Module Guide for CXR

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

January 17, 2025

1 Revision History

Date	Version	Notes
Jan 10	1.0	Ayman and Kelly worked on Anticipated changes and introduction
Jan 12	1.1	Reza worked on design timeline and the architecture
Jan 13	1.2	Nathan completed list of modules and the architecture, and Patrick worked on the module decomposition and the traceability with requirements

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

symbol	description
AC	Anticipated Change
DAG	Directed Acyclic Graph
M	Module
MG	Module Guide
OS	Operating System
R	Requirement
SC	Scientific Computing
SRS	Software Requirements Specification
CXR	Chest X-Ray scan Analysis
UC	Unlikely Change

Contents

1 Revision History **i**

2 Reference Material **ii**
2.1 Abbreviations and Acronyms . . . ii

3 Introduction **1**

4 Anticipated and Unlikely Changes **2**
4.1 Anticipated Changes . . . 2
4.2 Unlikely Changes . . . 2

5 Module Hierarchy **3**

6 Connection Between Requirements and Design **4**
6.1 Functional Requirements (FR) . . . 4
6.2 Non-Functional Requirements (NFR) . . . 6

7 Module Decomposition **8**
7.1 Hardware-Hiding Modules . . . 9
7.1.1 Web Application Server Module (M1) . . . 9
7.1.2 HTTP Server Module (M2) . . . 9
7.1.3 Disease Prediction Server Module (M3) . . . 9
7.1.4 Disease Progression Server Module (M4) . . . 9
7.2 Behaviour-Hiding Module . . . 10
7.2.1 User Authentication Module (M5) . . . 10
7.2.2 Patient List View Module (M6) . . . 10
7.2.3 Patient Overview View Module (M7) . . . 10
7.2.4 Disease Progression View Module (M8) . . . 11
7.2.5 Patient Medical Records List View Module (M9) . . . 11
7.2.6 X-Ray Report View Module (M10) . . . 11
7.3 Software Decision Module . . . 11
7.3.1 Disease Progression Model Module (M11) . . . 12
7.3.2 Disease Prediction Model Module (M12) . . . 12
7.3.3 Medical Record Management Module (M13) . . . 12
7.3.4 Data Persistence Module (M14) . . . 12

8 Traceability Matrix **13**

9 Use Hierarchy Between Modules **15**
9.1 Architecture Summary . . . 16

10 User Interfaces **17**

11 Timeline **17**

List of Tables

1 Module Hierarchy 4
2 Trace Between Functional Requirements and Modules 13
3 Trace Between Non-Functional Requirements and Modules 14
4 Trace Between Anticipated Changes and Modules 14

List of Figures

1 Use Hierarchy Diagram showing module interactions 15

3 Introduction

Decomposing a system into modules is a commonly accepted approachh to developing software. A module is a work assignment for a programmer or programming team (Parnas et al., 1984). We advocate a decomposition based on the principle of information hiding (Parnas, 1972). This principle supports design for change, because the "secrets" that each module hides represent likely future changes. Design for change is valuable in SC, where modifications are frequent, especially during initial development as the solution space is explored.

Our design follows the rules layed out by Parnas et al. (1984), as follows:

- System details that are likely to change independently should be the secrets of separate modules.
- Each data structure is implemented in only one module.
- Any other program that requires information stored in a module's data structures must obtain it by calling access programs belonging to that module.

After completing the first stage of the design, the Software Requirements Specification (SRS), the Module Guide (MG) is developed (Parnas et al., 1984). The MG specifies the modular structure of the system and is intended to allow both designers and maintainers to easily identify the parts of the software. potential readers of this document are as follows:

- New project members: This document can be a guide for a new project member to easily understand the overall structure and quickly find the relevant modules they are searching for.
- Maintainers: The hierarchical structure of the module guide improves the maintainers' understanding when they need to make changes to the system. It is important for a maintainer to update the relevant sections of the document after changes have been made.
- Designers: Once the module guide has been written, it can be used to check for consistency, feasibility, and flexibility. Designers can verify the system in various ways, such as consistency among modules, feasibility of the decomposition, and flexibility of the design.

