Module Interface Specification for CXR

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

January 12, 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 the application. Complementary documents include the Module Guide.

The full documentation and implementation can be found at CXR-Capstone.

4 Notation

The structure of the MIS for modules comes from Hoffman and Strooper (1995). The mathematical notation follows standard conventions for sets, sequences, and functions. The notation includes symbols for ML model operations and medical data processing.

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 (-∞, ∞)
tensor	$\mathbb{T}^n$	an n-dimensional array of numerical values used for ML computations
probability	$\mathbb{P}$	a real number in [0, 1] representing likelihood
matrix	$\mathbb{M}_{m,n}$	a 2D array of size m$\times$n containing numerical values
binary	$\mathbb{B}$	boolean values true, false

4.1 Data Types from Libraries

Data Type	Notation	Description
DICOM	DICOM	Digital Imaging and Communications in Medicine format, standard for medical imaging storage and transmission
PyTorch Tensor	torch.Tensor	Multi-dimensional matrix containing elements of a single data type, optimized for GPU operations
NumPy Array	np.ndarray	Multi-dimensional array for scientific computing and image processing
JWT	JWT	JSON Web Token for secure authentication and information transmission
HTTP Request	HTTPRequest	Object containing HTTP request information including headers, body, and method
HTTP Response	HTTPResponse	Object containing HTTP response information including status code, headers, and body
DataFrame	pd.DataFrame	2-dimensional labeled data structure for patient and medical records
Base64	Base64	Binary-to-text encoding scheme for transmitting binary image data
YAML	YAML	Human-readable data serialization format for configuration files

Table 1: Data types from libraries

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 2: Module Hierarchy

6 MIS of Web Application Server Module

6.1 Module

Web Application Server

6.2 Uses

N/A

6.3 Syntax

6.3.1 Exported Constants

N/A

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

- N/A

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 user 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
`getDoctorProfile`	userID	user Profile	`ProfileNotFoundException`
`getDoctorProfile`	userID	user Profile	`NotDoctorException`

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)
- **DEFAULT_SPECIALTY**: "Radiologist" (default specialty for doctors without a specified specialty)

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 Patient Overview Module

12.1 Other Modules the Current Module Uses

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

12.2 State Variables

- **patientOverview**: Stores the summary information of a selected patient, including personal details, recent medical history, and current treatment plans.

12.3 Exported Constants and Access Programs

12.3.1 Exported Access Programs

Name	In	Out	Exceptions
`getPatient`	Patient ID	Patient overview details	`OverviewNotFoundException`

12.3.2 Exported Constants

- **OVERVIEW_REFRESH_INTERVAL**: 15 minutes (interval for refreshing the patient overview data)

12.4 Environment Variables

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

12.5 Assumptions

- Assumes the User Authentication Module (M11) ensures that only authorized users can view the patient overview.
- Assumes the overview data is accurate and synchronized with the Data Persistence Module (M15).

12.6 Access Routine Semantics

12.6.1 viewPatientOverview(patientID)

- **Transition**: Retrieves the overview information of the specified patient.
- **Output**: Returns the patient overview details or throws an `OverviewNotFoundException` if the overview cannot be found.

12.7 Local Functions

- **fetchOverviewData(patientID)**: Retrieves overview data from the data source.
- **generateOverviewSummary(patientID)**: Creates a summary of the patient's current status.
- **logOverviewAccess(patientID)**: Logs each time a patient overview is accessed for auditing purposes.

13 Patient Diseases Progression View

13.1 Other Modules the Current Module Uses

- M1: Web Application Server Module
- M2: HTTP Server Module
- M11: User Authentication Module
- M12: Disease Progression Tracking Module
- M15: Data Persistence Module

13.2 State Variables

- **Title**: fill this

13.3 Exported Constants and Access Programs

13.3.1 Exported Access Programs

Name	In	Out	Exceptions
`viewProgression`	Patient ID	Progression data	`ProgressionNotFoundException`

13.3.2 Exported Constants

- **PROGRESSION_UPDATE_INTERVAL**: 12 hours (time interval for updating disease progression data)
- **DEFAULT_CHART_TEMPLATE**: "default_progression_chart.html" (default template for displaying disease progression charts)

13.4 Environment Variables

- **progressionDataPath**: Path to the data source containing disease progression information.
- **chartRenderingService**: URL or endpoint of the service used for rendering progression charts.

13.5 Assumptions

- Assumes the User Authentication Module (M11) ensures that only authorized users can view the disease progression.
- Assumes the data is regularly updated and maintained in the Data Persistence Module (M15).

