Module Interface Specification for CXR

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

January 10, 2025

1 Revision History

Date	Version	Notes
Date 1	1.0	Notes
Date 2	1.1	Notes

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at [give url —SS]

[Also add any additional symbols, abbreviations or acronyms —SS]

Contents

3 Introduction

The following document details the Module Interface Specifications for [Fill in your project name and description —SS]

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at .... [provide the url for your repo —SS]

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 CXR.

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 CXR uses some derived data types: sequences, strings, and tuples. Sequences are lists filled with elements of the same data type. Strings are sequences of characters. Tuples contain a list of values, potentially of different types. In addition, CXR uses functions, which are defined by the data types of their inputs and outputs. Local functions are described by giving their type signature followed by their specification.

5 Module Decomposition

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

Level 1	**Level 2**
Hardware-Hiding	
Behaviour-Hiding	Input Parameters Output Format Output Verification Temperature ODEs Energy Equations Control Module Specification Parameters Module
Software Decision	Sequence Data Structure ODE Solver Plotting

Table 1: Module Hierarchy

6 MIS of Web Application Server Module

6.1 Module

Web Application Server

6.2 Uses

M2: HTTP Server Module
M5: Doctor Profile View Module
M6: Patient List View Module

6.3 Syntax

6.3.1 Exported Constants

NA

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
handleRequest	HTTP request	HTTP response	InvalidRequestException

6.4 Semantics

6.4.1 State Variables

- **sessionData**: Stores current session information.
- **activeUsers**: Keeps track of currently active users.

6.4.2 Environment Variables

- **serverPort**: The port on which the server listens for incoming connections.
- **hostAddress**: The server's host address.

6.4.3 Assumptions

- Assumes the HTTP Server Module (M2) is properly configured and running.
- Assumes valid HTTP requests are received.

6.4.4 Access Routine Semantics

handleRequest(request)

- transition: Processes the incoming HTTP request and routes it to the appropriate view module.
- output: Returns the HTTP response based on the request.

6.4.5 Local Functions

- **parseRequest(request)**: Parses the incoming HTTP request to extract necessary information.
- **generateResponse(data)**: Constructs an HTTP response based on the processed data.
- **authenticateUser(credentials)**: Verifies the user's credentials before processing the request.

7 HTTP Server Module

7.1 Other Modules the Current Module Uses

- M1: Web Application Server Module
- M3: Disease Prediction Server Module
- M4: Disease Progression Tracking Server Module
- M5: Doctor Profile View Module
- M6: Patient List View Module
- Other view modules (M7-M10) for displaying data.

7.2 State Variables

- **requestQueue**: Holds incoming HTTP requests until they are processed.
- **responseQueue**: Holds outgoing HTTP responses that need to be sent back to clients.

7.3 Exported Constants and Access Programs

7.3.1 Exported Access Programs

Name	In	Out	Exceptions
`startServer`	None	None	`ServerStartException`
`stopServer`	None	None	`ServerStopException`
`processRequest`	`HTTP request`	`HTTP response`	`InvalidRequestException`

7.3.2 Exported Constants

- **SERVER_PORT**: 8080
- **MAX_CONNECTIONS**: 100

7.4 Environment Variables

- **serverPort**: The port on which the server listens for incoming HTTP connections.
- **maxConnections**: Maximum number of simultaneous connections the server can handle.

7.5 Assumptions

- Assumes the Web Application Server Module (M1) is properly configured and running.
- Assumes incoming HTTP requests are formatted correctly.

7.6 Access Routine Semantics

7.6.1 startServer()

- **Transition**: Starts the HTTP server, initializes necessary resources, and begins listening for incoming requests.
- **Output**: No output, but may throw a `ServerStartException` if the server cannot be started.

7.6.2 stopServer()

- **Transition**: Stops the HTTP server, gracefully shuts down connections.
- **Output**: No output, but may throw a `ServerStopException` if the server cannot be stopped.

7.6.3 processRequest(request)

- **Transition**: Takes an incoming HTTP request and processes it, routing it to the appropriate server module or view module.
- **Output**: Returns an HTTP response based on the processed request.