The rest of the document is organized as follows. Section 4 lists the anticipated and unlikely changes of the software requirements. Section 5 summarizes the module decomposition that was constructed according to the likely changes. Section 6 specifies the connections between the software requirements and the modules. Section 7 gives a detailed description of the modules. Section 8 includes two traceability matrices. One checks the completeness of the design against the requirements provided in the SRS. The other shows the relation between anticipated changes and the modules. Section 9 describes the use relation between modules.

4 Anticipated and Unlikely Changes

This section lists possible changes to the system. According to the likeliness of the change, the possible changes are classified into two categories. Anticipated changes are listed in Section 4.1, and unlikely changes are listed in Section 4.2.

4.1 Anticipated Changes

Anticipated changes are adjustments expected during the project lifecycle and are designed to minimize the impact on the overall system. These changes are encapsulated within specific modules to ensure that the modifications affect only the corresponding module. This strategy is referred to as "design for change."

AC1: The specific hardware configurations on which the AI model runs may evolve as technology advances, including GPU updates for faster computations or better memory handling.

AC2: The format or structure of input chest X-rays might vary, such as accommodating different resolutions or adding metadata for enhanced analysis.

AC3: Changes in healthcare regulations could require modifications to the model's outputs or data handling to ensure compliance with privacy and security standards.

AC4: Expanding or refining the list of detectable diseases as new medical research or datasets become available.

AC5: Adjustments to the front-end interface to make it more user-friendly for medical professionals, incorporating feedback from usability studies.

AC6: Introducing or refining mechanisms to manage edge cases, such as incomplete or corrupted input data.

AC7: Enhancements to image preprocessing, such as noise reduction, normalization, or resizing techniques, to improve the quality of input data.

4.2 Unlikely Changes

Unlikely changes are those that are not expected to occur during the project lifecycle. These changes typically involve fundamental shifts in the system's scope, technology, or purpose and would require significant effort to implement.

UC1: Switching from PyTorch to an entirely different deep learning framework, such as TensorFlow or JAX, is unlikely due to the associated complexity and cost.

UC2: A change in the primary use case, such as shifting from diagnostic support to research-oriented predictions, is considered unlikely.

UC3: Shifting the AI model's purpose from disease prediction to a completely different application, such as general image classification.

UC4: Eliminating the AI model and replacing it with traditional statistical methods or rule-based systems.

UC5: Expanding the system to predict non-chest-related diseases or conditions beyond its initial design focus or the project's current scope.

5 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 1. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

M1: Web Application

M2: HTTP Server

M3: Disease Prediction Server

M4: Disease Progression Server

M5: User Authentication

M6: Patients List View

M7: Patient Overview View

M8: Disease Progression View

M9: Medical Records List View

M10: X-Ray Report View

M11: Disease Progression Model

M12: Disease Prediction Model

M13: Medical Record Management

M14: Data Persistent

Level 1	Level 2
Hardware-Hiding Module	Web Application
	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
	Medical Record Management
	Data Persistent

Table 1: Module Hierarchy

6 Connection Between Requirements and Design

The design of the system is intended to satisfy the requirements developed in the SRS. In this stage, the system is decomposed into modules. The connection between requirements and modules is explained in detailed below and the traceability is listed in Table 2.

6.1 Functional Requirements (FR)

FR1: Accept chest X-ray images as input
Modules: M3, M4, M11, M12
Rationale: These modules are responsible for processing X-ray images for disease prediction and tracking progression. They handle the image input and its integration into the disease detection pipeline.

FR2: Input additional patient symptoms
Modules: M3, M10, M12
Rationale: The system requires patient symptoms for disease prediction. M3 and M12 handle the input, while M10 is responsible for displaying the results, including the additional symptoms.

FR3: Analyze chest X-rays for disease detection
Modules: M3, M12
Rationale: These modules perform the core task of analyzing X-rays and predicting disease based on the image content.

FR4: Monitor patient condition changes
Modules: M4, M11
Rationale: These modules track the progression of a disease over time and analyze changes in the patient's condition, ensuring that the system provides up-to-date insights into the patient's health.

