CSCI441 VA
Team D

Enhancing Electronic Medical Record Platform Security and Role-Based Access for IT Administrators

Report # 2

Submission Date: 03/09/2025
Team Members: Brandon Williams, Brittany Ritter, Christopher Pham, Riley Weaver

Project URL: https://github.com/ritterbrittany/CSCI441VATeamD.git

Table of Contents

Work Assignment

Contributions and Breakdown of Work

Topics	Brandon Williams	Brittany Ritter	Christopher Pham	Riley Weaver
Individual Contributions Breakdown	25%	25%	25%	25%
Customer Problem Statement	25%	25%	25%	25%
Goals, Requirements, and Analysis	25%	25%	25%	25%
Functional Requirement Specification and Use Cases	25%	25%	25%	25%
User Interface Specification	25%	25%	25%	25%
System Architecture and System Design	25%	25%	25%	25%
Design of Tests	25%	25%	25%	25%
Plan of Work	25%	25%	25%	25%
References	25%	25%	25%	25%

Section 1: Customer Problem Statement

Problem Statement:

Healthcare Administrator's/ IT Manager Perspective:

Managing an Electronic Medical Record System in a healthcare setting is crucial for making sure that patient information is accurate, secure and accessible to the correct people and the right time. However, with the current system it is difficult to manage the access permissions efficiently. Assigning roles to users (e.g., doctors, nurses, administrative staff) requires excessive manual configuration, increasing the risk of misconfigurations, unauthorized access, and compliance failures.

Furthermore, from the current standard of configuring what permissions are needed to be included in each of the various role assignments, it becomes extremely confusing for the us that must perform this since they are almost never familiar with the underlying code of the system and the corresponding terminology. For instance, a given healthcare administrator or IT manager would not know what the difference is between permission roles that says something like "Graph.API.Groups.Policy" versus "Groups.Policy.Admin" and what the difference is between the two without diving into dozens (or potentially hundreds) of pages of complex documentation. No matter how organized the documentation is, going through this many possible permissions would be mind-boggling for anyone other than the original developer. Unfortunately, that previously stated is all-to-common in the current industry. We need to be given the proper tools to create an effective system, and not end up going back and forth figuring out who needs access to what.

Besides role management, keeping track of evolving security issues and keeping up with regulatory compliance concerning HIPAA and other related standards/legal requirements is top priority. We need a system that helps me make sure that the correct individuals can access sensitive data without compromising its security. The current EMR platform does not allow the flexibility needed to set role-based access with the level of detail we need. I am worried that the current encryption system of AES-128 will not hold up to emerging quantum computing threats and we need a system that offers better protection against these threats.

There are far too many items that we need to have secured in our industry. We all know how many passwords we deal with on a daily basis, but what good is keeping track of all of these passwords if the data itself behind it isn't being dealt with securely? AES-128 may stand up to a degree of brute force attacks; however, I have also seen how many times healthcare systems have been compromised, and quantum computers being on the rise in the next 10-30 years gives me no peace of mind as for how safe our data is that we have now.

As an IT manager, I would appreciate a more intuitive process for role management and a more robust encryption system that would ensure our data remains secure for the long term. A modern EMR platform with role-based access control and better encryption would help alleviate these concerns. It would also make it much easier to complete my duties while also reducing the risk of human error and data breaches.

Doctors'/Nurses'/Caregivers' Perspective:

As a healthcare professional, the ability to access medical records in a timely and secure manner is vital to giving the best patient care. The current EMR system is difficult to navigate, requiring unnecessary steps to access critical information. This not only impacts workflow efficiency but also raises security concerns due to inconsistent role permissions. Additionally, I do not want to unintentionally access sensitive patient data that is irrelevant to my role, as this could result in compliance violations. The system should automatically restrict access to only the necessary data, minimizing risk while ensuring efficiency. We need a system that is easy to use and ensures that I have the access to the right patient information with the worry of seeing unnecessary data. It should also protect us from unauthorized access. We need a system that makes our workflow easier so that I am not having to take a lot of time to learn a complicated system. The ability to manage permissions on the fly and trust that the data is properly protected with the necessary encryption is essential for me to feel secure while doing my job.

If a system such as this were to exist, I believe that it would have to encompass all of the things I discussed here. For instance, we would need to be able to have a more direct and straightforward approach to knowing exactly what I should and shouldn't be able to see. Really, I would like to get this reassurance from my administrators or IT managers that I only have access to what I need to. I don't want to be responsible for having to participate in disputes I was never even involved in just because "my account technically might has access to view these certain parts of

data" - it is a waste of my time and raises a large eyebrow as to why people don't even know what all I do and don't have access to! In some areas it seems like I have access to every button under the sun, but in others I can see medications but not side-effects until my administrator or IT manager can figure out what "permission role" I need. As-is, it does not seem like the administrators or IT managers are too certain as to who has access to what, and I don't blame them - I blame the ERM. There is no reason why this ERM shouldn't have these features for to be able to get this organized for us healthcare professionals. And, on top of all of this I've said so far, another thing is that I would want reassurance that the data I am interacting with is secure in-and-of-itself - knowing that there is not someone watching every medical record we put in would give me contentment and the ability to do my job without having these thoughts lurk in the back of my head.

Patient's Perspective:

As a patient, I put my trust in the hospital and staff that my medical information will be stored securely and remain confidential. However, with the world today the ability for my information to get hacked and stolen has become an increasing concern. I want to know that my private medical information is secure and that my personal data is safe from attacks. The idea that my medical records could be exposed to unauthorized individuals is concerning. I would feel more comfortable and confident if I knew that the healthcare professionals that manage my records have a system that ensures only the right individuals have access to my data. The peace of mind that comes from knowing that my sensitive information is protected with the latest encryption standards and only those professionals that need the access to my records can view them, would help me build trust in the healthcare system.

Furthermore, even though it is essential and the first step to ensure that only the right people have access to view my data and not just a random support assistant can do so, it is also a matter of consequence that my data itself is shielded from any outside party trying to get in. I hear all the time about how different organizations are being compromised - even in this last year I heard about how nearly every American had their social security number leaked, and I sure don't want my medical information to be on the next tab over along with that!

The day I can sleep sound knowing that my healthcare provider is on the right system to keep my data secured, is the day that I know who I am sticking with for all of my health-related needs.

Decomposition into Sub-problems:

1. Role-Based Access Control:

Ensuring that only authorized personnel have access to specific medical data. This requires developing a mechanism that allows system administrators to assign clear and distinct roles with levels of data access. Requires developing role-specific permissions for accessing medical records and other sensitive data.

2. User Authentication and Authorization:

Ensuring that only authorized users can access the system to relevant portions of patient records. Which includes verifying user credentials securely with the appropriate role-based access. Requires implementing a secure authentication mechanism.

3. System Usability and Interface Design:

A critical component of this system is making sure that the user interface is easy to use and navigate for medical professionals. Ensuring that the design is simple, clean, and effective for role management, patient record viewing, and secure data entry.

4. Ensure Compliance with HIPAA Standards:

Ensuring encryption practices and data handling procedures align with evolving regulatory standards and follow audit requirements.

5. Implement Post-Quantum Cryptography (PQC) to Future Proof Data Security:

Achieving effective PQC that is not only secure but efficient is needed in order to deal with the emerging threats that come first to industries such as healthcare.

Glossary of Terms:

Electronic Medical Record (EMR) - an electronic record of health–related information on an individual that can be created, gathered, managed, and consulted by authorized clinicians and staff within one health care organization.

Role-Based Access Control (RBAC) - A security model where access rights are assigned based on the role of the user within an organization.

Data Encryption - The process of encoding data so that only authorized parties can read it.

Post-Quantum Cryptography (PQC) - Encryption algorithms designed to resist attacks from quantum computers.

- **CRYSTALS Kyber** - A PQC method for key encapsulation where a "key" is used to encrypt and decrypt small amounts of data. The key is asymmetrical meaning that if an attacker has the key to encrypt the data, it is nearly impossible to reverse-engineer the key and decrypt other parts of data.
- **Threefish-1024** - A PQC encryption algorithm that is excellent at encrypting large amounts of data such as pictures or large typed records while maintaining efficiency.

HIPAA (Health Insurance Portability and Accountability Act) - A U.S law that sets national standards for the protection of health information, ensuring the patient data is kept private and secure.

User Authentication - The process of verifying the identity of a user trying to access a system, such as login credentials like a username and password.

Security Role - A defined set of permissions assigned to users based on their job or function.

Nurses - A person trained to care for the sick or infirm, specifically a licensed health-care professional who practices independently.

Doctors - A qualified practitioner of medicine

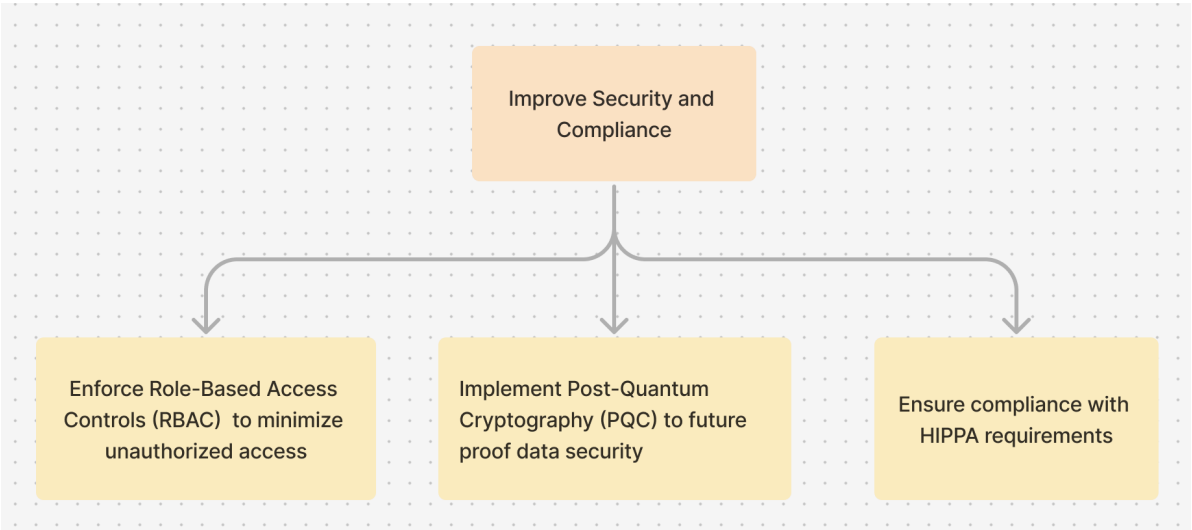
Healthcare Administrator - A healthcare professional that plans, directs, and coordinates healthcare services.

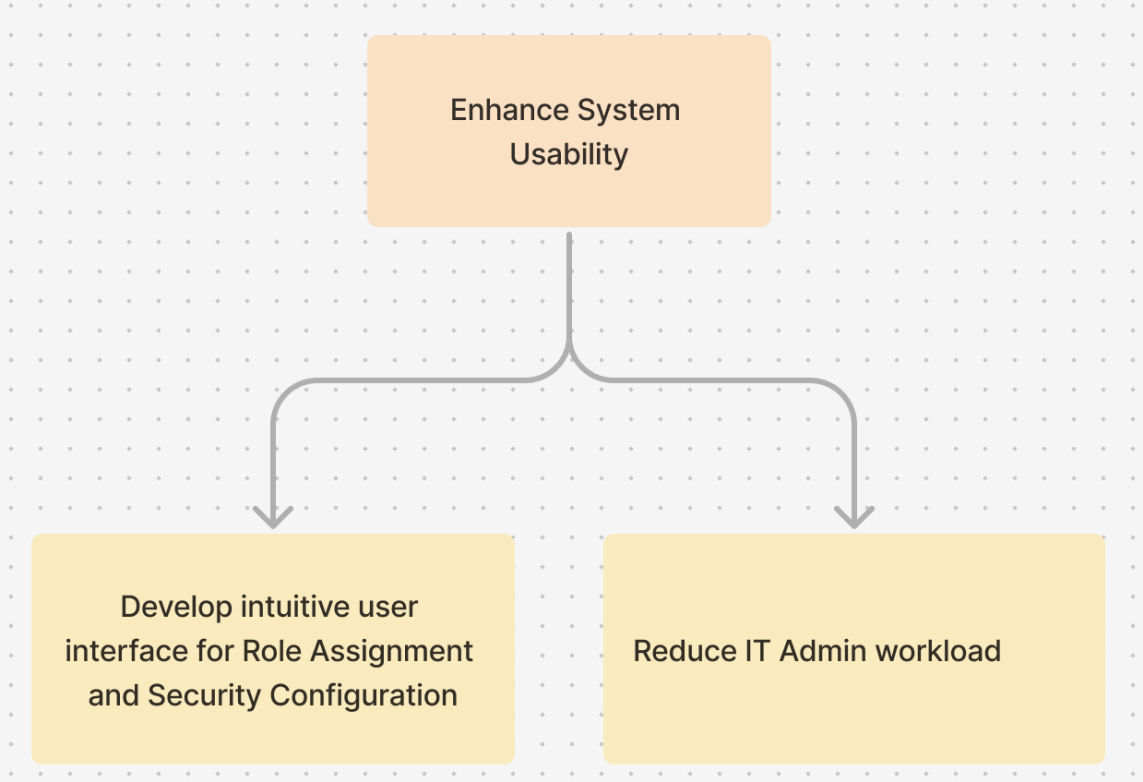
Medical Record - A medical record is any record which contains information relating to the physical or mental health or condition of an individual.

IT Manager - The lead person of an IT department, responsible for a company's system requirements.

Section 2: Goals, Requirements, and Analysis

Business Goals





Enumerated Functional Requirements

Identifier	Priority Weight	Requirement
REQ-1	5	The system shall allow IT administrators to create and modify user roles with specific permissions.
REQ-2	4	The system shall implement post-quantum cryptographic encryption.
REQ-3	4	End users of the system can perform all standard functionality of a typical ERM.
REQ-4	3	End users in the system have methods to verify that only the correct people have access to the respective needed areas.
REQ-5	3	End users should be able to input data properly.
REQ-6	3	End users should be able to receive output data properly.
REQ-7	2	UI functions and looks properly as-designed

Enumerated Nonfunctional Requirements

Identifier	Priority Weight	Requirement
NREQ-1	5	Application can be interfaced with using standard peripherals such as keyboard & mouse.
NREQ-2	4	Application must be reliable and responsive.
NREQ-3	4	Application contains desired PQC security implementations.
NREQ-4	3	Application must run in the major browsers.
NREQ-5	3	Application must be intuitive and easy to use.
NREQ-6	3	Application is scalable with regards to security, reliability, and maintainability.
NREQ-7	2	Application performance for normal tasks is adequate.
NREQ-8	2	Application performance for encryption/decryption operations is adequate.

User Interface Requirements

Identifier	Priority Weight	Requirement
UIREQ-1	5	There must be a categorized menu with primary and sub-categories to select which page and feature the user desires to access.
UIREQ-2	4	There must be various standard pages standard for an ERM (such as, but not limited to: Pass meds, medication list, reports, and all residents/patients).
UIREQ-3	4	GUI for assigning and customizing permissions must exist.
UIREQ-4	3	GUI must have a login screen/main screen for the user to enter account information
UIREQ-5	3	GUI must have the “Forgot password” option in case users forget their information.
UIREQ-6	3	GUI for handling staff members must exist.
UIREQ-7	2	App must contain an account page for end users’ account info.

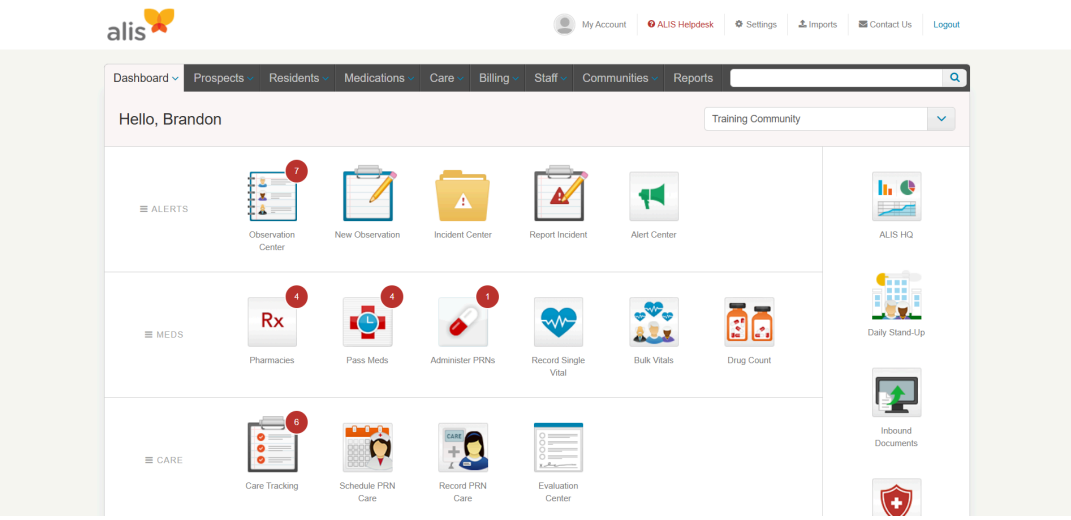
SEE NEXT PAGE →

User Interface Requirements Graphic Illustrations

UIREQ-1

"There must be a categorized menu with primary and sub-categories to select which page and feature the user desires to access."

