Module Guide for CXR

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

January 12, 2025

1 Revision History

Date	Version	Notes
Date 1	1.0	Notes
Date 2	1.1	Notes

2 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	Explanation of program name
UC	Unlikely Change
[etc. —SS]	[... —SS]

Contents

List of Tables

List of Figures

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 Server

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: Data Persistent

Level 1	Level 2
Hardware-Hiding Module	M1
	M2
	M3
	M4
Behaviour-Hiding Module	M5
	M6
	M7
	M8
	M9
	M10
Software Decision Module	M11
	M12
	M13

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 listed in Table 2.

[The intention of this section is to document decisions that are made "between" the requirements and the design. To satisfy some requirements, design decisions need to be made. Rather than make these decisions implicit, they are explicitly recorded here. For instance, if a program has security requirements, a specific design decision may be made to satisfy those requirements with a password. —SS]

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 structure and algorithm used to implement the virtual hardware.

Services: Serves as a virtual hardware used by the rest of the system. This module provides the interface between the hardware and the software. So, the system can use it to display outputs or to accept inputs.

Implemented By: OS

7.1.1 Web Application Server Module (M1)

Secrets: Infrastructure configuration, load balancing strategies, and server deployment patterns for handling concurrent medical requests.

Services: Manages web application hosting, handles HTTP requests routing, provides scalable infrastructure for serving the X-Ray Analysis interface, and ensures high availability.

Implemented By: Flask

7.1.2 HTTP Server Module (M2)

Secrets: Protocol implementation details, request/response handling mechanisms, and network-level security configurations.

Services: Handles HTTP/HTTPS protocol implementation, manages network connections, processes incoming requests, and implements security measures for medical data transmission.

Implemented By: Flask

7.1.3 Disease Prediction Server Module (M3)

Secrets: GPU memory allocation strategies, CUDA resource management patterns, and hardware-level optimizations for ML model inference.

Services: Manages GPU resources for disease prediction models, handles batch processing of X-ray images, coordinates model inference requests, and optimizes hardware utilization while acting as a controller for prediction workflows.

Implemented By: Python and CUDA

7.1.4 Disease Progression Server Module (M4)

Secrets: Multi-GPU resource allocation, parallel processing patterns, and memory management strategies for comparing temporal X-ray data.

Services: Coordinates progression analysis workflows, manages GPU resources for temporal analysis, handles parallel processing of multiple X-rays, and serves as a controller for progression detection pipelines.

Implemented By: Python and CUDA

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: Layout management and component composition strategies for organizing complex patient information.

Services: Presents a dashboard-style interface that organizes patient vitals, demographics, and current status in distinct sections. Implements responsive design to maintain usability across different screen sizes.

Implemented By: React and JavaScript

Type of Module: Abstract Object

7.2.4 Patient 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 Data Persistence Module (M13)

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 DynamoDB

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
R1	M??, M??, M??, M??
R2	M??, M??
R3	M??
R4	M??, M??
R5	M??, M??, M??, M??, M??, M??
R6	M??, M??, M??, M??, M??, M??
R7	M??, M??, M??, M??, M??
R8	M??, M??, M??, M??, M??
R9	M??
R10	M??, M??, M??
R11	M??, M??, M??, M??

Table 2: Trace Between Requirements and Modules

AC	Modules
AC1	M??
AC2	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??

Table 3: 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 2 illustrates the use relation between the modules.

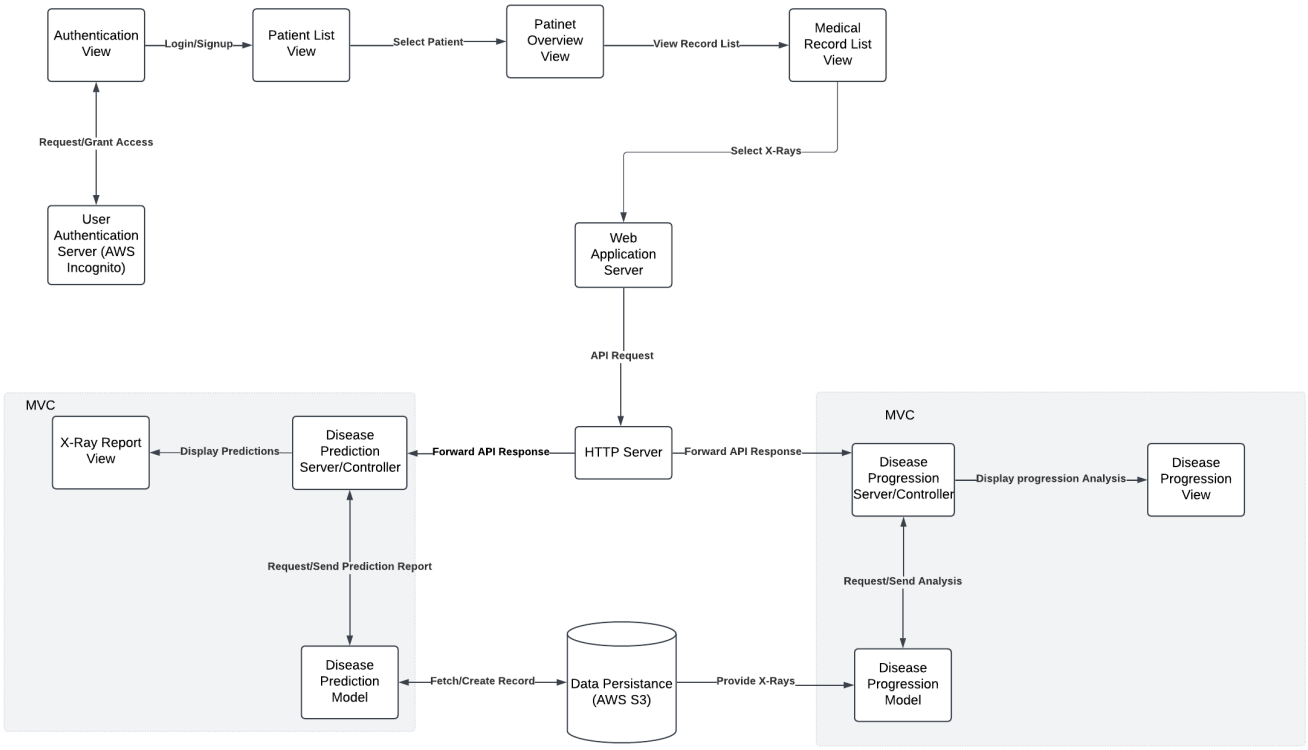

Figure 1: Use Hierarchy Diagram showing module interactions

The architecture follows a hybrid design pattern combining Model-View-Controller (MVC) for the machine learning components and client-server architecture for the view modules. The machine learning modules are structured with clear separation between the model logic, view presentation, and control flow. The view modules interact with patient data through requests to the web server module, which in turn communicates with the data persistence module. For simplicity, the diagram omits the request arrows between views and the web server module, as well as between modules and the data persistence layer. This modular design enables independent testing and development of components while maintaining clear interfaces between layers.

Figure 2: Use hierarchy among modules

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

- **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 Metrics and ROC/AUC 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: Dashboard Page 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 Module**
    - 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.