

Module Interface Specification for CXR

Team 27, Neuralyzers

Ayman Akhras

Nathan Luong

Patrick Zhou

Kelly Deng

Reza Jodeiri

January 17, 2025

1 Revision History

| Date | Version | Notes |
|--------|---------|-------|
| Date 1 | 1.0 | Notes |
| Date 2 | 1.1 | Notes |

2 Symbols, Abbreviations and Acronyms

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

Contents

| | | |
|----------|--|-----------|
| 1 | Revision History | i |
| 2 | Symbols, Abbreviations and Acronyms | ii |
| 3 | Introduction | 1 |
| 4 | Notation | 1 |
| 4.1 | Data Types from Libraries | 2 |
| 5 | Module Decomposition | 3 |
| 6 | Web Application Server Module | 4 |
| 6.1 | Module | 4 |
| 6.2 | Uses | 4 |
| 6.3 | Syntax | 4 |
| 6.3.1 | Exported Constants | 4 |
| 6.3.2 | Exported Access Programs | 4 |
| 6.4 | Semantics | 4 |
| 6.4.1 | State Variables | 4 |
| 6.4.2 | Environment Variables | 4 |
| 6.4.3 | Assumptions | 4 |
| 6.4.4 | Access Routine Semantics | 5 |
| 6.4.5 | Local Functions | 5 |
| 7 | HTTP Server Module | 6 |
| 7.1 | Other Modules the Current Module Uses | 6 |
| 7.2 | State Variables | 6 |
| 7.3 | Exported Constants and Access Programs | 6 |
| 7.3.1 | Exported Access Programs | 6 |
| 7.3.2 | Exported Constants | 6 |
| 7.4 | Environment Variables | 6 |
| 7.5 | Assumptions | 6 |
| 7.6 | Access Routine Semantics | 7 |
| 7.6.1 | startServer() | 7 |
| 7.6.2 | stopServer() | 7 |
| 7.6.3 | processRequest(request) | 7 |
| 7.7 | Local Functions | 7 |
| 8 | Disease Prediction Server Module | 7 |
| 8.1 | Other Modules the Current Module Uses | 7 |
| 8.2 | State Variables | 8 |
| 8.3 | Exported Constants and Access Programs | 8 |

| | | |
|-----------|---|-----------|
| 8.3.1 | Exported Access Programs | 8 |
| 8.3.2 | Exported Constants | 8 |
| 8.4 | Environment Variables | 8 |
| 8.5 | Assumptions | 8 |
| 8.6 | Access Routine Semantics | 9 |
| 8.6.1 | loadModel() | 9 |
| 8.6.2 | predictDisease(patientImageData) | 9 |
| 8.7 | Local Functions | 9 |
| 9 | User Authentication Module | 9 |
| 9.1 | Other Modules the Current Module Uses | 9 |
| 9.2 | State Variables | 9 |
| 9.3 | Exported Constants and Access Programs | 10 |
| 9.3.1 | Exported Access Programs | 10 |
| 9.3.2 | Exported Constants | 10 |
| 9.4 | Environment Variables | 10 |
| 9.5 | Assumptions | 10 |
| 9.6 | Access Routine Semantics | 10 |
| 9.6.1 | sign_up_user(user_email, password, user_detail) | 10 |
| 9.6.2 | sign_in_user(user_email, password) | 11 |
| 9.6.3 | get_self_user(bearer_token) | 11 |
| 9.6.4 | get_user_by_id(uuid) | 12 |
| 9.6.5 | update_user_by_id(uuid, updated_user_json) | 12 |
| 9.7 | Local Functions | 12 |
| 10 | Module NAME HERE!!! | 13 |
| 10.1 | Other Modules the Current Module Uses | 13 |
| 10.2 | State Variables | 13 |
| 10.3 | Exported Constants and Access Programs | 13 |
| 10.3.1 | Exported Access Programs | 13 |
| 10.3.2 | Exported Constants | 13 |
| 10.4 | Environment Variables | 13 |
| 10.5 | Assumptions | 13 |
| 10.6 | Access Routine Semantics | 13 |
| 10.6.1 | trackProgression(patientID, data) | 13 |
| 10.6.2 | | 13 |
| 10.7 | Local Functions | 13 |
| 11 | Doctor Profile View Module | 14 |
| 11.1 | Other Modules the Current Module Uses | 14 |
| 11.2 | State Variables | 14 |
| 11.3 | Exported Constants and Access Programs | 14 |
| 11.3.1 | Exported Access Programs | 14 |