7.7 Local Functions

- **parseRequest()**: Parses the incoming HTTP request to extract necessary information such as headers and parameters.
- **generateResponse()**: Constructs an HTTP response based on the processed data from the request.
- **handleError()**: Handles errors that arise during request processing and generates appropriate error responses.

8 Disease Prediction Server Module

8.1 Other Modules the Current Module Uses

- M1: Web Application Server Module

- M2: HTTP Server Module
- M4: Disease Progression Tracking Server Module
- M5: Doctor Profile View Module
- M6: Patient List View Module
- M7: Patient Diseases Progression View Module

8.2 State Variables

- **model**: The pre-trained model from `torchxrayvision` used for predicting lung diseases from X-ray images.
- **modelAccuracy**: Tracks the accuracy of the current model after training and validation.
- **predictionThreshold**: A constant threshold to determine the classification outcome (e.g., disease presence).
- **patientImageData**: Holds the chest X-ray image data used for prediction.

8.3 Exported Constants and Access Programs

8.3.1 Exported Access Programs

Name	In	Out	Exceptions
`loadModel`	None	Loaded model	`ModelLoadException`
`predictDisease`	X-ray image data	Disease prediction	`InvalidImageException`

8.3.2 Exported Constants

- **PREDICTION_THRESHOLD**: 0.75 (threshold for classification of disease presence)
- **MODEL_PATH**: Path to the pre-trained model (e.g., `./models/chest_xray_model.pth`)
- **MAX_PREDICTIONS**: 1000 (maximum number of predictions to handle concurrently)

8.4 Environment Variables

- **modelPath**: The path where the `torchxrayvision` pre-trained model is saved or loaded from.
- **predictionEndpoint**: The endpoint for making predictions using chest X-ray images.

8.5 Assumptions

- Assumes the pre-trained `torchxrayvision` model is available and compatible with the data provided.
- Assumes valid X-ray image data is available for predictions.
- Assumes the Web Application Server Module (M1) and HTTP Server Module (M2) are properly configured and running.

8.6 Access Routine Semantics

8.6.1 loadModel()

- **Transition**: Loads the pre-trained disease prediction model from the specified path after image is uploaded using `torchxrayvision`.

8.6.2 predictDisease(patientImageData)

- **Transition**: Uses the loaded model to make predictions based on the provided X-ray image data.
- **Output**: Returns the disease prediction (e.g., probability of a disease being present) or throws an `InvalidImageException` if the image is invalid.

8.7 Local Functions

- **loadModel()**: Loads the pre-trained model from disk or cloud storage using `torchxrayvision`'s functionality.
- **evaluateModel()**: Evaluates the model's performance with a test dataset to calculate metrics like accuracy and sensitivity.
- **preprocessImage()**: Preprocesses incoming X-ray image data to fit the model's input requirements (e.g., resizing, normalization).
- **postprocessPrediction()**: Processes the raw output from the model (e.g., probabilities) into a human-readable format (e.g., disease labels).

9 Disease Progression Tracking Server Module

9.1 Other Modules the Current Module Uses

- M1: Web Application Server Module
- M2: HTTP Server Module

- M3: Disease Prediction Server Module
- M5: Doctor Profile View Module
- M6: Patient List View Module
- M7: Patient Overview Module
- M8: Patient Diseases Progression View Module

9.2 State Variables

- **progressionData**: Stores historical data of disease progression for each patient.
- **timeStamps**: Records the dates and times when progression data is captured.
- **patientHistory**: Maintains a detailed history of each patient's disease states over time.

9.3 Exported Constants and Access Programs

9.3.1 Exported Access Programs

Name	In	Out	Exceptions
`trackProgression`	Patient ID, data	Confirmation of tracking	`DataNotFoundException`
`getProgressionHistory`	Patient ID	Progression history	`DataNotFoundException`

9.3.2 Exported Constants

- **DATA_RETENTION_PERIOD**: 5 years (duration for storing disease progression data)
- **TRACKING_INTERVAL**: 30 days (standard interval for recording progression data)
- **MAX_HISTORY_ENTRIES**: 10000 (maximum number of progression entries per patient)

9.4 Environment Variables

- **dataStoragePath**: The path where progression tracking data is stored.
- **updateInterval**: The time interval for automatically updating progression data.

