 Software Validation Plan

- **Developers**: After a feature is developed, the internal validation plan will includes the following steps
  - Developers will write tests, and come up with acceptance criteria for the feature.
  - The test caes then will be reviewd (on Github), and ran on Github Actions on every Pull Requests.
  - The software is consider validated, if all tests pass, including all old tests, and newly written tests.
  - If the test cases failed, the developer will need to fix the code, and notify the team to re-approve the Pull Requests.
- **Dr.Mehdi Moradi**: Since Dr.Mehdi Moradi is the primary stakeholder of the project, the team will report project progress to him on a bi-weekly basis. The team will also invite Dr.Mehdi to internal testing sessions.
- **Other Stakeholders**: During the development cycle of the application, User Acceptance Testing (UAT) will be performed periodically by the team. UATs will be performed to ensure that the system is working as expected and is ready for the stakeholders to test. The UAT will be performed in the following steps:
  - Coming up with business use-cases, by stakeholders, that the system must satisfy.
  - Selected stakeholders will be invited to use and test the beta system.
  - Stakeholders interaction with the system will be observed and feedback will be collected.
  - If the interactions and feedbacks are positive, the current system will be deployed to production. However, if the feedbacks are mostly negative, the team will make necessary changes to the system via GitHub Issues, and perform another round of UAT.

4.2.1 Patient Condition Progression Between Scans

This subsection covers Requirement #4 of the SRS document by testing that the system indicates whether a patient's condition has improved, worsened, or remained stable between scans.

1. **test-FR4-1**

   Type: Functional, Dynamic, Manual

   Initial State: The patient has previous chest X-ray images and analysis results stored in the system.

   Input: Uploading a new chest X-ray image for the same patient.

   Output: The system analyzes the new image, compares it with previous scans, and indicates the progression status (improved, worsened, or stable).

   Test Case Derivation: The system should accurately assess and report changes in the patient's condition over time.

   How the test will be performed:

   - *Step 1*: The tester logs into the system as a healthcare professional and selects the patient with prior scans.
   - *Step 2*: Upload the new chest X-ray image `xray_current.dcm`.
   - *Step 3*: Wait for the system to analyze the new image and retrieve previous analysis results.
   - *Step 4*: The system compares the new image with the previous one using progression algorithms.
   - *Step 5*: Review the progression status indicated by the system (e.g., "Condition has worsened").
   - *Step 6*: Verify the accuracy of the assessment by comparing it with known and reliable clinical data.

4.2.2 Visual Aid Generation Tests

This subsection covers Requirement #5 of the SRS document by testing that the system generates visual aids by highlighting affected areas on the chest X-ray images.

1. **test-FR5-1**

   Type: Functional, Dynamic, Manual

   Initial State: The system has completed analysis on a chest X-ray image with detected abnormalities.

   Input: Accessing the analysis results and viewing the image.

   Output: The image displays highlighted areas indicating the locations of the detected abnormalities.

   Test Case Derivation: Visual aids help clinicians quickly identify areas of concern on the images.

   How the test will be performed:

   - *Step 1*: The tester logs into the system and navigates to the patient's analysis results for `abnormal_xray.dcm`.
   - *Step 2*: Open the analyzed image with visual aids.
   - *Step 3*: Examine the highlighted areas on the image.
   - *Step 4*: Compare the highlighted areas with expected abnormalities based on known data.
   - *Step 5*: Assess whether the visual aids accurately represent the detected abnormalities.
   - *Step 6*: Document observations and any discrepancies.

2. **test-FR5-2**

   Control: Automatic

   Initial State: The system has completed analysis on a chest X-ray image (`normal_patient_xray.dcm`) with no detected abnormalities.

   Input: Accessing the analysis results for `normal_patient_xray.dcm`.

   Output: The image displays no highlights or visual markers.

   Test Case Derivation: The system should not produce false positives by highlighting areas in normal images.

   How the test will be performed:

   - *Step 1*: The automated test script will process `normal_patient_xray.dcm` through the system.

- *Step 2*: The system analyzes the image and generates an output image.
- *Step 3*: The script retrieves the output image from `.../images/outputs/`.
- *Step 4*: The script uses image comparison techniques to compare the output image with the original.
- *Step 5*: The script checks for any added visual markers or differences.
- *Step 6*: The script asserts that no highlights are present.

4.2.3 Structured Report Generation Tests

This subsection covers Requirement #6 of the SRS document by testing that the system produces a structured, human-readable report summarizing key findings, disease detection results, and progression status.

1. **test-FR6-1**

   Control: Automatic

   Initial State: The system has completed analysis on a patient's chest X-ray image, and all relevant data is available. The report generation module is integrated and configured to use an LLM API (e.g., BERT).

   Input: Request to generate a structured report summarizing the analysis.

   Output: A human-readable report containing key findings, detected diseases with confidence levels, progression status, and any input symptoms.

   Test Case Derivation: The system should produce comprehensive reports without errors, leveraging LLMs where appropriate.