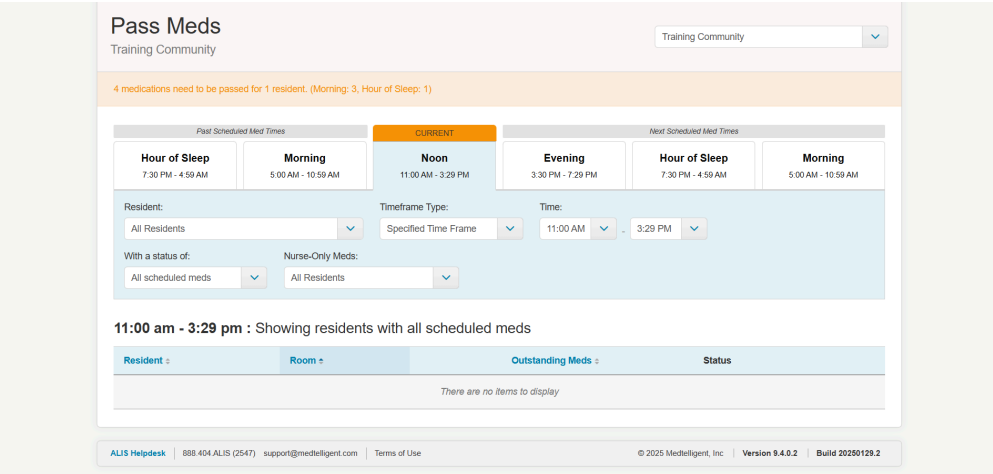
(**PICTURE IS FOR EXAMPLE/REFERENCE AND DOES NOT DEPICT EXACT DESIGN PLANS**)

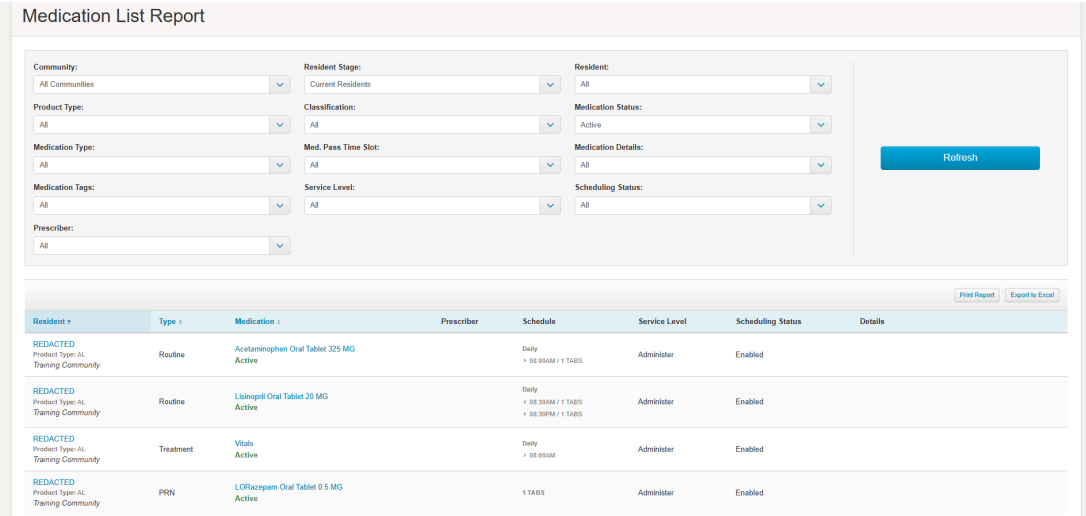


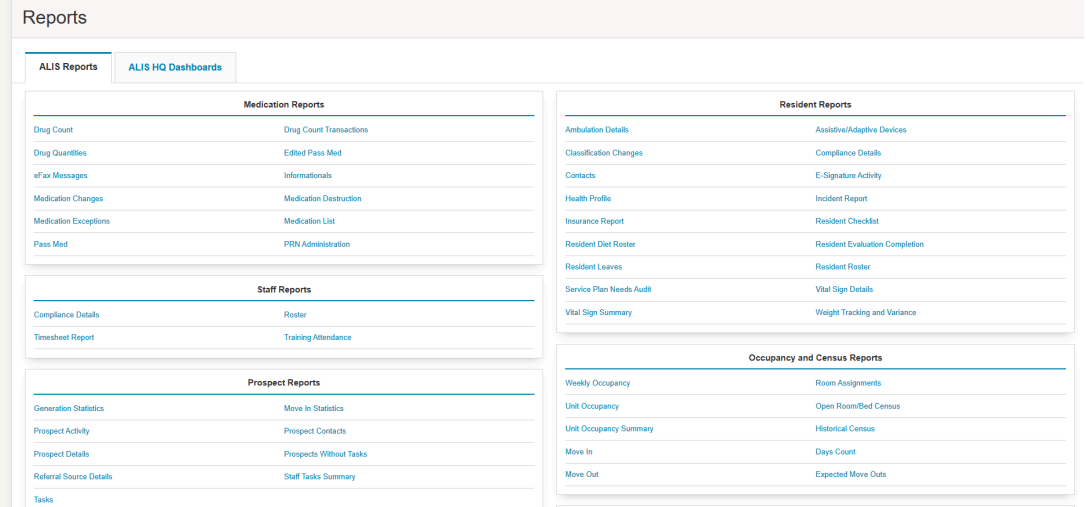
UIREQ-2

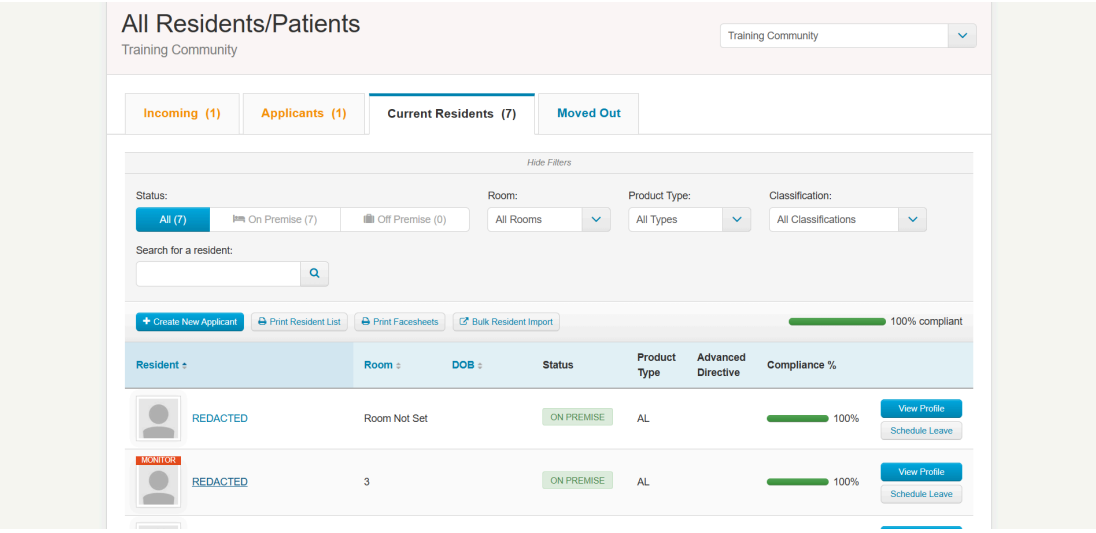
"There must be various standard pages standard for an ERM (such as, but not limited to: Pass meds, medication list, reports, and all residents/patients)."

(**PICTURE IS FOR EXAMPLE/REFERENCE AND DOES NOT DEPICT EXACT DESIGN PLANS**)




Medication List Report
Community:
All Communities
Resident Stage:
Current Residents
Resident:
All
Product Type:
All
Classification:
All
Medication Status:
Active
Medication Type:
All
Med. Pass Time Slot:
All
Medication Details:
All
Medication Tags:
All
Service Level:
All
Scheduling Status:
All
Prescriber:
All
Refresh
Print Report
Export to Excel
Resident
Type
Medication
Prescriber
Schedule
Service Level
Scheduling Status
Details
REDACTED
Product Type: AL
Training Community
Routine
Acetaminophen Oral Tablet 325 MG
Active
Daily
08:00AM / 1 TABS
Administer
Enabled
REDACTED
Product Type: AL
Training Community
Routine
Lisinopril Oral Tablet 20 MG
Active
Daily
08:30AM / 1 TABS
08:30PM / 1 TABS
Administer
Enabled
REDACTED
Product Type: AL
Training Community
Treatment
Vitals
Active
Daily
08:00AM
Administer
Enabled
REDACTED
Product Type: AL
Training Community
PRN
LORazepam Oral Tablet 0.5 MG
Active
1 TABS
Administer
Enabled

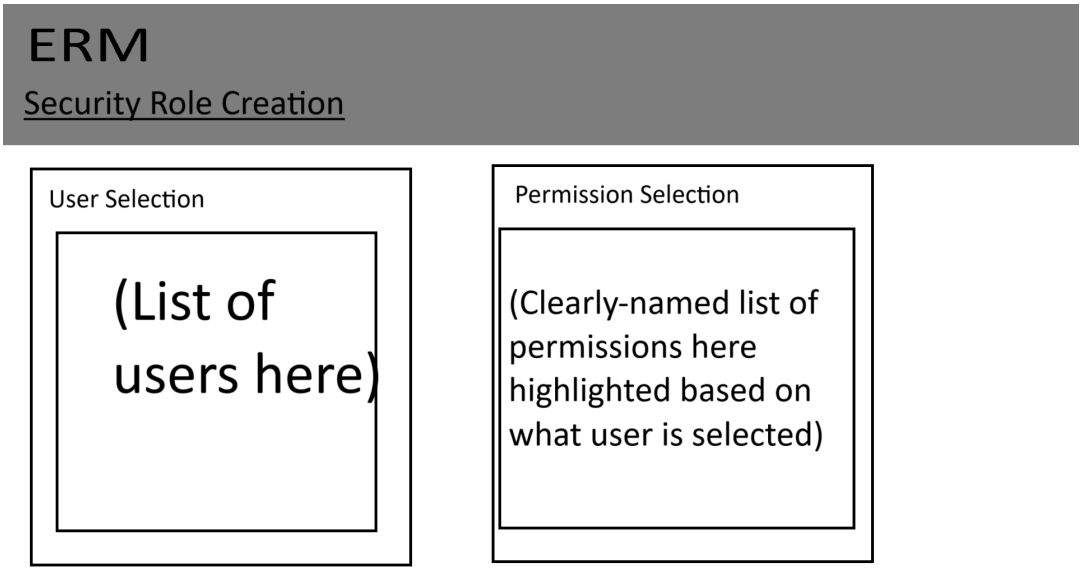

Reports
ALIS Reports
ALIS HQ Dashboards
Medication Reports
Drug Count
Drug Count Transactions
Drug Quantities
Edited Pass Med
eFax Messages
Informationals
Medication Changes
Medication Destruction
Medication Exceptions
Medication List
Pass Med
PRN Administration
Staff Reports
Compliance Details
Roster
Timesheet Report
Training Attendance
Prospect Reports
Generation Statistics
Move In Statistics
Prospect Activity
Prospect Contacts
Prospect Details
Prospects Without Tasks
Referral Source Details
Staff Tasks Summary
Tasks
Resident Reports
Ambulation Details
Assistive/Adaptive Devices
Classification Changes
Compliance Details
Contacts
E-Signature Activity
Health Profile
Incident Report
Insurance Report
Resident Checklist
Resident Diet Roster
Resident Evaluation Completion
Resident Leaves
Resident Roster
Service Plan Needs Audit
Vital Sign Details
Vital Sign Summary
Weight Tracking and Variance
Occupancy and Census Reports
Weekly Occupancy
Room Assignments
Unit Occupancy
Open Room/Bed Census
Unit Occupancy Summary
Historical Census
Move In
Days Count
Move Out
Expected Move Outs


All Residents/Patients
Training Community
Training Community
Incoming (1)
Applicants (1)
Current Residents (7)
Moved Out
Hide Filters
Status:
All (7)
On Premise (7)
Off Premise (0)
Room:
All Rooms
Product Type:
All Types
Classification:
All Classifications
Search for a resident:
Create New Applicant
Print Resident List
Print Facesheets
Bulk Resident Import
100% compliant
Resident
Room
DOB
Status
Product Type
Advanced Directive
Compliance %
REDACTED
Room Not Set
ON PREMISE
AL
100%
View Profile
Schedule Leave
MONITOR
REDACTED
3
ON PREMISE
AL
100%
View Profile
Schedule Leave

UIREQ-3

"GUI for assigning and customizing permissions must exist."

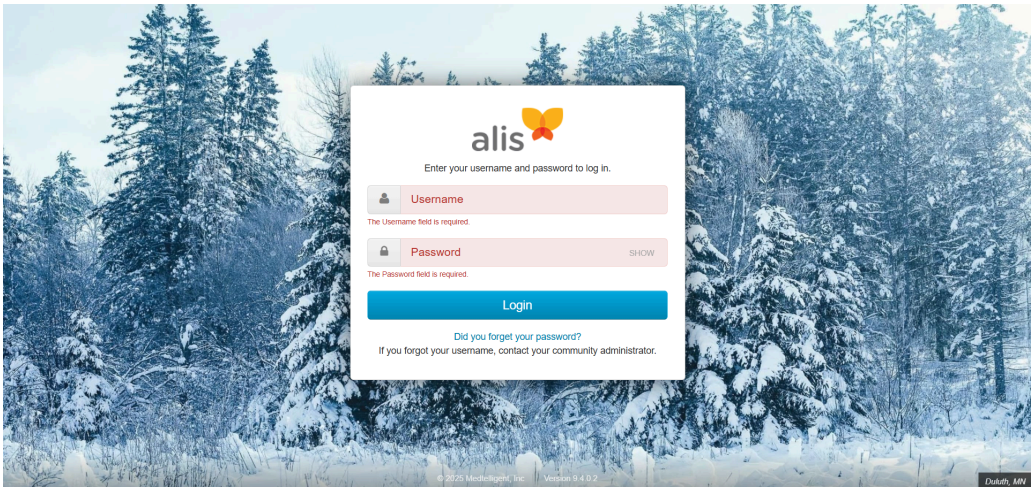
(**PICTURE IS FOR EXAMPLE/REFERENCE AND DOES NOT DEPICT EXACT DESIGN PLANS**)



UIREQ-4

"GUI must have a login screen/main screen for the user to enter account information"

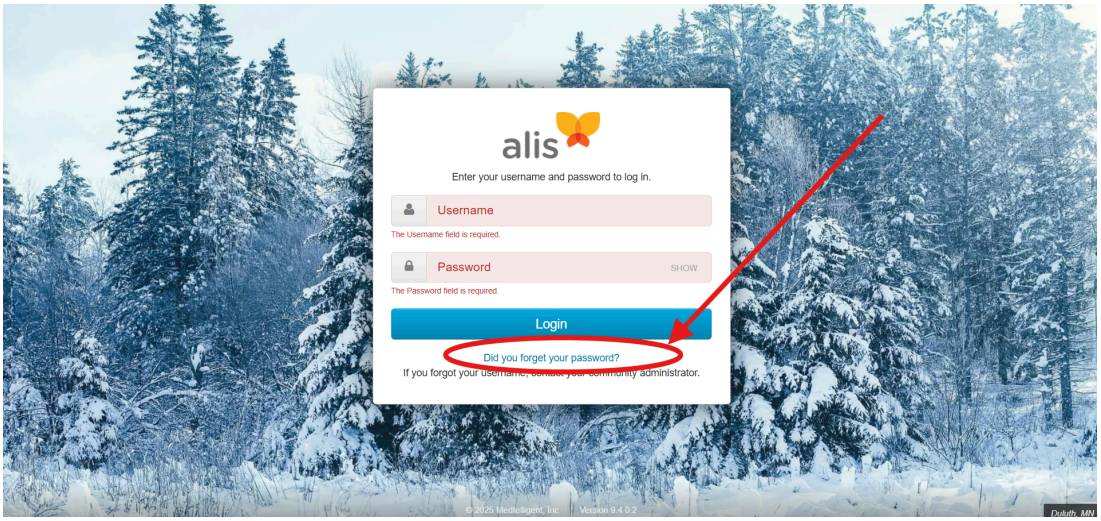
(**PICTURE IS FOR EXAMPLE/REFERENCE AND DOES NOT DEPICT EXACT DESIGN PLANS**)



UIREQ-5

"GUI must have the "Forgot password" option in case users forget their information."

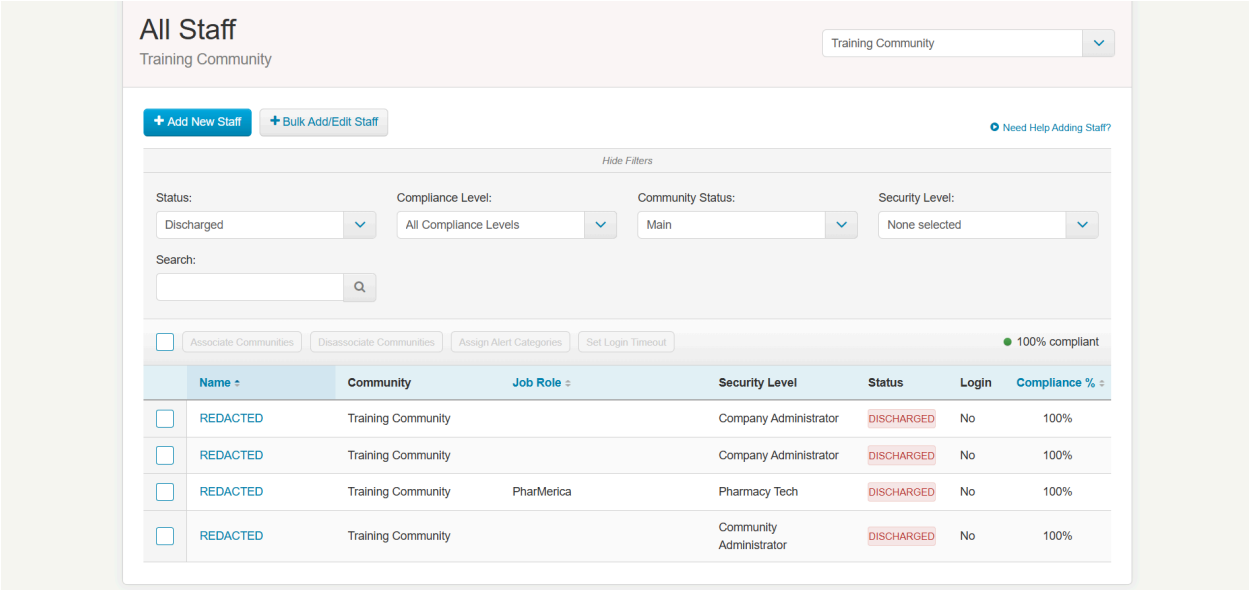
(**PICTURE IS FOR EXAMPLE/REFERENCE AND DOES NOT DEPICT EXACT DESIGN PLANS**)



UIREQ-6

"GUI for handling staff members must exist."

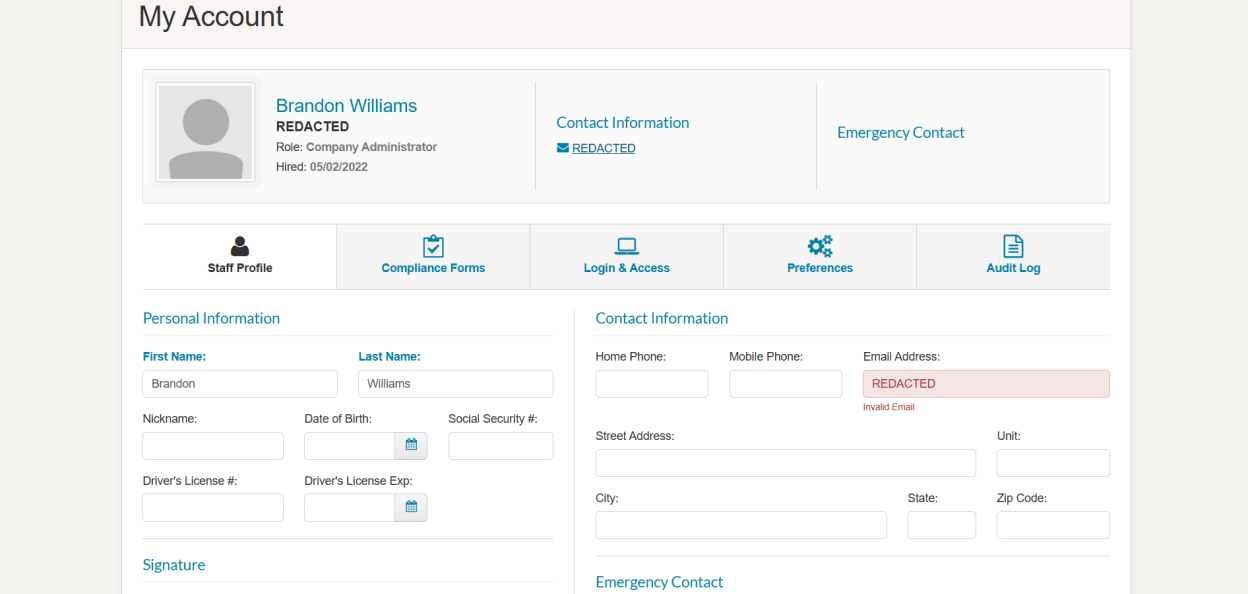
(**PICTURE IS FOR EXAMPLE/REFERENCE AND DOES NOT DEPICT EXACT DESIGN PLANS**)



UIREQ-7

"App must contain an account page for end users' account info."

(**PICTURE IS FOR EXAMPLE/REFERENCE AND DOES NOT DEPICT EXACT DESIGN PLANS**)



SEE NEXT PAGE →

Section 3: Functional Requirement Specification and Use Cases

Stakeholders

- ➢ Healthcare Administrators/It Managers- responsible for managing and configuring the system.
- ➢ Nurses/Doctors/Caregivers - end-users who require access to patient data in a secure and role-specific manner.
- ➢ Patients- users whose data must be securely stored and protected
- ➢ System Developers- responsible for implementing the system and maintaining its security and functionality.

Actors and Goals

Initiating Actor:

- ➢ IT Administrator
  - Role:
    - The IT administrator is responsible for the management of the system. This includes the creation and modification of user roles, assigning permissions, managing system configurations, and ensuring security protocols are followed.
  - Goals:
    - Ensure proper role-based access, security configurations, and compliance with regulatory standards.
    - Enforce compliance with healthcare regulations like HIPAA
    - Manage encryption methods and address future threats, such as those exposed by quantum computing.