FR5: Highlight affected areas on X-rays
Modules: M3, M10, M12
Rationale: M3 analyzes the X-rays and identifies affected areas, while M10 visualizes those areas for the user. M12 assists with disease prediction and highlighting.

FR6: Generate a structured report
Modules: M10
Rationale: This module is responsible for generating and displaying structured reports based on the analyzed X-rays and predicted diagnoses.

FR7: Store patient data securely
Modules: M14
Rationale: M14 is specifically designed to store and manage patient data securely, ensuring compliance with data protection regulations.

FR8: Provide alerts for significant condition changes
Modules: M4, M6, M7, M8
Rationale: These modules monitor changes in a patient's condition and provide notifications when significant changes are detected. They also handle displaying the alerts to users.

FR9: Allow treatment plan adjustments
Modules: M10
Rationale: M10 stores medical records, which includes treatment plans. This module enables viewing and adjusting the plans as necessary.

FR10: Integrate with EHR systems
Modules: M14
Rationale: Integration with EHR (Electronic Health Records) systems requires a secure and reliable data storage mechanism, handled by the M14 module.

FR11: Display confidence levels in UI

Modules: M10
Rationale: The confidence levels of AI predictions are displayed through the M10 to inform users about the accuracy of diagnoses.

FR12: Patient uploads for self-diagnosis (Canceled)
Rationale: This feature was canceled, so no module is connected to it.

FR13: Support user roles and access levels (Canceled)
Rationale: This feature was also canceled, so no module is connected to it.

FR14: Create a new copy of X-rays for analysis
Modules: M9
Rationale: The M9 module manages and stores medical records, including X-ray images, and enables creating new copies for further analysis.

FR15: Support AI model updates
Modules: M3, M4, M11, M12
Rationale: These modules rely on AI models for disease prediction and progression analysis. The system needs to support updates to these models as new data or improvements emerge.

6.2 Non-Functional Requirements (NFR)

LF1: Consistent user interface
Modules: M6, M7, M8, M9
Rationale: These modules are essential for maintaining a consistent UI across the system, ensuring a seamless user experience for viewing patient details and progress.

LF2: Ergonomic color schemes and fonts
Modules: M6, M7, M8, M9
Rationale: These modules focus on presenting patient data and disease progression in an easy-to-read, ergonomic manner. Consistent color schemes and font choices are critical for a good user experience.

UH1: Usability for common tasks
Modules:M6, M7, M8, M9
Rationale: These modules are designed with usability in mind to allow healthcare professionals to perform tasks such as searching for patients, viewing records, and analyzing disease progression efficiently.

UH2: Context-sensitive help
Modules: M5, M14

Rationale: The M5 module ensures that users can access the system securely, while M14 can provide relevant help regarding data management and privacy settings.

PR1: Process images within 30 seconds
Modules: M3, M12, M4, M11
Rationale: These modules must process images quickly to meet the 30-second time constraint for efficient diagnosis and progression tracking.

PR2: Maintain consistent availability
Modules: M1, M2
Rationale: These modules ensure that the application is available and responsive at all times, minimizing downtime for users who rely on the system.

PR3: Concurrent processing of 20 images
Modules: M3, M12, M4, M11
Rationale: The system is designed to handle the simultaneous processing of multiple images, enabling efficient analysis of large batches of X-rays or other diagnostic images.

OE1: Integration with PACS systems
Modules: M14
Rationale: PACS (Picture Archiving and Communication System) integration allows the system to interact with medical imaging databases, which is managed by M14.

OE2: Operate in variable network conditions
Modules: M1, M2
Rationale: These modules are built to handle different network conditions, ensuring that the web application can function reliably even with fluctuating internet speeds.

SR1: Encrypt patient data
Modules: M14
Rationale: M14 ensures that patient data is encrypted both in transit and at rest, satisfying security and privacy regulations.

MS1: Modular architecture
Modules: M10, M12, M11
Rationale: The modular architecture allows each module to be updated or replaced independently, facilitating maintenance and scalability.