   How the test will be performed:

   - *Step 1*: The automated test script will initiate the report generation process via the API endpoint `/generate_report` and corresponding patient ID.
   - *Step 2*: The script will monitor the API call to the LLM service, ensuring that:

  - The API token is included and valid.
  - The prompt sent to the LLM is correctly formatted and contains all necessary information.
- *Step 3*: The script will check the response from the LLM service, verifying:
  - The response status code is success (e.g., HTTP 200 OK).
  - The content includes the structured report.
- *Step 4*: The script will parse the generated report to ensure it includes:
  - Key findings from the AI analysis.
  - Detected diseases with confidence levels.
  - Progression status compared to previous scans.
  - Input symptoms provided by the user.
- *Step 5*: The script will verify that the report format adheres to predefined templates or standards (e.g., headings, sections).
- *Step 6*: The script will check for any placeholders or missing information that could indicate issues with LLM integration.
- *Step 7*: The test passes if the report is complete, accurate, and correctly formatted; otherwise, it fails.

2. **test-FR6-2**

   Control: Manual

   Initial State: The system is operational, and a report has been generated for a patient using the LLM integration.

   Input: The generated report for review.

   Output: Confirmation that the report does not contain misinformation and accurately reflects the patient's analysis results.

   Test Case Derivation: Reports must be accurate and free from misinformation to ensure patient safety.

   How the test will be performed:

   - *Step 1*: The tester (a qualified healthcare professional) will retrieve the generated report for patient ID `12345`.

- *Step 2*: The tester will review the report in detail, verifying:
  - All clinical findings are accurately reported.
  - Diagnoses match the AI model's outputs.
  - Confidence levels are correctly stated.
  - Progression status aligns with the comparative analysis.
- *Step 3*: The tester will cross-reference the report with the raw data and analysis results.
- *Step 4*: The tester will check for any signs of misinformation, such as incorrect interpretations or unsupported conclusions.
- *Step 5*: The tester will document any discrepancies or errors found.
- *Step 6*: The test passes if the report is accurate and free of misinformation; otherwise, it fails.

4.2.4 Patient Data Storage Tests

This subsection covers Requirement #7 of the SRS document by testing that the system securely stores patient data, including images and analysis results, for future reference.

1. **test-FR7-1**

   Control: Automatic

   Initial State: The system's secure database and storage solutions are operational. Access to patient data is regulated by role-based access controls.

   Input: Patient data including images (`test_patient_image.dcm`), analysis results, and reports to be stored.

   Output: Data is securely stored in the database and can be retrieved accurately by authorized users. Unauthorized access is prevented.

   Test Case Derivation: The system must ensure data integrity and security for patient information.

   How the test will be performed:

   - *Step 1*: The automated test script will simulate a user with appropriate permissions uploading patient data:

  - Image file: `test_patient_image.dcm`.
  - Analysis results: Generated by the AI model.
  - Report: Generated as per FR6.
- *Step 2*: The script will verify that the data is stored in the secure database:
  - Confirm entries in the database tables for patients, images, analyses, and reports.
  - Verify that images are stored in the secure storage directory (e.g., `/secure_storage/patient_images/`).
- *Step 3*: The script will retrieve the stored data using authorized credentials and compare it with the original inputs to ensure integrity:
  - Check that the image file retrieved matches the original file (e.g., via checksum comparison).
  - Verify that analysis results and reports are accurate.
- *Step 4*: The script will attempt to access the data using unauthorized credentials or as an unauthorized user:
  - Expect access to be denied.
  - Verify that appropriate error messages are displayed.
- *Step 5*: The script will check security measures:
  - Data at rest is encrypted (e.g., verify encryption settings in the database).
  - Access logs are updated with the retrieval attempts.
- *Step 6*: The test passes if data integrity is maintained, authorized access is successful, and unauthorized access is prevented.

4.2.5 Alert Mechanism Tests

This subsection covers Requirement #8 of the SRS document by testing that the system generates alerts to clinicians when significant changes in a patient's condition are detected.

1. **test-FR8-1**

   Control: Automatic

Initial State: The system has prior scans and analysis results for corresponding patient ID. Alerting mechanisms via email, SMS, and in-app notifications are configured.

Input: A new scan (`patient_xray_alert.dcm`) showing significant changes exceeding predefined thresholds.

Output: The system generates and delivers alerts to clinicians indicating the significant change.

Test Case Derivation: Clinicians should be promptly notified of critical changes in a patient's condition.

How the test will be performed:

- *Step 1*: The automated test script will upload `xray_alert.dcm`.
- *Step 2*: The system analyzes the new scan and compares it with previous scans.
- *Step 3*: The system detects changes exceeding thresholds (e.g., lesion size increase by 30%).
- *Step 4*: The system generates an alert message containing:
  - Patient ID and relevant details.
  - Description of the significant change.
  - Urgency level or recommended actions.
- *Step 5*: The system sends the alert via configured channels:
  - Email to the clinician's registered email address.
  - SMS to the clinician's registered phone number.
  - In-app notification within the clinician's dashboard.
- *Step 6*: The script will monitor each channel to verify delivery:
  - Check the email inbox for the alert message.
  - Use a mock SMS gateway to confirm SMS delivery.
  - Log into the system as the clinician to verify the in-app notification.
- *Step 7*: The script will record the time taken from detection to alert delivery to ensure it meets timeliness requirements (e.g., within 5 minutes).