- ➢ Healthcare Professionals (Doctors, Nurses, Caregivers)
  - Role:
    - Initiates the system to view/edit patient data based on their role-based access. Interact with the system to access and manage patient data according to their role and permission levels.
  - Goals:
    - Ensure that data relevant to their role is easily accessible while keeping unauthorized data hidden. Trust in system security.
    - Access only the patient data that is relevant to their role

        - Minimize workflow disruptions and compliance risks by accessing only the necessary data, ensuring efficiency and patient care.

- Patient-
    - Role:
        - Initiates system access through their healthcare provider for data protection and permissions assurance.
    - Goals:
        - Assurance that their medical records are securely protected and only accessible by authorized personnel.
        - Trust that any access or modification of their data is appropriately logged and controlled by the healthcare system and staff.
- System Developers

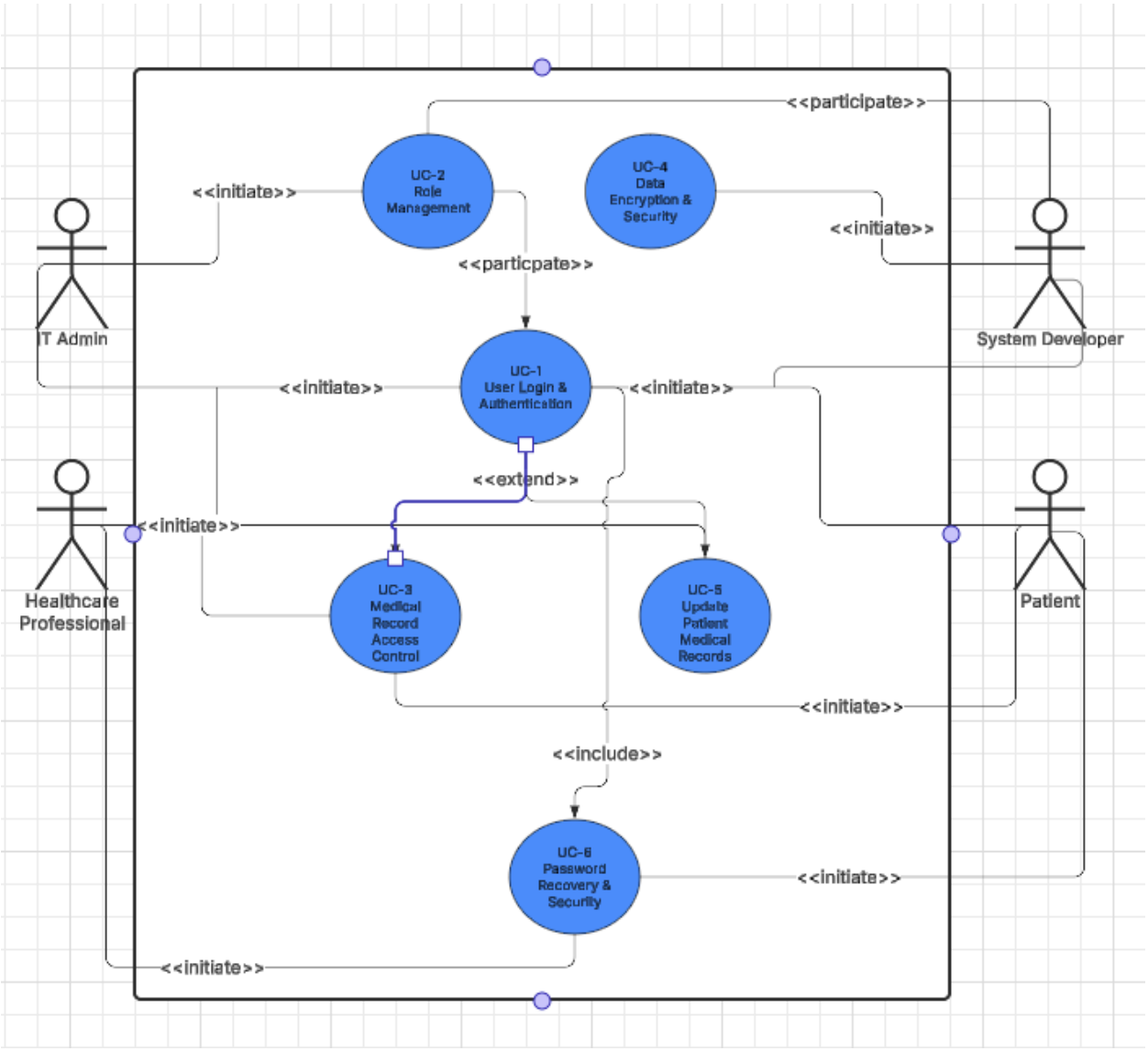
    Role:
        - The developers are responsible for implementing, maintaining, and updating the system's functionality and security measures. They ensure the backend operations are optimized, the user interface is intuitive, and security features are integrated and up to date.
    - Goals:
        - Ensure that the system works efficiently, securely, and with minimal issues.
        - Implement and maintain post-quantum cryptographic measures to future-proof the system.
        - Ensure the application is scalable and adaptable to evolving security needs, such as adapting encryption standards to quantum computing.

Use Cases

Casual Description

Identifier	Description	Related Requirement(s)
UC-1	**User Login & Authentication**: As a stakeholder, I expect to be able to log in / log out of the system utilizing a unique username and password to access patient data.	REQ-3, REQ-7
UC-2	**Role Management**: As a stakeholder, I expect to be able to create, modify, or delete user roles and set permissions in a straightforward and seamless process.	REQ-1, REQ-3
UC-3	**Medical Record Access Control**: As a stakeholder, I expect to be able to access patient records and only pertinent details based on my assigned role, ensuring data security.	REQ-1, REQ-2, REQ-4
UC-4	**Data Encryption & Security**: As a stakeholder, I expect the system to encrypt stored patient data and log access attempts for security and compliance.	REQ-2, REQ-3
UC-5	**Update Patient Medical Records**: As a stakeholder, I expect to be able to update patient records, so that I can keep medical information accurate and update to date.	REQ-4, REQ-5, REQ-6
UC-6	**Password Recovery & Security**: As a stakeholder, I expect a “Forgot Password” feature, to be able to securely recover/reset my account without IT intervention.	REQ-3, REQ-7

Use Case Diagram



Traceability Matrix

Requirement	PW	UC-1	UC-2	UC-3	UC-4	UC-5	UC-6
REQ-1	5		X	X			
REQ-2	4			X	X		
REQ-3	4	X	X		X		X
REQ-4	3			X		X	
REQ-5	3					X	
REQ-6	3					X	
REQ-7	2	X					X
MAX PW		4	5	5	4	3	4
TOTAL PW		6	9	12	8	9	6

SEE NEXT PAGE →

Fully Dressed Descriptions

UC1 - Login & Authentication	
Related Requirements:	REQ-3, REQ-7
Initiating Actor:	User (IT manager, healthcare administrator, caregiver, etc.)
Actor's Goals:	To access their account and continue for other actions.
Practicing Actors:	Interface/UI, Server, Database
Preconditions:	User account exists in the system (database, specifically).
Postconditions:	User gains access to their account and related functions.
Flow of Events for Main Success Scenario:	1. → User access application sign-in screen 2. → User types in and interface sends username/password for processing 3. ← Server receives sign-in request 4. ↔ Server computes password hash 5. → Server sends hashed request to database for authentication 6. ← Database receives contains requests 7. ↔ Database searches for and finds request 8. → Database sends verified request to server 9. ← Server receives database authentication 10. → Server sends authentication confirmation to interface with permissions to continue into next designated security roles assigned to user 11. ← Interface receives authentication request 12. → Interface/UI displays user proper next screen confirming sign-in request was accepted
Alternative Scenarios:	If the database does not find the request in step 7, then it sends back an invalid or error code back to the server, which relays it to the user and does not give permissions, but instead throws an error message in the interface/UI.

UC2 - Role Management	
Related Requirements:	REQ-1, REQ-3
Initiating Actor:	User (IT manager, healthcare administrator)
Actor's Goals:	To modify a security role assignment another user / other user(s).

Practicing Actors:	Interface/UI, Server, Database
Preconditions:	UC1 met and user account has permissions to create/modify other users' security roles.
Postconditions:	User successfully modifies a security assignment to another user.
Flow of Events for Main Success Scenario:	1. → User utilizes UC1 to sign into account 2. → User navigates to the security role management page 3. → Interface/UI sends requests to server to confirm user has permissions to access the security role management page 4. ← Server receives request 5. → Server sends request to database to check against what permission assignments the user has 6. ↔ Database receives, searches, and sends back to the server all hashed permission assignments that the user has 7. ← Server receives response from database 8. ↔ Server computes hashed permission assignments and compares against required assignment to access the security role management page to find that it is correct 9. → Server sends authenticated request to interface to allow user access to security role management page 10. ← Interface/UI receives request from server and displays page to user 11. → User uses interface/UI to select a user and choose which security roles ought to be assigned to the selected user 12. → User submits security role assignments and interface sends request to server 13. ← Server receives request 14. ↹ Server and database run steps 5-8 again to confirm that user does in-fact have permissions and there is no man-in-the-middle or account hijacking happening 15. ↔ Now verified, server computes hash of permission(s) that user is trying to apply to selected user 16. → Server sends hashed new role assignments to database alongside respective selected user to apply to 17. ↔ Database receives request, assigns permissions to be joined with designated selected user, and sends back confirmation to server 18. ↔ Server receives and relays confirmation to interface 19. ↔ Interface/UI receives confirmation and displays success message to user
Alternative Scenarios:	1. On step 8, if user does not have permissions to modify role assignments or view the page, then the server and interface/UI will throw an error message.

	2. On step 4, if server sees that the requested change is to the same user making the request, it will throw an error message back to the interface/UI.

UC4 - Data Encryption & Security	
Related Requirements:	REQ-2, REQ-3
Initiating Actor:	User
Actor's Goals:	To perform any operation securely.
Practicing Actors:	Users' Program, Interfarface/UI, Server, Database
Preconditions:	Post-quantum cryptographic process is implemented in system.
Postconditions:	User performs operation securely and efficiently.
Flow of Events for Main Success Scenario:	(NOTE: This will be an example success scenario to demonstrate searching for patient data. However, the cryptographic principles implemented here will be relevant for all operations.) 1. → User signs in using UC1, users' program loads backend server public key into cache for quick use 2. → User types "John" to search and submits 3. ↔ Users' program encrypts John with backend server public Kyber key 4. → Users' program sends ciphertext over HTTPS on TLS 1.2+ along with users' public Kyber key to backend server 5. ← Backend server authenticates user using JWT or OAuth 2.0 6. ↔ Backend server decrypts request using its private key that is on the TPM or HSM, private key is kept in cache only to be used by authenticated user for a predefined short-lived time such as 10 minutes of no activity 7. ↔ Backend server hashes plaintext name with a static SHA-256 secret key that is on the TPM or HSM to obtain hash ID index, the SHA-256 secret key is cached for a predefined short-lived time such as 10 minutes of no activity 8. → Backend server connects to database using persistent connection pooling to maintain database connection for a predefined short-lived time such as 10 minutes of no activity 9. → Backend server sends hash ID index to the database as a request for the encrypted row ID related to that hash ID index 10. ↔ Database finds and returns to backend server over HTTPS

	on TLS 1.2+ the random row ID key which matches the supplied hash ID index - the row ID must be encrypted using a backend server database-field-designated public Kyber key and NOT Threefish-1024 11. ↔ Backend server uses a database-field-designated Kyber private key to decrypt the row ID key and obtain real row ID; backend server should cache the real row ID for a predefined short-lived time such as 10 minutes of no activity 12. ↔ Backend server requests all fields related to the real row ID from the database 13. ↹ Database uses real row ID to find all corresponding fields for the initial request 14. → Database sends all ciphertext fields to backend server over HTTPS on TLS 1.2+ 15. ↔ Backend server uses a database-field-designated Kyber private key to decrypt the Kyber-encrypted Threefish-1024 key 16. ↔ Backend server decrypts all other ciphertext fields using the now-obtained patient-specific Threefish-1024 key 17. ↔ Backend server generates a Threefish-1024 key, stored twice: one temporarily for immediate use, one to be encrypted under the users' public Kyber key supplied from earlier step; backend server should cache the keys for a predefined short-lived time such as 10 minutes of no activity 18. ↔ Backend server encrypts all now plaintext patient data using the just-generated Threefish-1024 key 19. → Backend server sends the patient data Kyber-encrypted Threefish-1024 key and all respective encrypted Threefish-1024 data over HTTPS on TLS 1.2+ to users' program 20. ↔ Users' program uses private key stored on users' account in account database to decrypt Threefish-1024 key 21. ↔ Users' program uses Threefish-1024 key to decrypt all fields that the user requested 22. → Users' program displays plaintext of all fields requested by user 23. ↹ All other encrypted patient data fields are cached during users' session that expires every 10 minutes of no activity - AKA until the user signs out or is kicked out from inactivity, so if they request different data on the same patient it can be immediately decrypted simply by having the respective patient data Threefish-1024 key used to decrypt the cached encrypted patient data fields
Alternative Scenarios:	If at any point encryption/decryption fails, values aren’t found, or user(s) don’t have required permissions or data inputs, then an error

	will be cause in the program/interface/server/database and be ultimately relayed to the interface/UI for the user to know what happened.

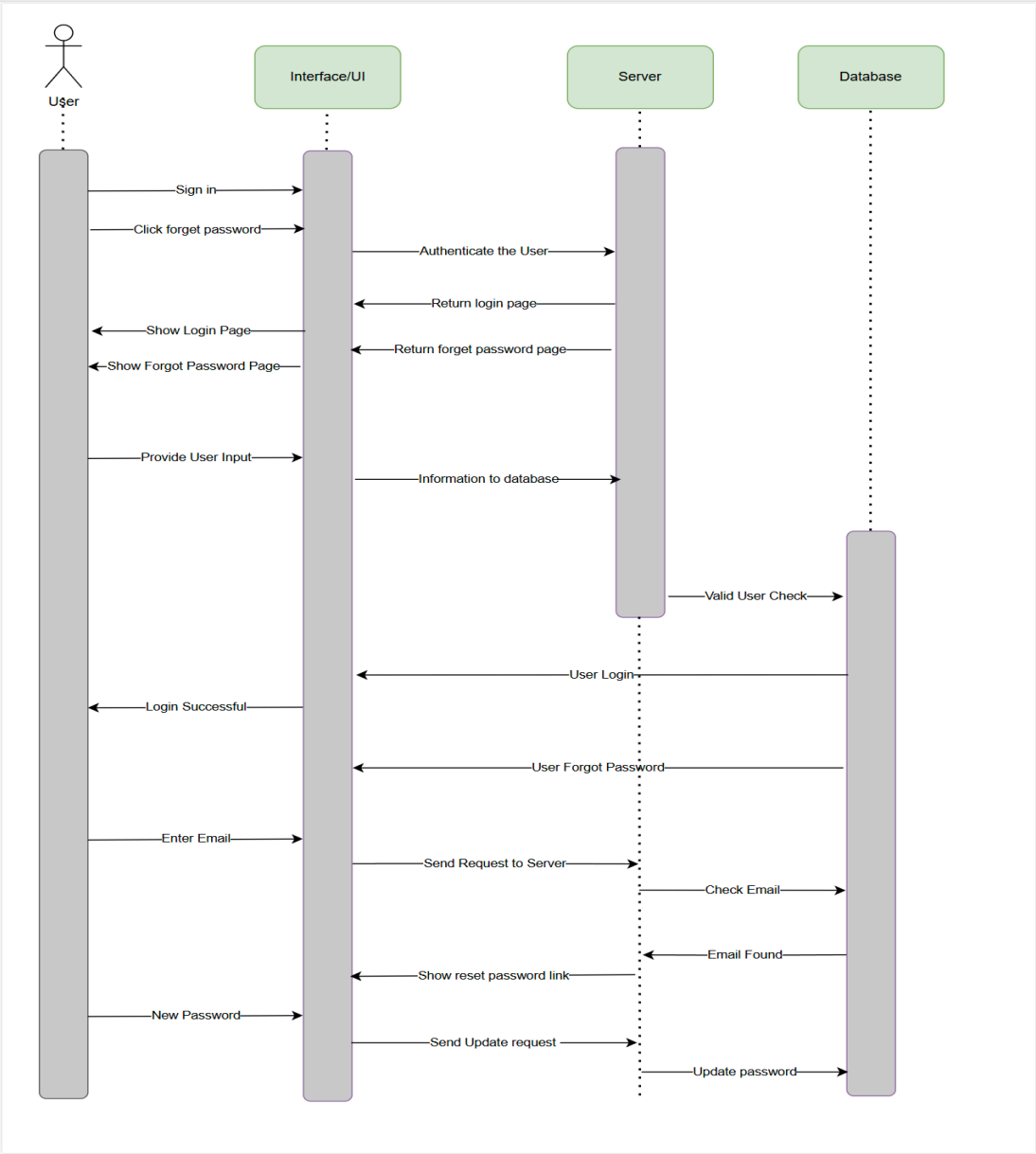
UC5 - Update Patient Medical Records	
Related Requirements:	REQ-4, REQ-5, REQ-6
Initiating Actor:	User (Healthcare administrator, caregiver, etc.)
Actor's Goals:	To access and update a users' stored medical record.
Practicing Actors:	Interface/UI, Server, Database
Preconditions:	User account exists and has permissions to view/modify patient records.
Postconditions:	User successfully modifies the desired patient record.
Flow of Events for Main Success Scenario:	1. → User signs in using UC1 2. → User navigates to search for the desired patient record, submits search 3. → Interface/UI sends requests to server to confirm user has permissions to process the given search request 4. ← Server receives request 5. → Server sends request to database to check against what permission assignments the user has 6. ↔ Database receives, searches, and sends back to the server all hashed permission assignments that the user has 7. ← Server receives response from database 8. ↔ Server computes hashed permission assignments and compares against required assignment to process the given search request to find that it is correct 9. → Server sends authenticated request to interface to allow user access to process the given search request 10. ← Interface/UI receives request from server and displays page to user 11. → User types in, uploads, or otherwise makes desired modifications to the patient's medical record and submits 12. → Interface/UI submits request to server for modifications 13. ← Server receives request 14. ↔ Server computes hash to make all patient data encrypted then sends to database 15. ← Database receives request 16. ↔ Database makes adjustments to given user's data and

	sends confirmation request to server once complete 17. ↔ Server receives confirmation request and relays confirmation to interface 18. ← Interface receives confirmation request 19. → Interface/UI lets user know that the request was submitted successfully
Alternative Scenarios:	1. On step 8, if user does not have permissions to process the search request or view the page, then the server and interface/UI will throw an error message. 2. If there is any error processing any of the user's inputted data (E.g., invalid characters), then the server and interface/UI will throw an error message.

