Module Interface Specification for Chest Scan

Team 16, Ace
Harrison Chiu
Hamza Issa
Ahmad Hamadi
Jared Paul
Gurnoor Bal

January 17, 2025

1 Revision History

Date	Version	Notes
01/07/2024	0.0	Initial Document.
01/08/2024	0.1	Added MIS structure and formatting; Started adding module definitions.
01/10/2024	0.2	Included Table from Module Guide (MG) into the Module Decomposition section.
01/12/2024	0.3	Expanded Symbols, Abbreviations, and Acronyms section; refined MIS formatting.
01/14/2024	0.4	Completed Introduction and Notation sections; refined function signatures in MIS.
01/16/2024	0.5	Updated MIS for HardwareAcceleration and DatasetHandler modules based on feedback and finished all of the other modules.

Table 1: Revision History

2 Symbols, Abbreviations and Acronyms

This section records the symbols, abbreviations, and acronyms information for easy reference for terms used in this document.

For information on most of the symbols, abbreviations, and acronyms referenced in this document, see the SRS Documentation at the following link:

GitHub SRS Documentation

The information on the rest of the symbols, abbreviations, and acronyms referenced in this document are shown in the table below.

symbol	**description**
AI/ML	Artificial Intelligence/Machine Learning
DICOM	Digital Imaging and Communications in Medicine; technical standard for digital storage/transmission of medical images and related information
GUI	Graphical User Interface
JPEG/JPG	Joint Photographic Experts Group; digital image compression standard, image format
M	Module
MG	Module Guide
MVC	Model-View-Controller Software Architecture
NLP	Natural Language Processing
SRS	Software Requirements Specification
Chest Scan	The Process of Designing and Developing Software; a reference to the software application described in this document

Contents

3 Introduction

The following document details the Module Interface Specifications of X-RayAssist. This software application is an AI-powered medical imaging software that processes chest X-ray images to identify, analyze, and reconstruct abnormal regions. The system utilizes advanced in-painting techniques to enhance visualization, aiding radiologists in interpreting affected areas. Additionally, it leverages machine learning (ML) models to generate synthetic X-ray images conditioned on specific disease characteristics, enabling better diagnosis and medical research. The processed results are then integrated into natural language radiology reports for further assessment.

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at:

GitHub Repository.

4 Notation

[You should describe your notation. You can use what is below as a starting point. —SS]

The structure of the MIS for modules comes from Hoffman and Strooper (1995), with the addition that template modules have been adapted from Ghezzi et al. (2003). The mathematical notation comes from Chapter 3 of Hoffman and Strooper (1995). For instance, the symbol := is used for a multiple assignment statement and conditional rules follow the form $(c_1 \Rightarrow r_1 | c_2 \Rightarrow r_2 | ... | c_n \Rightarrow r_n)$.

The following table summarizes the primitive data types used by Chest Scan.

Data Type	**Notation**	**Description**
character	char	a single symbol or digit
integer	$\mathbb{Z}$	a number without a fractional component in (-∞, ∞)
natural number	$\mathbb{N}$	a number without a fractional component in [1, ∞)
real	$\mathbb{R}$	any number in (-∞, ∞)

The specification of **X-RayChestScan** uses some derived data types, such as sequences, strings, and tuples. Sequences are lists containing elements of the same data type. Strings are sequences of characters. Tuples contain a list of values, potentially of different types.

Additionally, **X-RayChestScan** defines functions, where inputs and outputs are described by their data types. Local functions are defined using type signatures followed by their specifications.

5 Module Decomposition

The following table is taken directly from the Module Guide document for this project.

Level 1	**Level 2**
Hardware-Hiding	HardwareAcceleration
Behaviour-Hiding	DataPreprocessing SyntheticImageGen DatasetHandler EvaluationMetrics LoggingAndMonitoring UserInterface Login
Software Decision	DiffusionModel ImageExport IntegrationModule

Table 2: Module Hierarchy

6 MIS of HardwareAcceleration Module

6.1 Module

HardwareAcceleration
This module provides hardware-level optimizations using GPUs or TPUs for accelerating the computational tasks of the diffusion model, ensuring efficient training and generation of synthetic chest X-ray images.

