System Verification and Validation Plan for CXR

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

April 4, 2025

Revision History

Date	Version	Notes
October 22	1.0	Ayman completed section 2.
October 27	1.1	Nathan completed sections 3.4–3.7.
October 27	1.2	Kelly completed Nonfunctional Requirement tests.
October 28	1.3	Reza completed Functional Requirement tests.
October 29	1.4	Ayman completed sections 3.1–3.3.
November 3	1.5	Patrick completed the appendix and usability questions.

Contents

List of Tables

1 Symbols, Abbreviations, and Acronyms

This section records information for easy reference and aims to reduce ambiguity in understanding key concepts used in the project.

1.1 Table of Units

Throughout this document, SI (Système International d'Unités) is employed as the unit system. In addition to basic units, several derived units are used as described below. For each unit, the symbol is given, followed by a description of the unit and the SI name.

Symbol	**Unit**	**SI**
s	Time	Second
GB	Data	Gigabyte
MB	Data	Megabyte
LOC	Quantity	Lines of Code

1.2 Definitions

This subsection provides a list of terms that are used in the subsequent sections and their meanings, with the purpose of reducing ambiguity and making it easier to correctly understand the requirements:

- **Artificial Intelligence (AI) Model**: A program that analyzes datasets to identify patterns and make predictions. Used extensively in medical image analysis for automating diagnostics.

- **Convolutional Neural Network (CNN)**: A deep learning algorithm that processes images by assigning weights and biases, allowing it to identify patterns and features in medical images such as chest X-rays.

- **DICOM (Digital Imaging and Communications in Medicine)**: The international standard for medical images, defining formats for image exchange that ensure clinical quality.

- **Containerized Application**: A portable version of an application that can be run on a container run-time, such as Docker.

- **Machine Learning (ML)**: A subset of AI focusing on using data and algorithms to mimic human learning, improving accuracy over time.

- **Picture Archiving and Communication System (PACS)**: A system for acquiring, storing, transmitting, and displaying medical images digitally, providing a filmless clinical environment.

- **PHI**: Personal Health Information - Private and confidential data that must be protected under the HIPPA act.

- **HIPPA**: Health Insurance Portability and Accountability Act, a set of standards protecting sensitive health information from disclosure without patient's consent.

- **AWS - Amazon Web Services**: A public cloud provider, offering all HIPAA-compliance cloud services that helps Neuralanalyzer host, manage, and scale our application.

- **AWS ECS**: AWS Elastic Cloud Service: - An AWS managed service for managing and maintaining application containers at run-time.

- **AWS ECR**: AWS Elastic Container Registry - An AWS managed service for storing and managing container images.

- **AWS Fargate**: An AWS managed service for running containerized applications.

- **AWS Cognito**: An AWS managed service for authentication logic, handling user and password management.

- **React**: A web front-end framework, written in Javascript.

- **Flask**: An HTTP-based server framework, written in Python.

- **Finite State Machine (FSM)**: A computation model that simulates sequential logic using state transitions, applied in processes like user authentication and backend workflows.

- **ROC Curve (Receiver Operating Characteristic Curve)**: A graph that shows the performance of a classification model by plotting the true positive rate against the false positive rate at various threshold levels.

- **Service-Level Agreement (SLA)**: Defines the guaranteed uptime of the system, such as ensuring the availability of the AI service for 99.99% of operational hours.

- **Software as a Medical Device (SaMD)**: Software classified as a medical device under regulatory frameworks, such as those defined by the Food and Drugs Act.

- **TorchXRAYVision**: An open-source library for classifying diseases based on chest X-ray images, offering pre-trained models to accelerate the development process.

- **X-ray**: A form of high-energy electromagnetic radiation used in medical imaging to produce images of the inside of the body, enabling the diagnosis of conditions through radiographic film or digital detectors.

- **MIT License**: An open-source software license that allows for the free use, modification, and distribution of software.

- **Training Data**: Refers to the dataset of labeled chest X-ray images used to train the AI model. In this project, the dataset size is approximately 471.12 GB.

1.3 Abbreviations and Acronyms

Symbol	Description
SRS	Software Requirements Specification
AI	Artificial Intelligence
CNN	Convolutional Neural Network
DICOM	Digital Imaging and Communications in Medicine
VnV	Verification and Validation
ML	Machine Learning
PACS	Picture Archiving and Communication System
SaMD	Software as a Medical Device
ROC	Receiver Operating Characteristic Curve
SLA	Service-Level Agreement
FR	Functional Requirement
NFR	Non-Functional Requirement
FSM	Finite State Machine
CXR	Chest X-Ray Project
POC	Proof of Concept
TM	Theoretical Model
AWS	Amazon Web Services
ECS	Elastic Container Service
ECR	Elastic Container Registry

2 General Information

2.1 Summary

There are multiple software components that are required for the team 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)

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

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

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

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.

2.1.5 Automated Reporting

After the AI model completes the analysis of medical images, the system automatically generates structured, human-readable reports.

- 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

Referring back to the critical assumptions and system boundaries mentioned in the Hazard Analysis (?) document, we have identified both in-scope and out-of-scope objectives that will be considered in the VnV plan.The main focus will be on the AI model and the software components developed by our team, which revolve around the Functional and Nonfunctional requirements described in SRS (?) document. External components to our project, such as the libraries (e.g., TorchXRayVision) and APIs or quality of the X-Ray images, will not be tested as they are assumed to have been verified by their implementation teams. Below is further justification for the objectives:

2.2.1 Verification of AI Model Accuracy and Correctness (In scope)

- Confirm that the AI model meets the specified minimum accuracy threshold (e.g., 75% accuracy) as defined in the SRS.
- Validate the model's capability to correctly predict various lung conditions (e.g., pneumonia, lung cancer) from X-ray images, ensuring reliable clinical decision-making.
- Verifying the correctness of the AI's predictions is essential to ensure reliable clinical decision-making.

2.2.2 Verification of User Interface Usability (In scope)

- Ensure that all interface elements are responsive.
- Ensure that key functionalities—such as uploading X-rays, viewing diagnostic probabilities, and generating reports—are easily accessible and operable.

2.2.3 Ensure security of Patient Data Security (In scope)

- Ensure that the software complies with relevant privacy regulations (e.g., PIPEDA for Canada, HIPAA for the US).
- Verify that patient data is securely stored and transferred, and that appropriate access control mechanisms are in place.