| | | |
|-----------|--|-----------|
| 11.3.2 | Exported Constants | 14 |
| 11.4 | Environment Variables | 14 |
| 11.5 | Assumptions | 15 |
| 11.6 | Access Routine Semantics | 15 |
| 11.6.1 | viewDoctorProfile(doctorID) | 15 |
| 11.7 | Local Functions | 15 |
| 12 | Patient List View Module | 15 |
| 12.1 | Other Modules the Current Module Uses | 15 |
| 12.2 | State Variables | 15 |
| 12.3 | Exported Constants and Access Programs | 16 |
| 12.3.1 | Exported Access Programs | 16 |
| 12.3.2 | Exported Constants | 16 |
| 12.4 | Environment Variables | 16 |
| 12.5 | Assumptions | 16 |
| 12.6 | Access Routine Semantics | 16 |
| 12.6.1 | viewPatientList(doctorID, filters) | 16 |
| 12.6.2 | applyFilter(filters) | 17 |
| 12.6.3 | sortList(criteria) | 17 |
| 12.7 | Local Functions | 17 |
| 13 | Patient Overview Module | 17 |
| 13.1 | Other Modules the Current Module Uses | 17 |
| 13.2 | State Variables | 17 |
| 13.3 | Exported Constants and Access Programs | 17 |
| 13.3.1 | Exported Access Programs | 17 |
| 13.3.2 | Exported Constants | 18 |
| 13.4 | Environment Variables | 18 |
| 13.5 | Assumptions | 18 |
| 13.6 | Access Routine Semantics | 18 |
| 13.6.1 | viewPatientOverview(patientID) | 18 |
| 13.7 | Local Functions | 18 |
| 14 | Diseases Progression View | 18 |
| 14.1 | Other Modules the Current Module Uses | 18 |
| 14.2 | State Variables | 19 |
| 14.3 | Exported Constants and Access Programs | 19 |
| 14.3.1 | Exported Access Programs | 19 |
| 14.3.2 | Exported Constants | 19 |
| 14.4 | Environment Variables | 19 |
| 14.5 | Assumptions | 19 |
| 14.6 | Access Routine Semantics | 20 |
| 14.6.1 | viewDiseaseProgression(patientID) | 20 |

| | |
|---|-----------|
| 14.7 Local Functions | 20 |
| 15 Medical Records List View Module | 20 |
| 15.1 Other Modules the Current Module Uses | 20 |
| 15.2 State Variables | 20 |
| 15.3 Exported Constants and Access Programs | 20 |
| 15.3.1 Exported Access Programs | 20 |
| 15.3.2 Exported Constants | 21 |
| 15.4 Environment Variables | 21 |
| 15.5 Assumptions | 21 |
| 15.6 Access Routine Semantics | 21 |
| 15.6.1 viewMedicalRecordsList(patientID) | 21 |
| 15.7 Local Functions | 21 |
| 16 Medical Record View Module | 22 |
| 16.1 Other Modules the Current Module Uses | 22 |
| 16.2 State Variables | 22 |
| 16.3 Exported Constants and Access Programs | 22 |
| 16.3.1 Exported Access Programs | 22 |
| 16.3.2 Exported Constants | 22 |
| 16.4 Environment Variables | 22 |
| 16.5 Assumptions | 23 |
| 16.6 Access Routine Semantics | 23 |
| 16.6.1 viewMedicalRecord(recordID) | 23 |
| 16.7 Local Functions | 23 |
| 17 X-Ray Report View | 23 |
| 17.1 place holder | 23 |
| 17.2 State Variables | 23 |
| 17.3 Exported Constants and Access Programs | 24 |
| 17.3.1 Exported Access Programs | 24 |
| 17.3.2 Exported Constants | 24 |
| 17.4 Environment Variables | 24 |
| 17.5 Assumptions | 24 |
| 17.6 Access Routine Semantics | 24 |
| 17.6.1 place holder(place holder) | 24 |
| 17.7 Local Functions | 24 |
| 18 Disease Progression Module | 24 |
| 18.1 Other Modules the Current Module Uses | 24 |
| 18.2 State Variables | 25 |
| 18.3 Exported Constants and Access Programs | 25 |
| 18.3.1 Exported Access Programs | 25 |