MS2: Automated testing suites
Modules: M3, M12, M4, M11, M14
Rationale: These modules require robust automated testing to ensure their functionality and accuracy in disease prediction, progression tracking, and data management.

CR1: Support English and French languages
Modules: M6, M7, M8, M9
Rationale: These modules display patient information and reports, which must support multiple languages for broader user accessibility.

LR1: Compliance with healthcare data protection laws
Modules: M14
Rationale: M14 ensures compliance with healthcare regulations by securely handling patient data in accordance with legal requirements.

LR2: Adherence to ISO 13485
Modules: M14
Rationale: M14 ensures that all data handling and storage comply with ISO 13485 standards for medical devices.

HS1: Radiologist validation of AI-generated diagnoses
Modules: M10
Rationale: The M10 module provides a platform for radiologists to review AI-generated diagnoses and confirm their accuracy before taking action.

HS2: Clear disclaimers about AI limitations
Modules: M10
Rationale: M10 includes disclaimers in the generated reports to inform users about the limitations of AI-based predictions.

7 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by Parnas et al. (1984). The *Secrets* field in a module decomposition is a brief statement of the design decision hidden by the module. The *Services* field specifies *what* the module will do without documenting *how* to do it. For each module, a suggestion for the implementing software is given under the *Implemented By* title. If the entry is *OS*, this means that the module is provided by the operating system or by standard programming language libraries. *CXR* means the module will be implemented by the CXR software.
Only the leaf modules in the hierarchy have to be implemented. If a dash (–) is shown, this means that the module is not a leaf and will not have to be implemented.

7.1 Hardware-Hiding Modules

Secrets: The data structures and algorithms used to implement the virtual hardware layer that abstracts infrastructure complexities.

Services: Provides virtualized hardware interfaces to the rest of the system, enabling hardware-independent operation.

Implemented By: –

7.1.1 Web Application Server Module (M1)

Secrets: The web server implementation and configuration details for handling web UI requests.

Services: Provides a platform for hosting and serving the web application to users.

Implemented By: Docker Engine

7.1.2 HTTP Server Module (M2)

Secrets: The implementation details of HTTP protocol handling and network communication.

Services: Provides HTTP communication interface for client-server interactions.

Implemented By: Docker Engine

7.1.3 Disease Prediction Server Module (M3)

Secrets: The implementation details of GPU resource management and ML model pipeline.

Services: Provides computational resources for disease prediction model execution.

Implemented By: Docker Engine and CUDA

7.1.4 Disease Progression Server Module (M4)

Secrets: The implementation details of parallel processing and GPU resource allocation for temporal progression analysis.

Services: Provides computational resources for disease progression analysis.

Implemented By: Docker Engine and CUDA

Type of Module: Hardware-Hiding

7.2 Behaviour-Hiding Module

Secrets: The contents of the required behaviours.

Services: Includes programs that provide externally visible behaviour of the system as specified in the software requirements specification (SRS) documents. This module serves as a communication layer between the hardware-hiding module and the software decision module. The programs in this module will need to change if there are changes in the SRS.

Implemented By: –

7.2.1 User Authentication Module (M5)

Secrets: AWS Cognito integration patterns and frontend authentication workflows that handle secure medical staff access.

Services: Provides a secure gateway for medical personnel to access the system through login forms, handles session persistence, and manages role-based access control.

Implemented By: React, JavaScript, and AWS Cognito

Type of Module: Abstract Object

7.2.2 Patient List View Module (M6)

Secrets: Component architecture for displaying and managing tabular patient data with sorting and filtering capabilities.

Services: Renders an interactive table interface displaying patient records with customizable columns. Enables medical staff to quickly sort, filter, and search through patient lists while maintaining responsive performance.

Implemented By: React and JavaScript

Type of Module: Abstract Object

7.2.3 Patient Overview View Module (M7)

Secrets: Dynamic data presentation based on patient's overall medical condition and historical data.

Services: Displays a summary of the patient's information, including diagnoses, medications, and treatment progress .

Implemented By: React and JavaScript

Type of Module: Abstract Object

7.2.4 Disease Progression View Module (M8)

Secrets: Side by side comparison of 2 X-Ray records.