2.2.4 Verification of Image Modalities and Preprocessing (In Scope)

- Verify that the system accepts multiple image modalities, including DICOM files and other common formats, as specified in the SRS.
- Validate that incoming images are correctly preprocessed before being fed into the deep learning model. This includes checking that images are resized, normalized, and formatted as required by the model's training pipeline.
- Ensure that the pre-processing pipeline reliably transforms raw input data into a form that maintains the integrity and usability of the data for accurate model predictions.

2.2.5 Exclusions: External Components and Data Quality (Out of Scope)

- Verification does not extend to external libraries, APIs, or the inherent quality of patient-provided data, as these are assumed to be validated by their respective providers.

2.3 Challenge Level and Extras

2.3.1 Challenge Level

The challenge level of this project is classified as advanced, reflecting the complexity of integrating AI for diagnostic purposes, the necessity for robust data handling and privacy compliance, and the development of 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 from the Computing and Software department and relevant medical datasets, ensuring we can manage the data and AI components of the project effectively.

2.3.2 Extras

Research Report
Our research paper will propose a unified Transformer-based framework for comprehensive chest X-ray (CXR) analysis that emphasizes anatomical localization. Using a Detection Transformer (DETR), it will identify anatomical regions and supports downstream tasks like disease classification and progression monitoring.

Norman Principle Report
Norman Principle Report will evaluate the user interface of the system based on Don Norman's seven usability principles. Each principle will be used as

a lens to assess how intuitive, user-friendly, and efficient the interface is.

2.4 Relevant Documentation

The three main relevant documentation that helped guide us in System Verification and Validation Plan (VnV) was Development Plan, Software Requirements and Specification, and Hazard Analysis.

Development Plan:

This document outlined the approach and resources needed to achieve our project goals, providing a clear roadmap for the development team. The problem statement within the development plan identified the specific challenges to be addressed, helping us focus on critical areas and allocate tasks efficiently. By defining milestones, deliverables, and timelines, the development plan ensured that all team members were aligned and aware of their responsibilities throughout the project lifecycle.

Software Requirements Specification (SRS):

The SRS clearly outlined the software's intended functionality, performance requirements, and design constraints, providing a comprehensive guide for developers and helping us understand our stakeholders' needs. The SRS specified both functional and non-functional requirements, ensuring that the system met user expectations and complied with regulatory standards. It served as a foundational document that informed the design and implementation phases, allowing for a structured and organized development process.

Hazard Analysis (HA):

By systematically analyzing possible hazards, we were able to incorporate additional mitigation strategies into our implementation plan. For example, we recognized that server downtime could impact the Service Level Agreement due to system unavailability. Through the HA, we developed countermeasures to ensure that such risks were mitigated, such as implementing failover mechanisms and robust backup solutions. This proactive approach helped us design a more resilient and secure system.

These documents significantly impacted the Verification and Validation (VnV) process by providing clear guidelines and objectives for testing. The SRS helped ensure that the project's objectives were well-defined, guiding the VnV process by defining acceptance criteria and enabling the creation of test cases linked to specific requirements. The establishment of a traceability matrix allowed us to ensure that all functionalities were thoroughly tested. The HA contributed by highlighting potential risks that needed to be addressed during testing, leading to more comprehensive and robust test scenarios. Additionally, the SRS and HA aided in managing changes, identifying risks, and ensuring compliance with regulations like PIPEDA or HIPAA (**?**).

3 Plan

This section introduces the verification and validation team with their associated responsibilities, and provides verification plans for SRS, system design, VnV plan, and system implementation. In addition, it introduces several automated testing and verification tools that are planned to be utilized by the testing team during the verification and validation process.

3.1 Verification and Validation Team

The verification and validation process is a collective responsibility within the team, with each member actively participating.

3.1.1 Medical Data Analysis

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 PyTorch Model Verification

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 Interface Verification

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

3.2 SRS Verification Plan

To ensure the Software Requirements Specification (SRS) document is comprehensive, clear, and meets all stakeholder needs, we have developed a structured verification plan. This plan includes peer reviews, consultations with supervisors and teaching assistants (TAs), and a detailed SRS checklist to guide the verification process.

3.2.1 Review Process

Our peer review process involves multiple stages to gather feedback and ensure the SRS document is of high quality:

- **Ad Hoc Feedback:** Team members will initially review the SRS document internally, providing immediate feedback on content, structure, and clarity. This step allows for quick identification of obvious issues or omissions.

- **Individual Review and Issue Creation:** Each team member is responsible for thoroughly reading the SRS, evaluating its clarity, and checking for missing or conflicting requirements. For any identified issues, team members will create GitHub issues detailing the concerns and suggesting improvements or additions. This systematic approach ensures all potential problems are documented and addressed.

- **External Peer Reviews:** We will invite classmates, our supervisor, stakeholders, and other peers to review the SRS document. Their feedback will provide diverse perspectives, ensuring the document is understandable to all stakeholders, including those unfamiliar with technical aspects of the project.

- **Pre-Meeting Preparation:** Before meetings with the supervisor or TAs, the team will update the SRS based on internal reviews and ensure it is ready for external feedback. We will create a checklist based on the project rubric and key questions, such as:

    – Are all functional and non-functional requirements included and clearly defined?

    – Are the constraints appropriately identified and in scope?

    – Does the document address all stakeholder needs and regulatory requirements?

  This preparation ensures productive meetings and focused feedback sessions.

- **Consultation and Feedback Sessions:** During meetings with the supervisor and TAs, we will present the SRS and actively seek their input on scope, clarity, and feasibility. We will ask targeted questions to verify alignment with project objectives, such as:
  - Do the non-functional requirements align with the project's quality objectives?
  - Are there any concerns regarding the feasibility of the proposed AI models and techniques?
  - Is the level of detail sufficient for each requirement?
- **Post-Meeting Actions:** After each consultation, the team will compile notes and feedback into a prioritized action list. This list will serve as a checklist of items to address, ensuring that all suggestions and concerns are systematically incorporated into the SRS.