| | | |
|-----------|--|-----------|
| 18.3.2 | Exported Constants | 25 |
| 18.4 | Environment Variables | 25 |
| 18.5 | Assumptions | 25 |
| 18.6 | Access Routine Semantics | 26 |
| 18.6.1 | loadProgressionModel(modelPath) | 26 |
| 18.6.2 | updateProgressionStage(patientID, stageData) | 26 |
| 18.6.3 | getProgressionStatus(patientID) | 26 |
| 18.6.4 | resetProgressionModel() | 26 |
| 18.6.5 | forecastProgression(patientID) | 26 |
| 18.7 | Local Functions | 26 |
| 19 | Disease Prediction Module | 27 |
| 19.1 | Other Modules the Current Module Uses | 27 |
| 19.2 | State Variables | 27 |
| 19.3 | Exported Constants and Access Programs | 27 |
| 19.3.1 | Exported Access Programs | 27 |
| 19.3.2 | Exported Constants | 27 |
| 19.4 | Environment Variables | 27 |
| 19.5 | Assumptions | 28 |
| 19.6 | Access Routine Semantics | 28 |
| 19.6.1 | loadModel(modelPath) | 28 |
| 19.6.2 | predictDisease(patientData) | 28 |
| 19.6.3 | updateModel(newModel) | 28 |
| 19.6.4 | getModelStatus() | 28 |
| 19.6.5 | resetModel() | 28 |
| 19.7 | Local Functions | 29 |
| 20 | Data Persistent Module | 29 |
| 20.1 | Other Modules the Current Module Uses | 29 |
| 20.2 | State Variables | 29 |
| 20.3 | Exported Constants and Access Programs | 29 |
| 20.3.1 | Exported Access Programs | 29 |
| 20.3.2 | Exported Constants | 29 |
| 20.4 | Environment Variables | 30 |
| 20.5 | Assumptions | 30 |
| 20.6 | Access Routine Semantics | 30 |
| 20.6.1 | get_item_by_id(uuid) | 30 |
| 20.6.2 | create_new_item(item) | 30 |
| 20.6.3 | update_item_by_id(uuid, item) | 31 |
| 20.6.4 | get_s3_upload_presigned_url(file_name) | 31 |
| 20.7 | Local Functions | 32 |
| 20.8 | Other Modules the Current Module Uses | 32 |
| 20.9 | State Variables | 32 |

| | | |
|---------|--|----|
| 20.10 | Exported Constants and Access Programs | 32 |
| 20.10.1 | Exported Access Programs | 32 |
| 20.10.2 | Exported Constants | 32 |
| 20.11 | Environment Variables | 33 |
| 20.12 | Assumptions | 33 |
| 20.13 | Access Routine Semantics | 33 |
| 20.13.1 | connectDB(dbPath) | 33 |
| 20.13.2 | saveData(data) | 33 |
| 20.13.3 | loadData(query) | 33 |
| 20.13.4 | updateData(dataID, newData) | 33 |
| 20.13.5 | deleteData(dataID) | 34 |
| 20.13.6 | backupDatabase(backupPath) | 34 |
| 20.13.7 | restoreDatabase(backupPath) | 34 |
| 20.14 | Local Functions | 34 |

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 $(-\infty, \infty)$ |
| natural number | \mathbb{N} | a number without a fractional component in $[1, \infty)$ |
| real | \mathbb{R} | any number in $(-\infty, \infty)$ |
| 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 | <code>torch.Tensor</code> | Multi-dimensional matrix containing elements of a single data type, optimized for GPU operations |
| NumPy Array | <code>np.ndarray</code> | Multi-dimensional array for scientific computing and image processing |
| HTTP Request | <code>HTTPRequest</code> | Object containing HTTP request information including headers, body, and method |
| HTTP Response | <code>HTTPResponse</code> | Object containing HTTP response information including status code, headers, and body |
| DataFrame | <code>pd.DataFrame</code> | 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 Module | Web Application Server |
| | HTTP Server |
| | Disease Prediction Server |
| | Disease Progression Server |
| Behaviour-Hiding Module | User Authentication |
| | Patients List View |
| | Patient Overview View |
| | Disease Progression View |
| | Medical Records List View |
| | X-Ray Report View |
| Software Decision Module | Disease Progression Model |
| | Disease Prediction Model |
| | Data Persistent |