Services: Displays the disease progression of the patient over 2 selected X-Ray records.

Implemented By: React and JavaScript

Type of Module: Abstract Object

7.2.5 Patient Medical Records List View Module (M9)

Secrets: Virtual scrolling implementation and lazy loading patterns for efficient record display.

Services: Manages the presentation of paginated medical records detailing the patient's medical history.

Implemented By: React and JavaScript

Type of Module: Abstract Object

7.2.6 X-Ray Report View Module (M10)

Secrets: Form layout and validation logic for structured medical report display.

Services: Presents AI generated reports in a structured format with detailing the findings of the X-Ray.

Implemented By: React and JavaScript

Type of Module: Abstract Object

7.3 Software Decision Module

Secrets: The design decision based on mathematical theorems, physical facts, or programming considerations. The secrets of this module are *not* described in the SRS.

Services: Includes data structure and algorithms used in the system that do not provide direct interaction with the user.

Implemented By: –

7.3.1 Disease Progression Model Module (M11)

Secrets: The criteria and methods for detecting and quantifying changes in disease manifestation between X-ray images over time.

Services: Analyzes temporal changes in specific lung regions using DETR(Detection Transformer) for precise disease progression monitoring.

Implemented By: PyTorch

Type of Module: Abstract Object

7.3.2 Disease Prediction Model Module (M12)

Secrets: The algorithm used to identify diseases and their severity from X-ray image features.

Services: Processes chest X-ray images to provide disease classification and severity scores using a fine-tuned ResNet model.

Implemented By: PyTorch

Type of Module: Abstract Object

7.3.3 Medical Record Management Module (M13)

Secrets: Record data structure organization, state management patterns, and client-side caching strategies for efficient record handling and display.

Services: Manages medical record data operations including creation, updates, and retrieval of patient records, prescriptions, and associated medical history while maintaining data consistency.

Implemented By: Flask

Type of Module: Abstract Object

7.3.4 Data Persistence Module (M14)

Secrets: The organization and relationships between different types of patient data, and the rules for maintaining data consistency and accessibility.

Services: Manages data storage and retrieval using AWS S3 for Patient Data including X-Ray Images, prescriptions, clinical notes, and DynamoDB for structured data.

Implemented By: AWS S3 and DynamoD

Type of Module: Abstract Object

8 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes.

Req.	**Modules**
FR1: Accept chest X-ray images as input	M3, M4, M11, M12
FR2: Input additional patient symptoms	M3, M12
FR3: Analyze chest X-rays for disease detection	M3, M12
FR4: Monitor patient condition changes	M4, M11
FR5: Highlight affected areas on X-rays	M3, M10, M12
FR6: Generate a structured report	M10
FR7: Store patient data securely	M14
FR8: Provide alerts for significant condition changes	M4, M6, M7, M8
FR9: Allow treatment plan adjustments	M10
FR10: Integrate with EHR systems	M14
FR11: Display confidence levels in UI	M10
FR12: Patient uploads for self-diagnosis (Canceled)	
FR13: Support user roles and access levels (Canceled)	
FR14: Create a new copy of X-rays for analysis	M9
FR15: Support AI model updates	M3, M4, M11, M12

Table 2: Trace Between Functional Requirements and Modules

Requirement	Modules
LF1: Consistent user interface	M6,M7,M8,M9
LF2: Ergonomic color schemes and fonts	M6,M7,M8,M9
UH1: Usability for common tasks	M6,M7,M8,M9
UH2: Context-sensitive help	M5, M14
PR1: Process images within 30 seconds	M3, M12, M4, M11
PR2: Maintain consistent availability	M1, M2
PR3: Concurrent processing of 20 images	M3, M12, M4, M11
OE1: Integration with PACS systems	M14
OE2: Operate in variable network conditions	M1, M2
SR1: Encrypt patient data	M14
SR2: Role-based access control (Canceled)	
MS1: Modular architecture	M10, M12, M11
MS2: Automated testing suites	M3, M12, M4, M11, M14
CR1: Support English and French languages	M6,M7,M8,M9,M**??**,M10
LR1: Compliance with healthcare data protection laws	M14
LR2: Adherence to ISO 13485	M14
HS1: Radiologist validation of AI-generated diagnoses	M10
HS2: Clear disclaimers about AI limitations	M10