3.2.2 SRS Checklist

1. **Completeness**
   - **Purpose and Scope:** Clearly outlines the project's goal to predict diseases from X-ray images using AI models.
   - **Stakeholders:** Identifies and defines relevant stakeholders, including healthcare professionals, patients, and regulatory bodies.
   - **Functional Requirements:** Describes core functionalities like image processing, disease prediction accuracy, and report generation.
   - **Non-Functional Requirements:** Covers essential factors such as system performance, data security, and regulatory compliance.
2. **Clarity**
   - **Clear Terminology:** Ensures that technical terms (e.g., deep learning, classification, X-ray modalities) are defined and used consistently.
   - **Glossary Completeness:** Includes all necessary acronyms and terminology relevant to AI, healthcare, and radiology.
3. **Consistency**

- **Consistent Terminology:** Uses standardized language across all documentation to avoid misinterpretation.
- **No Conflicting Requirements:** Checks for conflicts, especially between performance and security constraints.

4. **Verifiability**
   - **Testable Requirements:** Ensures all functions, such as prediction accuracy and response time, are measurable and testable.
   - **Acceptance Criteria:** Establishes specific benchmarks for model performance and prediction reliability.
5. **Traceability**
   - **Unique Identifiers:** Assigns identifiers to each requirement for tracking through development.
   - **Source of Requirements:** Links each requirement back to stakeholder needs and healthcare standards.
6. **Feasibility**
   - **Technical Feasibility:** Confirms that AI model and image processing techniques are achievable within the project's scope.
   - **Practical Constraints:** Assesses budget, data availability, and timeline constraints to ensure project viability.
7. **Security and Privacy**
   - **Data Retention Policy:** Aligns with data privacy laws, focusing on secure handling of patient X-ray images.
   - **Access Control Requirements:** Specifies requirements for user authentication, ensuring only authorized access to sensitive data.
8. **Modifiability**
   - **Organized Structure:** Structures requirements logically to allow future updates and integration of new AI models.

- **Avoiding Redundancy:** Ensures there are no duplicate or conflicting requirements to streamline modifications.

9. Compliance and Ethics

- **Ethical and Legal Standards:** Addresses compliance with healthcare standards, patient privacy, and ethical AI usage.

By systematically applying this checklist during the SRS verification process, we aim to produce a high-quality, comprehensive, and reliable specification document that will guide the successful development of the project.

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

1. Document Review

- **Design Alignment:** Ensure the design aligns with all functional and non-functional requirements specified in the SRS.
- **Documentation Accuracy:** Confirm that all documentation accurately reflects the implementation details for image processing, classification, and prediction algorithms.
- **Discrepancy Recording:** Record any discrepancies between the design documents and the requirements for resolution.

2. Prototype Testing

- **System Performance**
    - **X-ray Image Processing:** Verify accurate processing of X-ray images, ensuring they meet the input specifications of the AI model.
    - **Performance Under Load:** Test system performance under various load conditions to assess scalability and reliability.

  - **Prediction Response Time:** Check that the response time for generating predictions meets the specified performance requirements.
  - **Scalability:** Ensure the system can scale to accommodate future increases in user demand and data volume.

- **Disease Prediction Accuracy**
  - **Benchmark Testing:** Test the AI model's accuracy using a benchmark dataset to validate performance metrics.
  - **Complex Case Evaluation:** Evaluate the algorithm's effectiveness on rare and complex cases to ensure robustness.
  - **Prediction Consistency:** Check for consistency of predictions across multiple runs with the same input data.

- **User Interface (UI)**
  - **Intuitiveness:** Confirm that the UI is intuitive and user-friendly for healthcare professionals.
  - **Clear Display:** Ensure that disease predictions and analysis results are displayed clearly and concisely.
  - **Data Visualization:** Verify that data visualization tools (e.g., charts, highlighted images) are easy to interpret.
  - **System Integration:** Check for seamless integration with existing clinical systems and workflows.

- **Failure Handling**
  - **Incomplete Data Response:** Test the system's response to incomplete or corrupted data inputs.
  - **Error Messages:** Verify that error messages are clear, informative, and guide the user towards corrective action.
  - **Network Failure Simulation:** Simulate network failures to assess system recovery capabilities and data integrity.
  - **Error Logging:** Check that errors are properly logged for troubleshooting and auditing purposes.

- **Security and Compliance**
  - **Encryption and Privacy:** Confirm that patient data is encrypted during transmission and storage, adhering to privacy protections.
  - **Regulatory Compliance:** Verify compliance with healthcare data regulations such as HIPAA and PIPEDA.
  - **Access Control:** Ensure that access control mechanisms are in place, restricting data access to authorized users only.

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.).

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.).

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.), and Functional Requirements.

Periodic Health Checks

- The front-end, back-end, and ML model will be checked periodically to measure the system's uptime. Measured uptime must satisfy the agreed-upon SLA (95%).
- Periodic Health Checks will be performed regardless of the system's software version and state.
- In addition to uptime, the following performance parameters will be monitored:
  * API response time must remain under 500 milliseconds on average.
  * System memory usage must remain below 80% capacity during normal operations.
  * Server CPU usage must remain below 75% under standard load conditions.
  * Database query response times must be under 1 second for 95% of queries.
- Threshold violations will trigger alerts and initiate diagnostic procedures to ensure system stability and performance.

Updated based on peer review feedbacks

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.).

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.

3.7 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 and supervisor 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 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`).~~ Removed as per peer review addressed, user log in is already in the initial state

   - *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 Inference 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 75% or higher.~~

1. **~~test-FR3-1~~**

   ~~Control: 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 75%, correctly identifying at least 750 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 75%, using assertions.~~

Removed to non-functional requirement

4.1.4 Patient Condition Progression Tests

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**

   Control: 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.

4.1.5 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.

4.1.6 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.

4.1.7 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.

4.1.8 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).

4.1.9 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.

4.1.10 ~~Role Based Access Control Tests~~

~~This subsection covers Requirement #10 of the SRS (?) document by testing that the system enforces role-based access control, ensuring users only access permitted features.~~

1. ~~**test-FR10-1**~~

   ~~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.~~

Removed FR10 due to change in current scope of the project (corresponding to SRS)

4.1.11 Preservation of Original Data Tests

This subsection covers Requirement #11 of the SRS (?) document by testing that the system preserves the original chest X-ray images by creating copies for analysis.

1. **test-FR11-1**

   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.

4.1.12 Multiple Modality Support Tests

This subsection covers Requirement #12 of the SRS (?) document by testing that the system supports multiple medical imaging modalities beyond chest X-rays.

1. ~~**test-FR12-1**~~

   ~~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.~~

Removed FR12 due to change in current scope of the project (corresponding to SRS)

4.1.13 Regular AI Model Update Tests

This subsection covers Requirement #13 of the SRS (**?**) document by testing that the system can update the AI model regularly without disrupting ongoing services.