Table 2: Module Hierarchy

6 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
- M14: Disease Prediction Model

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 User Authentication Module

9.1 Other Modules the Current Module Uses

- M15: Data Persistence Module

9.2 State Variables

- **cognito_idp_client:** AWS Cognito client, initialized via the AWS boto3 library.
- **user_pool_id:** Identifier of the database at which AWS Cognito stores user's information such as username, password, first name, etc.
- **client_id:** Identifier of the User Authentication Module to authenticate with AWS Cognito (via OAuth's client credential grant).
- **client_secret:** Password of the User Authentication Module, used in conjunction with `client_id` to authenticate with AWS Cognito.

9.3 Exported Constants and Access Programs

9.3.1 Exported Access Programs

| Name | In | Out | Exceptions |
|-------------------|-----------------------------------|--------------|-----------------------------|
| sign_up_user | user_email, password, user_detail | user | Unauthorized, UserAlreadyEx |
| sign_in_user | user_email, password | bearer_token | Unauthorized, IncorrectCred |
| get_self_user | bearer_token | user | Unauthorized, InternalServe |
| get_user_by_id | uuid | user | Unauthorized, UserNotFoundE |
| update_user_by_id | uuid, updated_user_json | user | Unauthorized, UserNotFoundE |

9.3.2 Exported Constants

This module does not export any constants

9.4 Environment Variables

- **AWS_ACCESS_KEY_ID**: Access key to authenticate backend with AWS
- **AWS_SECRET_ACCESS_KEY**: Secret key to authenticate backend with AWS
- **AWS_REGION**: AWS Region of the backend infrastructure

9.5 Assumptions

- AWS Cognito returns consistent json results with user fields, since neuralanalyzer backend expect certain fields from user object (custom:firstName, custom:organization, etc.)
- AWS Cognito always stays online, since without this service, users wont be able to login into the application.

9.6 Access Routine Semantics

9.6.1 sign_up_user(user_email, password, user_detail)

- **Inputs:**
 - **user_email**: Email address of the user.
 - **password**: Password for the user's account.
 - **user_detail**: Additional user details, such as first name and organization, formatted as Python Dictionary.
- **Outputs**: A user object containing the created user's details.

- **Preconditions:**

- The provided email is valid and not already registered.
- Password meets AWS Cognito’s security requirements.

- **Postconditions:**

- The user is successfully created and stored in the Cognito user pool.

9.6.2 `sign_in_user(user_email, password)`

- **Inputs:**

- **user_email:** Email address of the user.
- **password:** Password for the user’s account.

- **Outputs:** A bearer token for authenticated access to other services.

- **Preconditions:**

- The user is registered in the Cognito user pool.
- The provided credentials are correct.

- **Postconditions:**

- The user is successfully authenticated.
- A bearer token is generated and returned for future authenticated requests.

9.6.3 `get_self_user(bearer_token)`

- **Inputs:**

- **bearer_token:** User’s Bearer token.

- **Outputs:** A user object containing the created user’s details.

- **Preconditions:**

- The `bearer_token` must be valid and not expired.

- **Postconditions:**

- The user is successfully retrieved from AWS Cognito.

9.6.4 `get_user_by_id(uuid)`

- **Inputs:**
 - **uuid:** The unique identifier of the user (string)
- **Outputs:** A user object containing the user's details.
- **Preconditions:**
 - User's UUID must be valid, and attached to a registered user.
- **Postconditions:**
 - The user is successfully retrieved from AWS Cognito.