Table 3: Trace Between Non-Functional Requirements and Modules

AC	Modules
AC1	M3, M4, M11, M12, M14
AC2	M3, M4, M7, M8, M14, M12, M11
AC3	M14
AC4	M3, M4, M12, M11
AC5	M6, M7, M8, M9,M10
AC6	M3, M4, M11, M12
AC7	M3, M4, M11, M12

Table 4: Trace Between Anticipated Changes and Modules

9 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. Parnas (1978) said of two programs A and B that A *uses* B if correct execution of B may be necessary for A to complete the task described in its specification. That is, A *uses* B if there exist situations in which the correct functioning of A depends upon the availability of a correct implementation of B. Figure 1 illustrates the use relation between the modules. It can be seen that the graph is a directed acyclic graph (DAG). Each level of the hierarchy offers a testable and usable subset of the system, and modules in the higher level of the hierarchy are essentially simpler because they use modules from the lower levels. For viewing the image with better clarity please cleas

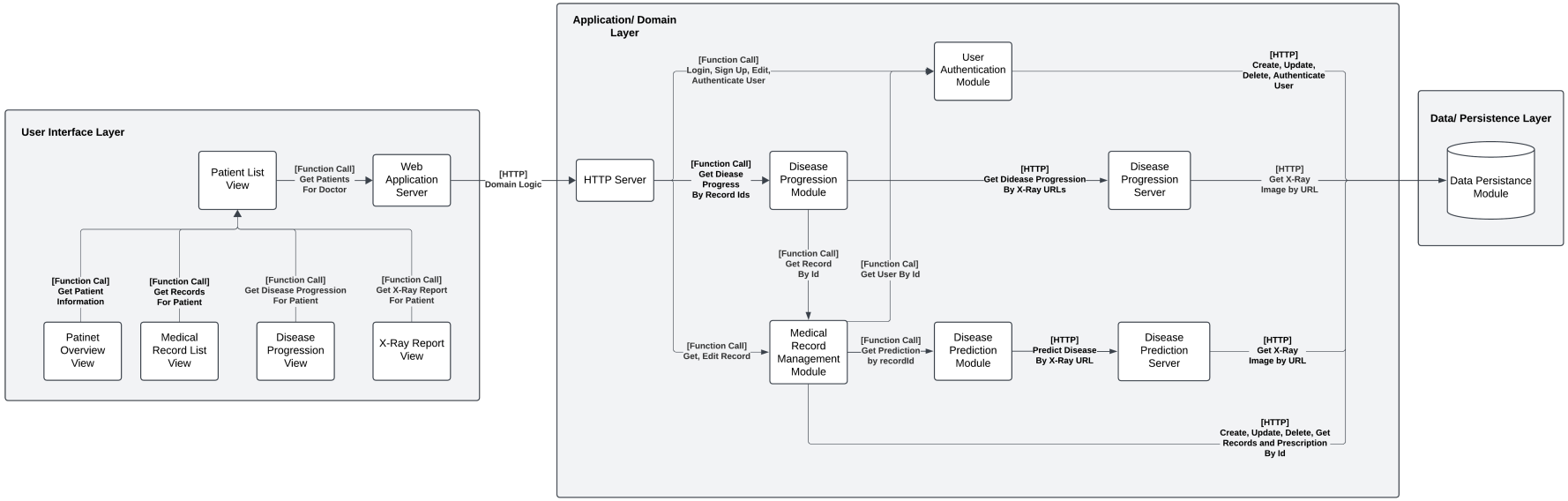

Figure 1: Use Hierarchy Diagram showing module interactions

9.1 Architecture Summary

This system is organized into three main layers—a User Interface layer, an Application/Domain layer, and a Data Persistence layer—each with clear responsibilities and well-defined interfaces:

1. **User Interface Layer**
   This layer provides all patient-facing and clinician-facing views. For example, the Patient List View, Patient Overview View, Medical Record List View, Disease Progression View, and X-Ray Report View each issue function calls (e.g., "Get Patient Information," "Get Records for Patient," etc.) to request data and operations.These views communicate with the Web Application Server in the Application/Domain layer.