6.2 Uses

- Relies on backend frameworks like CUDA, TensorFlow, or PyTorch for communication with hardware accelerators.
- Used by other modules that require intensive computation to offload tasks for performance optimization.

6.3 Syntax

6.3.1 Exported Constants

None.

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
initializeHardware	Device ID, Memory Allocation	Initialization Confirmation	InvalidConfigError
executeTask	Task Configuration, Input Data	Computed Results	ResourceExhaustionError
monitorUsage	None	Hardware Utilization Statistics	None

6.4 Semantics

6.4.1 State Variables

- **allocatedResources**: Tracks the allocated GPU/TPU resources.
- **taskQueue**: Stores tasks pending execution that require performance optimization.

6.4.2 Environment Variables

- Access to GPU/TPU hardware for execution.

6.4.3 Assumptions

- The execution environment includes at least one GPU or TPU with sufficient memory.
- Backend frameworks are correctly installed and configured.

6.4.4 Access Routine Semantics

initializeHardware(DeviceID, MemAllocParams):

- **Transition:** Updates **allocatedResources** with details of the initialized hardware.
- **Output:** Returns confirmation of hardware initialization.
- **Exceptions:** Throws **InvalidConfigError** if initialization parameters are incorrect or not supported.

executeTask(TaskConfig, InputData):

- **Transition:** Updates **taskQueue** with a new task. Executes computation using available hardware resources.
- **Output:** Computation results for the task.
- **Exceptions:** Throws **ResourceExhaustionError** if memory or compute limits are exceeded.

monitorUsage():

- **Transition:** None.
- **Output:** Current utilization metrics of GPU/TPU resources (e.g., memory, compute usage).
- **Exceptions:** None.

6.4.5 Local Functions

- **allocateResources()**: Dynamically allocates GPU/TPU resources based on task requirements.
- **scheduleTask()**: Handles task scheduling and prioritization in the task queue.
- **cleanupResources()**: Frees resources post-computation to avoid memory leaks.

7 MIS of DataPreprocessing Module

7.1 Module

DataPreprocessing
This module is responsible for preparing raw chest X-ray image datasets by applying data cleaning, augmentation, and transformation techniques to produce clean, ready-to-use data for input into the diffusion model.

7.2 Uses

- Model requires data that is preprocessed and clean before it is trained for the most effective result.

7.3 Syntax

7.3.1 Exported Constants

None.

7.3.2 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
preprocessData	Dataset, preprocess config	Preprocessed Dataset	TransformationError
augmentData	Dataset, augmentation config	Augmented Dataset	AugmentationError

7.4 Semantics

7.4.1 State Variables

- **rawDataset**: Stores the raw dataset being processed.
- **preprocessConfig**: Tracks the configuration for data preprocessing and augmentation operations.

7.4.2 Environment Variables

- None.

7.4.3 Assumptions

- Configuration files for transformations and augmentations are valid and correctly specified.

7.4.4 Access Routine Semantics

preprocessData(DatasetObj, TransformConfig):

- **Transition:** Applies specified transformations (e.g., normalization, resizing) to the dataset. Updates the raw dataset with the transformed dataset.
- **Output:** Preprocessed dataset.
- **Exceptions:** Throws **TransformationError** if any transformation fails (e.g., invalid parameters).

augmentData(DatasetObj, AugmentConfig):

- **Transition:** Applies data augmentation techniques (e.g., rotations, flips) to the dataset. Updates the preprocessed dataset with the augmented dataset.
- **Output:** Augmented dataset object.
- **Exceptions:** Throws **AugmentationError** for failed augmentations.

7.4.5 Local Functions

- **validateDataset()**: Ensures the dataset meets input requirements before processing.