SEE NEXT PAGE →

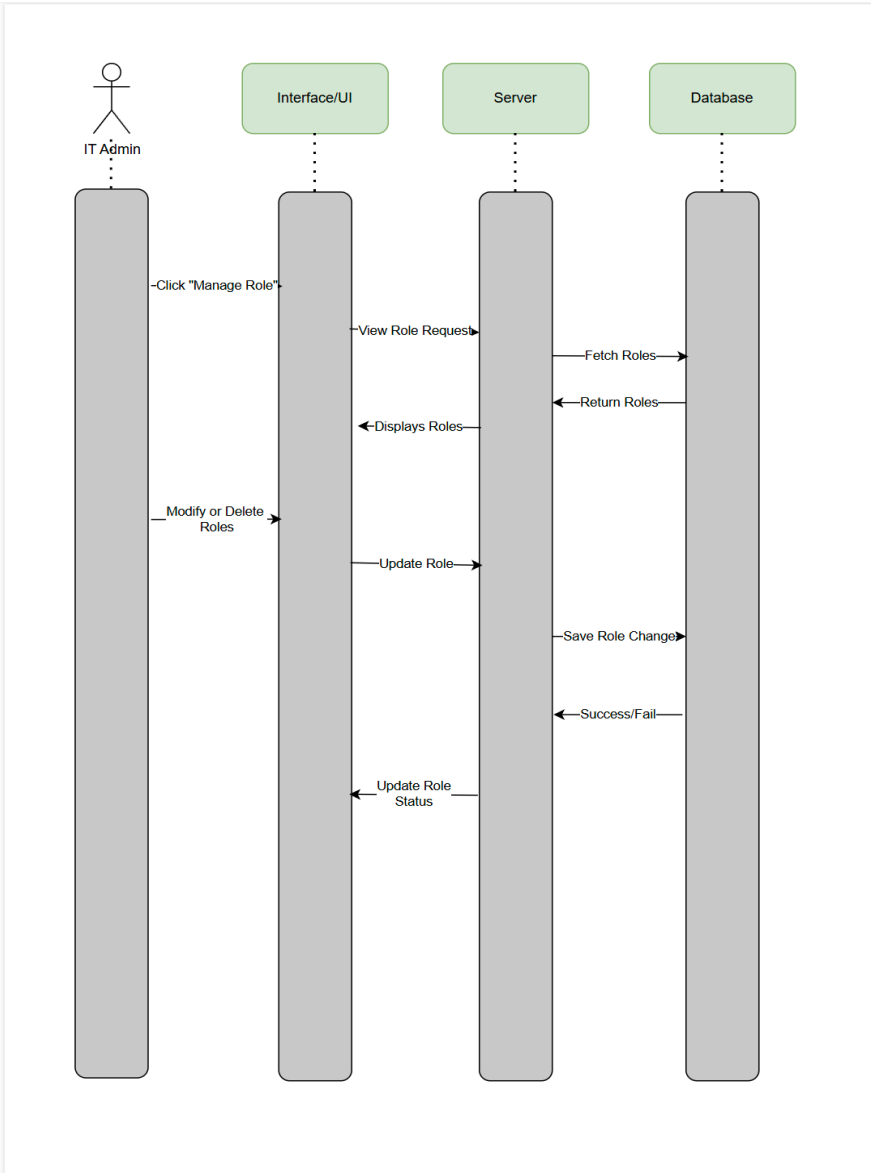
System Sequence Diagrams

1. UC1 - Login & Authentication



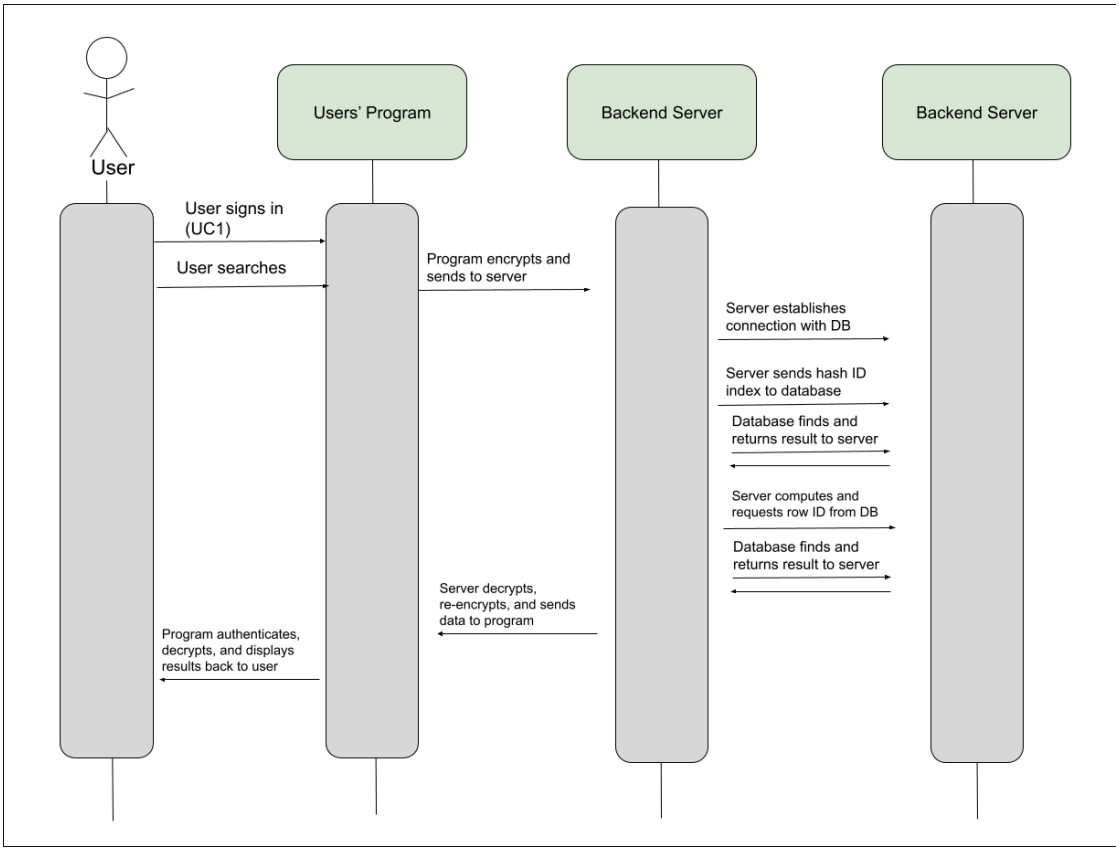
In the diagram above, the user either signs in with valid credentials or selects “Forget Password.” For login, the server checks the database to validate the user’s username and password. If valid, the user is granted access, otherwise, an error will occur. If the user clicks forget password, the system requests a valid email. If found in the database, a reset link or code is sent, allowing the user to create a new password, which is then stored in the database.

2. UC2 - Role Management



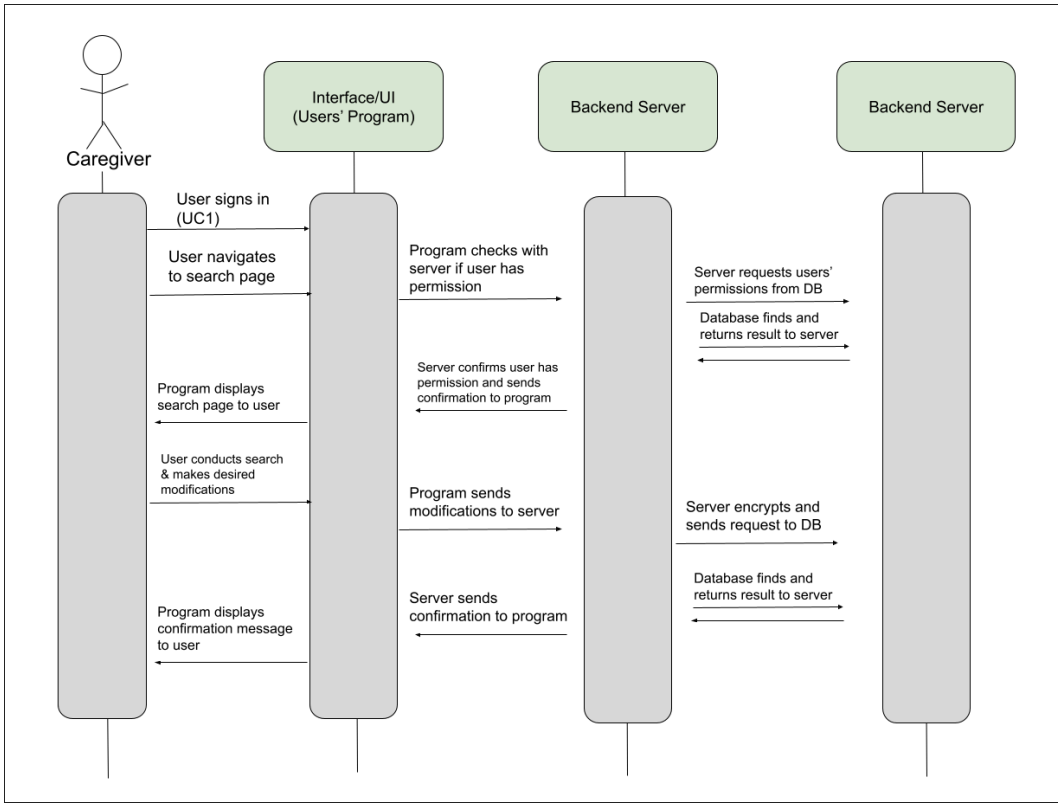
In the diagram above , the IT admin begins by clicking “Manage Roles” from the interface. Then the UI sends a request to the server to retrieve existing roles. The server fetches roles from the database, and returns it to the UI for the admin to view or edit permissions. When the admin does this the UI sends updated role information to the server and saves those changes to the database. The admin will be notified if it was successful or not.

3. UC4 - Data Encryption & Security



In the diagram above, it is intended to focus on how the cryptographic implementation may resemble the given fully-dressed description outline. The user signs in using the UC1 procedure. From here, the user conducts a search, similar to how UC5 below functions, the program sends this request to the backend server. Once there, the backend server authenticates the user and establishes a secure pooling connection with the database. Once the connection is established, the server can than effectively request the desired hash ID index and retrieve it from the database to then decrypt and determine the real row ID. Now, the database can request all data from the real row ID in the database. Once this is done, the backend server can decrypt the data and re-encrypt using the respective proper keys and algorithms at each point, and send the encrypted data back to the user. The user then uses their private key to decrypt the data finally to be displayed back to the original user.

4. UC5 - Update Patient Medical Records

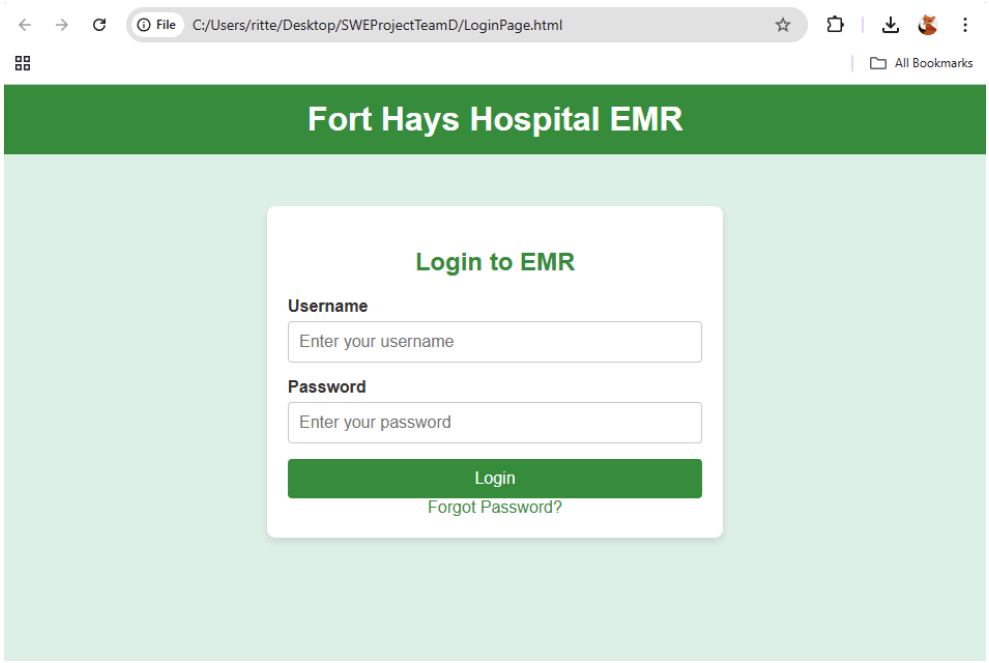


In the diagram above, it represents the process similar to how it is outlined in the fully-dressed description for UC5. Essentially, what is being carried out here is that a caregiver (doctor, nurse, etc.) signs into the system following the UC1 procedures, then the caregiver proceeds to navigate to the search page. The backend server authenticates the user with the database to ensure the user has permission to conduct searches and which exact permissions the user has, then gives confirmation back to the program to allow the user to continue the search. Now, once the user conducts the search and proper modifications to the desired patient, the backend server takes these modifications, conducts the required encryptions, and sends them to be updated in the database. Once the database has updated the field, it is then relayed back to the backend server and back to the user to receive confirmation of the update.

Section 4: User Interface Specification

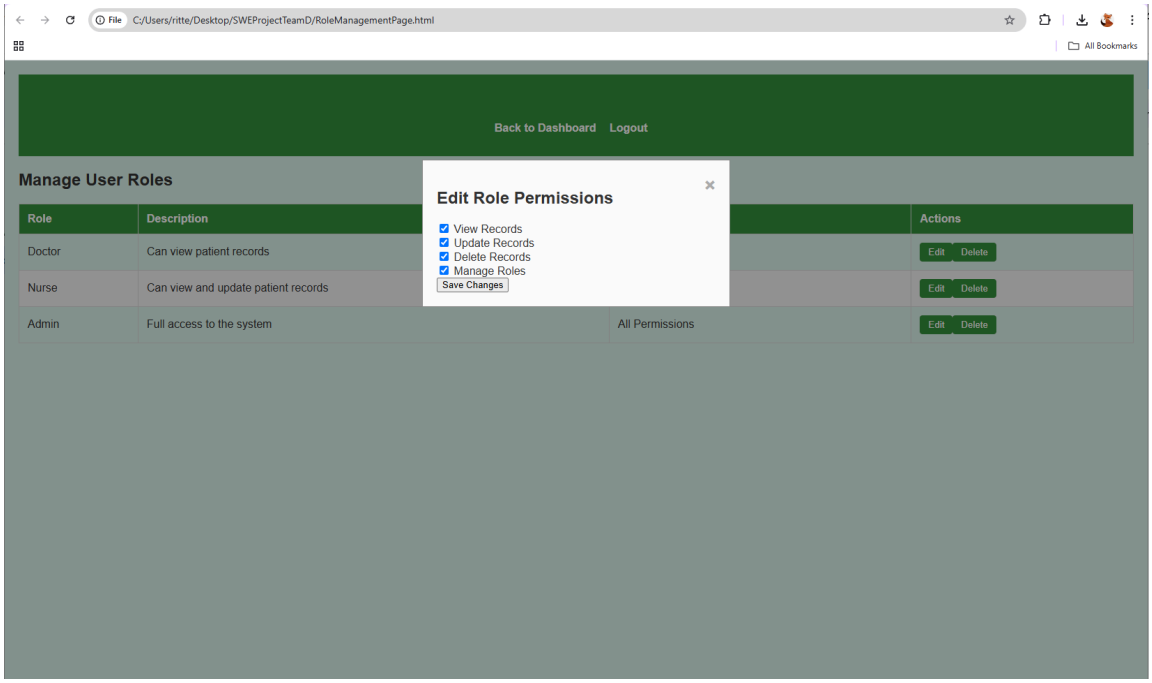
Preliminary Design:

1. **Login and Authentication: User Login**
   The user navigates to the login page where they input their username and password. Authentication occurs using a secure backend system, if the authentication fails an error message is displayed. Upon successful login, the user is directed to their role-based homepage.
   * mock up image not reflected for final design



2. **Role Management (For IT Administrators) : Assign/ Modify Roles and Permissions**
   From the admin dashboard, the user selects “manage roles” from a categorized menu. A list of roles such as doctor, nurse and admin is displayed. The administrator can select any role to modify its permissions. Permissions are listed with checkboxes, allowing the admin to enable or disable specific access for each role. Changes will be saved via a “Save Changes” button, which will update role configurations in real time.

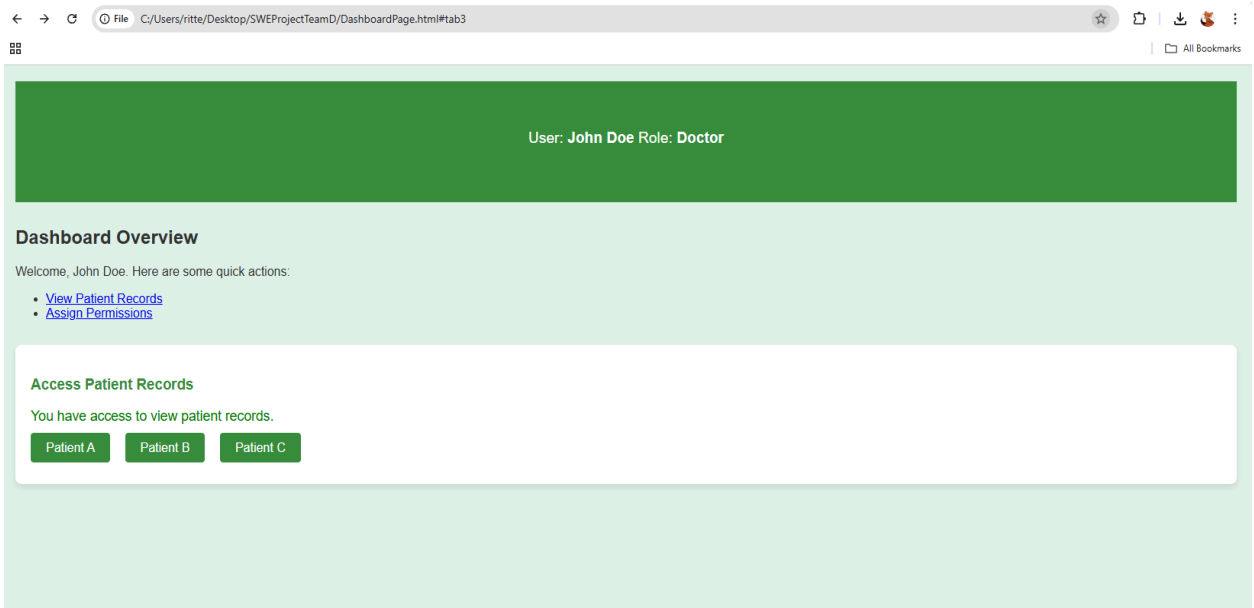
*mock up image not reflected for final design



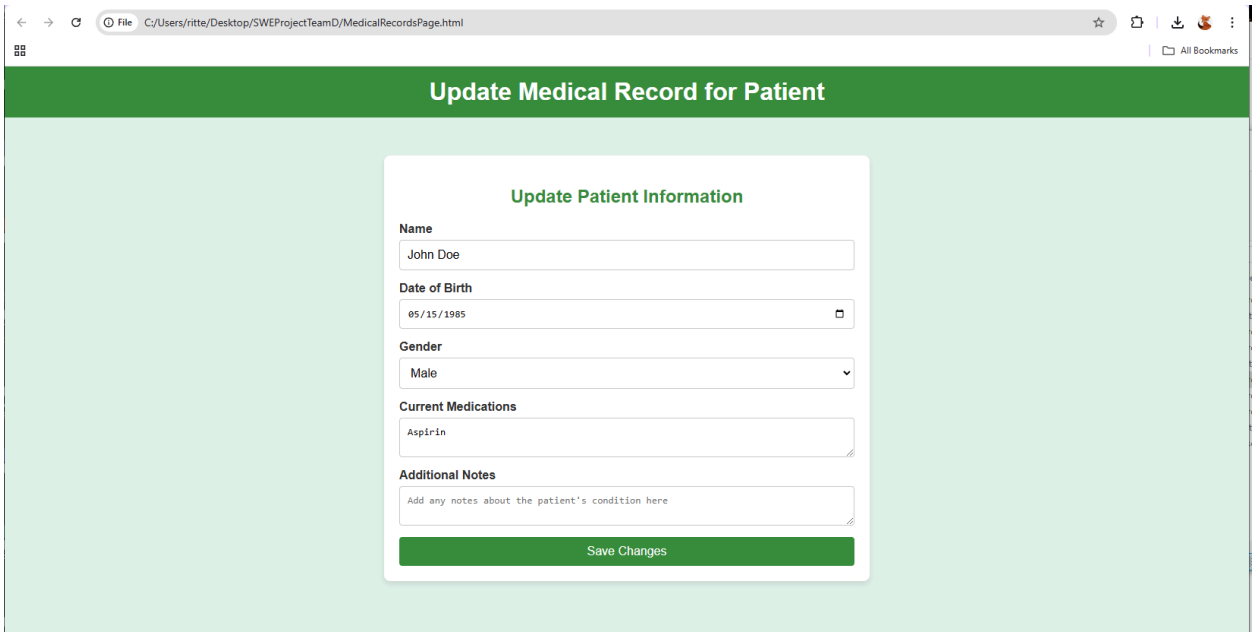
3. **Medical Record Access Control (For Healthcare Professionals) : Access Medical Records Based on Role**

The user logs in and is automatically granted access to the medical records due to their role. The user can view patient records only for those who are authorized to see. If unauthorized access is attempted, the system displays a “Permission Denied” message. The system presents patient information with a clean, easy to navigate interface and the information is categorized and presented in tabs.