(a) ~~**test-FR13-1**~~

~~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.~~

Removed FR13 due to change in current scope of the project (corresponding to SRS)

4.1.14 Image Preprocessing Tests

This subsection covers Requirement #14 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.

i. ~~**test-FR14-1**~~
   ~~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_tes`~~

~~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.~~

ii. **~~test-FR14-2~~**

~~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").~~

Removed FR14 due to change in current scope of the project (corresponding to SRS)

4.2 Tests for Nonfunctional Requirements

This section contains tests for the Nonfunctional Requirements. The subsections for these tests were created based on the subsections of the Nonfunctional Requirements listed in the SRS (?) . Each test was created based on the Fit Criterion of the requirements that they covered.

4.2.1 Look and Feel

This subsection's tests cover all Look and Feel requirements listed in the SRS (?) by verifying that the user interface matches standard medical imaging software layouts and ensuring that the interface is comfortable and accessible for users during typical workflows.

A. NFR-LF1

   Type: Non-Functional, Dynamic, Manual
   Initial State: The system must be installed and accessible.
   Input/Condition: Adjust window/level settings. Navigate between other tabs. Apply basic tools (zoom, pan, measure). Export or save images from the viewer.
   Output/Result: The UI layout should be similar to standard medical imaging software (with menu positions, toolbar layout, and terminology consistent with common practices).
   How this test will be performed: Ensure the system under test and reference medical imaging software are installed and configured. Launch the system under test and the reference software side-by-side. Compare the menu structure, toolbars, icons, panels, and overall layout. Verify whether the placement of key elements matches standard layouts. Review menu labels, tooltips, and buttons to ensure they use medical imaging terms. Adjust window/level settings. Use zoom, pan, and measurement tools.

B. NFR-LF2

   Type: Non-Functional, Dynamic, Manual
   Initial State: The system under test is installed and

accessible on the target device. The monitor brightness is set to a comfortable level to ensure consistency during the test.
Input/Condition: User interaction with the interface for typical workflows.
Output/Result: Color contrast ratio is 4.5:1 or higher for text. Font sizes at least 12-14 points for normal text.
How this test will be performed: Launch the system and ensure the default color schemes and fonts are applied. Perform tasks continuously for 1-2 hours while interacting with different parts of the system interface (e.g., forms, menus). Check for signs of fatigue or frustration (e.g., rubbing eyes, slowing down). Use tools to measure the color contrast ratio.

4.2.2 Usability and Humanity

This subsection's tests cover all Usability and Humanity requirements listed in the SRS (**?**) by verifying that users can successfully perform key tasks without assistance and that the interface provides adequate help and tool-tips to guide users through the system.

A. NFR-UH1

Type: Non-Functional, Dynamic, Automated
Initial State: User accounts and access credentials are prepared for all healthcare professionals participating in the test. Any required files or images are pre-loaded and accessible for the tasks.
Input/Condition: Automated script with a list of specified tasks and user accounts.
Output/Result: Percentage of user accounts that successfully completed all tasks without assistance: Success Rate = $\frac{\text{Number of Successful Participants}}{\text{Total Number of Participants}} \times 100$.
How this test will be performed: The automated script will perform the following tasks with each user account and calculate the success rate: upload a chest X-ray im-

age to the system, view analysis results or processed reports, adjust image settings (e.g., window/level, zoom), and export the analysis or report to a local folder.

B. ~~NFR-UH2~~

~~Type: Non-Functional, Dynamic, Automated~~
~~Initial State: List of user accounts and access credentials are prepared. Full access to the user interface. Interface elements (buttons, icons, menus) are accessible for interaction.~~
~~Input/Condition: Automated script with a list of interactions specified to interact with the tabs and other listed functionalities.~~
~~Output/Result: Log whether tool-tips and help content were displayed correctly for each element tested.~~
~~How this test will be performed: The test script will detect if the tool-tip appears after a short delay (0.5-1 second) when it hovers over an element. When it clicks on a help icon (e.g., "?" symbol) or help link, the test script checks if the system provides relevant information about that part of the interface. When the test script finishes, it will provide a log including if tool-tips and help content were displayed correctly for each element tested.~~
Removed NFR-UH2 due to change in current scope of the project (corresponding to SRS)

4.2.3 Performance

This subsection's tests cover all Performance requirements listed in the SRS (?) by testing the processing time of image analysis, system availability under various conditions, and system performance when handling simultaneous image uploads.

A. NFR-PR1

Type: Non-Functional, Dynamic, Automated
Initial State: User is logged in. Standard chest X-ray images are available for testing in PNG/JPG format

(depending on system requirements).
Input/Condition: Automated test script interacts with the system.
Output/Result: Percentage of the number of the chest X-ray images that are analyzed correctly.
How this test will be performed: The test script will upload a chest X-ray image through the interface. It will monitor the system to ensure it processes and analyzes the image. The moment the results or report are displayed or the system indicates completion, it checks the timing (30 seconds). The process is repeated multiple times (at least 5 trials) to account for variations in processing time.

B. NFR-PR2

Type: Non-Functional, Dynamic, Automated
Initial State: A monitoring system is set up to log availability metrics, such as downtime events, mean time to recovery, and uptime percentage.
Input/Condition: Server outages, network issues, high number of concurrent users.
Output/Result: Uptime percentage; downtime events (log the number, duration, and cause of any disruptions or downtime); mean time to recovery.
How this test will be performed: Two automated scripts will start the monitoring system for a defined period (e.g., 24 hours or 1 week). One will simulate server outages or network issues by disabling a service temporarily and record the downtime, mean time to recovery. The other will use load testing tools (like Apache JMeter or Locust) to simulate a high number of concurrent users during peak hours and monitor response times. The two scripts will combine their results and return the output once they all finish.

C. NFR-PR3

Type: Non-Functional, Dynamic, Automated

Initial State: All dependencies (database, network, storage, and image processing engine) are configured and operational. Monitoring tools are available to track system performance, including CPU and memory usage, image processing times.
Input/Condition: Collections of 20 identical or varied images, multiple user accounts. Baseline for each image processing is 20 seconds.
Output/Result: Processing time (float), system resource utilization (float), images uploading success/fail (boolean).
How this test will be performed: The automated script will simulate different users to upload 20 images simultaneously and start the stopwatch or monitoring tool when the uploads begin. It will keep track of the CPU, memory, and I/O usage during the processing of the 20 images. When image processing is finished, it will calculate and return the average processing time for the 20 images, return CPU usage, memory usage, and check for errors or failures.