4.2.6 Treatment Plan Adjustment Tests

This subsection covers Requirement #9 of the SRS document by testing that the system allows healthcare professionals to adjust a patient's treatment plan based on analysis results.

1. **test-FR9-1**

   Control: Manual

   Initial State: A healthcare professional is logged into the system and has access to patient ID `12345`'s analysis results.

   Input: Adjustment to the patient's treatment plan based on the analysis results.

   Output: The system allows the professional to modify the treatment plan, linking the changes to the corresponding X-ray analysis results.

   Test Case Derivation: Clinicians should be able to update treatment plans within the system, ensuring continuity of care.

   How the test will be performed:

   - *Step 1*: The tester logs into the system as a clinician with credentials.
   - *Step 2*: The tester navigates to patient's profile and reviews the analysis results.
   - *Step 3*: The tester accesses the treatment plan section and makes adjustments:
     - Changes medication dosage.
     - Adds a new therapy recommendation.
   - *Step 4*: The tester saves the changes.
   - *Step 5*: The system validates the changes according to clinical guidelines.
   - *Step 6*: The system updates the patient's treatment plan and logs the changes with timestamps and the clinician's ID.
   - *Step 7*: The tester verifies that the updated treatment plan is correctly reflected in the patient's records.

- *Step 8*: The tester checks that the changes are linked to the latest analysis results.
- *Step 9*: The test passes if the treatment plan is updated accurately and linked appropriately; otherwise, it fails.

4.2.7 Disclaimer Message Tests

This subsection covers Requirement #10 of the SRS document by testing that the system allows patients to upload their own chest X-ray images for analysis, providing appropriate disclaimers.

1. **test-FR10-1**

   Control: Automatic

   Initial State: A patient user account (`username: patient_user`, `password: PatientPass123`) is registered and the user is logged out.

   Input: The patient logs in, uploads a chest X-ray image (`patient_image.dcm`), and requests analysis.

   Output: The system displays an appropriate disclaimer, requires acknowledgment before proceeding, accepts the image, and provides analysis results.

   Test Case Derivation: Patients should be informed about limitations and must consent before using the service.

   How the test will be performed:

   - *Step 1*: The automated test script simulates the patient logging into the system.
   - *Step 2*: The script navigates to the "Upload Image" section and uploads an image
   - *Step 4*: The script verifies that a disclaimer is presented:
     - Check that a modal or page displays the disclaimer text, including:
       * The analysis is not a substitute for professional medical advice.
       * The need to consult a healthcare professional.
       * Information about data usage and privacy.

  - Ensure that the "I Agree" or acknowledgment checkbox/button is present.

- *Step 5*: The script attempts to proceed without acknowledging the disclaimer:
  - Click "Continue" without checking "I Agree".
  - Verify that the system prevents progression and displays a prompt to acknowledge the disclaimer.
- *Step 6*: The script acknowledges the disclaimer:
  - Check the "I Agree" box.
  - Click "Continue".
  - Verify that the system accepts the acknowledgment and proceeds to process the image.
- *Step 7*: The script monitors the analysis process:
  - Check for a progress indicator or status message.
  - Wait for the analysis to complete.
- *Step 8*: After analysis completion, the script verifies that results are displayed:
  - Confirm that detected diseases and confidence levels are shown.
  - Ensure that a reminder is present advising consultation with a healthcare professional.
- *Step 9*: The script attempts to access features restricted to clinicians:
  - Try to access the "Patient Records" section.
  - Verify that access is denied with an appropriate message.
- *Step 10*: The test passes if all steps function as expected and the disclaimer process cannot be bypassed; otherwise, it fails.

4.2.8 Role Based Access Control Tests

This subsection covers Requirement #11 of the SRS document by testing that the system enforces role-based access control, ensuring users only access permitted features.

1. **test-FR11-1**

   Control: Automatic

   Initial State: The system is operational with user accounts assigned different roles:

   - Physician: `username: physician_user`, `password: PhysicianPass123`.
   - Radiologist: `username: radiologist_user`, `password: RadiologistPass123`.
   - Patient: `username: patient_user`, `password: PatientPass123`.

   Input: Users attempt to access features outside their permissions.

   Output: Access is granted only to features appropriate for each role; unauthorized access is denied with an appropriate error message.

   Test Case Derivation: Role-based access control is essential for security and compliance.

   How the test will be performed:

   - *Step 1*: For each user role, the automated test script will perform the following steps:
     - **Physician User**:
       * Log in using physician credentials.
       * Access permitted features:
         · View and edit assigned patient records.
         · Adjust treatment plans.
         · Verify access is granted and functions correctly.
       * Attempt to access restricted features:
         · Administrative settings.
         · Other clinicians' patient records.
         · Verify access is denied with appropriate error messages.
       * Log out.
     - **Radiologist User**:
       * Log in using radiologist credentials.
       * Access permitted features:
         · View imaging studies.