8 MIS of SyntheticImageGen Module

8.1 Module

SyntheticImageGen
This module generates high-quality synthetic chest X-ray images that mimic real-world data. It enhances training datasets and simulates diverse imaging conditions for robust model evaluation.

8.2 Uses

- Used by model training and evaluation modules to provide augmented datasets for improved performance and robustness.
- Utilizes diffusion models to simulate realistic medical images.

8.3 Syntax

8.3.1 Exported Constants

None.

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
InitializeGenerator	ConfigParameters	Generator Instance	InvalidConfigError
generateImages	GeneratorInstance, Image specifications	Synthetic Image Dataset	GenerationError
saveGeneratedImages	Synthetic Image Dataset, Output file Path	Confirmation that images are saved	FileWriteError

8.4 Semantics

8.4.1 State Variables

- **generatorConfig**: Stores configuration details for the image generation process.
- **generatedImages**: Tracks synthetic images created during the session.

8.4.2 Environment Variables

- Requires access to GPU resources for computationally intensive image generation.
- File system access for saving synthetic datasets.

8.4.3 Assumptions

- Input configurations are valid and specify all necessary parameters.
- The environment has the required computational resources to execute the process.

8.4.4 Access Routine Semantics

initializeGenerator(ConfigParams):

- **Transition:** Sets up the image generator based on configuration parameters. Updates **generatorConfig**.
- **Output:** Generator instance ready for image generation.
- **Exceptions:** Throws **InvalidConfigError** if the configuration parameters are invalid.

generateImages(GenInstance, ImageParams):

- **Transition:** Uses the generator instance to create synthetic images based on the specified parameters.
- **Output:** A dataset of synthetic chest X-ray images.
- **Exceptions:** Throws **GenerationError** if the image generation fails.

saveGeneratedImages(SyntheticDataset, OutputPath):

- **Transition:** Saves the synthetic dataset to the specified file/folder.
- **Output:** Receives confirmation that the dataset was successfully saved.
- **Exceptions:** Throws **FileWriteError** if the save operation fails (e.g., non-existing file path).

9 MIS of DatasetHandler Module

9.1 Module

DatasetHandler
This module manages the storage, retrieval, and organization of large volumes of medical imaging data. It ensures seamless compatibility with preprocessing and other essential pipelines while maintaining efficient data handling practices.

9.2 Uses

- Utilized by preprocessing, training, and analysis modules for accessing and managing medical imaging datasets.
- Provides a consistent interface for handling large-scale data in a backend storage system.

9.3 Syntax

9.3.1 Exported Constants

None.

9.3.2 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
storeData	Dataset, Destination path	None	DataStorageError
retrieveData	Data parameters	Retrieved Dataset	DataRetrievalError
deleteData	Dataset Identifier	None	DeletionError

9.4 Semantics

9.4.1 State Variables

- **storageConfig**: Stores the configuration details for the backend data storage system.
- **dataRegistry**: Tracks metadata and identifiers for all datasets managed by the system.

9.4.2 Environment Variables

- File system or cloud storage access for storing and retrieving datasets.
- Credentials for connectivity with the backend data management system.

9.4.3 Assumptions

- Input datasets are in a supported format (e.g., DICOM).
- Sufficient storage space is available for dataset operations.
- The backend system is correctly configured and operational.

9.4.4 Access Routine Semantics

storeData(Dataset, DestinationPath):

- **Transition:** Stores the dataset in the specified location. Updates **dataRegistry** with metadata for that dataset.
- **Output:** None.
- **Exceptions:** Throws **DataStorageError** for issues like insufficient storage or invalid paths.

retrieveData(DataParams):

- **Transition:** Fetches the dataset based on the provided parameters.
- **Output:** Retrieved dataset.
- **Exceptions:** Throws **DataRetrievalError** if the query is invalid or the dataset cannot be found.

deleteData(DatasetID):

- **Transition:** Removes the specified dataset from storage. Updates the metadata in **dataRegistry**.
- **Output:** None.
- **Exceptions:** Throws **DeletionError** if the dataset cannot be found or deletion fails.