9.6.5 `update_user_by_id(uuid, updated_user_json)`

- **Inputs:**
 - **uuid:** The unique identifier of the user (string)
 - **updated_user_json:** Python Dictionary object that contains the field which the caller wants to update the user with.
- **Outputs:** A user object containing the updated user's details.
- **Preconditions:**
 - User's UUID must be valid, and attached to a registered user.
 - User's updated fields must belong to list of fields users are allowed to edit (ie: first and last name, role, organization, etc.)
- **Postconditions:**
 - User with UUID must have the fields in `updated_user_json` updated on AWS Cognito.

9.7 Local Functions

- **`_parse_user_json_from_cognito_user`:** Convert user object returned from AWS Cognito, to a more application-friendly internal user object.
- **`_parse_cognito_attr_user_from_user_json`:** Convert the internal user object to the user data format that AWS Cognito requires.
- **`_get_username_from_email`:** Converting user's email into their username. (ie: `nathan@gmail.com` will be converted to `nathan_gmail`)
- **`_get_user_by_email`:** Fetching user from AWS Cognito via their email.

10 Module NAME HERE!!!

10.1 Other Modules the Current Module Uses

- fill this

10.2 State Variables

- **Title:** fill this

10.3 Exported Constants and Access Programs

10.3.1 Exported Access Programs

| Name | In | Out | Exceptions |
|--------|--------|--------|------------|
| fill 1 | fill 2 | fill 3 | fill 4 |
| fill 1 | fill 2 | fill 3 | fill 4 |

10.3.2 Exported Constants

- **Title:** file this

10.4 Environment Variables

- fill this

10.5 Assumptions

- fill this

10.6 Access Routine Semantics

10.6.1 trackProgression(patientID, data)

- fill this

10.6.2

- fill this

10.7 Local Functions

- fill this

11 Doctor Profile View Module

11.1 Other Modules the Current Module Uses

- M1: Web Application Server Module
- M2: HTTP Server Module
- M5: User Authentication Module
- M7: Patients Overview View
- M10: Medical Records List View
- M15: Data Persistence Module

11.2 State Variables

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

11.3 Exported Constants and Access Programs

11.3.1 Exported Access Programs

| Name | In | Out | Exceptions |
|------------------|--------|--------------|--------------------------|
| getDoctorProfile | userID | user Profile | ProfileNotFoundException |
| getDoctorProfile | userID | user Profile | NotDoctorException |

11.3.2 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)

11.4 Environment Variables

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

11.5 Assumptions

- Assumes that the User Authentication Module (M5) ensures only authorized doctors can view their profiles.
- Assumes the profile data is accurately stored and updated in the Data Persistence Module (M15).

11.6 Access Routine Semantics

11.6.1 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.

11.7 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.

12 Patient List View Module

12.1 Other Modules the Current Module Uses

- M1: Web Application Server Module
- M2: HTTP Server Module
- M5: User Authentication
- M7: Patient Overview View
- M15: Data Persistent

12.2 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.

12.3 Exported Constants and Access Programs

12.3.1 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 |

12.3.2 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)

12.4 Environment Variables

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

12.5 Assumptions

- Assumes that the User Authentication Module (M5) 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).

12.6 Access Routine Semantics

12.6.1 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.

12.6.2 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.

12.6.3 sortList(criteria)

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

12.7 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.

13 Patient Overview Module

13.1 Other Modules the Current Module Uses

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

13.2 State Variables

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

13.3 Exported Constants and Access Programs

13.3.1 Exported Access Programs

| Name | In | Out | Exceptions |
|------------|------------|--------------------------|---------------------------|
| getPatient | Patient ID | Patient overview details | OverviewNotFoundException |

13.3.2 Exported Constants

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

13.4 Environment Variables

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

13.5 Assumptions

- Assumes the User Authentication Module (M5) ensures that only authorized users can view the patient overview.
- Assumes the overview data is accurate and synchronized with the Data Persistence Module (M15).

13.6 Access Routine Semantics

13.6.1 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.

13.7 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.

14 Diseases Progression View

14.1 Other Modules the Current Module Uses

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

- M11: User Authentication Module
- M12: X-Ray Report View
- M15: Data Persistence Module

14.2 State Variables

- **Title:** fill this

14.3 Exported Constants and Access Programs

14.3.1 Exported Access Programs

| Name | In | Out | Exceptions |
|-----------------|------------|------------------|------------------------------|
| viewProgression | Patient ID | Progression data | ProgressionNotFoundException |

14.3.2 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)

14.4 Environment Variables

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

14.5 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).