        - Perform image analyses.
        - Verify access is granted and functions correctly.
      * Attempt to access restricted features:
        - Modify treatment plans.
        - Access administrative settings.
        - Verify access is denied with appropriate error messages.
      * Log out.
    - **Patient User**:
      * Log in using patient credentials.
      * Access permitted features:
        - View own medical records.
        - Upload images for analysis.
        - Verify access is granted and functions correctly.
      * Attempt to access restricted features:
        - Other patients' records.
        - Clinical tools and analysis features.
        - Administrative settings.
        - Verify access is denied with appropriate error messages.
      * Log out.
  - *Step 2*: The script will record all access attempts and the system's responses.
  - *Step 3*: The script will verify that:
    – Authorized actions are allowed without errors.
    – Unauthorized actions are blocked with clear error messages.
  - *Step 4*: The test passes if access controls function correctly for all roles; otherwise, it fails.

4.2.9 Preservation of Original Data Tests

This subsection covers Requirement #12 of the SRS document by testing that the system preserves the original chest X-ray images by creating copies for analysis.

1. **test-FR12-1**

   Control: Automatic

   Initial State: A chest X-ray image (`original_image.dcm`) is uploaded and stored in the system's image repository (`/var/www/images/uploads/`).

   Input: Initiation of the AI model analysis on the uploaded image.

   Output: The system creates a new copy of the image for analysis, preserving the original unaltered.

   Test Case Derivation: Data integrity is maintained by not altering original images.

   How the test will be performed:

   - *Step 1*: The automated test script uploads `original_image.dcm` to the system:
     - Use API endpoint `/api/upload` with authorized credentials.
     - Record the file size and compute the checksum of `original_image.dcm`.
   - *Step 2*: The script initiates the analysis process:
     - Call the analysis API endpoint `/api/analyze` with the image ID returned from the upload.
   - *Step 3*: The script monitors the processing directory (`.../images/processing/`):
     - Verify that a copy of the image (`processing_image.dcm`) is created for analysis.
     - Record the file size and compute the checksum of `processing_image.dcm`.
   - *Step 4*: The script compares the original and copied images:
     - Confirm that the checksums of the two files match, ensuring the copy is identical before processing.
   - *Step 5*: After analysis completion, the script re-computes the checksum of `original_image.dcm`:
     - Verify that the checksum matches the original value from Step 1.
     - Confirm that the file size remains unchanged.
   - *Step 6*: The script verifies that all processing operations were performed on `processing_image.dcm`:

  - Check logs or metadata to confirm the processed image's ID matches `processing_image.dcm`.
- *Step 7*: The test passes if the original image is unmodified and a copy was used for analysis; otherwise, it fails.

4.2.10 Multiple Modality Support Tests

This subsection covers Requirement #13 of the SRS document by testing that the system supports multiple medical imaging modalities beyond chest X-rays.

1. **test-FR13-1**

   Control: Automatic

   Initial State: The system is configured to accept multiple medical imaging modalities, and processing modules for CT and MRI images are operational.

   Input: Uploading a CT scan image (`sample_ct_scan.dcm`) and an MRI image (`sample_mri_image.dcm`).

   Output: The system accepts and processes each image correctly, providing accurate analysis results specific to each modality.

   Test Case Derivation: The system should handle different imaging modalities appropriately.

   How the test will be performed:

   - *Step 1*: The automated test script logs into the system with clinician credentials.
   - *Step 2*: The script uploads `sample_ct_scan.dcm`:
     - Use the API endpoint `/api/upload` with the image file.
     - Include metadata indicating the modality (if not automatically detected).
   - *Step 3*: The script verifies that the system recognizes the image as a CT scan:
     - Check the response from the server for confirmation.
     - Verify that the image is stored in the appropriate directory (e.g., `/images/ct_scans/`).

- *Step 4*: The script initiates the analysis process for the CT scan:
  - Call the analysis API endpoint `/api/analyze` with the CT image ID.
- *Step 5*: The script monitors the analysis pipeline to ensure the CT-specific processing module is used:
  - Check logs or system messages indicating the module invoked.
- *Step 6*: Upon analysis completion, the script retrieves the results:
  - Verify that the results are appropriate for CT images.
  - Compare results against expected findings for `sample_ct_scan.dcm`.
- *Step 7*: The script repeats Steps 2-6 for `sample_mri_image.dcm`, ensuring that:
  - The MRI image is correctly recognized.
  - The MRI-specific analysis module is used.
  - Results are accurate and modality-specific.
- *Step 8*: The test passes if both images are processed correctly with accurate results; otherwise, it fails.

4.2.11 Regular AI Model Update Tests

This subsection covers Requirement #14 of the SRS document by testing that the system can update the AI model regularly without disrupting ongoing services.

1. **test-FR14-1**

   Control: Automatic, ¡anual

   Initial State: The system is functional with the current AI model version (`model_v1.0`). A new AI model version (`model_v1.1`) is ready for deployment.

   Input: Deployment of the updated AI model to the system generated by the .ipynb file.