9.4.5 Local Functions

- **validateStorageConfig()**: Verifies the storage configuration parameters are valid.
- **updateRegistry()**: Manages metadata and tracks changes to the datasets.
- **optimizeStorage()**: Performs optimization tasks (e.g., compressing or reorganizing data) to optimize the use of storage.

9.4.6 Local Functions

- **validateGeneratorConfig()**: Validates the generator configurations for synthetic image creation.
- **validateImageParams()**: Validates the input parameters for synthetic image generation.

10 MIS of EvaluationMetrics Module

10.1 Module

EvaluationMetrics
This module provides metrics and evaluation techniques to assess the quality of synthetic images and model performance. It computes quantitative evaluations, ensuring the system's effectiveness is monitored and validated.

10.2 Uses

- Used by analysis and validation modules to measure the system's performance and the quality of synthetic images.
- Provides feedback for model improvement and fine-tuning.

10.3 Syntax

10.3.1 Exported Constants

None.

10.3.2 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
computeMetrics	Ground Truth, Predictions, Metric Type	Computed Metric Score	MetricComputationError
genEvaluationReport	Evaluation Results, Output Path	Report Confirmation	FileWriteError

Table 3: Exported Access Programs for EvaluationMetrics

10.4 Semantics

10.4.1 State Variables

- **evaluationResults**: Stores the results of computed evaluation metrics.

10.4.2 Environment Variables

- Requires access to ground truth data and predictions for evaluation.
- File system access for saving evaluation reports.

10.4.3 Assumptions

- Input data for metrics computation is correctly formatted (e.g., binary or multi-class labels).
- The evaluation system operates in an environment with access to sufficient computational resources.

10.4.4 Access Routine Semantics

computeMetrics(GroundTruth, Predictions, MetricType):

- **Transition**: Computes the selected evaluation metric (accuracy, recall, F1-score, etc.) based on the provided ground truth and prediction data.
- **Output**: Computed metric score as a floating-point value.
- **Exceptions**: Throws `MetricComputationError` if inputs are incompatible or computation fails.

generateEvaluationReport(EvaluationResults, OutputPath):

- **Transition**: Writes the computed evaluation results to a formatted report file in the specified output path.
- **Output**: Confirmation message for successful report generation.
- **Exceptions**: Throws `FileWriteError` if the file cannot be written due to permission or storage issues.

11 MIS of LoggingAndMonitoring Module

11.1 Module

LoggingAndMonitoring
This module tracks system performance, logs activities, and monitors errors or anomalies in real-time. It ensures the system's reliability and maintainability by providing actionable insights into its operation.

11.2 Uses

- Utilized by the system to log activities, detect anomalies, and monitor performance metrics.
- Provides support for debugging, maintenance, and performance tuning.

11.3 Syntax

11.3.1 Exported Constants

None.

11.3.2 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
logEvent	Log Message, Severity Level	None	LogWriteError
monitorPerformance	None	Performance Metrics	MonitoringError
reportAnomalies	Anomaly Details	Anomaly Report	AnomalyReportError

Table 4: Exported Access Programs for LoggingAndMonitoring Module

11.4 Semantics

11.4.1 State Variables

- **logRecords**: Stores logged messages and associated metadata.
- **monitoringData**: Tracks real-time system performance metrics.

11.4.2 Environment Variables

- Access to the file system or cloud-based logging services for storing logs and reports.
- Integration with performance monitoring tools or frameworks.

11.4.3 Assumptions

- The system is configured to allow monitoring of key metrics and resources.