9.5 Assumptions

- Assumes valid and accurate disease prediction data is available from M3.
- Assumes patients' data is regularly updated.
- Assumes the Web Application Server Module (M1) and HTTP Server Module (M2) are properly configured and running.

9.6 Access Routine Semantics

9.6.1 trackProgression(patientID, data)

- **Transition**: Stores new disease progression data for the given patient.
- **Output**: Returns confirmation of data storage or throws a `DataNotFoundException` if the patient data is not found.

9.6.2 getProgressionHistory(patientID)

- **Transition**: Retrieves historical progression data for the specified patient.
- **Output**: Returns the progression history or throws a `DataNotFoundException` if no history is found.

9.7 Local Functions

- **updateProgressionData()**: Updates the disease progression data at regular intervals based on new predictions or patient information.
- **analyzeProgressionTrends()**: Analyzes progression data to identify trends or anomalies in disease progression.
- **archiveOldData()**: Moves data older than the retention period to an archive for long-term storage.

10 Doctor Profile View Module

10.1 Other Modules the Current Module Uses

- M1: Web Application Server Module
- M2: HTTP Server Module
- M11: User Authentication Module
- M15: Data Persistence Module

10.2 State Variables

- **doctorProfile**: Stores the profile information of the currently logged-in doctor, including name, specialty, contact details, and credentials.

10.3 Exported Constants and Access Programs

10.3.1 Exported Access Programs

Name	In	Out	Exceptions
`viewDoctorProfile`	Doctor ID	Doctor profile details	`ProfileNotFoundException`

10.3.2 Exported Constants

- **PROFILE_UPDATE_INTERVAL**: 24 hours (time interval for automatic profile updates)
- **DEFAULT_PROFILE_PICTURE**: "default_doctor.png" (default profile picture for doctors without a custom one)

10.4 Environment Variables

- **profileDataPath**: Path to the data source containing doctor profile information.
- **authenticationService**: URL or endpoint of the service used for user authentication.

10.5 Assumptions

- Assumes that the User Authentication Module (M11) ensures only authorized doctors can view their profiles.
- Assumes the profile data is accurately stored and updated in the Data Persistence Module (M15).

10.6 Access Routine Semantics

10.6.1 viewDoctorProfile(doctorID)

- **Transition**: Retrieves the profile information of the specified doctor.
- **Output**: Returns the doctor profile details or throws a `ProfileNotFoundException` if the profile cannot be found.

10.7 Local Functions

- **fetchProfileData(doctorID)**: Retrieves profile data from the data source.
- **updateProfilePicture(doctorID, picturePath)**: Updates the profile picture of the doctor.
- **logProfileAccess(doctorID)**: Logs each time a doctor accesses their profile for auditing purposes.

11 Patient List View Module

11.1 Other Modules the Current Module Uses

- M1: Web Application Server Module
- M2: HTTP Server Module
- M11: Patient Medical Record View Module
- M15: Patient Medical Record Update Module

11.2 State Variables

- **patientList**: Stores a list of patients assigned to a doctor, including their basic information such as name, age, and medical condition.
- **searchFilters**: Stores the current filters applied to the patient list for sorting and searching purposes.

11.3 Exported Constants and Access Programs

11.3.1 Exported Access Programs

Name	In	Out	Exceptions
`viewPatientList`	Doctor ID, filters	List of patient details	`PatientListNotFoundException`
`applyFilter`	Filter parameters	Filtered patient list	`InvalidFilterException`
`sortList`	Sorting criteria	Sorted patient list	None

11.3.2 Exported Constants

- **MAX_PATIENTS_PER_PAGE**: 20 (maximum number of patients displayed per page in the list)
- **DEFAULT_SORT_ORDER**: "alphabetical" (default order in which patients are listed)

11.4 Environment Variables

- **patientDataPath**: Path to the data source containing patient records.
- **authenticationService**: URL or endpoint of the service used for user authentication.