*mock up image not reflected for final design

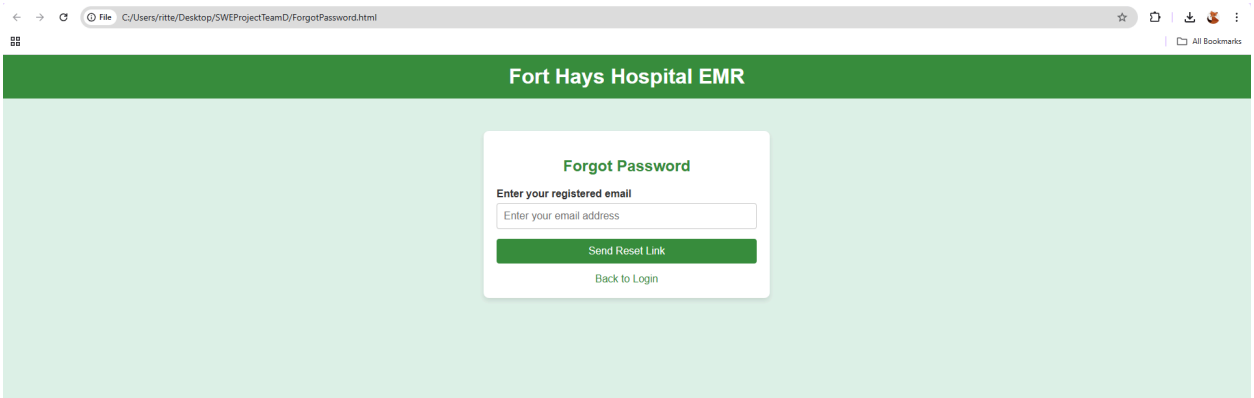


4. **Data Encryption and Security: Data Encryption**
   All patient records are encrypted using post-quantum cryptography standards. Any communication between the client and server is secured with HTTPS.
5. **Updating Medical Records : Add/Modify Patient Data**
   The user can input updates to patient records via a form. The system pre-fills known information and the user inputs new details. The user then has to save the changes and it will update automatically for the next log in.



6. **Password Recovery- Forgot Password**
   If a user forgets their password, they click on the “Forgot Password” link on the login page. A secure, encrypted email is sent to the registered email address with instructions to reset their password. Once the password is reset, the user can log in with the new credentials.

*mock up image not reflected for final design



User Effort Estimation:

1. **User Login & Authentication**

Flow of Events:

1. Click in the username field
    - Navigation: 1 click
2. Type Username (Lets just use admin for the example)
    - Data Entry: 5 keystrokes
3. Click in password field
    - Navigation: 1 click
4. Type password
    - Data Entry: 5 keystrokes
5. Click “Login” button
    - Navigation: 1 click

Total:

Navigation = 3 events
Data Entry = 10 events
Overall = 13 events

Navigation = 23.1%
Clerical Data Entry = 76.9%

2. **Role Management (IT Admin Editing a Role)**

Flow of Events:

1. Click the “Manage Roles” option on the admin dashboard
    - Navigation: 1 click
2. Click on specific role from the role list that is displayed
    - Navigation: 1 click
3. Toggle permission checkbox to modify the selected roles displayed
    - We are going to assume that the admin wants to change 4 permission settings
    - Navigation: 4 clicks
4. Click “Save Changes” button to update role configuration
    - Navigation: 1 click

Total:

Navigation = 7 events
Data Entry = 0 events
Overall = 7 events

Navigation = 100%
Clerical Data Entry = 0%

3. **Updating a Patient's Medical Record**

Flow of Events:

1. Click the "Patient's Record" option on the dashboard
    - Navigation: 1 click
2. Click the "Update" button fo the patient
    - Navigation: 1 click
3. Click into the Current Medication field
    - Navigation: 1 clicks
4. Type the Medication ("Aspirin")
    - Data Entry: 7 keystrokes
5. Click into additional notes
    - Navigation: 1 click
6. Type a note for the medication ("Take with food")
    - Data Entry: 14 keystrokes
7. Click "Save Changes" button" to update the information
    - Navigation: 1 click

Total:

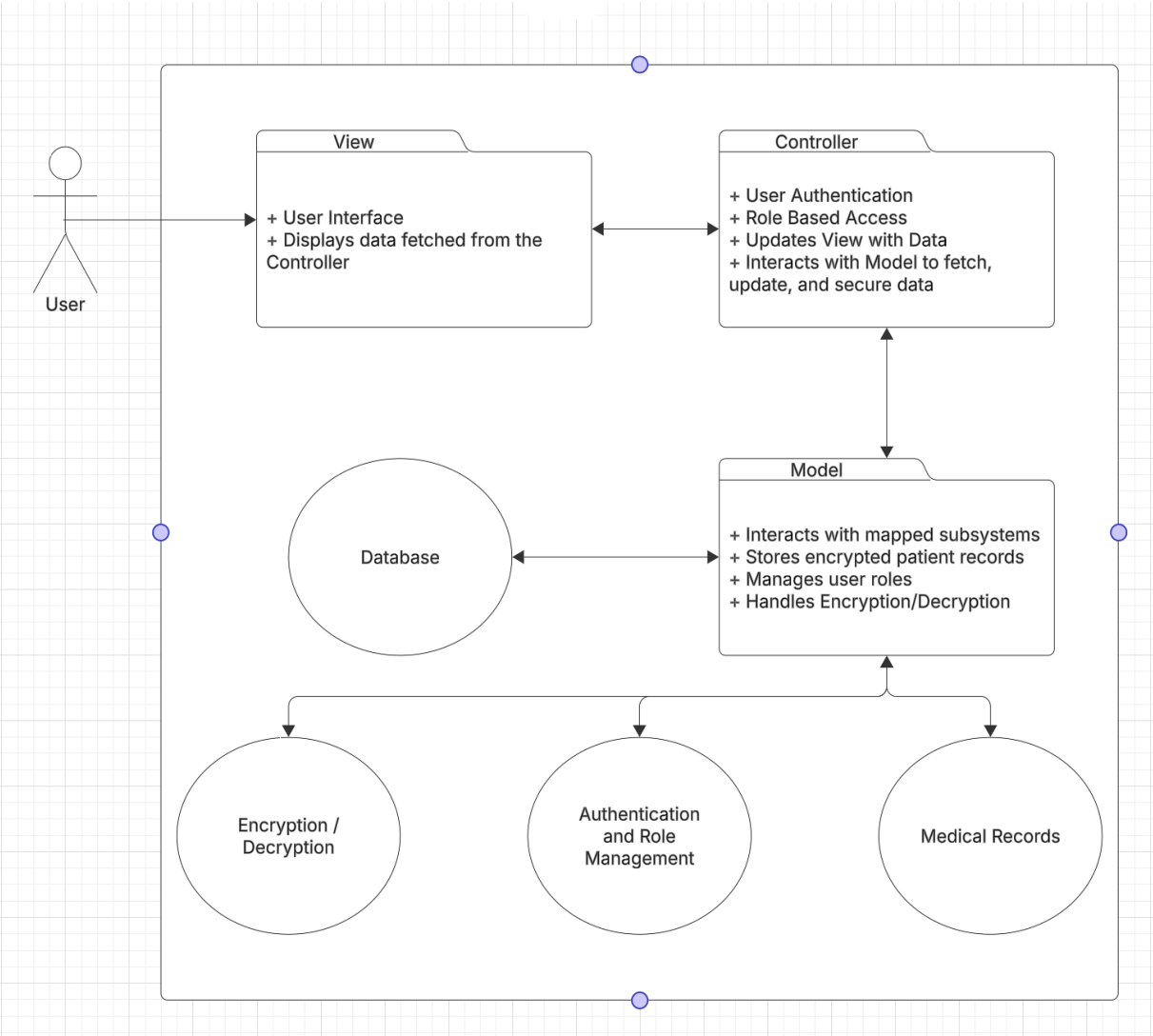
Navigation = 5 events
Data Entry = 21 events
Overall = 26 events

Navigation = 19.2%
Clerical Data Entry = 80.8%

SEE NEXT PAGE →

Section 5: System Architecture and System Design

Identifying Subsystems



The Enhanced Electronic Medical Record (EMR) Platform Security and Role-Based Access for IT Administrators follows a layered MVC design pattern the main subsystems (MVC) and supporting subsystems.

The Model is responsible for storing encrypted patient records, manage user roles, and handle encryption/decryption of data. It will also interact with the database and other supporting subsystems.

The View is responsible for the interaction with the user through the UI and interact with the Controller.

The Controller is responsible for handling the authentication, managing the access, and routing the data across the EMR.

Architecture Styles

The system is structured using a Model-View-Controller (MVC) architectural pattern, which will ensure a clear separation of concerns, modularity, and scalability. The system will consist of multiple interconnected subsystems, each responsible for a specific function as depicted in diagram above.

As a whole, the EMR is designed to ensure secure, role-specific access to patient data while maintaining regulatory compliance (HIPAA) requirements. It incorporates Post-Quantum Cryptographic (PQC) encryption to enhance data security and protect sensitive medical records from emerging threats.

Mapping Subsystems to Hardware

The system architecture is centralized on a master server that hosts the following subsystems:
Database: Stores all data, including medical records, user information, and logs.
Web Application and API: Serves the web application interface and handles requests between the client-side and the back-end API. The client-side access includes healthcare professionals, administrators, and managers using desktop computers to access the concierge web application through a browser.

Connectors and Network Protocols

Our system comprises multiple parts that must all effectively communicate with each other. Initially, there is the client's device that connects to the initial web server / web application; from here, the web application communicates with a backend server for certain operations and performance requirements that then organizes various related data in a connected SQL database. We can see these connections as so:

Client to Web Server - HTTPS over TLS 1.2/1.3 with the endpoint web server being a cloud hosted Flask environment utilizing OAuth2/JWT session token authentication and persistence. This part will also be using JSON for data formatting and HTML/CSS for UI.

Web Server to Backend Server - Now from the flask web server to backend Flask service, authentication is validated with the incoming OAuth2/JWT Flask web server session tokens. The web server sends encrypted data to the backend server over HTTPS on TLS 1.2/1.3 for PQC processing and related operations including items such as key generation, encryption, and signature verification, as needed.

Backend Server to Database - The backend server secures a connection with the database using SQL over TCP/IP on TLS 1.2/1.3 and utilizes connection pooling to optimize operations during a predefined time limit (E.g., 10 minutes of no activity). All queries are executed using prepared statements to reduce the risk of SQL injection.

Global Control Flow

Event driven - No order, users can login and send requests needed (login, view records, update records, or manage roles). In any order that is needed. The system will wait for form submission and process accordingly. No single path that every user has to follow. The system must react to dynamic user requests for access to patient data, with each user's flow dependent on their role and security clearance.. Actions such as login, data access, or role management trigger security checks and user validation.

Real-Time Execution- Time constraints aren't stringent on all operations, but core functionality (data access, role assignment) must happen in real-time.

Periodic Updates- Some background tasks like compliance audits, system updates, and encryption policy checks should be run periodically, but they should not affect immediate user flows.

Time Dependency - To keep up with HIPAA needs a session timeout (10 mins of inactivity). Not a real time system as it is a ERM, responds as it occurs.

Hardware Requirements

The Enhanced Electronic Medical Record (EMR) Platform Security and Role-Based Access for IT Administrators will be reliant on the following hardware:

Screen Display: The EMR will be able to run on any device that has access to the internet through a web browser. The initial design will be implemented for laptop/desktop computers with the functionality using an iPad/tablet as a fast follow.

Database Server: A database server with sufficient resources will also be required in order to store and secure patient data.

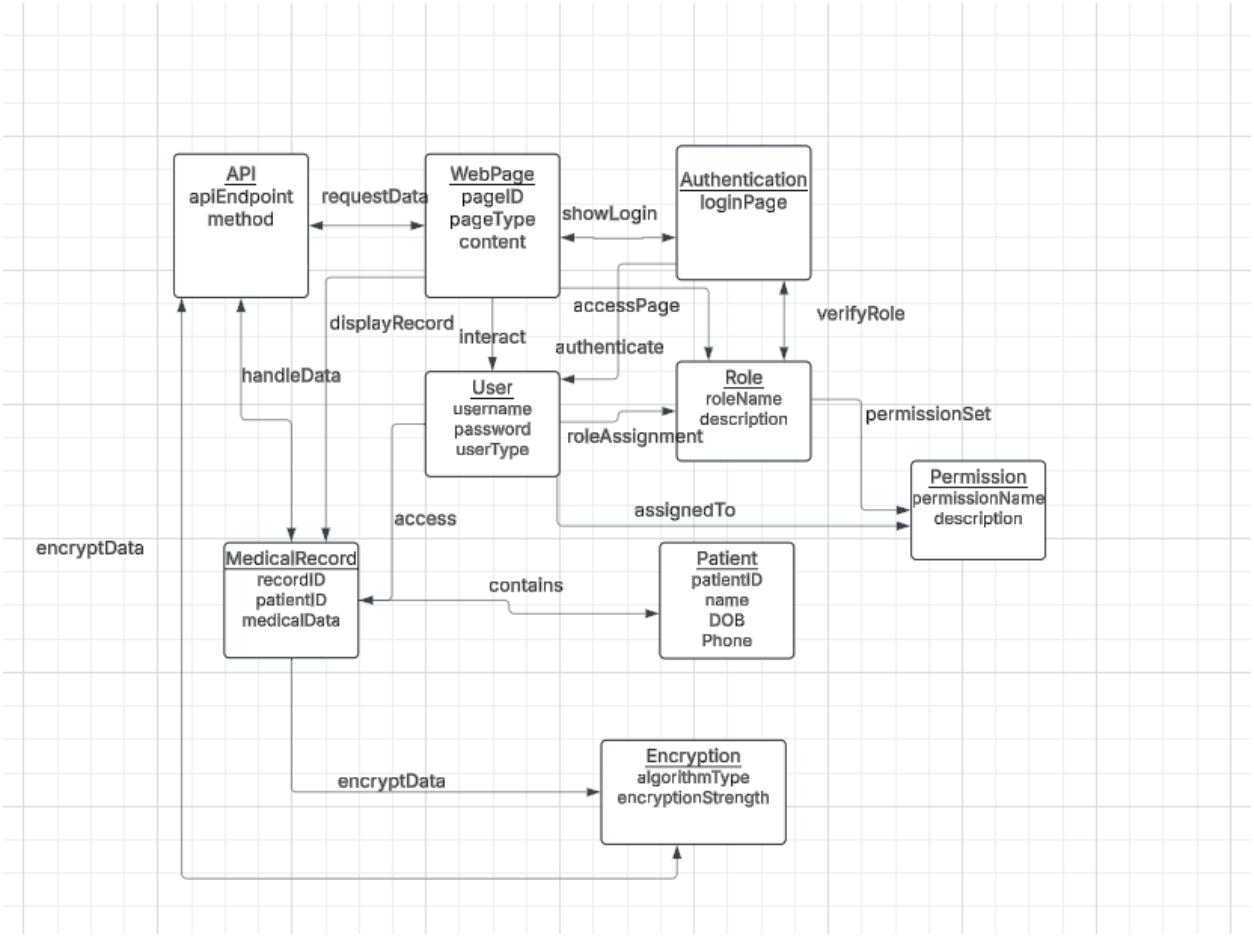
Communication Network: A stable internet connection for establishing and accessing the EMR and its subsystems.

Disk Storage: The database and other subsystems will require adequate disk storage.

Section 6:Analysis and Domain Modeling

Conceptual Model

In this section, we present the Conceptual Model for our EMR project. The Conceptual Model forms the foundational framework of our software's architecture, with the concept definitions, association definitions, attribute definitions, and traceability matrix.



Concept Definitions:

Concept Name	**Responsibility Description**	**Type (D-doing, K-knowing)**
User	Represents the individuals who interact with the system	K (knowing)
Administrator	Specialized User who manages roles, permissions, and access	K (knowing)
Role	Defines a set of permissions granted to a user for system access	K (knowing)
Permission	The individual rights or access levels a user has in the system	K (knowing)
MedicalRecord	Represents a patient's medical information stored in the system.	K (knowing)
Patient	Represents the individuals whose medical records are stored.	K (knowing)
Encryption	The encryption standard used for securing sensitive data.	K (knowing)
Post-Quantum Cryptography(PQC)	Future-proof encryption standard to protect against quantum computing threats.	D(doing)
WebPage	The interface where users interact with the system (UI).	D (doing)
API	The backend service that handles interactions with the database.	D (doing)
Authentication	The process of verifying a user's credentials before access.	D (doing)

Association Definitions

Concept Pair	Association Definition	Association Name
User⇔Role	A user is assigned one or more roles which determines the rights.	roleAssignment
Role⇔Permission	A role has a set of permissions that define what a user can do in the system.	permissionSet
User⇔MedicalRecord	A user accesses medical records of patients	access
Administrator⇔Role	An administrator manages the roles and assigns permissions to users.	manageRoles
Administrator⇔MedicalRecord	An administrator manages the medical records and ensures compliance.	manageRecords
MedicalRecord⇔Patient	Each medical record belongs to one patient.	contains
User⇔Authentication	Users are authenticated before accessing the system.	authenticate
Administrator⇔API	Administrators interact with the API to manage roles, permissions, and medical records	manageAPI
Encryption⇔MedicalRecord	Medical records are encrypted to ensure confidentiality.	encryptData
WebPage⇔ User	WebPages allow users to interact with the systems interface	interact
WebPage⇔Role	Different WebPages might be accessible depending on the role of the user.	accessPage

WebPage⇔MedicalRecord	WebPages display and allow modification of medical records	displayRecord
WebPage⇔Authentication	WebPages include authentication processes	showLogin
WebPage⇔API	WebPages communicate with the backend API for data retrieval and interaction	requestData

Attribute Definitions

Concept Name	**Attribute**	**Description**
User	userID	Unique identifier for the user.
	userName	Name of the user.
	password	User's secure login password.
Administrator	adminID	Unique identifier for the administrator.
	contactInfo	Contact information.
Role	roleName	Name of the role
	roleID	Unique identifier for the role
	description	Description of the role's responsibilities.
Permission	permissionName	Name of the permission
	description	Description of the permission.
MedicalRecord	recordID	Unique identifier for the medical record.
	patientID	The ID of the patient associated with the medical record.
Encryption	algorithmType	The encryption algorithm used.

	encryptionLevel	The level of encryption.
WebPage	pageID	Unique identifier for each webpage.
	pageType	Type of the page.
	content	The content that the page displays.
API	apiEndpoint	The URL or endpoint for API calls.
	method	HTTP method used for the API
Authentication	loginPage	The WebPage that handles user login.