11.4.4 Access Routine Semantics

logEvent(LogMessage, SeverityLevel):

- **Transition**: Adds a log entry to `logRecords` with the specified message and severity level.
- **Output**: None.
- **Exceptions**: Throws `LogWriteError` for file system issues or invalid log inputs.

monitorPerformance():

- **Transition**: Collects and updates `monitoringData` with current performance metrics (e.g., CPU usage).
- **Output**: Performance metrics in a structured format.
- **Exceptions**: Throws `MonitoringError` if metrics cannot be determined.

reportAnomalies(AnomalyDetails):

- **Transition**: Generates a detailed anomaly report based on provided details and updates `logRecords`.
- **Output**: Anomaly report.
- **Exceptions**: Throws `AnomalyReportError` if the reporting process fails.

12 MIS of DiffusionModel Module

12.1 Module

DiffusionModel
This module implements the core diffusion model architecture, which is used for generating and refining chest X-ray images.

12.2 Uses

- Used by synthetic image generation and enhancement modules to create realistic medical images.
- Relies on underlying mathematical models and GPU/TPU acceleration for efficient computations.

12.3 Syntax

12.3.1 Exported Constants

None.

12.3.2 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
initializeModel	Config Parameters	Model Instance	ModelInitError
generateImage	Model Instance, Input Noise, Parameters	Synthetic Image	ImageGenerationEr
refineImage	Model Instance, Input Image, Parameters	Refined Image	RefinementError
trainModel	Training Data, Hyperparameters	Training Metrics	TrainingError

Table 5: Exported Access Programs for DiffusionModel Module

12.4 Semantics

12.4.1 State Variables

- **modelParameters**: Stores the configuration details and parameters of the diffusion model.
- **trainingState**: Tracks the training progress, including loss metrics and checkpoints.

12.4.2 Environment Variables

- Access to computational resources (e.g., GPU) for training and generation tasks.

12.4.3 Assumptions

- Input data for training and generation is preprocessed and in the required format.
- The execution system contains sufficient computational resources for this process.

12.4.4 Access Routine Semantics

initializeModel(ConfigParams):

- **Transition**: Sets up the diffusion model based on the provided configuration parameters. Updates `modelParameters`.
- **Output**: Model instance ready for image generation or training.
- **Exceptions**: Throws `ModelInitError` if configuration is invalid or the initialization fails.

generateImage(ModelInstance, InputNoise, Parameters):

- **Transition**: Uses the diffusion model to generate a synthetic image from input noise.
- **Output**: A synthetic chest X-ray image.
- **Exceptions**: Throws `ImageGenerationError` if the generation process fails.

refineImage(ModelInstance, InputImage, Parameters):

- **Transition**: Enhances the input image to improve its quality and realism.
- **Output**: Refined chest X-ray image.
- **Exceptions**: Throws `RefinementError` for invalid inputs or failed refinement operations.

trainModel(TrainingData, Hyperparameters):

- **Transition**: Updates `trainingState` based on training progress.
- **Output**: Training metrics, including loss and accuracy values.
- **Exceptions**: Throws `TrainingError` for invalid data or parameter issues.

13 MIS of ImageExport Module

13.1 Module

ImageExport
This module handles the formatting and export of generated or processed images in required formats (e.g., DICOM, PNG). It ensures compatibility with medical imaging standards and external tools.

13.2 Uses

- Used by synthetic image generation and refinement modules to save outputs in desired formats.
- Ensures exported images are compliant with medical imaging standards for further analysis.

13.3 Syntax

13.3.1 Exported Constants

None.

13.3.2 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
initializeExporter	Config Parameters	Exporter Instance	ExportInitError
exportImage	Image Object, Format, Destination Path	Export Confirmation	ExportError
validateExport	Image Object, Format	Validation Status	ValidationError

Table 6: Exported Access Programs for ImageExport Module

13.4 Semantics

13.4.1 State Variables

- **exportConfig**: Stores configuration details for image export formats and destinations.
- **exportedImages**: Tracks metadata of all exported images.

13.4.2 Environment Variables

- Access to file systems or cloud storage for saving exported images.
- Compatibility with external libraries or tools for format conversions (e.g., `pydicom` for DICOM).