D. NFR-PR4

Type: Non-Functional, Automated
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 75%, correctly identifying at least 750 out of 1,000 cases.
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 75%, using assertions.

4.2.4 Operational and Environmental

This subsection's tests cover all Operational and Environmental requirements listed in the SRS (?) by verifying the system's ability to integrate with the hospital's PACS system and its performance under varying network conditions.

E. ~~NFR-OE1~~

~~Type: Non-Functional, Dynamic, Manual~~
~~Initial State: The AI system and the PACS are both connected to the same hospital network and the DICOM configuration for both systems is correctly set.~~
~~Input/Condition: AI-processed results including annotations or diagnostic report as an image (in DICOM format). Metadata with patient ID (string) and instance number (string).~~
~~Output/Result: Boolean indicating whether the PACS successfully stores the AI-processed results with correct format (annotated image or report). The stored result is associated with the correct patient ID and instance number in the PACS.~~
~~How this test will be performed: An automated script will process the retrieved chest X-ray image and generate an annotated image or diagnostic report in DICOM format. It will then send the annotated result to the PACS and verify the response from the PACS to ensure that the transfer was successful. Then it will log into the PACS and search for the patient ID to confirm that~~

~~the stored result is associated with the correct instance number. The automated script can be run multiple times depending on the need.~~

Removed NFR-OE1 due to change in current scope of the project (corresponding to SRS)

F. NFR-OE2

Type: Non-Functional, Dynamic, Automated
Initial State: The network is configured normally with minimal latency and no packet loss initially. Necessary patient data and chest X-ray studies are available for retrieval.
Input/Condition: Retrieve and process a chest X-ray image, store the processed result, and send the data to external.
Output/Result: Boolean indicating if latency is within a reasonable time (190ms 220ms), logs include latency and packet loss statistics.
How this test will be performed: The automated script will simulate network latency and packet loss using a network emulation tool. It will then process the retrieved chest X-ray image and generate an annotated image or diagnostic report, meanwhile monitoring the time taken for operations and the number of retries, and checking the system logs for any warnings or errors related to packet loss or latency. It will check if the system raises alerts if packet loss exceeds 1% or if latency impacts performance significantly and return the results.

4.2.5 Security and Privacy

This subsection's tests cover all Security and Privacy requirements listed in the SRS (?) by verifying that patient data is securely encrypted during storage and transmission, and that user access is properly controlled according to assigned roles.

G. NFR-SR1

Type: Non-Functional, Dynamic, Automated
Initial State: AES-256 encryption libraries (such as OpenSSL or Cryptography in Python) are configured for the system. Patient data (including DICOM images and reports) is stored on the system or transmitted over the network.
Input/Condition: X-ray image and diagnostic report with patient ID and instance number. Secret key and initialization vector (IV) for AES-256 encryption.
Output/Result: All patient data, including images and reports, is encrypted using AES-256 both during storage and transmission.
How this test will be performed: The automated system will trigger the AI system to retrieve a chest X-ray image and process it. It will check if transmitted data (images and reports) appears as encrypted bytes and no readable patient information or image data should be visible. After processing, the AI system stores the data and the automated script will try to open a stored DICOM image or report directly without using the decryption mechanism, which should appear as unreadable, encrypted content (random bytes). When the AI system decrypts and accesses the stored image or report, the automated script will detect if the decrypted data matches the original data.

H. ~~NFR-SR2~~

~~Type: Non-Functional, Dynamic, Automated~~
~~Initial State: The AI system has different user accounts with different assigned roles. Each role has specific permissions defined.~~
~~Input/Condition: Role 1: Radiologist—can view diagnostic results and patient images. Role 2: Administrator—can manage user accounts and system configurations (user_radiologist, user_admin).~~

~~Output/Result: Radiologist should be able to view diagnostic reports and images and should not be able to manage user accounts. Administrator should be able to create and manage user accounts and should not have access to patient diagnostic reports.~~
~~How this test will be performed: The automated script will log in as either one of the users (user_radiologist or user_admin). For example, the automated script logs in as user_radiologist and views diagnostic reports and downloads patient images, which should be successful. It will then attempt to manage user accounts, which access should be denied. The same process is repeated for user_admin.~~
Removed NFR-SR2 due to change in current scope of the project (corresponding to SRS)

4.2.6 Maintainability and Support

This subsection's tests cover all Maintainability and Support requirements listed in the SRS (?) by verifying that the system's codebase is modular and well-documented, and that automated testing with code coverage is in place to facilitate future maintenance and updates.

I. NFR-MS1

Type: Non-Functional, Static, Manual
Initial State: The AI system code-base is deployed in a version-controlled environment. Documentation for each module (e.g., README files, API references, inline comments) is available in the repository.
Input/Condition: All modules within the code repository are accessible.
Output/Result: All key functionalities (e.g., data ingestion, inference, reporting) are encapsulated in separate, well-defined modules. Modules can function independently with minimal coupling. Dependencies between modules are well-documented.
How this test will be performed: The test person should

clone the repository of this AI system and identify and list all modules, review dependencies to ensure minimal coupling, and make sure each module is self-contained and can be modified without breaking other parts of the system.

J. NFR-MS2

Type: Non-Functional, Dynamic, Automated
Initial State: Code repository contains the entire AI system, automated testing framework (e.g., JUnit, Pytest) is installed and configured in the project. Tests include unit tests, integration tests, system tests, and end-to-end tests. Code coverage tool (e.g., coverage.py, JaCoCo) is integrated.
Input/Condition: Unit tests, integration tests, and end-to-end tests.
Output/Result: Code coverage report.
How this test will be performed: The automated test script will navigate to the project directory and execute the automated test suite using the testing framework (e.g., Pytest or JUnit).