   Output: The system integrates the new AI model seamlessly without disrupting ongoing services and continues processing user requests during the update.

Test Case Derivation: Regular updates should not impact system availability.

How the test will be performed:

- *Step 1*: The automated test script triggers the deployment of `model_v1.1` via the CI/CD pipeline:
  - Use deployment scripts to initiate the update.
- *Step 2*: While the deployment is in progress, the script simulates continuous user activity:
  - Upload test images at regular intervals (e.g., every 30 seconds).
  - Initiate analysis requests for each uploaded image.
- *Step 3*: The script monitors system performance metrics:
  - Response times for API calls.
  - Error rates or any failed requests.
  - Server resource utilization (CPU, GPU, memory).
- *Step 4*: After deployment completion, the script verifies that the new model is active:
  - Check the model version via a dedicated API endpoint (`/api/model_version`).
  - Confirm that `model_v1.1` is returned.
- *Step 5*: The script analyzes the results from test images submitted during deployment:
  - Verify that all analysis requests were processed successfully.
  - Compare results before and after the model update for consistency or expected improvements.
- *Step 6*: The script checks for any logged errors or warnings during deployment.
- *Step 7*: The test passes if there are no service disruptions, the new model is correctly integrated, and all user requests are handled successfully; otherwise, it fails.

4.2.12 Image Preprocessing Tests

This subsection covers Requirement #15 of the SRS document by testing that the system preprocesses chest X-ray images by normalizing pixel values and resizing images to the input dimensions required by the AI model.

1. **test-FR15-1**

   Control: Automatic

   Initial State: The preprocessing module is operational and correctly configured.

   Input: A batch of raw chest X-ray images with varying resolutions and pixel intensity ranges, located in `./test_data/preprocessing_test_set/`.

   Output: Preprocessed images resized to 224x224 pixels with normalized pixel values suitable for input into the AI model.

   Test Case Derivation: Proper preprocessing ensures consistency and compatibility with the AI model.

   How the test will be performed:

   - *Step 1*: The automated test script loads the batch of raw images:
     - Read images from `./test_data/preprocessing_test_set/`.
     - Record original metadata for each image (dimensions, pixel value ranges).
   - *Step 2*: The script passes each image through the preprocessing module:
     - Use the preprocessing API or function.
     - Capture any preprocessing logs or outputs.
   - *Step 3*: For each preprocessed image, the script verifies:
     - Image Dimensions:
       * Confirm that the image dimensions are 224x224 pixels.
     - Pixel Value Normalization:
       * Check that pixel values are within the expected range (e.g., 0 to 1).
   - *Step 4*: The script compares the preprocessed images to expected outputs:

  - Use reference images or calculated expected values.
  - Compute the difference between preprocessed images and references.
  - Verify that any differences are within acceptable tolerances.
- *Step 5*: The test passes if all images meet the specified preprocessing requirements; otherwise, it fails.

2. **test-FR15-2**

   Control: Automatic

   Initial State: The preprocessing module is operational.

   Input: An image file with an unsupported format (`invalid_image.bmp`) and a corrupted image file (`corrupted_image.dcm`).

   Output: The system handles the errors gracefully without crashing and logs informative error messages.

   Test Case Derivation: The system should be robust against invalid inputs and maintain stability.

   How the test will be performed:

   - *Step 1*: The automated test script attempts to preprocess `invalid_image.bmp`:
     - Pass the file to the preprocessing module.
     - Monitor the module's response.
   - *Step 2*: The script verifies that the preprocessing module:
     - Detects the unsupported file format.
     - Does not crash or throw unhandled exceptions.
     - Logs an error message indicating the issue, including the file name and reason (e.g., "Unsupported file format: .bmp").
   - *Step 3*: The script repeats the process with `corrupted_image.dcm`:
     - Attempt to preprocess the corrupted file.
     - Monitor for exceptions or errors.
   - *Step 4*: The script verifies that the preprocessing module:
     - Detects the file corruption.
     - Handles the error without crashing.
     - Logs an informative error message (e.g., "Failed to read image data: corrupted or incomplete file").

4.3 Tests for Nonfunctional Requirements

[The nonfunctional requirements for accuracy will likely just reference the appropriate functional tests from above. The test cases should mention reporting the relative error for these tests. Not all projects will necessarily have nonfunctional requirements related to accuracy. —SS]

[For some nonfunctional tests, you won't be setting a target threshold for passing the test, but rather describing the experiment you will do to measure the quality for different inputs. For instance, you could measure speed versus the problem size. The output of the test isn't pass/fail, but rather a summary table or graph. —SS]

[Tests related to usability could include conducting a usability test and survey. The survey will be in the Appendix. —SS]

[Static tests, review, inspections, and walkthroughs, will not follow the format for the tests given below. —SS]

[If you introduce static tests in your plan, you need to provide details. How will they be done? In cases like code (or document) walkthroughs, who will be involved? Be specific. —SS]

4.3.1 Area of Testing1

Title for Test

1. test-id1

   Type: Functional, Dynamic, Manual, Static etc.

   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:

4.3.2 Area of Testing2

...