11.5 Assumptions

- Assumes that the User Authentication Module (M11) is responsible for verifying that the user is a doctor with access to the list.
- Assumes the patient data is up-to-date and synchronized with the Data Persistence Module (M15).

11.6 Access Routine Semantics

11.6.1 viewPatientList(doctorID, filters)

- **Transition**: Retrieves a list of patients associated with the given doctor, applying the specified filters.
- **Output**: Returns the list of patient details or throws a `PatientListNotFoundException` if no patients are found.

11.6.2 applyFilter(filters)

- **Transition**: Applies the given filters to the current patient list.
- **Output**: Returns the filtered patient list or throws an `InvalidFilterException` if the filters are invalid.

11.6.3 sortList(criteria)

- **Transition**: Sorts the current patient list based on the provided criteria.
- **Output**: Returns the sorted patient list.

11.7 Local Functions

- **filterPatients(filters)**: Filters the patient list according to the specified parameters.
- **sortPatients(criteria)**: Sorts the patient list based on the provided criteria.
- **logListAccess(doctorID)**: Logs each time a doctor accesses the patient list for auditing purposes.

12 Module NAME HERE!!!

12.1 Other Modules the Current Module Uses

- fill this

12.2 State Variables

- **Title**: fill this

12.3 Exported Constants and Access Programs

12.3.1 Exported Access Programs

Name	In	Out	Exceptions
`fill 1`	fill 2	fill 3	`fill 4`
`fill 1`	fill 2	fill 3	`fill 4`

12.3.2 Exported Constants

- **Title**: file this

12.4 Environment Variables

- fill this

12.5 Assumptions

- fill this

12.6 Access Routine Semantics

12.6.1 trackProgression(patientID, data)

- fill this

12.6.2

- fill this

12.7 Local Functions

- fill this

13 Module NAME HERE!!!

13.1 Other Modules the Current Module Uses

- fill this

13.2 State Variables

- **Title**: fill this

13.3 Exported Constants and Access Programs

13.3.1 Exported Access Programs

Name	In	Out	Exceptions
`fill 1`	fill 2	fill 3	`fill 4`
`fill 1`	fill 2	fill 3	`fill 4`

13.3.2 Exported Constants

- **Title**: file this

13.4 Environment Variables

- fill this

13.5 Assumptions

- fill this

13.6 Access Routine Semantics

13.6.1 trackProgression(patientID, data)

- fill this

13.6.2

- fill this

13.7 Local Functions

- fill this

14 Module NAME HERE!!!

14.1 Other Modules the Current Module Uses

- fill this

14.2 State Variables

- **Title**: fill this

14.3 Exported Constants and Access Programs

14.3.1 Exported Access Programs

Name	In	Out	Exceptions
`fill 1`	fill 2	fill 3	`fill 4`
`fill 1`	fill 2	fill 3	`fill 4`

14.3.2 Exported Constants

- **Title**: file this

14.4 Environment Variables

- fill this

14.5 Assumptions

- fill this

14.6 Access Routine Semantics

14.6.1 trackProgression(patientID, data)

- fill this

14.6.2

- fill this

14.7 Local Functions

- fill this

15 Patient Medical Record View Module

15.1 Other Modules the Current Module Uses

- fill this

15.2 State Variables

- **Title**: fill this

15.3 Exported Constants and Access Programs

15.3.1 Exported Access Programs

Name	In	Out	Exceptions
`fill 1`	fill 2	fill 3	`fill 4`
`fill 1`	fill 2	fill 3	`fill 4`

15.3.2 Exported Constants

- **Title**: file this

15.4 Environment Variables

- fill this

15.5 Assumptions

- fill this

15.6 Access Routine Semantics

15.6.1 trackProgression(patientID, data)