13.4.3 Assumptions

- Input image objects are correctly formatted and meet the requirements of the specified export format.
- The target destination has sufficient storage and appropriate permissions.

13.4.4 Access Routine Semantics

initializeExporter(ConfigParams):

- **Transition**: Sets up the exporter instance based on configuration parameters. Updates `exportConfig`.
- **Output**: Exporter instance ready for image export tasks.
- **Exceptions**: Throws `ExportInitError` if initialization fails or configuration parameters are invalid.

exportImage(ImageObj, Format, DestPath):

- **Transition**: Formats the image as specified and saves it to the target destination. Updates `exportedImages` metadata.
- **Output**: Confirmation of successful image export.
- **Exceptions**: Throws `ExportError` if the export operation fails (e.g., unsupported format or file system issues).

validateExport(ImageObj, Format):

- **Transition**: Validates the image and format compatibility.
- **Output**: Validation status indicating success or failure.
- **Exceptions**: Throws `ValidationError` if the image or format fails compatibility checks.

14 MIS of UserInterface

14.1 Module

UserInterface
This module provides the graphical user interface (GUI) for the system, enabling users to input data, view results, and manage the system in an intuitive and user-friendly manner.

14.2 Uses

- Used by system users to interact with features such as data input, image generation, and result visualization.
- Relies on frontend frameworks (e.g., React, PyQt) for implementation and rendering.

14.3 Syntax

14.3.1 Exported Constants

None.

14.3.2 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
displayDashboard	None	Rendered Dashboard	RenderError
handleUserInput	User Actions, Input Data	Response	InputHandlingErro
updateUI	Updated Data	Updated Interface	UpdateError

Table 7: Exported Access Programs for UserInterface Module

14.4 Semantics

14.4.1 State Variables

- **inputCache**: Temporarily stores user input data before it is processed.

14.4.2 Environment Variables

- Browser or local desktop environment for rendering the GUI.
- Integration with backend services for real-time updates and data processing.

14.4.3 Assumptions

- The execution environment supports the frontend framework used.
- Backend services and APIs are operational and accessible to the frontend.

14.4.4 Access Routine Semantics

displayDashboard():

- **Transition**: Renders the main dashboard view with all relevant components.
- **Output**: A fully functional dashboard interface.
- **Exceptions**: Throws `RenderError` if rendering fails (e.g., missing assets or framework issues).

handleUserInput(UserActions, InputData):

- **Transition**: Processes user actions and inputs, updating the system accordingly. Updates `inputCache` as needed.
- **Output**: Response to the user's action, such as a confirmation or updated view.
- **Exceptions**: Throws `InputHandlingError` for invalid actions or data.

updateUI(UpdatedData):

- **Transition**: Updates the UI components with new data (e.g., refreshed results or status updates).
- **Output**: Successfully updated interface.
- **Exceptions**: Throws `UpdateError` if the update operation fails.

15 MIS of IntegrationModule

15.1 Module

IntegrationModule
This module implements strategies and mechanisms for integrating the system with external tools, medical databases, and IT infrastructure. It provides APIs and connectors to ensure seamless communication and interoperability with external systems.

15.2 Uses

- Used to establish connections with external systems for data exchange, authentication, and collaboration.
- Supports integration with medical databases and external APIs to enhance system functionality.

15.3 Syntax

15.3.1 Exported Constants

None.

15.3.2 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
initializeIntegration	Config Parameters	Integration Instance	IntegrationInitErro
connectToDatabase	Database Config	Connection Confirmation	DatabaseConnectio
callExternalAPI	API Endpoint, Request Parameters	API Response	APIRequestError

Table 8: Exported Access Programs for IntegrationModule

15.4 Semantics

15.4.1 State Variables

- **integrationConfig**: Stores the configuration details for integrations, including API keys and database credentials.
- **connectionPool**: Manages active connections to external systems.

15.4.2 Environment Variables

- Requires access to external networked resources (e.g., APIs, databases).
- Integration with authentication mechanisms for secure communication.