13.6 Access Routine Semantics

13.6.1 viewDiseaseProgression(patientID)

- **Transition**: Retrieves the disease progression details for the specified patient.
- **Output**: Returns the progression details or throws a `ProgressionNotFoundException` if the data cannot be found.

13.7 Local Functions

- **fetchProgressionData(patientID)**: Retrieves disease progression data from the data source.
- **generateProgressionChart(patientID)**: Generates a visual chart of the disease progression.
- **logProgressionAccess(patientID)**: Logs access to a patient's disease progression data for auditing purposes.

14 Patient Medical Records List View Module

14.1 Other Modules the Current Module Uses

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

14.2 State Variables

- **medicalRecordsList**: Contains a list of all medical records associated with a specific patient, including dates, record types, and summaries.

14.3 Exported Constants and Access Programs

14.3.1 Exported Access Programs

Name	In	Out	Exceptions
`viewMedicalRecordsList`	Patient ID	List of medical records	`RecordsNotFoundException`

14.3.2 Exported Constants

- **RECORDS_REFRESH_INTERVAL**: 10 minutes (interval for refreshing the list of medical records)
- **DEFAULT_LIST_TEMPLATE**: "default_records_list_template.html" (default template for displaying the list of medical records)

14.4 Environment Variables

- **recordsDataPath**: Path to the data source containing the patient's medical records.
- **recordSummaryService**: URL or endpoint of the service used for summarizing medical records.

14.5 Assumptions

- Assumes the User Authentication Module (M11) ensures that only authorized users can view the medical records.
- Assumes the records data is accurate and up-to-date in the Data Persistence Module (M15).

14.6 Access Routine Semantics

14.6.1 viewMedicalRecordsList(patientID)

- **Transition**: Retrieves the list of medical records for the specified patient.
- **Output**: Returns the list of medical records or throws a `RecordsNotFoundException` if no records are found.

14.7 Local Functions

- **fetchMedicalRecords(patientID)**: Retrieves the list of medical records from the data source.
- **generateRecordsSummary(patientID)**: Creates summaries for each record in the list.
- **logRecordsAccess(patientID)**: Logs access to a patient's medical records list for auditing purposes.

15 Patient Medical Record View Module

15.1 Other Modules the Current Module Uses

- M1: Web Application Server Module
- M2: HTTP Server Module
- M11: User Authentication Module
- M13: Disease Prediction Module
- M15: Data Persistence Module

15.2 State Variables

- **medicalRecordDetails**: Stores detailed information about a specific medical record, including the date, diagnosis, treatment, and doctor notes.

15.3 Exported Constants and Access Programs

15.3.1 Exported Access Programs

Name	In	Out	Exceptions
`viewMedicalRecord`	Record ID	Medical record details	`RecordNotFoundException`

15.3.2 Exported Constants

- **DEFAULT_RECORD_TEMPLATE**: "default_record_template.html" (default template for displaying a medical record)
- **MAX_RECORD_DISPLAY_LENGTH**: 5000 characters (maximum length for displaying record content)

15.4 Environment Variables

- **recordDataPath**: Path to the data source containing the specific medical record.
- **recordDetailService**: URL or endpoint of the service used for fetching detailed medical record data.

15.5 Assumptions

- Assumes the User Authentication Module (M11) ensures that only authorized users can view the detailed medical record.
- Assumes the data for medical records is accurately maintained and readily accessible through the Data Persistence Module (M15).

15.6 Access Routine Semantics

15.6.1 viewMedicalRecord(recordID)

- **Transition**: Retrieves the details of the specific medical record identified by `recordID`.
- **Output**: Returns the medical record details or throws a `RecordNotFoundException` if the record cannot be found.

15.7 Local Functions

- **fetchMedicalRecord(recordID)**: Retrieves the details of the specified medical record from the data source.
- **formatRecordDetails(recordID)**: Formats the medical record details for display.
- **logRecordAccess(recordID)**: Logs access to the specific medical record for auditing purposes.

16 Disease Prediction Module

16.1 Other Modules the Current Module Uses

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

16.2 State Variables

- **currentModel**: Model and The currently loaded disease prediction model.
- **modelStatus**: ModelStatus and Status of the prediction model (e.g., loaded, unloaded, error).
- **predictionCache**: Cache and Caches recent prediction results for quick access.

16.3 Exported Constants and Access Programs

16.3.1 Exported Access Programs

Name	In	Out	Exceptions
`loadModel`	modelPath : String	-	`ModelLoadException`
`predictDisease`	patientData : PatientData	PredictionResult	`InvalidInputException`
`updateModel`	newModel : Model	-	`ModelUpdateException`
`getModelStatus`	-	ModelStatus	-
`resetModel`	-	-	-

16.3.2 Exported Constants

- **DEFAULT_PREDICTION_MODEL**: "default_model.pkl" (path to the default prediction model)
- **MAX_INPUT_SIZE**: 1024 (maximum size for input data)

16.4 Environment Variables

- **MODEL_PATH**: Path to the directory containing prediction models.
- **CACHE_SIZE**: Maximum number of predictions to cache.