4.4 Traceability Between Test Cases and Requirements

[Provide a table that shows which test cases are supporting which requirements. —SS]

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]

Document is not completed yet.

[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 Unit Testing Scope

The modules which the unit test focuses on will be divided into two categories: **Front-end** and **Back-end**.

The following modules are considered outside the scope of unit testing:

Third-Party Modules: Any third-party libraries or frameworks integrated into the application will not be subjected to unit testing. This includes modules developed by external sources, such as libraries for data visualization or authentication. These modules will be relied upon for their documented functionality, but no direct verification will be performed.

5.3 Rationale for Ranking

The prioritization of modules for testing is based on the following criteria:

- **Impact on Core Functionality**: Modules that are integral to the application's main features will receive higher priority for unit testing.
- **User Experience**: Features that directly affect user interactions and overall experience will be tested more rigorously.
- **Risk Assessment**: Modules with higher associated risks, such as security and data handling, will be prioritized to mitigate potential issues.

By focusing our testing efforts on high-impact modules, we aim to ensure a robust and reliable application while efficiently utilizing our resources.

5.4 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]

For unit testing, we're dividing the process into frontend and backend sections. UI/UX interactions, such as buttons and links, will be tested manually to ensure they function as expected. Backend functions, especially those involving data fetching, will be tested automatically.

Frontend testing: UI/UX elements such as buttons, links, and other interactive components will undergo manual testing to confirm that they work as intended from a user perspective. This includes verifying that each interactive element responds accurately and meets usability standards.

Backend testing: functions that handle data fetching and processing will be validated through automated tests. These tests will simulate various scenarios to ensure that data retrieval, processing, and any associated logic work as expected, helping us catch issues early and maintain data integrity. This combination of manual and automated testing ensures comprehensive coverage and reliability across the application. (Possibly by Pytest library)

5.4.1 Front End

[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]

The frontend testing will encompass all possible user interactions, covering any element that the user can click or interact with. This includes, but is not limited to, buttons, links, dropdowns, modals, and form fields. Each interactive element will be manually tested to confirm that it behaves as expected in various scenarios, such as different screen sizes and user flows. This ensures that the user experience is smooth, intuitive, and functional across all devices and use cases. Anything that deals with username/password/upload/download should be securely designed.

1. FR-FE-1
   (FR-7, FR-13)

   Type: Manual

   Initial State: Automated

   Input: Username & Password

   Output: User is successfully logged in and redirected to the home page if the username and password are correct. If incorrect, an error message is displayed, preventing login.

Test Case Derivation: The output is expected to confirm whether the login process works correctly. A successful login with correct credentials should redirect the user to the home page, while incorrect credentials should display an error message, preventing access. This expected outcome ensures that the system only grants access to users with valid login information, maintaining security and usability standards.

How test will be performed: An automated

[Refrence to Login Form UI]

2. FR-FE-2
(FR-7, FR-13)

   Type: Automated

   Initial State: Register Page

   Input: All form fields: Include but not limited to username/email, first name, last name, occupation, organization, location, password and confirmed password.

   Output: User is successfully registered and redirected to the login page if all fields are valid; otherwise, appropriate error messages are displayed for invalid inputs.

   Test Case Derivation: This test verifies that the registration functionality correctly processes all required fields. The expected output is based on whether the provided data meets validation criteria and does not contain duplicates.

   How test will be performed: Automated scripts will fill in all form fields with valid and invalid data, submitting the registration form. The outputs will be checked to ensure successful registration for valid data and the correct error messages for invalid inputs. [Refrence to Register Form UI]

3. FR-FE-3
(FR-1, FR-12, FR-15)

   Type: Automated

   Initial State: Upload Page

   Input: One or more pictures in various formats (e.g., JPG, PNG, etc.) and within constrained size.

Output: Images are successfully uploaded, and confirmation messages are displayed. For unsupported formats, appropriate error messages are shown.

Test Case Derivation: This test verifies the image upload functionality, ensuring that the system accepts valid image formats while rejecting unsupported ones. The expected outcome is determined by the format and size of the uploaded images.

How test will be performed: Automated scripts will upload a mix of supported and unsupported image formats and sizes to the upload page. The results will be checked for successful uploads or correct error messages for unsupported formats. [Refrence to Upload Page UI]

4. FR-FE-4
(FR-3)

   Type: Automated

   Initial State: Upload Page (After Upload)

   Input: Symptoms

   Output: Symptoms are successfully recorded and displayed on the user interface, confirming submission; an error message appears for invalid input.

   Test Case Derivation: This test verifies that the system correctly processes and records user-submitted symptoms after an image upload.

   How test will be performed: Automated scripts will input various symptom descriptions, including valid and invalid examples, to the appropriate field. The results will be checked to ensure successful recording or the display of correct error messages for invalid entries.

5. FR-FE-5
(FR-1, FR-12, FR-15)

   Type: Automated

   Initial State: Upload Page (Any)

   Input: Delete, Save or Submit

   Output: User's symptoms and uploaded images are deleted, User's symptoms and uploaded images are saved under profile; User's symptoms and uploaded images are submitted to backed. Confirmation

message will be displayed or if there are validation errors, appropriate error messages are shown.