4.2.7 ~~Cultural~~

~~This subsection's tests cover all Cultural requirements listed in the SRS (?) by ensuring that the system supports both English and French languages throughout the interface and reports, with no untranslated or misaligned content.~~

K. ~~NFR-CR1~~

~~Type: Non-Functional, Dynamic, Manual~~
~~Initial State: The language options (English and French) are available in the settings menu.~~
~~Input/Condition: Switch language between English and French. Generate report in English and French.~~
~~Output/Result: No untranslated or misaligned content should appear.~~
~~How this test will be performed: The person conducting~~

~~the test will switch the language from English to French (and vice versa) in the settings menu and inspect menus, buttons/labels, error messages, notifications, and other necessary elements for correctness. The test person will then use the system to generate a diagnostic report in English (and French) for a chest X-ray image. He/She will review the report for medical findings and patient info to verify that both reports contain the same findings but in the correct language. He/She will also trigger an error condition (e.g., entering invalid patient data) to verify that the error message appears in the selected language (English or French).~~

Removed NFR-CR1 due to change in current scope of the project (corresponding to SRS)

4.2.8 Legal

This subsection's tests cover all Legal requirements listed in the SRS (**?**) by verifying that the system's design and development processes comply with relevant regulations and standards such as HIPAA, PIPEDA, and ISO 13485.

L. ~~NFR-LR1~~

~~Type: Non-Functional, Static, Manual~~
~~Initial State: HIPAA and PIPEDA documentation are in place, development artifacts and source code are available.~~
~~Input/Condition: Development artifacts, system design documentation.~~
~~Output/Result: System design aligns with the requirements of HIPAA (US) and PIPEDA (Canada). If not, all software risks are identified, evaluated, and mitigated.~~
~~How this test will be performed: Compliance audits will verify the system design and related documentation to see if the AI system meets data protection and regulation requirements.~~
Removed NFR-LR1 due to change in current scope of the project (corresponding to SRS)

M. NFR-LR2

   Type: Non-Functional, Static, Manual
   Initial State: ISO 13485 documentation is in place, development artifacts are available (verification and validation plans, risk analysis reports, requirements, design documents, test plans, etc.).
   Input/Condition: Development artifacts, processes (software lifecycle management process, design review process, etc.).
   Output/Result: All development processes align with the requirements of ISO 13485 and if not, all software risks are identified, evaluated, and mitigated.
   How this test will be performed: The person conducting the test will first review the development artifacts and processes to confirm that development processes follow ISO 13485 guidelines and risk management procedures are aligned with ISO 14971. He/She will verify that requirements documents are available and cover all key system functionalities, and the system shall adhere to the ISO 13485 standard for medical device software development.

4.2.9 Health and Safety

This subsection's tests cover all Health and Safety requirements listed in the SRS (**?**) by ensuring that the AI-generated diagnostic reports are reviewed and confirmed by qualified radiologists before being finalized, and that appropriate disclaimers are displayed to inform users about the AI's role.

N. NFR-HS1

   Type: Non-Functional, Dynamic, Manual
   Initial State: User roles are configured within the system, including Radiologist, Clinician, etc.
   Input/Condition: AI-generated report, Radiologist user account.

Output/Result: Logs including radiologist's review action (confirm/reject) with a timestamp.
How this test will be performed: The test person will log in using a Radiologist user account and upload a test chest X-ray image to the AI system. The AI system analyzes the image and generates a report (flagged as preliminary). The test person will then access the preliminary report through the system's dashboard and confirm/reject the diagnosis. If confirmed, the report should be marked as final and made available to clinicians/patients. If rejected, the system should log the feedback and trigger a re-analysis request.

O. NFR-HS2

Type: Non-Functional, Dynamic, Manual
Initial State: The AI system is deployed and accessible to radiologists, clinicians, and other users.
Input/Condition: Access the AI-generated diagnostic report and view the user interface displaying AI results.
Output/Result: Disclaimer is prominently displayed and easy to read.
How this test will be performed: The test person should generate a diagnostic report using the AI system and open the generated report. He/She should check if the disclaimer is present on the report, either at the top, bottom, or in a prominent section, and the disclaimer text matches the predefined wording. He/She should also access the user interface which displays AI results and select a patient report, checking if the disclaimer is clearly displayed alongside the AI result and cannot be dismissed.

4.3 Traceability Between Test Cases and Requirements

Test ID	FR1	FR2	FR3	FR4	FR5	FR6	FR7	FR8	FR9	FR11
test-FR1-1-1	×									
test-FR1-2-2	×									
test-FR2-1-1		×								
test-FR3-1-1			×							
test-FR4-1-1				×						
test-FR5-1-1					×					
test-FR5-2-2					×					
test-FR6-1-1						×				
test-FR6-2-2						×				
test-FR7-1-1							×			
test-FR8-1-1								×		
test-FR9-1-1									×	
test-FR11-1-1										×

Table 1: Functional Requirements Traceability (Updated)

NFR ID	**LF**		**UH**	**PR**				**OE**	**SR**	**MS**	
	1	2	1	1	2	3	4	2	1	1	2
LF 1	x										
LF 2		x									
UH 1			x								
PR 1				x							
PR 2					x						
PR 3						x					
PR 4							x				
OE 2								x			
SR 1									x		
MS 1										x	
MS 2											x

Table 2: Updated Nonfunctional Requirements Traceability Matrix

5 Unit Test Description

Please note that this section is a placeholder for the unit test description. The unit test modules will be updated as the project progresses and Module Interface Specification (MIS) documents are developed.

5.1 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.

5.2 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.

5.3 Tests for AWS Cognito Module

Testing for test_aws_cognito.py

test_get_self_user():

- Retrieves a user's full profile using a unique identifier and checks for the presence and accuracy of core fields such as name, gender, and organization. Ensures backend correctly maps and fetches user data using UUIDs.
- **Test Goal:** To confirm that user data retrieval via user ID returns a complete and accurate profile.

test_sign_up_doctor():

- Registers a doctor by using their email, password, and additional personal details like occupation and organization. Ensures user creation includes correct username formatting and password handling.
- **Test Goal:** To verify that doctor registration creates the correct user account with accurate details and login credentials.

test_create_patient():

- Creates a new patient entry linked to a specific doctor by capturing essential patient details like name, gender, and birthdate. Verifies the auto-generation of a username and password for the patient.
- **Test Goal:** To confirm that patient creation works correctly by linking the patient to a doctor and generating the correct username and password.

test_sign_in_user():

- Signs in a user with their email and password, ensuring that authentication tokens are generated. Verifies the presence of the access and refresh tokens in the response.

- **Test Goal:** To ensure the sign-in process correctly generates authentication tokens for the user.

test_get_self_user():

- Fetches the authenticated user's details using their access token, ensuring that all essential information like email, name, and occupation are returned accurately.
- **Test Goal:** To confirm that the correct and complete user profile information is returned when accessing the authenticated user's data.

test_get_user_by_id():

- Retrieves a user's profile by their unique identifier (UUID) to check for accuracy in their details such as name, gender, and occupation.
- **Test Goal:** To ensure that user information is correctly fetched by user ID and includes all necessary fields.

test_update_user_by_id():

- Updates a user's profile information using their unique identifier and validates that changes to attributes like name and occupation are applied correctly in the backend.
- **Test Goal:** To verify that profile updates via user ID are successful and reflect the correct changes in the database.