14.6 Access Routine Semantics

14.6.1 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.

14.7 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.

15 Medical Records List View Module

15.1 Other Modules the Current Module Uses

- M1: Web Application Server Module
- M2: HTTP Server Module
- M5: User Authentication Module
- M11: Medical Record View Module
- M15: Data Persistence Module

15.2 State Variables

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

15.3 Exported Constants and Access Programs

15.3.1 Exported Access Programs

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

15.3.2 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)

15.4 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.

15.5 Assumptions

- Assumes the User Authentication Module (M5) 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).

15.6 Access Routine Semantics

15.6.1 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.

15.7 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.

16 Medical Record View Module

16.1 Other Modules the Current Module Uses

- M1: Web Application Server Module
- M2: HTTP Server Module
- M11: User Authentication Module
- M12: X-Ray Report View
- M15: Data Persistence Module

16.2 State Variables

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

16.3 Exported Constants and Access Programs

16.3.1 Exported Access Programs

| Name | In | Out | Exceptions |
|-------------------|-----------|------------------------|-------------------------|
| viewMedicalRecord | Record ID | Medical record details | RecordNotFoundException |

16.3.2 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)

16.4 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.

16.5 Assumptions

- Assumes the User Authentication Module (M5) 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).

16.6 Access Routine Semantics

16.6.1 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.

16.7 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.

17 X-Ray Report View

17.1 place holder

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

17.2 State Variables

- **place holder:** place holder

17.3 Exported Constants and Access Programs

17.3.1 Exported Access Programs

| Name | In | Out | Exceptions |
|--------------|--------------|--------------|--------------|
| place holder | place holder | place holder | place holder |

17.3.2 Exported Constants

- place holder: "place holder"

17.4 Environment Variables

- place holder: place holder

17.5 Assumptions

- place holder
- place holder

17.6 Access Routine Semantics

17.6.1 place holder(place holder)

- place holder: place holder

17.7 Local Functions

- place holder(place holder): place holder

18 Disease Progression Module

18.1 Other Modules the Current Module Uses

- M1: Web Application Server Module
- M2: HTTP Server Module
- M5: User Authentication Module
- M15: Data Persistence Module

18.2 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.

18.3 Exported Constants and Access Programs

18.3.1 Exported Access Programs

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

18.3.2 Exported Constants

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

18.4 Environment Variables

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

18.5 Assumptions

- Assumes that the Data Persistence Module (M15) provides reliable access to patient data.
- Assumes that the User Authentication Module (M5) ensures that only authorized users can update and view disease progression.
- Assumes that progression models are regularly updated and maintained for accuracy.

18.6 Access Routine Semantics

18.6.1 loadProgressionModel(modelPath)

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

18.6.2 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.

18.6.3 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.

18.6.4 resetProgressionModel()

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

18.6.5 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.

18.7 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`.

19 Disease Prediction Module

19.1 Other Modules the Current Module Uses

- M1: Web Application Server Module
- M2: HTTP Server Module
- M5: User Authentication Module
- M15: Data Persistence Module

19.2 State Variables

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

19.3 Exported Constants and Access Programs

19.3.1 Exported Access Programs

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

19.3.2 Exported Constants

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

19.4 Environment Variables

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

19.5 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.

19.6 Access Routine Semantics

19.6.1 loadModel(modelPath)

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

19.6.2 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.

19.6.3 updateModel(newModel)

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

19.6.4 getModelStatus()

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

19.6.5 resetModel()

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

19.7 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.

20 Data Persistent Module

20.1 Other Modules the Current Module Uses

- None

20.2 State Variables

- **dynamodb**: AWS DynamoDB client (managed NoSQL service), initialized via aws boto3 library.
- **table**: Name of the DynamoDB table.
- **s3_client**: AWS S3 client (managed blob storage service), initialized via aws boto3 library.