- fill this

15.6.2

- fill this

15.7 Local Functions

- fill this

16 Module NAME HERE!!!

16.1 Other Modules the Current Module Uses

- fill this

16.2 State Variables

- **Title**: fill this

16.3 Exported Constants and Access Programs

16.3.1 Exported Access Programs

Name	In	Out	Exceptions
`fill 1`	fill 2	fill 3	`fill 4`
`fill 1`	fill 2	fill 3	`fill 4`

16.3.2 Exported Constants

- **Title**: file this

16.4 Environment Variables

- fill this

16.5 Assumptions

- fill this

16.6 Access Routine Semantics

16.6.1 trackProgression(patientID, data)

- fill this

16.6.2

- fill this

16.7 Local Functions

- fill this

17 Module NAME HERE!!!

17.1 Other Modules the Current Module Uses

- fill this

17.2 State Variables

- **Title**: fill this

17.3 Exported Constants and Access Programs

17.3.1 Exported Access Programs

Name	In	Out	Exceptions
`fill 1`	fill 2	fill 3	`fill 4`
`fill 1`	fill 2	fill 3	`fill 4`

17.3.2 Exported Constants

- **Title**: file this

17.4 Environment Variables

- fill this

17.5 Assumptions

- fill this

17.6 Access Routine Semantics

17.6.1 trackProgression(patientID, data)

- fill this

17.6.2

- fill this

17.7 Local Functions

- fill this

18 Module NAME HERE!!!

18.1 Other Modules the Current Module Uses

- fill this

18.2 State Variables

- **Title**: fill this

18.3 Exported Constants and Access Programs

18.3.1 Exported Access Programs

Name	In	Out	Exceptions
`fill 1`	fill 2	fill 3	`fill 4`
`fill 1`	fill 2	fill 3	`fill 4`

18.3.2 Exported Constants

- **Title**: file this

18.4 Environment Variables

- fill this

18.5 Assumptions

- fill this

18.6 Access Routine Semantics

18.6.1 trackProgression(patientID, data)

- fill this

18.6.2

- fill this

18.7 Local Functions

- fill this

19 Module NAME HERE!!!

19.1 Other Modules the Current Module Uses

- fill this

19.2 State Variables

- **Title**: fill this

19.3 Exported Constants and Access Programs

19.3.1 Exported Access Programs

Name	In	Out	Exceptions
`fill 1`	fill 2	fill 3	`fill 4`
`fill 1`	fill 2	fill 3	`fill 4`

19.3.2 Exported Constants

- **Title**: file this

19.4 Environment Variables

- fill this

19.5 Assumptions

- fill this

19.6 Access Routine Semantics

19.6.1 trackProgression(patientID, data)

- fill this

19.6.2

- fill this

19.7 Local Functions

- fill this

20 Module NAME HERE!!!

20.1 Other Modules the Current Module Uses

- fill this

20.2 State Variables

- **Title**: fill this

20.3 Exported Constants and Access Programs

20.3.1 Exported Access Programs

Name	In	Out	Exceptions
`fill 1`	fill 2	fill 3	`fill 4`
`fill 1`	fill 2	fill 3	`fill 4`

20.3.2 Exported Constants

- **Title**: file this

20.4 Environment Variables

- fill this

20.5 Assumptions

- fill this

20.6 Access Routine Semantics

20.6.1 trackProgression(patientID, data)

- fill this

20.6.2

- fill this

20.7 Local Functions

- fill this

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.

21 Appendix

[Extra information if required —SS]

Appendix — Reflection

[Not required for CAS 741 projects —SS]

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

1. What went well while writing this deliverable?

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

3. Which of your design decisions stemmed from speaking to your client(s) or a proxy (e.g. your peers, stakeholders, potential users)? For those that were not, why, and where did they come from?

4. While creating the design doc, what parts of your other documents (e.g. requirements, hazard analysis, etc), it any, needed to be changed, and why?

5. What are the limitations of your solution? Put another way, given unlimited resources, what could you do to make the project better? (LO_ProbSolutions)

6. Give a brief overview of other design solutions you considered. What are the benefits and tradeoffs of those other designs compared with the chosen design? From all the potential options, why did you select the documented design? (LO_Explores)