5.4 Tests for S3 Presigned URL Handler Module

Testing for test_aws_s3.py
test_generate_url():

- Tests if ‘generate_presigned_url‘ returns the mocked S3 URL, confirming that the function correctly generates a URL.
- **Test Goal:** To verify that the ‘generate_presigned_url‘ function correctly returns a valid URL in string format.

test_upload_file():

- Tests if ‘upload_file‘ returns the expected success response with status and message, confirming that the upload operation works as intended.
- **Test Goal:** To ensure the ‘upload_file‘ function correctly simulates a successful file upload and returns the appropriate success response.

5.5 Tests for Report Generation Service Module

Testing for test_report_generation_service.py
test_openai_api_key_exists():

- Tests if the OpenAI API key is correctly set and not None or empty, ensuring that the application has valid API credentials for interacting with OpenAI.
- **Test Goal:** To verify that the OpenAI API key is not None or an empty string, confirming that valid credentials are used.

test_user_id_exists():

- Tests if the user ID is set and not None or empty, ensuring that a valid user ID is available for generating personalized reports.
- **Test Goal:** To confirm that the user ID is set correctly and is not empty or None.

test_image_url_exists():

- Tests if the image URL is correctly set and is not None or empty, ensuring that a valid URL is provided for image processing.
- **Test Goal:** To ensure that the image URL is set correctly and is not None or an empty string.

test_generate_report():

- Tests the ‘generate_report‘ method of the ‘ReportGenerationService‘ to check if it returns a valid report with findings, impressions, and predictions.
- **Test Goal:** To verify that the ‘generate_report‘ method returns a complete report, including findings, impression, and prediction for the user.

5.6 Tests for Database Module

Testing for test_db.py

test_get_paginated_record_by_userId():

- Tests if the ‘get_paginated_record_by_userId‘ method fetches paginated medical records for a given user ID, ensuring that data retrieval is correct and pagination is handled.
- **Test Goal:** To confirm that the method returns valid and paginated medical records based on the user ID.

test_get_record_by_id():

- Tests if the ‘get_record_by_id‘ method fetches a medical record by its unique identifier, ensuring that it retrieves the correct record.
- **Test Goal:** To verify that the method correctly returns the medical record based on the given ID.

test_create_new_record():

- Tests if the ‘create_new_record‘ method creates a new medical record for the user, ensuring that the record is created successfully in the system.
- **Test Goal:** To ensure that a new medical record is correctly created for the user with valid details.

test_update_record_by_id():

- Tests if the ‘update_record_by_id‘ method updates a medical record by its ID, verifying that the record is modified correctly.
- **Test Goal:** To verify that the medical record is successfully updated by its unique identifier.

test_link_report_to_record():

- Tests if the ‘link_report_to_record‘ method links a report to a medical record, ensuring that the link between the two is established correctly.
- **Test Goal:** To ensure that the report is successfully linked to the corresponding medical record.

test_get_paginated_prescription_by_recordId():

- Tests if the ‘get_paginated_prescription_by_recordId‘ method fetches paginated medical prescriptions for a given record ID, ensuring that data retrieval and pagination are correct.
- **Test Goal:** To confirm that the method returns valid and paginated prescriptions based on the record ID.

test_get_prescription_by_id():

- Tests if the ‘get_prescription_by_id‘ method fetches a prescription by its unique identifier, ensuring that it retrieves the correct prescription.
- **Test Goal:** To verify that the method correctly returns the prescription based on the given ID.

test_create_new_prescription():

- Tests if the 'create_new_prescription' method creates a new prescription for a given medical record, ensuring the prescription is successfully created.
- **Test Goal:** To ensure that a new prescription is correctly created for the medical record.

test_update_prescription_by_id():

- Tests if the 'update_prescription_by_id' method updates a prescription by its ID, verifying that the prescription details are modified correctly.
- **Test Goal:** To verify that the prescription is successfully updated by its unique identifier.

References

6 Appendix

6.1 Symbolic Parameters

Parameter	**Description**
Uptime	99.99% – Ensures near-continuous availability of the AI system, minimizing downtime for uninterrupted analysis and diagnostics.
Training Photos	471.12 GB (zipped) – A substantial dataset of chest X-rays used to train the AI model, contributing to its robustness and accuracy.
Accuracy Rate	90% – The model achieves a high accuracy rate in classifying and diagnosing chest X-rays, balancing precision with speed for practical clinical use.
Response Time	Less than 1 second per image – The system is optimized to provide immediate feedback on X-ray results, significantly reducing wait times.
Batch Size	32 images per batch – Defines the number of images processed together during each training iteration.
Learning Rate	0.001 – The step size during gradient descent, impacting model training speed.
Number of Epochs	50 – The number of times the model processes the entire training dataset.
Input Image Size	224x224 pixels – Suitable for CNN architectures, balancing computational efficiency and image detail.
Threshold for ROC Analysis	0.75 – The probability cut-off for positive classification, balancing sensitivity and specificity.
Precision	92% – Reflects the percentage of true positive cases among all cases predicted positive.
Recall	88% – Reflects the percentage of true positive cases among all actual positive cases.

Table 3: **Symbolic Parameters for AI System**

6.2 Usability Survey Questions

P. On a scale of 1 to 5, how would you rate your overall experience with the chest X-ray analysis product?

Q. On a scale of 1 to 5, how would you rate the ease of use and navigation of the interface?

R. On a scale of 1 to 5, how satisfied are you with the accuracy of the product's disease detection and report generation?

S. How confident are you in the reliability of the predictions provided by the product? (1 = Not confident, 5 = Very confident)

T. Are there any specific features you would like added to improve the accuracy or relevance of the analysis?

U. What feature of the product do you find most helpful in interpreting the chest X-ray images and understanding the diagnosis?

V. Which feature, if any, did you find least helpful or unnecessary for your workflow?

W. How likely are you to recommend this product to other healthcare professionals? (1 = Not likely, 5 = Very likely)

X. How intuitive do you find the product's interface for navigating and analyzing X-ray images? (1 = Not intuitive, 5 = Very intuitive)

Y. On a scale of 1 to 5, how well do the generated reports and predictions align with your expectations and clinical experience?