Traceability Matrix

Use Case	User	Admin	Role	Permission	Medical Record	Patient	API	Authentication	Encryption	Web page	PQC
UC-1	X						X	X		X	
UC-2		X	X	X			X			X	
UC-3	X			X	X	X				X	X
UC-4					X				X		
UC-5	X			X	X	X	X			X	
UC-6	X							X		X	

System Operation Contracts

Operation	Registration
Use Case:	UC-1: User Login & Authentication
Pre-Condition	○ User submits a valid userID and password via login page ○ Authentication service is available
Post-Condition	○ Session is established if credentials are correct ○ Authentication log entry is generated

Operation	Registration
Use Case:	UC-2: Role Management
Pre-Condition	○ Administrator is logged into the system and role management interface is accessible ○ Administrator has the required permissions
Post-Condition	○ System updates the role ○ The updated role is immediately available to the system

Operation	Registration
Use Case:	UC-3: Access Medical Record
Pre-Condition	○ User is authenticated ○ User has role with valid permissions for request medical record ○ Medical record exist
Post-Condition	○ Medical record is displayed if user's permissions are valid ○ If not valid permissions a error is shown

Operation	Registration
Use Case:	UC-4: Data Encryption
Pre-Condition	○ Systems encryption module is available and properly configured ○ Data can be encrypted or decrypted is valid
Post-Condition	○ Specific data is encrypted or decrypted as requested ○ Data remains secure in storage or in transit, due to security policies

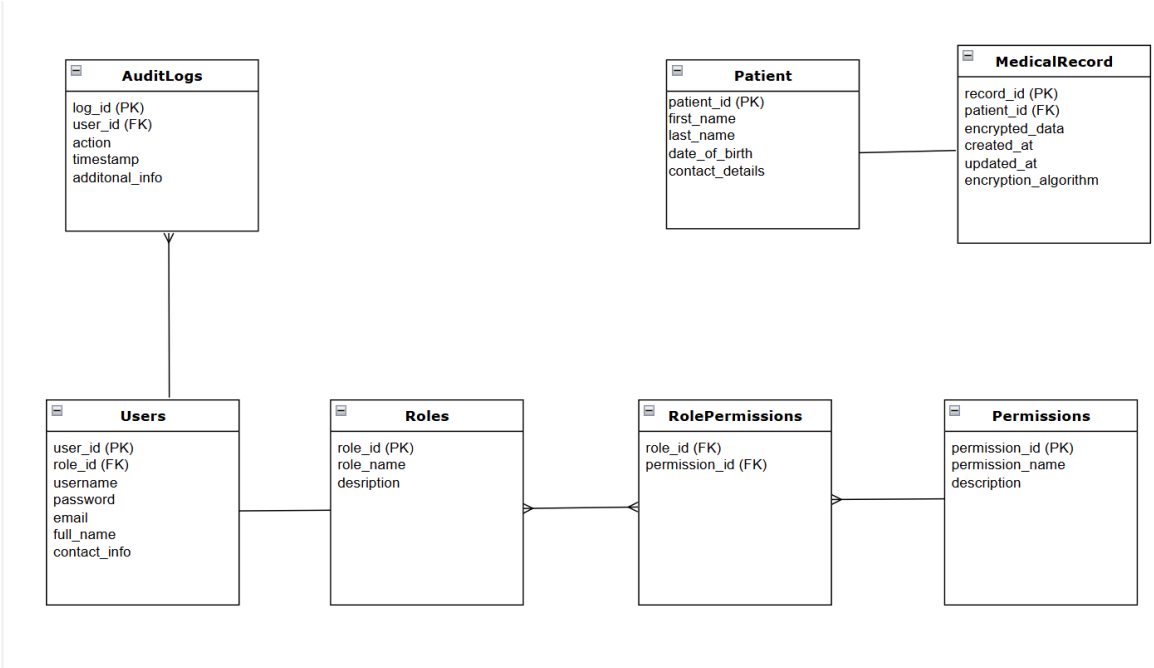
Operation	Registration
Use Case:	UC-5: Update Medical Record
Pre-Condition	○ User logged in and has a role allowing record updates ○ Medical record exist in system ○ User inputs valid data
Post-Condition	○ Medical record is updated and saved in system storage

Operation	Registration
Use Case:	UC-6: Password Recovery
Pre-Condition	○ User can verify their identity ○ Password recovery interns=face is accessible ○ Systems email service is available
Post-Condition	○ Secure reset link is sent to user's verified contact method ○ User can update their password upon proving valid verification ○ User is notified of the change

Data Model and Persistent Data Storage

The EMR platform deals with sensitive, real-world patient health records and data, meaning data must be persistent across system executions. This is critical because:

1. Medical Records are long-term assets - patient data must be preserved for years to meet legal and regulatory requirements.
2. Audit trails must be persistent - system actions should be logged to be aligned with compliance auditioning requirements.
3. User Accounts & Roles must be persistent - role-based access control is meaningless if user roles do not persist across sessions.
4. Encryption Keys and Security must be persistent - since the system uses Post-Quantum Cryptography (PQC), the encryption metadata must be retained.



The Users table stores account credentials, role assignments, and contact information, making it the core entity for all authenticated interactions. Each user can have multiple roles, which is managed through the UserRoles table. This table maps users to roles, enabling flexible role-based access control.

The Roles table defines each system role (such as Doctor, Nurse, Admin), while the Permissions table defines granular access rights (such as "View Records" or "Edit Records"). The RolePermissions table links each role to multiple permissions, ensuring roles can evolve without directly modifying users.

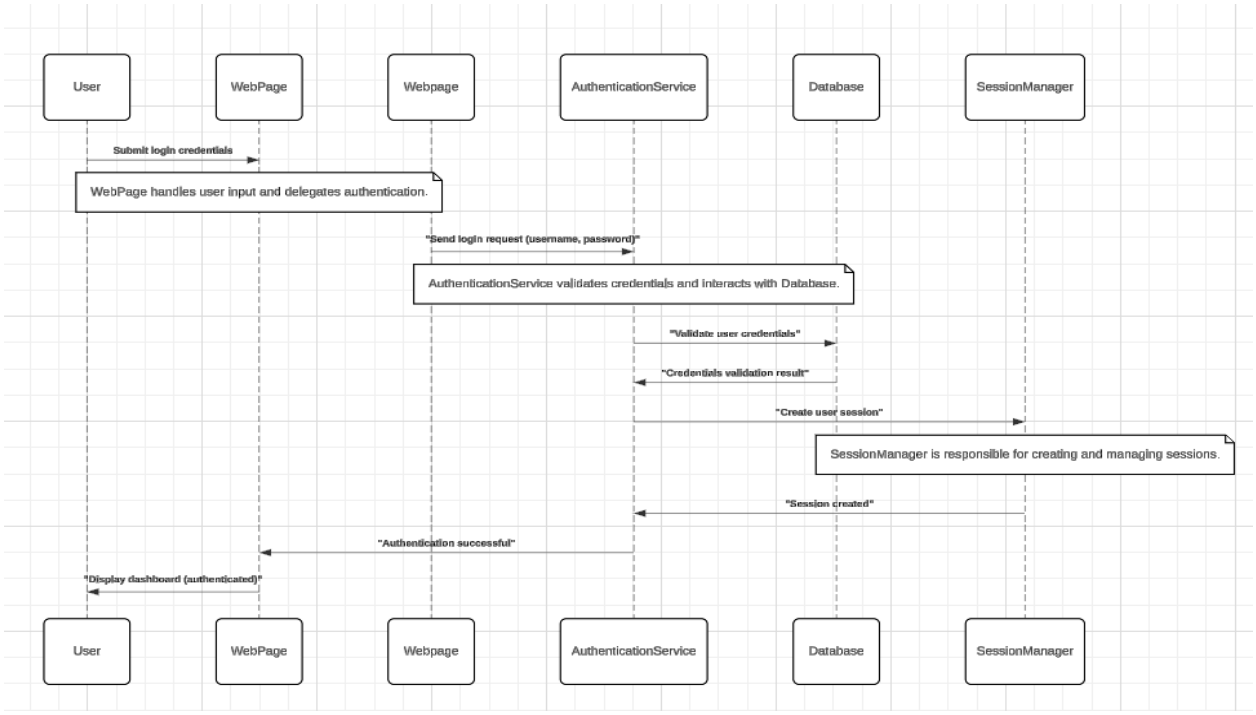
The Patients table stores core patient demographic data, while the MedicalRecord table stores individual encrypted health records linked to patients. Each record also captures timestamps and encryption metadata, ensuring full traceability and security.

The AuditLogs table tracks every sensitive system action (logins, record accesses, updates), providing a complete compliance audit trail. Each log entry records the responsible user, action, timestamp, and any additional relevant details.

All these tables form a tightly integrated schema ensuring data integrity, role-based security enforcement, regulatory compliance, and efficient querying.

Section 7: Interaction Diagrams

UC-1 Login and Authentication



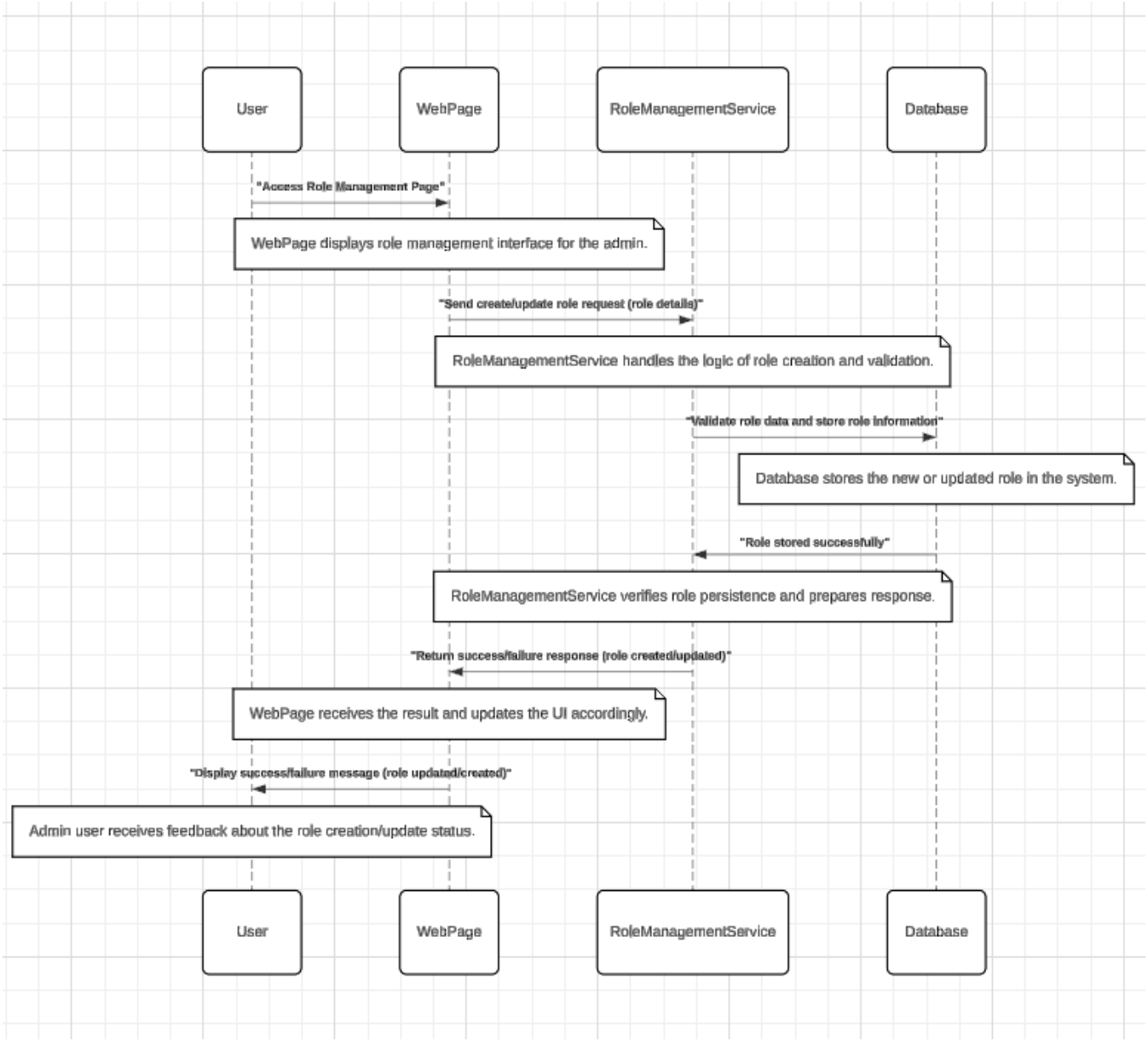
Design Principles for UC-1: User Login & Authentication:

High Cohesion Principle: The WebPage is focused solely on handling the user's input, and it delegates authentication responsibilities to the AuthenticationService, which handles user validation. The SessionManager is responsible only for creating and managing sessions, which keeps the system components cohesive and ensures that each part of the system has a single responsibility.

Expert Does Principle: The AuthenticationService is the expert in user authentication and validation because it has the necessary knowledge about the credentials and how to verify them. The SessionManager holds the required knowledge about session handling and is responsible for creating and managing sessions.

Single Responsibility Principle: Each component performs only its designated task. WebPage handles the presentation layer, AuthenticationService manages user authentication, and SessionManager is in charge of session creation.

UC-2 Role Management



Design Principles for UC-2: Role Management:

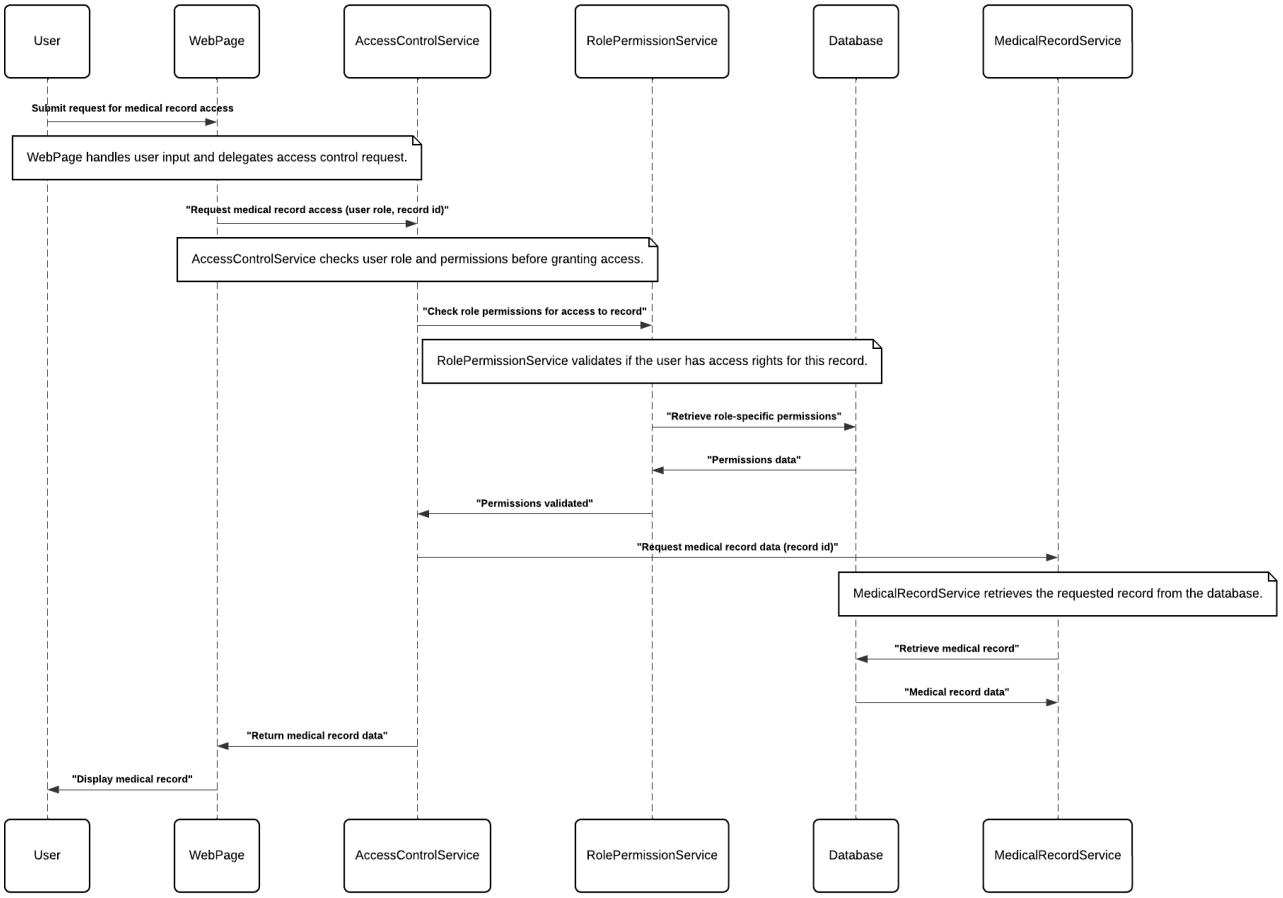
High Cohesion Principle: The AccessControlService is highly cohesive, focusing solely on checking and validating access control rules for the medical records. The RolePermissionService is responsible for handling role-based permission checks, ensuring that role and access control logic is not scattered across multiple components.

Expert Does Principle: The AccessControlService is the expert in verifying if a user is authorized to access a particular medical record based on their role and permissions. The RolePermissionService is the expert in managing and validating the roles and

permissions of users. Each service possesses the necessary knowledge to effectively carry out its responsibilities.

Separation of Concerns: Each component handles a separate concern: the AccessControlService for access management, the RolePermissionService for role validation, and the MedicalRecordService for retrieving medical records. This separation enhances system modularity and maintainability.

UC-3 Medical Record Access Control



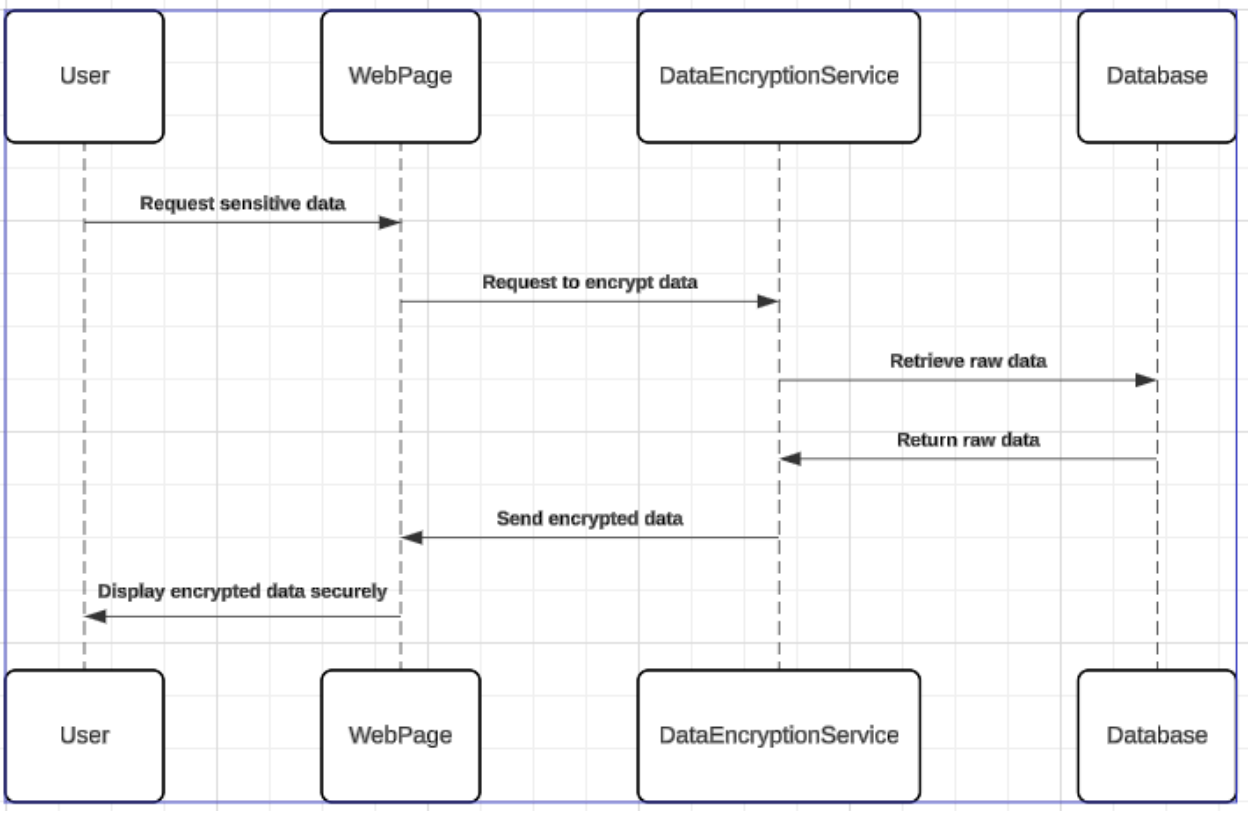
Design Principles for UC-3: Medical Record Access Control:

High Cohesion Principle: The MedicalRecordService is highly cohesive, focused only on managing and retrieving medical records. The AccessControlService is also cohesive, responsible for validating whether the user has the right to view the medical record based on their permissions.