20.3 Exported Constants and Access Programs

20.3.1 Exported Access Programs

| Name | In | Out | Exceptions |
|-----------------------------|------------|---------------|----------------------------------|
| get_item_by_id | uuid | item | InternalServerError, ClientError |
| create_new_item | item | uuid, item | InternalServerError, ClientError |
| update_item_by_id | uuid, item | item | InternalServerError, ClientError |
| get_s3_upload_presigned_url | file_name | presigned_url | InternalServerError, ClientError |

20.3.2 Exported Constants

This module does not export any constants

20.4 Environment Variables

- **AWS_ACCESS_KEY_ID**: Access key to authenticate backend with AWS
- **AWS_SECRET_ACCESS_KEY**: Secret key to authenticate backend with AWS
- **AWS_REGION**: AWS Region of the backend infrastructure

20.5 Assumptions

- AWS DynamoDB and AWS S3 returns consistent json results and error codes.
- AWS DynamoDB and AWS S3 always stays online, without these services, users wont be able to create, view, update prescriptions and records.

20.6 Access Routine Semantics

20.6.1 `get_item_by_id(uuid)`

- **Inputs:**
 - **uuid**: Unique identifier of the item to retrieve.
- **Outputs:**
 - The item object corresponding to the given ‘uuid’.
- **Preconditions:**
 - The ‘uuid’ exists in the DynamoDB table.
- **Postconditions:**
 - The item object is successfully retrieved from the database.

20.6.2 `create_new_item(item)`

- **Inputs:**
 - **item**: A Python Dictionary object containing the details of the item to be created.
- **Outputs:**
 - **uuid**: The unique identifier of the created item.
 - **item**: The created item object.
- **Preconditions:**
 - The ‘item’ dictionary is well-formed.

- **Postconditions:**

- The new item is successfully added to the DynamoDB table.

20.6.3 `update_item_by_id(uuid, item)`

- **Inputs:**

- **uuid:** The unique identifier of the item to be updated.
- **item:** A Python Dictionary containing the updated fields for the item.

- **Outputs:**

- The updated item object.

- **Preconditions:**

- The ‘uuid’ corresponds to an existing item in the DynamoDB table.
- The ‘item’ Dictionary contains valid fields for the update.

- **Postconditions:**

- The item is successfully updated in the database, and returned to user.

20.6.4 `get_s3_upload_presigned_url(file_name)`

- **Inputs:**

- **file_name:** The name of the file to be uploaded to S3 (string).

- **Outputs:**

- **presigned_url:** A pre-signed URL allowing authorized upload of the specified file to AWS S3.

- **Preconditions:**

- The ‘file_name’ is a valid, non-empty string
- The S3 bucket configured in the `s3_client` is accessible and writable by the authenticated user.

- **Postconditions:**

- A valid pre-signed URL for uploading the specified file is generated and returned.
- The generated pre-signed URL is only valid for 1 minute.

20.7 Local Functions

- None

20.8 Other Modules the Current Module Uses

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

20.9 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.

20.10 Exported Constants and Access Programs

20.10.1 Exported Access Programs

| Name | In | Out | Exceptions |
|-----------------|---------------------------------------|------------------|---------------------|
| connectDB | dbPath : String | ConnectionStatus | ConnectionException |
| 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 |

20.10.2 Exported Constants

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

20.11 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.

20.12 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.

20.13 Access Routine Semantics

20.13.1 connectDB(dbPath)

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

20.13.2 saveData(data)

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

20.13.3 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.

20.13.4 updateData(dataID, newData)

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

20.13.5 deleteData(dataID)

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

20.13.6 backupDatabase(backupPath)

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

20.13.7 restoreDatabase(backupPath)

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

20.14 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>.

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?

Our existing documentation, such as 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.

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

We faced notable challenges while designing our Behavior Hiding Module, especially given our limited experience with standard healthcare user interface practices. To address this, we created an initial MVP concept UI and then conducted interviews with several healthcare professionals. Their feedback highlighted the need for straightforward interactions with patient profiles, a smooth process to upload chest X-ray scans, and a hassle-free way to create new records. In response, we iterated on our designs multiple times, adjusting layout structures, refining navigation flows, and simplifying labeling conventions to ensure the module felt intuitive and clinically relevant. Each revision brought us closer to delivering a user-centric interface that balances simplicity with the technical rigor necessary in medical settings, resulting in a solution that healthcare professionals can adopt with minimal friction.

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?