15.4.3 Assumptions

- External systems (e.g., APIs, databases) are operational and accessible.
- The configuration for integrations, including credentials, is valid and up-to-date.

15.4.4 Access Routine Semantics

initializeIntegration(ConfigParams):

- **Transition**: Sets up the integration module based on the provided configuration parameters. Updates `integrationConfig`.
- **Output**: Integration instance ready to interact with external systems.
- **Exceptions**: Throws `IntegrationInitError` for invalid configuration or initialization failures.

connectToDatabase(DatabaseConfig):

- **Transition**: Establishes a connection to the specified medical database. Updates `connectionPool`.
- **Output**: Confirmation of a successful database connection.
- **Exceptions**: Throws `DatabaseConnectionError` if the connection fails due to incorrect configuration or network issues.

callExternalAPI(APIEndpoint, RequestParams):

- **Transition**: Sends a request to an external API and retrieves the response.
- **Output**: API response containing requested data or status.
- **Exceptions**: Throws `APIRequestError` for request failures, such as timeout or invalid endpoints.

16 MIS of Login Module

16.1 Module

Login

This module provides the login functionality for users by displaying a login portal and authenticating credentials. It ensures secure access to the application for authorized users.

16.2 Uses

- Provides the entry point for user authentication and access control.
- Used by application modules requiring user-specific access or session management.

16.3 Syntax

16.3.1 Exported Constants

None.

16.3.2 Exported Access Programs

Name	In	Out	Exceptions
handleAuthentication	Username, Password	Authentication Status, Session Token	AuthenticationError, SessionCreationError
manageSession	Session Token, Action Type (Create/Logout)	Session Status	LogoutError, SessionCreationError

Table 9: Exported Access Programs for Login Module

16.4 Semantics

16.4.1 State Variables

- **activeSession**: Stores the current user's session details, including session token and expiration time.

16.4.2 Environment Variables

- Integration with the backend authentication system or database for credential verification.
- Access to a secure storage system for managing session tokens.

16.4.3 Assumptions

- User credentials are stored securely in a database or authentication service.
- The application environment supports encryption for secure transmission of user credentials.

16.4.4 Access Routine Semantics

handleAuthentication(Username, Password):

- **Transition**: Verifies user credentials against stored records. If valid, a session token is generated and stored in `activeSession`.
- **Output**: Returns authentication status (success/failure) and a session token upon success.
- **Exceptions**: Throws `AuthenticationError` for invalid credentials or backend issues. Throws `SessionCreationError` if session token generation fails.

manageSession(SessionToken, ActionType):

- **Transition**: If `ActionType` is "Create," a new session token is generated and stored in `activeSession`. If "Logout," the session token is invalidated.
- **Output**: Returns the updated session status.
- **Exceptions**: Throws `SessionCreationError` if token generation/storage fails. Throws `LogoutError` if logout fails.

References

Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.

Daniel M. Hoffman and Paul A. Strooper. *Software Design, Automated Testing, and Maintenance: A Practical Approach*. International Thomson Computer Press, New York, NY, USA, 1995. URL http://citeseer.ist.psu.edu/428727.html.
Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.

Daniel M. Hoffman and Paul A. Strooper. *Software Design, Automated Testing, and Maintenance: A Practical Approach*. International Thomson Computer Press, New York, NY, USA, 1995. URL http://citeseer.ist.psu.edu/428727.html.

17 Appendix

[Extra information if required —SS]

18 Appendix

18.1 Reflection Questions

The information in this section will be used to evaluate the team members on the graduate attribute of Problem Analysis and Design.

The purpose of reflection questions is to assess individual and team learning experiences and identify areas for future improvement. Reflection is an essential component of the software development process, ensuring continuous improvement and better decision-making.

18.1.1 1. What went well while writing this deliverable?