16.5 Assumptions

- Assumes that the Data Persistence Module (M15) provides reliable access to patient data.
- Assumes that the User Authentication Module (M11) ensures that only authorized users can perform predictions.
- Assumes that prediction models are compatible with the system's architecture and dependencies.

16.6 Access Routine Semantics

16.6.1 loadModel(modelPath)

- **Transition**: Loads the disease prediction model from the specified `modelPath`.
- **Exception**: Throws `ModelLoadException` if the model cannot be loaded.

16.6.2 predictDisease(patientData)

- **Transition**: Processes `patientData` using the current prediction model to generate a prediction.
- **Output**: Returns a `PredictionResult` containing the predicted disease information.
- **Exception**: Throws `InvalidInputException` if `patientData` is malformed or incomplete.

16.6.3 updateModel(newModel)

- **Transition**: Updates the current prediction model with `newModel`.
- **Exception**: Throws `ModelUpdateException` if the update fails.

16.6.4 getModelStatus()

- **Output**: Returns the current status of the prediction model (`ModelStatus`).
- **Exception**: None.

16.6.5 resetModel()

- **Transition**: Resets the prediction model to the default state.
- **Exception**: None.

16.7 Local Functions

- **validatePatientData(patientData)**: Validates the structure and completeness of `patientData`.
- **loadDefaultModel()** : Loads the default prediction model.
- **cachePrediction(result)**: Stores the `PredictionResult` in the `predictionCache`.
- **clearCache()** : Clears all entries from the `predictionCache`.

17 Disease Progression Module

17.1 Other Modules the Current Module Uses

- M1: Web Application Server Module
- M2: HTTP Server Module
- M11: User Authentication Module
- M13: Disease Prediction Module
- M15: Data Persistence Module

17.2 State Variables

currentProgressionModel: Model – The currently loaded disease progression model.

progressionStatus: ProgressionStatus – Status of the progression model (e.g., loaded, unloaded, error).

progressionCache: Cache – Caches recent progression results for quick access.

17.3 Exported Constants and Access Programs

17.3.1 Exported Access Programs

Name	In	Out	Exceptio
`loadProgressionModel`	modelPath : String	-	`ModelLoad`
`updateProgressionStage`	patientID : String, stageData : StageData	-	`InvalidSt`
`getProgressionStatus`	patientID : String	ProgressionStatus	`PatientNo`
`resetProgressionModel`	-	-	-
`forecastProgression`	patientID : String	ForecastResult	`ForecastE`

17.3.2 Exported Constants

- **DEFAULT_PROGRESSION_MODEL**: `"progression_model.pkl"` – Path to the default progression model.
- **MAX_STAGES**: 10 – Maximum number of disease progression stages.

17.4 Environment Variables

- **PROGRESSION_MODEL_PATH**: Path to the directory containing progression models.
- **CACHE_SIZE**: Maximum number of progression forecasts to cache.

17.5 Assumptions

- Assumes that the Data Persistence Module (M15) provides reliable access to patient data.
- Assumes that the User Authentication Module (M11) ensures that only authorized users can update and view disease progression.
- Assumes that progression models are regularly updated and maintained for accuracy.

17.6 Access Routine Semantics

17.6.1 loadProgressionModel(modelPath)

- **Transition**: Loads the disease progression model from the specified `modelPath`.
- **Exception**: Throws `ModelLoadException` if the model cannot be loaded.

17.6.2 updateProgressionStage(patientID, stageData)

- **Transition**: Updates the disease progression stage for the patient identified by `patientID` with the provided `stageData`.
- **Exception**: Throws `InvalidStageException` if `stageData` is invalid.

17.6.3 getProgressionStatus(patientID)

- **Transition**: Retrieves the current disease progression status for the patient identified by `patientID`.
- **Output**: Returns a `ProgressionStatus` object.
- **Exception**: Throws `PatientNotFoundException` if the patient does not exist.

17.6.4 resetProgressionModel()

- **Transition**: Resets the disease progression model to the default state.
- **Exception**: None.

17.6.5 forecastProgression(patientID)

- **Transition**: Generates a forecast of the disease progression for the patient identified by `patientID`.
- **Output**: Returns a `ForecastResult` containing predicted progression data.
- **Exception**: Throws `ForecastException` if the forecast cannot be generated.