Test Case Derivation: The expected output depends on the action taken, ensuring that data is either saved locally or submitted to the backend as appropriate.

How test will be performed: Automated scripts will execute the Save or Submit actions and subsequently assess the actual outcomes against the expected results to ensure compliance with the specified requirements.

6. FR-FE-6
(FR-13)

   Type: Manual

   Initial State: Home Page

   Input: Actions to navigate to other pages. (Upload, Profile, Logout-redirects to Login page etc)

   Output: The user is successfully redirected to the selected page.

   Test Case Derivation: This test ensures that navigation actions from the Home Page function correctly, leading users to the appropriate pages without errors.

   How test will be performed: All the buttons will be manually tested, and observed for functionality.

7. FR-FE-7
(FR-1, FR-12, FR-15)

   Type: Automated

   Initial State: Report Page (Saved)

   Input: Delete or Submit

   Output: Report page will be deleted or Report page will go to submitted state.

   Test Case Derivation: This test verifies that the system correctly processes the actions of deleting or submitting a report, ensuring appropriate status changes and user feedback.

   How test will be performed: Automated scripts will simulate the selection of "Delete" and "Submit" actions, checking for the expected

outcomes—either the removal of the report or the confirmation of submission.

8. FR-FE-8
   (FR-5 FR-6 FR-7 FR-11 FR-13)

   Type: Automated

   Initial State: Report Page (Submitted)

   Input: N/A

   Output: Fetch and display the report from the database; otherwise, an error message indicates an invalid or expired session.

   Test Case Derivation: This test ensures the system retrieves and displays reports correctly and handles invalid Session ID scenarios (Security).

   How test will be performed: Automated scripts will input valid and invalid datasets, verifying the correct report display.

9. FR-FE-9
   (FR-1, FR-12, FR-15)

   Type: Automated

   Initial State: Report Page (Saved)

   Input: Delete or Submit

   Output: If 'Delete' is selected, the report page is removed with confirmation. If 'Submit' is selected, it transitions to a submitted state with confirmation.

   Test Case Derivation: This test verifies that the system correctly processes the actions of deleting or submitting a report, ensuring appropriate status changes and user feedback.

   How test will be performed:Automated scripts will simulate the selection of Delete and Submit actions, checking for the expected outcomes—either the removal of the report or the confirmation of submission.

10. FR-FE-10
    (FR-7 FR-10 FR-13)

Type: Automated

Initial State: Profile Page (Patient)

Input: N/A

Output: Displays all relevant information, such as the treatment plan and reports.

Test Case Derivation: This test verifies that the Profile Page correctly displays all pertinent patient information, ensuring data accuracy and completeness.

How test will be performed: Automated scripts will access the Profile Page and check that all expected information, including the treatment plan and reports, is displayed correctly.

11. FR-FE-11
(FR-4 FR-7 FR-8 FR-10 FR-13)

    Type: Automated

    Initial State: Profile Page (Doctor)

    Input: Actions to edit a patient's profile, including treatment plan, comments, etc.

    Output: Displays all patients and their respective information.

    Test Case Derivation: This test ensures that the Profile Page for the doctor accurately displays all patients and their details after any editing actions are performed.

    How test will be performed: Automated scripts will simulate actions to edit patient profiles and verify that the updated information is displayed correctly for all patients.

5.4.2 Back End

The backend testing will focus on validating the accuracy, reliability, and performance of all server-side functions and data processes. This includes testing data-fetching endpoints, ensuring secure and correct handling of requests and responses, and verifying data integrity in various scenarios. Automated tests will simulate multiple use cases to ensure that business logic, such as data processing, authentication, and error handling, works as expected. Performance tests will also be conducted to confirm that the backend maintains

speed and efficiency under load, supporting seamless interactions between the frontend and backend components. This comprehensive backend testing aims to prevent data inconsistencies and ensure robust, secure, and scalable operations.

1. FR-BE-1
   (FR-16)

   Type: Automated

   Initial State: Start

   Input: N/A

   Output: Initialize backed systems and returns result.

   Test Case Derivation: This test verifies that the backend systems initialize correctly and that the expected result is returned, indicating successful setup. (Go to S0)

   How test will be performed: Automated scripts will execute the initialization process and validate that the system returns the expected results without errors.

2. FR-BE-2
   (FR-1 FR-2 FR-12)

   Type: Automated

   Initial State: S0

   Input: Unloaded Form data (Images, symptoms etc)

   Output: Transition to S1 if the upload is successful; otherwise, remain at S0.

   Test Case Derivation: This test verifies that the system successfully transitions from S0 to S1 upon a successful upload of form data and stays at S0 if the upload fails.

   How test will be performed: Automated scripts will simulate the upload of form data and check for the correct transition to S1 when the upload is successful or confirm that the system remains at S0 when the upload fails.

3. FR-BE-3
   (FR-7 FR-15)

   Type: Automated

   Initial State: S1

   Input: Uploaded Form data

   Output: If the form data is valid, transition to S2; otherwise, return to S0.