2. **Application/Domain Layer**
   At this layer, the Web Application Server routes requests to the HTTP Server, which in turn orchestrates calls to the core modules:

   - **User Authentication Module** – Manages user login, sign-up, edit, and authentication.
   - **Disease Progression Module** – Retrieves and processes progression data, possibly delegating to the external Disease Progression Server via HTTP.
   - **Disease Prediction Module** – Handles machine-learning predictions for patient records, again leveraging an external Disease Prediction Server.
   - **Medical Record Management Module** – Allows creating, reading, editing, and deleting patient records and prescriptions.

   These modules may also request X-ray images from the Data Persistence layer or send them to the Disease Progression / Prediction servers for further analysis.

3. **Data Persistence Layer**
   The Data Persistence Module handles storage and retrieval of patient information, records, and X-ray images. All modules in the Application/Domain layer can communicate with this layer to read or persist data as needed.

By separating concerns into these three layers, the architecture remains modular, enabling independent development, testing, and maintenance of each layer while also decoupling the disease prediction and progression machine learning models.

10 User Interfaces

The user interface design consists of several key pages that have been prototyped in Figma:

- A login page for secure user authentication
- A dashboard showing critical patient statistics and recent X-rays
- A patient records page displaying medical history and X-ray images
- A detailed X-ray analysis page with disease detection results
- A report generation page for doctors to document findings and prescriptions
- A disease progression tracking interface to monitor changes over time

The complete interactive mockups can be viewed at our Figma design.

11 Timeline

NOTE: M# in this case stands for milestones.

- **Week 1 - 2: Jan 15 - Jan 28, 2025**
  - **M1: Backend Integration Module**
    - Setup initial backend services for user data, medical records, and X-ray storage (*Nathan*)
    - Integrate existing frontend structure with backend endpoints (ensure authentication, data fetching) (*Nathan*)
    - Basic testing to confirm data flow and API correctness (*Nathan, Ayman*)
  - **M2: Report Page Module**
    - Design the UI/UX for the doctor's report page (wireframes, user flow) (*Ayman*)
    - Implement editable fields (findings, prescriptions, clinical notes) and "Generate Report" functionality (*Ayman*)
    - Basic integration tests to ensure the generated report is saved/retrieved properly from backend (*Ayman*)
- **Week 3 - 4: Jan 29 - Feb 11, 2025**
  - **M3: Disease Model Implementation Module**
    - Fine-tune a baseline ML model (e.g., EfficientNet or similar) for disease detection (*Patrick*)

    - Incorporate clinical notes or prescription data as additional input (if feasible) (*Patrick*)
    - Initial inference endpoint set up in the backend (possibly behind an API route) (*Nathan*)
    - Minimal validation tests: confirm the model loads, infers, and returns results (*All as needed*)
  - **M4: Disease Progression Feature Module**
    - Research and define approach for tracking disease progression over time (*Reza*)
    - Implement backend logic to compare multiple patient X-rays or relevant data points (*Reza*)
    - Update UI to support selecting multiple images or timepoints (*Reza*)
    - Basic tests for progression calculations (mock data sets, verifying progression output) (*Reza, Nathan*)

- **Week 5 - 6: Feb 12 - Feb 25, 2025**
  - **M5: ML related Module**
    - Collaborate with *Patrick* to gather model outputs for test sets (*Kelly*)
    - Develop scripts or a mini-dashboard to compute ROC/AUC, precision/recall, etc. (*Kelly, Patrick*)
    - Display these metrics in a UI page or console logs for internal evaluation (*Kelly, Ayman*)
    - Run spot checks on real or synthetic data to validate model performance (*Kelly, Patrick*)
  - **M6: Records list Module**
    - Design a dashboard that displays quick stats: critical cases, aggregated disease predictions, etc. (*Kelly, Ayman*)
    - Implement data retrieval from backend to highlight urgent patient statuses (*Kelly, Nathan*)
    - Optional advanced features (if time allows): interactive charts, patient filtering (*Kelly, Ayman*)