In our previous meeting with our supervisor, Dr. Moradi, we decided to prioritize designing a user-friendly interface specifically tailored for the use of radiologists. Based on their feedback, we focused on creating an intuitive and easy-to-navigate interface. Most of our time and research was dedicated to understanding and adhering to standard healthcare web interface guidelines to ensure the design aligns with the requirements

of a healthcare setting. We decided to prioritize interpretability and transparency of AI-generated report (explainable AI) which comes from the discussions with stakeholder emphasizing trust in AI results. This would help doctors/radiologists to better understand the prediction results. As for non-client-informed decisions, we decided to implement a disease progression feature that is separated from the prediction model. It traces the patient disease progression over a certain period, allowing comparing different X-rays and reports to give a more straightforward understanding of patient condition overtime while also reviewing the effectiveness of the prescriptions given to the patient in that specific period. By separating two models, they work independently and will still function if the other one goes down, allowing higher fault tolerance.

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

In the process of drafting the design document, we identified the need to revise several aspects of our other documents, including the Software Requirements Specification (SRS) and the Hazard Analysis document, to align with newly introduced features and interface design changes. One of the key additions was the disease progression monitoring feature, implemented using a Detection Transformer (DETR). This feature required us to add new requirements in the SRS, detailing how disease progression should be visualized, the data flow for tracking multiple X-rays over time, and the integration of DETR-based models for reliable prediction. Furthermore, the design now incorporates processing DICOM metadata to extract relevant information, such as clinical history, imaging parameters, and patient demographics, which ensures that critical contextual data is seamlessly integrated into the system. This step enhances the accuracy and usability of the interface by providing doctors with a comprehensive view of patient history alongside imaging results. Additionally, we refined our interface design to primarily cater to doctors instead of only patients, shifting the focus to provide structured, clinically relevant data and user-friendly tools for interpreting complex results. This change necessitated updating user interface requirements in the SRS to include features such as advanced filtering, annotation tools, and report generation. Furthermore, the incorporation of these features introduced new safety and ethical considerations, leading us to revisit and expand the Hazard Analysis document. For example, we identified potential risks related to false positives or negatives in disease progression detection, which could impact clinical decision-making. To mitigate these risks, we documented fallback mechanisms such as manual review workflows and alerts for ambiguous cases.

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)

A significant limitation of our solution lies in the inherent bias and fairness concerns within our AI model. Training exclusively on the MIMIC-CXR dataset from Beth

Israel Deaconess Medical Center means our model may exhibit performance disparities across different patient demographics. The limited diversity in our training data regarding age, gender, ethnicity, and socioeconomic status could result in reduced prediction accuracy for underrepresented populations, potentially compromising our system's ability to provide equitable healthcare solutions. Furthermore, since our training data originates from a single institution's imaging equipment, the model may demonstrate reduced effectiveness when processing X-rays captured using different machines, resolutions, or configurations. This technical limitation could disproportionately affect healthcare facilities utilizing different or older equipment specifications. Given unlimited resources, we would prioritize expanding our training dataset to encompass a broader spectrum of patient populations from multiple healthcare institutions and diverse imaging equipment. We would implement comprehensive testing protocols to evaluate model performance across various demographic groups and establish robust monitoring systems to track performance metrics across different population segments and equipment configurations. While our current implementation demonstrates promising results, addressing these limitations would be essential for future iterations to ensure the system delivers consistent and equitable care across all patient populations and healthcare settings.

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)

We considered both MVC and a layered architecture for our medical imaging system, particularly to accommodate disease prediction and progression pipelines. While MVC separates data, business logic, and presentation, it can introduce tight coupling between controllers and views, ultimately complicating upgrades and integrations of AI components. In contrast, a layered approach naturally decouples these processes into discrete layers such as presentation, application, and data—ensuring each layer can be maintained, replaced, or updated independently. This flexibility is especially vital given our need to integrate complex machine learning and imaging algorithms that may evolve or expand over time. Additionally, layering simplifies the implementation of security, logging, and regulatory compliance by localizing those concerns within specific strata of the system. By providing a clear, top-to-bottom sequence for data flow, the layered model also reduces communication complexity, making it easier to integrate and manage our disease prediction and progression modules. Ultimately, we selected layered architecture for its modularity, maintainability, and scalability.