- Tasks were effectively divided among team members, ensuring steady progress.
- Using the **Module Guide (MG)** and **SRS** as references helped maintain consistency across documents.
- Regular team meetings and discussions clarified misunderstandings, ensuring each module was well-defined and fit into the overall architecture.
- The design process was clearly outlined, allowing for efficient documentation of our **diffusion model** approach.
- Our prior research and understanding of **painting and bounding boxes for diffusion models** helped us articulate the technical aspects clearly.

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

- Ensuring **consistency across documents** was challenging, as some module definitions overlapped or lacked clarity.
  - **Resolution:** We refined definitions in the **MIS** to remove ambiguities and iteratively reviewed sections for alignment.
- **Formatting in LaTeX** was another hurdle, especially when working with tables and cross-references.
  - **Resolution:** Using **Overleaf** streamlined collaboration and debugging.
- **Defining functional vs. non-functional requirements** was tricky due to the performance-based and interpretability-based nature of diffusion models.
  - **Resolution:** We categorized requirements explicitly after team discussions.
- **Bounding box accuracy** in diffusion models presented challenges in defining precision.

  - **Resolution:** We referred to industry benchmarks and medical imaging standards to define thresholds.

18.1.3 3. Which design decisions stemmed from clients/peers, and which came from research?

Decisions influenced by stakeholders:

- **HardwareAcceleration**: Ensured compatibility with GPUs/TPUs for efficiency based on stakeholder feedback.
- **Bounding boxes for interpretability**: Based on radiologists' need for clear anomaly indicators.
- **Integration with PACS**: Shaped by discussions on existing workflows in medical imaging.

Decisions based on research:

- **Diffusion models**: Chosen for high-quality synthetic X-rays based on recent literature.
- **Evaluation metrics**: Adopted standard quality assessments from medical AI research.
- **Training dataset selection**: Publicly available datasets were selected due to limited access to proprietary hospital data.

18.1.4 4. What parts of other documents needed changes, and why?

- **SRS Functional Requirements**: Clarified the **difference between DatasetHandler and DataPreprocessing** to avoid redundancy.
- **Traceability Matrix**: Adjusted **anticipated changes (AC5)** to include **IntegrationModule** for handling data exports.
- **Hazard Analysis**: Expanded to include risks related to false positives and false negatives in AI-based diagnostics.
- **Mission Goals (MG)**: Revised to better reflect user workflows.
- **User Interaction Model**: Refined to ensure radiologists could override AI suggestions when necessary.

18.1.5 5. What are the limitations of your solution?

- **Hardware constraints**: Dependence on GPUs/TPUs may limit accessibility for smaller institutions.
- **Data availability**: Limited access to real medical datasets impacts training quality.
- **Regulatory compliance**: More effort is required to meet **HIPAA/GDPR** standards for handling medical data.
- **AI explainability**: Additional interpretability tools could provide clearer reasoning behind outputs.
- **Computational efficiency**: Diffusion models require significant GPU resources, limiting real-time applications.
- **Robustness**: Model sensitivity to dataset biases requires a more diverse dataset.

18.1.6 6. What alternative designs were considered, and why was this one chosen?

Design	Pros	Cons
Monolithic	Simple, easier to debug	Hard to scale, tightly coupled components
Microservices	Scalable, allows independent updates	Increased complexity in API management
Event-Driven	Efficient for real-time data processing	More difficult to debug, not necessary for our use case
CNN-based classification	Faster inference, well-documented in medical AI	No spatial localization, prone to overfitting
U-Net for segmentation	Better at localizing abnormalities	Requires large labeled datasets, high annotation costs
Transformer-based models	Models long-range dependencies effectively	Computationally expensive, less mature in medical AI
Diffusion model with inpainting	High fidelity, better at uncertainty estimation	Slower than CNNs, requires fine-tuning

Table 10: Comparison of Alternative Design Choices

We chose **modular architecture with diffusion models** because it balances **flexibility, maintainability, and scalability** without adding unnecessary complexity. The diffusion models' ability to handle **uncertainty modeling** and provide **spatial localization** with bounding boxes aligned with stakeholder needs.