17.7 Local Functions

- **validateStageData(stageData)**: Validates the structure and completeness of `stageData`.
- **loadDefaultProgressionModel()**: Loads the default progression model.
- **cacheProgressionResult(result)**: Stores the `ForecastResult` in the `progressionCache`.
- **clearProgressionCache()**: Clears all entries from the `progressionCache`.

18 Data Persistence Module

18.1 Other Modules the Current Module Uses

- M1: Web Application Server Module
- M2: HTTP Server Module
- M11: User Authentication Module
- M13: Disease Prediction Module
- M14: Disease Progression Module

18.2 State Variables

dbConnection: DBConnection – Represents the active connection to the database.

connectionStatus: ConnectionStatus – Indicates the status of the database connection.

retryCount: int – Tracks the number of retry attempts for failed operations.

18.3 Exported Constants and Access Programs

18.3.1 Exported Access Programs

Name	In	Out	Exceptions
`connectDB`	dbPath : String	ConnectionStatus	`ConnectionExceptio`
`saveData`	data : DataObject	-	`SaveException`
`loadData`	query : Query	DataObject	`LoadException`
`updateData`	dataID : String, newData : DataObject	-	`UpdateException`
`deleteData`	dataID : String	-	`DeleteException`
`backupDatabase`	backupPath : String	-	`BackupException`
`restoreDatabase`	backupPath : String	-	`RestoreException`

18.3.2 Exported Constants

- **DEFAULT_DB_PATH**: `"/var/data/persistence.db"` – Default database file path.
- **MAX_RETRIES**: 5 – Maximum number of retry attempts for database operations.

18.4 Environment Variables

- **DB_PATH**: Path to the primary database file.
- **BACKUP_PATH**: Path to store database backups.
- **RETRY_LIMIT**: Maximum number of retry attempts for database operations.

18.5 Assumptions

- Assumes that the database server is accessible and properly configured.
- Assumes that sufficient storage is available for data persistence and backups.
- Assumes that the User Authentication Module (M11) handles access control for database operations.

18.6 Access Routine Semantics

18.6.1 connectDB(dbPath)

- **Transition**: Establishes a connection to the database located at `dbPath`.
- **Output**: Returns the current `ConnectionStatus`.
- **Exception**: Throws `ConnectionException` if the connection fails.

18.6.2 saveData(data)

- **Transition**: Saves the provided `data` object to the database.
- **Exception**: Throws `SaveException` if the data cannot be saved.

18.6.3 loadData(query)

- **Transition**: Executes the `query` to retrieve data from the database.
- **Output**: Returns the resulting `DataObject`.
- **Exception**: Throws `LoadException` if the data cannot be retrieved.

18.6.4 updateData(dataID, newData)

- **Transition**: Updates the data entry identified by `dataID` with `newData`.
- **Exception**: Throws `UpdateException` if the update fails.

18.6.5 deleteData(dataID)

- **Transition**: Deletes the data entry identified by `dataID` from the database.
- **Exception**: Throws `DeleteException` if the deletion fails.

18.6.6 backupDatabase(backupPath)

- **Transition**: Creates a backup of the database at the specified `backupPath`.
- **Exception**: Throws `BackupException` if the backup process fails.

18.6.7 restoreDatabase(backupPath)

- **Transition**: Restores the database from the backup located at `backupPath`.
- **Exception**: Throws `RestoreException` if the restoration process fails.

18.7 Local Functions

- **validateData(data)**: Validates the structure and integrity of `data`.
- **retryOperation(operation)**: Attempts to retry a failed `operation` up to `MAX_RETRIES`.
- **logDatabaseActivity(activity)**: Logs database operations for auditing and debugging purposes.
- **initializeConnection()**: Initializes the database connection using environment variables.

References

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.

19 Appendix

[Extra information if required —SS]

Appendix — Reflection

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)

1. When writing this deliverable, several things went really well that made the whole process smoother and more organized. Our existing documentation, like the detailed list of functional and non-functional requirements and the Module Interface Specification (MIS) we created, provided a solid roadmap for how the system should behave. This clarity made it easier to develop the design blueprint and the specifications for each module because we already knew what each part needed to do and how they should interact. Another big plus was the team's early focus on a modular architecture, such as separating the machine learning (ML) services from the core backend. This approach kept our design documents organized and manageable, allowing us to develop, test, and maintain each module independently without getting overwhelmed. Additionally, having the core implementation set up during the Proof of Concept (POC) phase gave us a great direction for developing the specifications for our modules. It provided a tested foundation to build on, reducing uncertainties and making our specification development more precise and targeted.