Expert Does Principle: The AccessControlService is the expert in validating whether a user has the necessary permissions to access a medical record. The MedicalRecordService is the expert in retrieving medical record data from the database. The RolePermissionService is the expert in checking if a user's role grants them the required permissions to access the records.

Single Responsibility Principle: The MedicalRecordService is solely responsible for interacting with the database to retrieve medical records, while the AccessControlService is responsible for ensuring that users are authorized to access the requested records, and the RolePermissionService handles permission validation. This clear separation of concerns aligns with the Single Responsibility Principle.
Publisher-Subscriber Pattern : If any changes to user roles or permissions need to be updated across the system in real time, a Publisher-Subscriber pattern could be applied, where the RolePermissionService acts as the Publisher, notifying relevant components (Subscribers) about changes in roles and permissions. This would ensure that all parts of the system remain synchronized with user access rights.

UC-4 Data Encryption



Design Principles for UC-4: Data Encryption:

High Cohesion Principle: The DataEncryptionService is solely responsible for handling encryption and decryption of sensitive data. By grouping all related functionality within one service, the system's design becomes easier to maintain and understand.

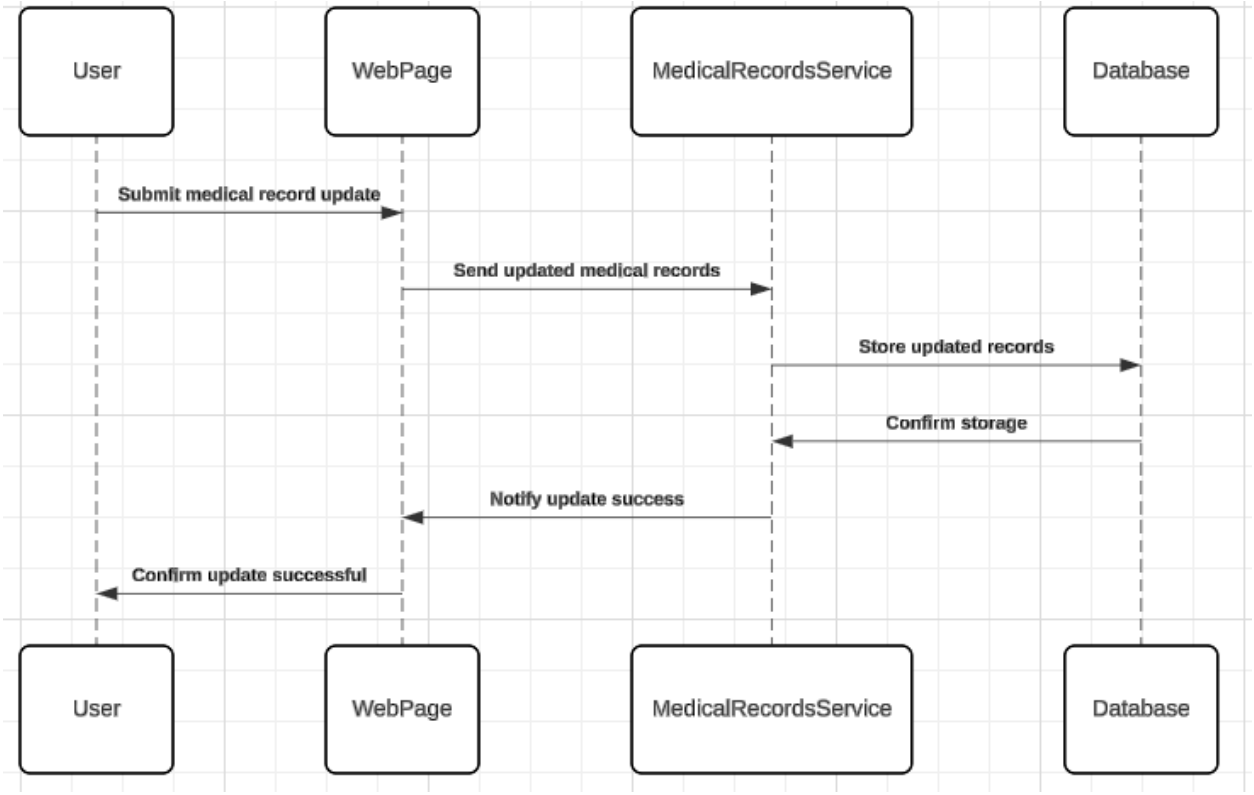
Expert Does Principle: The DataEncryptionService is the expert in encrypting data, so it takes responsibility for ensuring that data is encrypted properly before being sent to the user. This separation of concerns allows for a clean architecture.

Low Coupling Principle: The WebPage interacts with the DataEncryptionService and does not need to worry about how encryption is done. The WebPage remains focused on user interface concerns, while the DataEncryptionService handles encryption. This keeps the system flexible and easier to maintain.

Single Responsibility Principle : Each component has a distinct responsibility: the WebPage handles the presentation layer, the DataEncryptionService handles the data security concerns, and the Database handles data storage. This clear division of responsibilities ensures that the system remains modular

.*Secure Design Principle:* The encryption process ensures that sensitive patient data remains secure both during storage and transmission. By following secure design principles, the system protects users' data from unauthorized access.

UC-5 Updated Medical Records



Design Principles for UC-5 Updated Medical Records:

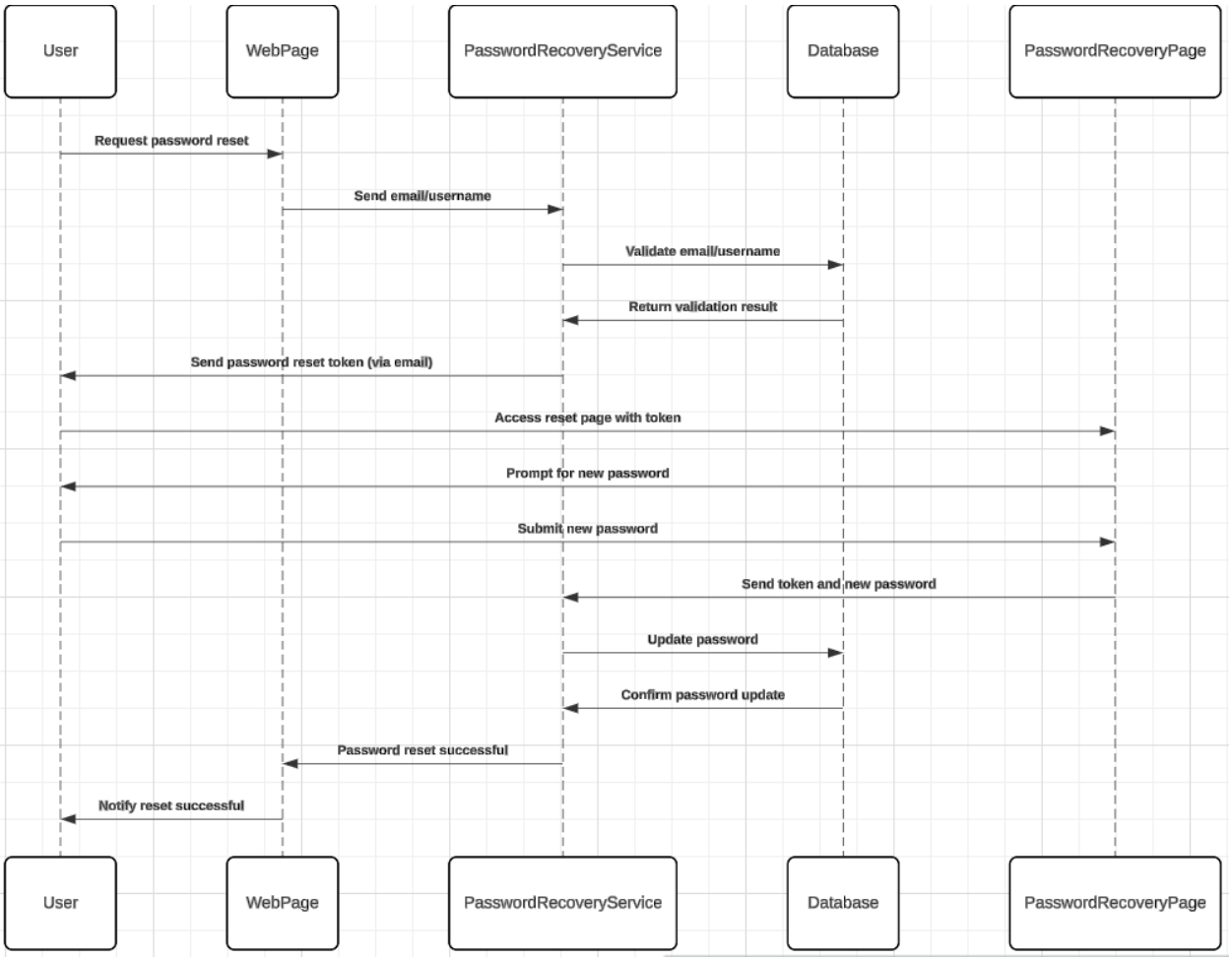
High Cohesion Principle: The MedicalRecordsService handles all functionality related to the updating and validation of medical records. This centralization ensures that related functionality remains cohesive and easier to maintain.

Expert Does Principle: The MedicalRecordsService is assigned the responsibility of managing medical records because it has the necessary knowledge and logic to update and validate them correctly. The Database stores and manages data, while the MedicalRecordsService handles the business logic.

Low Coupling Principle: The WebPage only communicates with the MedicalRecordsService, not directly with the Database. This ensures that the system is loosely coupled, with each component focusing on its area of expertise.

Single Responsibility Principle : Each component in the system focuses on one responsibility. The WebPage deals with the user interface, the MedicalRecordsService with business logic related to medical records, and the Database with data storage. This clear separation makes the system easier to maintain.

UC- 6 Password Recovery and Security



Design Principles for UC-6: Password Recovery and Security

High Cohesion Principle: The PasswordRecoveryService manages the password recovery process, which includes validating the reset token and updating the password. This centralization of functionality makes the system more cohesive and easier to maintain.

Expert Does Principle: The PasswordRecoveryService is responsible for password recovery because it has the necessary logic and expertise to validate tokens and securely update passwords.

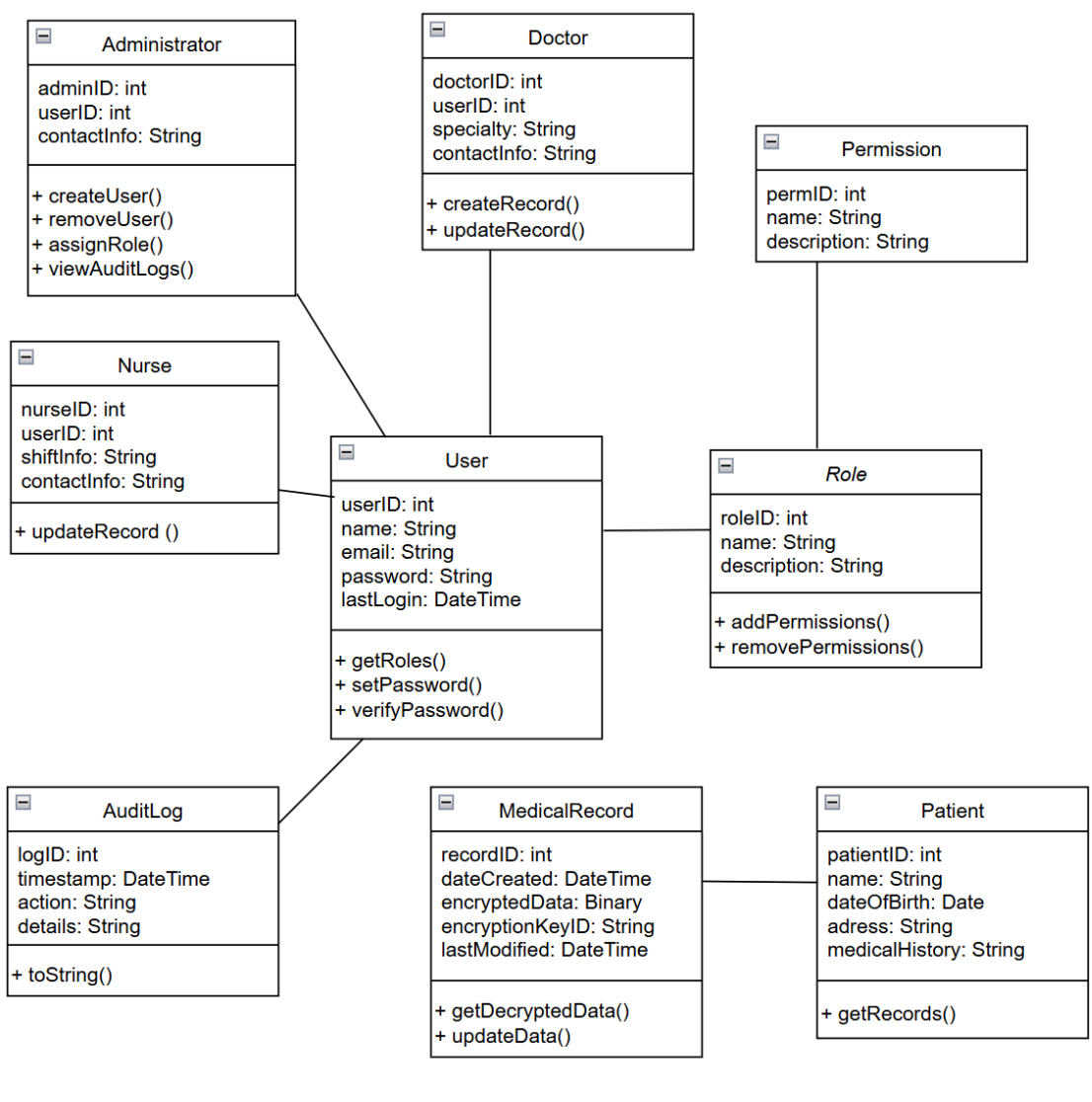
Low Coupling Principle: The WebPage interacts only with the PasswordRecoveryService, and not with the database or other internal services. This decoupling makes the system more flexible and easier to modify.

Single Responsibility Principle : The WebPage focuses on handling the user interface, while the PasswordRecoveryService focuses on the password recovery business logic. The Database handles data storage. Each component is focused on a single responsibility, promoting system modularity.

Secure Design Principle: This use case incorporates secure password recovery methods, using a reset token and ensuring that the password is securely updated in the Database. It reduces the risk of unauthorized access and ensures that only the user with the correct credentials can update their password.

Section 8: Class Diagram and Interface Specification

Class Diagram



Data Types and Operation Signatures

Class: Administrator

Attributes:

- adminID: Int - A unique, specific-length identifier for the administrator as a numerical ID prepended to the user's userID.
- userID: Int - A unique identifier for the administrator user as a numerical ID.
- contactInfo: String - The relevant contact information of the administrator user as a string.

Operations:

+ createUser(): Int - Creates a new user in the system and adds all relevant information to related databases, returning 0 for success and 1 for failure.
+ removeUser(): Int - Removes a user from the system in all relevant databases, returning 0 for success and 1 for failure.
+ assignRole(): Int - Assigns a role to a current user to set security access permissions, returning 0 for success and 1 for failure.
+ viewAuditLogs(): String - Retrieves relevant queried information from the related areas (e.g., user names from the user account database, login data, etc.), returning a string containing all relevant information.

Meaning of Class:

- The meaning and purpose of the "Administrator" class is to perform the needed functions of any administrator in this EMR, whether it be a healthcare administrator, IT manager, or other designated administrator user accounts.

Table:

Administrator
-adminID: Int -userID: Int -contactInfo: String
+createUser(): Int +removeUser(): Int +assignRole(): Int +viewAuditLogs(): String

Class: Doctor

Attributes:

- doctorID: Int - A unique, specific-length identifier for the doctor as a numerical ID prepended to the user's userID.
- userID: Int - A unique identifier for the doctor user as a numerical ID.
- specialty: String - A description of the doctor's designated specialty.

- contactInfo: String - The relevant contact information of the doctor user as a string.

Operations:

+ createRecord(): Int - Creates a new record in the relevant databases (e.g., creates a new medical history record for a given patient), returning 0 for success and 1 for failure.
+ updateRecord(): Int - Updates a current record in its respective database, returning 0 for success and 1 for failure.

Meaning of Class:

- The meaning and purpose of the "Doctor" class is to perform the needed functions of any doctor in this EMR, whether it be creating new patient records or modifying existing records.

Table:

Doctor
-doctorID: Int -userID: Int -specialty: String -contactInfo: String
+createRecord(): Int +updateRecord(): Int

Class: Nurse

Attributes:

- nurseID: Int - A unique, specific-length identifier for the nurse as a numerical ID prepended to the user's userID.
- userID: Int - A unique identifier for the nurse user as a numerical ID.
- shiftInfo: String - Detailed information logged from the nurse for what was performed during a given shift as a string.
- contactInfo: String - The relevant contact information of the doctor user as a string.

Operations:

+ updateRecord(): Int - Updates a current record in its respective database, returning 0 for success and 1 for failure.

Meaning of Class:

- The meaning and purpose of the "Nurse" class is to perform the needed functions of any nurse in this EMR, such as updating current medical records of given patients.

Table:

Nurse
-nurseID: Int -userID: Int -shiftInfo: String -contactInfo: String
+updateRecord(): Int

Class: AuditLog

Attributes:

- logID: Int - A unique identifier for the log as a numerical ID.
- timestamp: DateTime - The current date/time of the audit log and/or of the data being retrieved from the audit as a datetime.
- action: String - Description of the current action request or being performed by the audit log as string data.
- details: String - Details of the audit log and relevant information being retrieved from it.

Operations:

+ toString(): String - Converts any encrypted or other formatted data into a string format, returning a string.

Meaning of Class:

- The meaning and purpose of the “AuditLog” class is to operate any needed audits that ought to be performed within an EMR system, such as gathering user sign-in logs or record modification logs.

Table:

AuditLog
-logID: Int -timestamp: DateTime -action: String -details: String
+toString(): String

Class: Permission

Attributes:

- permID: Int - A unique identifier for the permission as a numerical ID.
- name: String - The display name of a given permission in a simple, readable form for the user as a string.
- description: String - The description of the respective permissions in a clear, concise, and readable form for the user as a string.

Operations:

+ N/A

Meaning:

- The meaning and purpose of the “Permission” class is to utilize any related permissions within the EMR systems, such as permissions set to give access controls to specified users.

Table:

Permission
-permID: Int -name: String -description: String

Class: *Role

Attributes:

- roleID: Int - A unique identifier for the role as a numerical ID.
- name: String - The display name of a given role in a simple, readable form for the user as a string.
- description: String - The description of the respective permissions in a clear, concise, and readable form for the user as a string.

Operations:

+ addPermissions(): Int - Assigns permissions to a given user (e.g., allowing a user to view a record search page but without modifications privilege), returning 0 for success and 1 for failure.
+ removePermissions(): Int - Removes permissions from a current user (e.g., removing a users’ ability to edit records while maintaining the ability to view), returning 0 for success and 1 for failure.

Meaning of Class:

- The meaning and purpose of the “Role” class is to utilize any related roles within the EMR systems, such as predefined roles like doctors, nurses, etc.

Table:

**Role*
-roleID: Int -name: String -description: String
+addPermissions(): Int +removePermissions(): Int

Class: User

Attributes:

- userID: Int - A unique identifier for the user as a numerical ID.
- name: String - The display name of a given user as a string.
- email: String - The email address of a given user as a string.
- password: String - The encrypted password of a given user as a string.
- lastLogin: DateTime - The stored last sign-on date/timestamp of a given user as a datetime datatype.

Operations:

+ getRoles(): String - Retrieves the roles/permissions assigned to a given user to establish the proper viewing environment (e.g., if a doctor signs in then their system should display extra buttons for creating new records), returning string data that contains all relevant information to then be parsed by the program into the relevant data types.
+ setPassword(): Int - Assigns a password to a current user, returning 0 for success and 1 for failure.
+ verifyPassword(): Int - Verifies if a given password matches the respective users' current password on their account, returning 0 for success and 1 for failure.

Meaning of Class:

- The meaning and purpose of the "User" class is to store and utilize any related information related to users the EMR systems, such as the current roles/permissions of a user or what their encrypted password hash is.