Z. Did you encounter any difficulties in completing diagnostic tasks or viewing the analysis results? If yes, please describe.

. On a scale of 1 to 5, how useful do you find the automated report summaries in supporting your decision-making?

. How secure do you feel using this product in terms of patient data privacy? (1 = Not secure, 5 = Very secure)

. Are there any other medical conditions or features you would like the product to detect or analyze in future updates?

Appendix — Reflection

. **What went well while writing this deliverable?**
Writing this deliverable enabled us to delve deeply into the design and verification aspects of our AI-based chest X-ray analysis system.The abundance of ongoing research in the medical imaging domain and machine learning provided us with access to established validation frameworks and testing tools specifically designed for medical imaging applications and AI models. This made it straightforward to develop detailed system tests for both functional and non-functional requirements, ensuring that we thoroughly examined the core functionalities of our system.

. **What pain points did you experience during this deliverable, and how did you resolve them?**
One of the main challenges we encountered was designing comprehensive system tests that effectively validate both the core functionalities and the additional features of our system. Ensuring that each functional requirement was adequately tested required great attention to detail and a thorough understanding of the system's expected behavior. Initially, we found it difficult to formulate test cases for complex features like AI model integration, security protocols, and handling of patient data. To overcome this, we revisited our System Requirements Specification (SRS) to clarify the requirements and ensure a precise alignment between requirements and tests. We also faced difficulties in balancing the scope of our extras, particularly the secondary features, within our project timeline and resource constraints. To resolve these issues, we held team discussions to prioritize features based on their feasibility and impact. We incorporated a traceability matrix to systematically link test cases to requirements, which greatly improved our ability to identify gaps and redundancies. Additionally, seeking feedback from our supervisor, Dr.Moradi, helped us refine our testing strategies

and address any overlooked areas.

. **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.**

. **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?**

The following 2 questions are answered here for each team member:

. Patrick

- Responsibility: Implementing and validating PyTorch models, ensuring model robustness and accuracy.
- Knowledge and Skills Needed:
  - Advanced proficiency in PyTorch for deep learning model development and testing.
  - Dynamic testing techniques for AI models, including unit testing and integration testing.
- Approaches to Acquiring Skills:
- Online Courses and Tutorials: Enrolling in advanced PyTorch courses (e.g., Coursera, Udemy) focusing on model testing and validation, such as "Deep Learning with PyTorch" or "AI Testing with PyTorch."
- PyTorch Documentation and Community Forums: Studying official documentation and engaging with the PyTorch community to learn best practices in model validation and debugging.

- Chosen Approach: Patrick will pursue the Online Courses and Tutorials approach.
- Reason for Choice: Structured courses provide comprehensive coverage of advanced topics and include hands-on projects that reinforce learning. This method allows Patrick to systematically build upon his existing knowledge and directly apply new skills to our project, ensuring the AI model meets the required standards for accuracy and reliability in the V&V plan.

. Reza

- Responsibility: Conducting static code analysis and maintaining code quality across the project.
- Knowledge and Skills Needed:
  - Proficiency in static code analysis tools (e.g., Pylint for Python, SonarQube).
  - Understanding of coding standards and best practices for maintainability and security.
- Approaches to Acquiring Skills:
- Hands-on Practice with Tools: Installing and experimenting with static analysis tools on sample codebases to gain practical experience.
- Workshops and Webinars: Participating in workshops focused on code quality and static analysis techniques.
- Chosen Approach: Reza will pursue the Hands-on Practice with Tools approach.
- Reason for Choice: Practical experience is essential for mastering static analysis tools, and applying them directly to our project's codebase will provide immediate benefits. This approach allows Reza to tailor his learning to the specific tools and languages used in our project, ensuring that code quality and security are upheld in accordance with the V&V plan.

. Kelly

- Responsibility: Setting up and managing continuous integration (CI) and automation workflows using GitHub Actions.
- Knowledge and Skills Needed:
  - Proficiency in CI/CD pipelines and GitHub Actions.
  - Automation of testing procedures and deployment processes.
- Approaches to Acquiring Skills:

• GitHub Learning Lab Courses: Completing interactive courses specifically focused on GitHub Actions and CI/CD practices.

• Project-Based Learning: Implementing CI/CD pipelines in a controlled environment or sample projects to gain practical experience.

- Chosen Approach: Kelly will combine both approaches but will focus on the GitHub Learning Lab Courses first.
- Reason for Choice: The structured courses provide essential knowledge and best practices, after which she will apply this knowledge through project-based learning to reinforce and expand her skills. This dual approach ensures that our V&V processes are automated efficiently and effectively.

. **Ayman**

- Responsibility: Developing effective data visualization techniques for medical data analysis and assisting in web interface development.
- Knowledge and Skills Needed:
  - Proficiency in data visualization libraries (e.g., Matplotlib, Seaborn, Plotly).
  - Understanding of usability testing methodologies for web applications.
  - Frontend development skills (e.g., React, Angular) for web interface testing.

- Approaches to Acquiring Skills:

- Online Tutorials and Documentation: Learning visualization libraries through official documentation and community tutorials.
- Case Studies and Best Practices: Studying case studies on effective medical data visualization and user interface design.

- Chosen Approach: Ayman will pursue the Case Studies and Best Practices approach.

- Reason for Choice: By understanding how visualization is effectively used in existing medical applications, Ayman can apply proven techniques to our project. This approach also aids in designing user-friendly interfaces, crucial for the V&V plan's usability testing component.

. Nathan

- Responsibility: Conducting usability testing to ensure the web application is user-friendly and meets the needs of healthcare professionals.

- Knowledge and Skills Needed:
  - Understanding of usability testing methodologies and best practices.
  - Proficiency in designing and conducting user surveys and interviews.
  - Ability to analyze user feedback and identify areas for improvement.

- Approaches to Acquiring Skills:

- User Research and Surveys: Engaging in user research activities to gather insights into user needs and preferences.
- Usability Testing Workshops: Participating in workshops focused on usability testing techniques and user-centered design principles.

- Chosen Approach: Nathan will pursue the Usability Testing Workshops approach.

- Reason for Choice: Attending workshops will provide Nathan with structured training and hands-on experience in usability testing, enabling him to effectively evaluate and enhance the user interface based on real user feedback. This approach ensures that Nathan can apply best practices directly to our V&V plan, improving the overall user experience for healthcare professionals.