   Test Case Derivation: This test verifies that the upload form processes data correctly, ensuring successful transitions between states based on the upload outcome.

   How test will be performed: Automated scripts will simulate form submissions with valid and invalid data, checking that the system transitions and returns the appropriate error code for failures.

4. FR-BE-4
   (FR-14)

   Type: Automated

   Initial State: S2

   Input: Valid Form data

   Output: Processed and normalized form data (in the case of image data) if success transition into S3, else failure state.

   Test Case Derivation: This test verifies that the system accurately processes and normalizes valid form data, ensuring that the output meets the specified requirements for further use.

   How test will be performed: Automated scripts will input valid form data and verify that the system produces correctly processed and normalized output, checking for data integrity and conformity to expected formats.

5. FR-BE-5
   (FR-3)

   Type: Automated

   Initial State: S3

Input: Normalized data

Output: Diagnosis prediction data if success transition into S4, else failure state.

Test Case Derivation: If the normalized data is successfully processed by the model, transition to S4; otherwise, enter a failure state.

How test will be performed: Automated scripts will input normalized data into the model and verify that the diagnosis prediction data is produced correctly. The scripts will check for a successful transition to S4 or confirm entry into a failure state based on the processing outcome.

6. FR-BE-6
(FR-4 FR-5 FR-6 FR-8)

   Type: Automated

   Initial State: S4

   Input: Diagnosis prediction data

   Output: Generated report data if sucess transition into S5, else failure state.

   Test Case Derivation: This test will generate a concise report based on the diagnosis prediction data provided.

   How test will be performed: Automated scripts will input the diagnosis prediction data and verify that a report is generated successfully. The scripts will check for a successful transition to S5 or confirm entry into a failure state if the report generation fails.

7. FR-BE-7
(FR3)

   Type: Automated

   Initial State: Failure

   Input: Error code

   Output: Appropriate error message and system logs.

   Test Case Derivation: This is the exit state of the process, indicating that an error has occurred, and the system should provide relevant feedback based on the error code received.

How test will be performed: Automated scripts will input various error codes and verify that the system generates the correct error messages and logs the details appropriately. The test will ensure that the system behaves as expected in the failure state.

8. FR-BE-8
   (FR-11 FR-13)

   Type: Automated

   Initial State: S5

   Input: Generated report data

   Output: A confidence score; if low, transition to S6 (if the user is under a doctor); otherwise, proceed to End (if the doctor has approved or the user is not under any doctor).

   Test Case Derivation: The confidence score is derived from the generated report data, determining the subsequent state based on its value and the user's doctor status.

   How test will be performed: Automated scripts will process the generated report data, calculate the confidence score, and verify the correct state transition based on the score and user status.

9. FR-BE-9
   (FR-9 FR-13)

   Type: Automated

   Initial State: S6

   Input: Edited data (Doctor)

   Output: JSON data containing relevant details of reports, diagnosis data, and status.

   Test Case Derivation: The output will depend on the doctor's assessment of the confidence score and the generated report data, resulting in manually annotated information.

   How test will be performed: Automated scripts will submit the confidence score and generated report data for manual review, verifying that the doctor's annotations are accurately recorded and linked to the corresponding report.

10. FR-BE-10
    (FR-3 FR7)

    Type: Automated

    Initial State: End

    Input: Approval/Disapproval (1/0)

    Output: JSON data containing relevant details of reports, diagnosis data, and status.

    Test Case Derivation: The output will vary based on the input (1 or 0), returns nothing or include all pertinent report and diagnosis information.

    How test will be performed: Automated scripts will input approval/disapproval and validate that the returned JSON data matches the expected format and includes all relevant details.

5.5 Tests for Non-Functional Requirements

5.6 Traceability Between Test Cases and Modules

This section provides evidence that all modules have been considered in the test case design. Each test case is mapped to its corresponding module to ensure comprehensive coverage and verification of functionality.

Table 1: Traceability Matrix

Test Case ID	**Module**	**Description**	**Status**
test-id1	Module A (e.g., Upload)	Tests the upload functionality	Pending
test-id2	Module B (e.g., Profile)	Verifies profile editing capabilities	In Progress
test-id3	Module C (e.g., Reports)	Confirms report generation accuracy	Completed
test-id4	Module D (e.g., Diagnosis)	Assesses diagnosis prediction output	Pending

Analysis:

- Each test case directly corresponds to specific modules within the application, ensuring that all critical functionalities are covered.
- The status of each test case reflects its current progress, allowing for easy tracking of testing efforts.
- System logs will be generated during the execution of each test case to provide a detailed record of operations. These logs will capture input parameters, output results, and any errors encountered, aiding in debugging and validation.
- By correlating the test case outcomes with system logs, we can ensure accountability and traceability in the testing process, making it easier to identify and address issues.

References

Author Author. System requirements specification. https://github.com/..., 2019.

6 Appendix

This is where you can place additional information.

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.

6.2 Usability Survey Questions?

[This is a section that would be appropriate for some projects. —SS]

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.

1. What went well while writing this deliverable?

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

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.

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?