- **Week 7 - 8: Feb 26 - Mar 10, 2025**
  - **M7: CI/CD Pipeline and Testing Related Modules**
    - Set up automated build and deployment scripts (GitHub Actions, Jenkins, or similar) (*Ayman*)
    - Establish unit and integration test frameworks in both frontend and backend (*Ayman, All team members*)
    - Generate code coverage reports and ensure critical paths are tested (*Ayman*)

        * Basic load or stress tests on the main endpoints to ensure reliability (*All team members as needed*)
    - **M8: Finishing Incomplete UI Pages**
        * Complete Settings/Help page: user preferences, contact info, help content (*Ayman, All team members*)
        * Finalize Profile pages: doctor or patient profile editing, access controls (*Ayman, All team members*)
        * Ensure consistent styling/theme across all UI pages (*All team members*)

- **Week 9 - 10: Mar 11 - Mar 24, 2025**
    - **M9: Additional Sidebar Features**
        * Add quick links (e.g., "New Report," "View Dashboard," "Disease Progression") strictly relevant to the X-ray project (*Ayman, All team members*)
        * Remove or hide unrelated placeholders (appointments, scheduling) to keep UI focused (*Ayman*)
        * Small UX improvements (icon updates, rearranging submenus, etc.) (*Ayman*)
    - **M10: Consolidation and Light Testing**
        * Review each module's functionality to ensure "Rev 0" completeness (*All team members*)
        * Perform partial integration testing across backend-frontend flows (uploading X-rays, generating reports, disease progression) (*All team members*)
        * Document known issues and quick fixes for final refinements (*All team members*)

- **Week 11 - 12: Mar 25 - Apr 7, 2025**
    - **M11: Final Checks and Rev 0 Confidence**
        * Compile partial test results to ensure stable performance (*Ayman, All team members*)
        * Confirm ML model accuracy is acceptable; refine or retrain if major gaps found (*Kelly*)
        * Ensure final UI/UX for the doctor's workflow is coherent (report generation, patient overview, dashboard) (*Ayman*)
        * If time allows, address any low-priority enhancements or bugs (*All team members*)

Notes and Assignments Summary:

- *Nathan (Backend Integration)*: AWS S3 setup, database connections, API routes, ensuring the frontend can properly fetch and save data.

- *Ayman (Frontend UI/UX)*: Report page design/coding, dashboard, overall UI refinement.
- *Patrick (Disease Model)*: ML model development, fine-tuning, integration into backend inference endpoints.
- *Reza (Disease Progression)*: Algorithm/design for progression logic, backend integration.
- *Kelly (ML Metrics, Dashboard Features)*: ROC/AUC, precision/recall metrics, supporting the model team, building quick metrics display.
- *Ayman (CI/CD, Testing)*: Automate builds, set up test frameworks, code coverage, minimal performance checks.
- *All Team Members*: Contribute to tests for each feature, help refine UI and fix bugs as needed.

By the end of **Week 12**, we aim for a functional "Rev 0" where doctors can:

- Log in to the system
- Upload or view X-ray images
- Receive disease predictions and progression analysis
- Generate and edit a final report (with prescriptions/notes), automatically linked in the patient's record
- See basic metrics (ROC/AUC) for the ML model's performance
- Use a partially tested, but stable, CI/CD pipeline that automates builds/deployments

Additional refinement or advanced features can be scheduled for post-Rev-0 milestones if time permits. formation is included there

References

David L. Parnas. On the criteria to be used in decomposing systems into modules. *Comm. ACM*, 15(2):1053–1058, December 1972.

David L. Parnas. Designing software for ease of extension and contraction. In *ICSE '78: Proceedings of the 3rd international conference on Software engineering*, pages 264–277, Piscataway, NJ, USA, 1978. IEEE Press. ISBN none.

D.L. Parnas, P.C. Clement, and D. M. Weiss. The modular structure of complex systems. In *International Conference on Software Engineering*, pages 408–419, 1984.