Table:

User
-userID: Int -name: String -email: String -password: String -lastLogin: DateTime
+getRoles(): String +setPassword(): Int +verifyPassword(): Int

Class: Patient

Attributes:

- patientID: Int - A unique identifier for the patient user as a numerical ID.
- name: String - The patient-given legal name of a given patient as a string.
- dateOfBirth: Date - The patient-given legal data of birth of a given patient as a date datatype.

- address: String - The patient-given legal current address of a given patient as a string.
- medicalHistory: String - The patient-given and currently-documented medical history and all relevant information of a given patient as an encrypted string.

Operations:

+ getRecords(): String - Retrieves the records of a given patient, returning string data of all related information.

Meaning of Class:

- The meaning and purpose of the “Patient” class is to store and utilize any related information related to patients the EMR systems, such as the current address of a given patient or what their medical history is.

Table:

Patient
-patientID: Int -name: String -dateOfBirth: Date -address: String -medicalHistory: String
+getRecords(): String

Class: MedicalRecord

Attributes:

- recordID: Int - A unique identifier for the record as a numerical ID.
- dateCreated: DateTime - Designated stored date/time of record creation date as a datetime data format.
- encryptedData: Binary - Stored data of a given medical record encrypted in threefish-1024 post-quantum cryptographic methods as binary data for ease-of-use between systems and throughout time.
- encryptionKeyID: String - A small asymmetrical CYSTALS Kyber key used to encrypt/decrypt data stored as a string.
- lastModified: DateTime - Designated last modification stored date/time of a given record as a datetime data format.

Operations:

+ getDecryptedData(): String - Retrieves data and processes decryption of it to display to the user (e.g., obtains encrypted unreadable data from a database through the backend server, then the client/web server runs quick hashes on this to display it to the user), returning string data for further processing.
+ updateData(): Int - Updates/modifies data of a given medical record, returning 0 for success and 1 for failure.

Meaning of Class:

- The meaning and purpose of the “MedicalRecord” class is to store and utilize any related information related to medical records of the EMR systems, such as the encrypted data within it.

Table:

MedicalRecord
-recordID: Int -dateCreated: DateTime -encryptedData: Binary -encryptionKeyID: String -lastModified: DateTime
+getDecryptedData(): String +updateData(): Int

Traceability Matrix

Use Case	User	Admin	Role	Permission	Medical Record	Patient	API	Authentication	Encryption	Web page	PQC
UC-1	X						X	X		X	
UC-2		X	X	X			X			X	
UC-3	X			X	X	X				X	X
UC-4					X				X		
UC-5	X			X	X	X	X			X	
UC-6	X							X		X	

Domain Concept	Derived Classes	Explanation
User	User, Administrator, Doctor, Nurse, Patient	The "User" concept evolved into multiple specialized classes to differentiate access levels and functionalities within the EMR system. Administrators manage the system, while Doctors, Nurses, and Patients interact with medical records based

		on predefined permissions.
Medical Record	MedicalRecord	The "Medical Record" concept directly maps to the MedicalRecord class, which includes patient data storage, timestamps, and encryption metadata for security and compliance.
Role Based Access Control	Role, Permission	RBAC was expanded into three separate entities: Role (user roles like Doctor, Nurse), Permission (granular access rights like "View Record")
Audit Log	AuditLog	The "Audit Log" domain concept maps directly to AuditLog, which records system activities such as logins, data access attempts, and permission changes for compliance tracking.

Section 9: Algorithms and Data Structures

Algorithms

While the entire system in general does not rely on any type of mathematical algorithms in any major way, there is a significant usage of post-quantum cryptographic (PQC) structured algorithms that are based on Bruce Schneier's Threefish-1024 algorithm for bulk data encryption and CRYSTALS Kyber algorithm for key encapsulation. Throughout all operations, such as user authentication, role-based access control, or dealing with database entries, these algorithms will be utilized as the primary means for data security.

To further elaborate on these specific algorithms, we can start with a focus on Bruce Schneier's algorithm that we will rely on. Threefish-1024 is an extremely useful PQC method that has its basis in using 1024-bit symmetrical keys and only simple operations (namely, XOR, addition, and rotation) on 1024-bit blocks. Additionally, it also uses what is called a "tweak value" that, while known, is used for further randomization similar to running the program through a second key, internally. Finally, this algorithm enhances it's hardened security by doing 80 rounds of processing through each data before obtaining the final result.

Shifting over to CRYSTALS Kyber, this is a more simple, though very robust, PQC encryption "algorithm" (more accurately, a "key encapsulation method") designed specifically for creating asymmetrical keys so that a public and private key may exist within our EMR system, thus (when paired with Threefish-1024) achieving a similar security standard as the current industry uses with AES-256. Moreover, this algorithm works by using lattice-based encryption, which is essentially creating points on a graph and randomly assigning them connections to one-another in an unpredictable manner to obtain a generated "key".

Now, aside from the aforementioned post-quantum encryption methods and how it intertwines itself with other aspects of the environment, the EMR system does not rely on any other complex or mathematical algorithms: it uses simple methods for user authentication, role-based access control, and record updates. For user authentication, when a user login the system will hash the password provided and compare it with the hashed password in the database. If they match, access is granted. For role-based access control, once the user is authenticated the system checks the user's role and verifies if that role has the permission to do certain actions. For record updates, the system will compare old vs. new and save only valid changes. All-in-all, these aspects work together well in the system to satisfy the need of algorithms without introducing an intense overcomplexity to the system.

Data Structures

The core of our EMR rests on a relational SQL database that stores everything. It has a users table that has user details, a roles and permission table that define each role (admin, nurse, doctor) and what they can do (view, update, delete records, edit roles etc.), patient and medical records that hold patient data and the corresponding medical data, and a Logs table that captures events like login attempts and modifying patients records.

In addition, to improve performance for common lookups (like permission checks) the system loads each user's role and permissions into a hash table at login, so there will be no need to repeat database queries. Also arrays are used to hold temporary lists like displaying a list of patients in the UI.

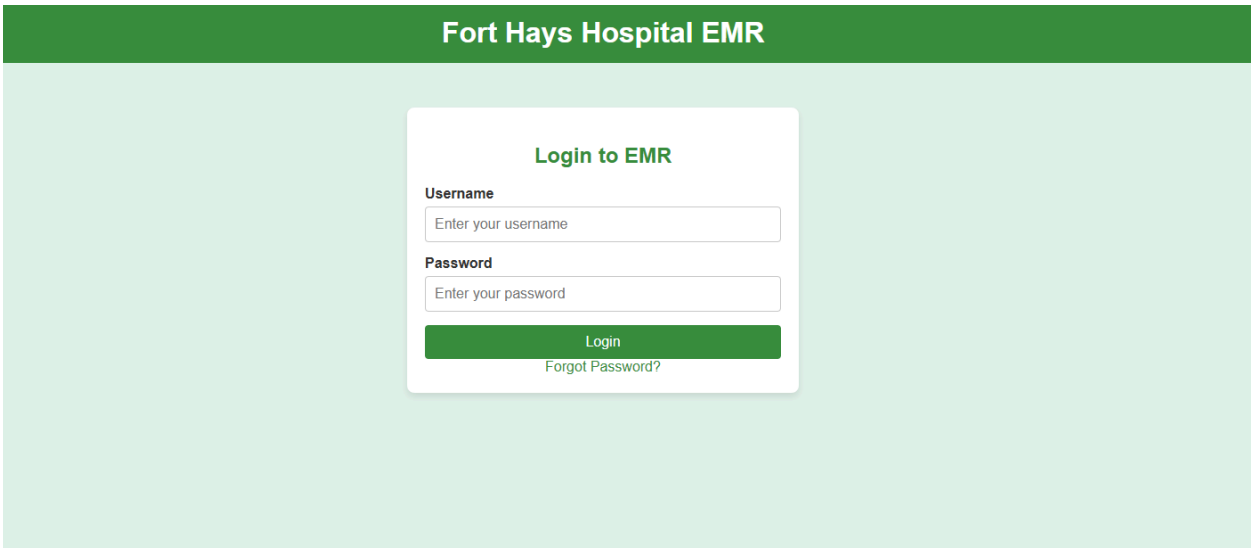
Concurrency

The EMR does not have complex concurrency. One request at a time per user session. This leads to the system operating on a one request per user sessions model, ensuring sequential processing.

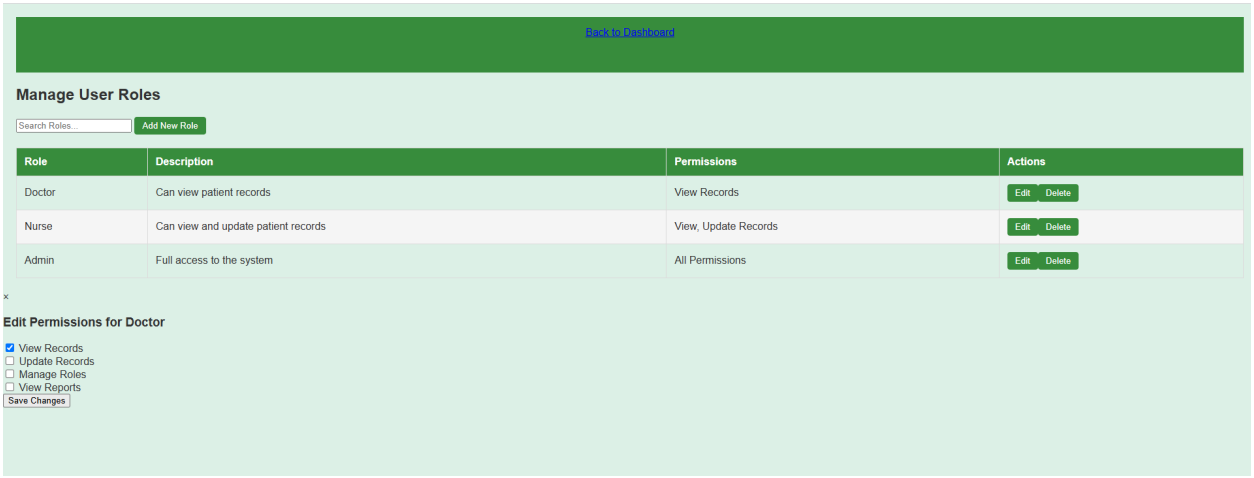
Section 10: User Interface Design and Implementation

Design: A few changes were implemented to the initial mock-ups for the EMR application. A few of the key changes were:

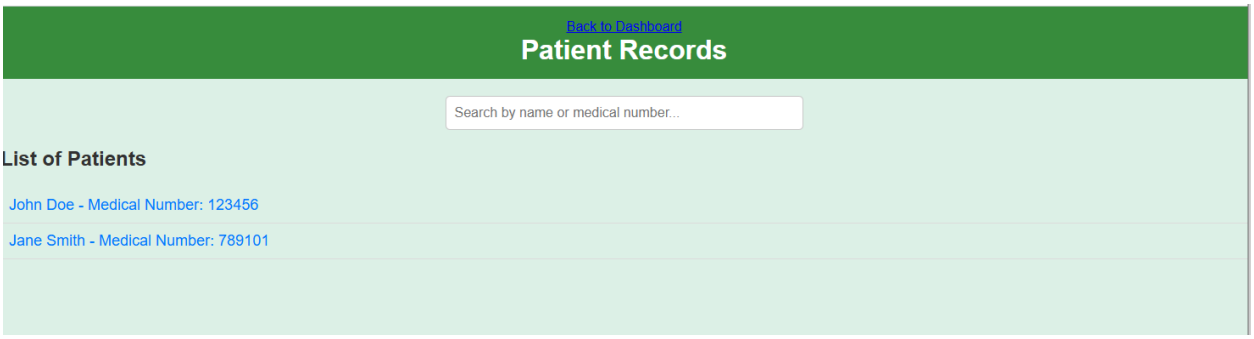
- Creating search bars for the User roles as well as the Patient Records page, to allow the user to be able to navigate to what they need more quickly.
- Allowing the permissions to be viewed at the bottom of the page for the role management.



UC-1 - Keeping the same edits from before to ensure a very straight forward approach. Entering Username and Password for user with a Login button and Forgot Password Link.



UC-2. Updated the Role Management page to allow for the user to search for roles and create a new role. Also allowed the edit of the permissions to be viewed on the bottom of the page to give access to create and remove permissions for that role.



Updated the Patient Records page so that the user may search for the patient by their name or their medical number. The list of patients will be implemented by a database, this is just for a mock up of what the design will look like.

Section 11: Test Designs

Unit Testing

Unit testing will be performed to verify that individual components of the EMR system function correctly. The test cases below focus on ensuring that core system functionalities operate as expected.

Test Case ID	Test Name	Use Case Tested	Description	Expected Result
TC-01	User Authentication	UC-1	Verify login with valid and invalid credentials	Users can log in with correct credentials; error displayed for invalid inputs
TC-02	Role Assignment	UC-2	Ensure administrators can assign and update roles	Roles are assigned correctly and unauthorized users cannot modify roles
TC-03	Medical Record Retrieval	UC-3	Verify patient records can be accessed based on user permissions	Authorized users can retrieve records, unauthorized access is denied or not presented
TC-04	Data Encryption	UC-4	Ensure medical records are encrypted before storage	Encrypted data is stored, and decrypted is possible only with the correct key
TC-05	Audit Log Generation	UC-3, UC-5	Confirm system generates logs for role updates, record access, and log in attempts	Logs correctly tracked user activities
TC-06	Password Reset	UC-6	Ensure users can securely reset passwords through verification	Password reset functionality works as expected

Test Coverage

Test coverage will ensure that the following aspects of the system are thoroughly validated:

- ➢ Functional Testing: Verifying all user interactions with authentication, role management, and medical records.
- ➢ Security Testing: Ensuring unauthorized access is prevented and that encryption mechanisms are intact.
- ➢ Performance Testing: Evaluating response time and efficiency of database queries.
- ➢ Exception Handling: Testing scenarios of failure, such as invalid user credentials and incorrect encryption keys.

Integration Testing Strategy

Integration testing will validate the communication between different system modules, ensuring smooth interoperability.

Planned Integration Tests:

1. Authentication & Role Based Access Control - verify that user roles restrict access to medical records.
2. Medical Record Updates & Encryption - ensure that updating a record also re-encrypts its contents before storage.
3. Audit Log System - Confirm that audit entries are generated when accessing or modifying sensitive data.
4. Session Management & Logout - validate that active sessions expire correctly
5. Database & UI Interaction - ensure that the frontend correctly display stored medical records with proper security checks.

System Testing Strategy

System testing will verify that the entire EMR system meets the non-functional requirements including performance usability, and compliance with industry standards/regulations.

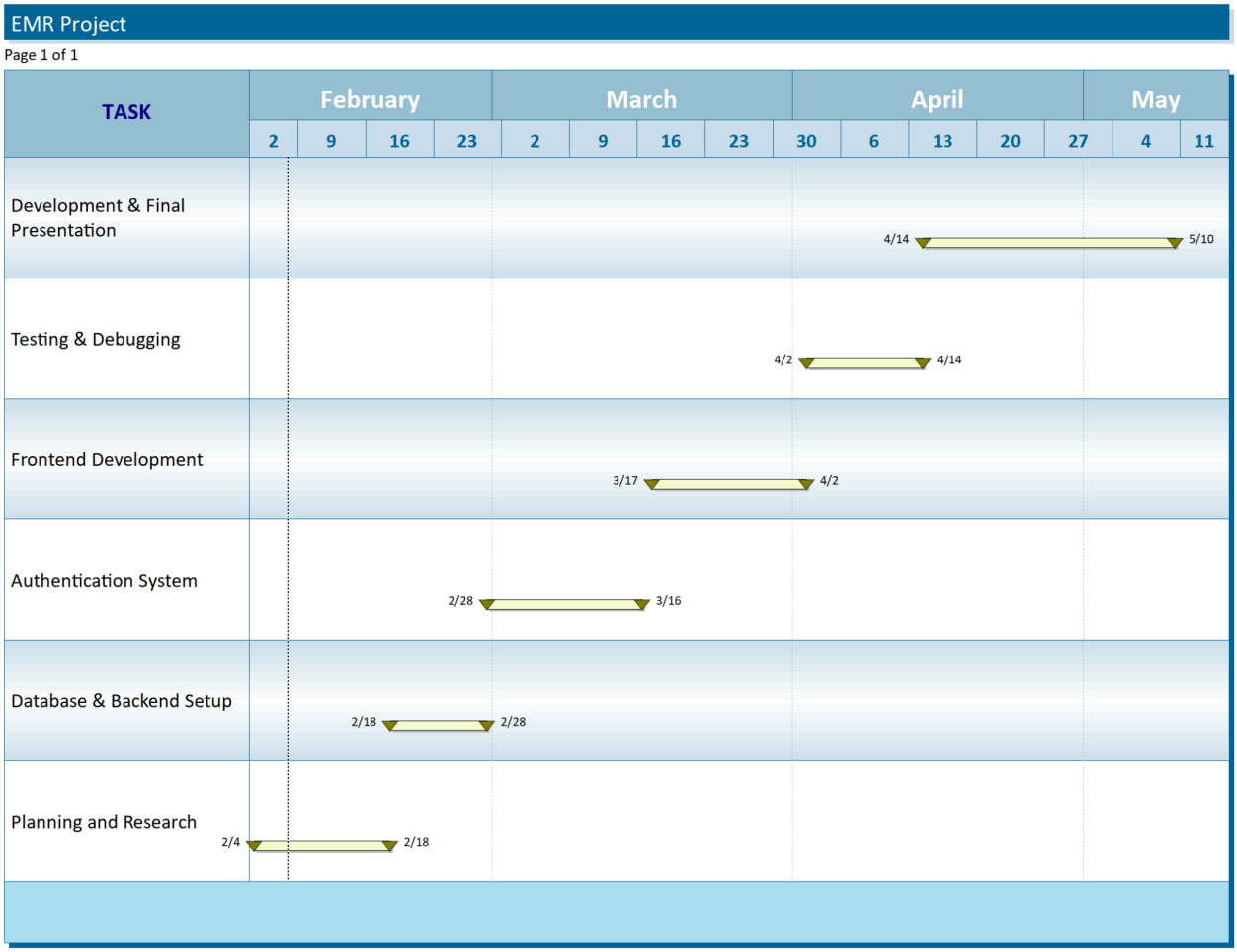
Planned System Tests:

1. Algorithm Testing - validate the encryption/decryption performance under real time conditions. Verify that search and retrieval of patient records operate efficiently.
2. Non-Functional Requirement Testing:
    a. HIPAA Compliance Check: Ensure all data transmission and storage meets standards.
    b. Load Testing: Simulate multiple users accessing the system simultaneously to test scalability.

    c. Data Integrity & Consistency: Verify that database transactions are accurate
3. User Interface Testing:
    a. Confirm responsive UI design across different devices
    b. Ensure error messages and security warnings are properly displayed.

SEE NEXT PAGE →

Section 12: Project Management and Plan of Work



Project Roadmap

February:

- Design and create database and setup backend with encryption (E.g., Foreign key set-up, ensure unique ids, add roles to each person)
- Test each and ensure everything works properly before moving on to implement the authentication system.

March:

- Continue role base login and authentication setup and implementation
- Implement frontend and connection to back-end (Create user interface, connect with backend server, implement CRUD for pulling data to user interface)

April:

- Finish frontend implementation and development
- Test and debug
- Work on final development and presentation

Project Phases

Phase	Task	Start Date	End Date	Team Member
1.Planning & Research	Define Requirements, select encryption method, role-base model	Feb 5th	Feb 18th	All Team members
2. Database & Backend Setup	Design database schema, implement encryption, connect backend to database	Feb 18th	Feb 28th	All Team Member
3. Authentication System	Develop role-based authentication, integrate with backend	Feb 28th	Mar 16th	All team members
4. Frontend Development	Create UI for login, user roles, dashboard integration	Mar 17th	Apr 2nd	All team members
5. Testing & Debugging	Test security, bug fixes, optimize performance	Apr 2nd	Apr 14th	All team members
6. Development & Final Presentation	Final testing, documentation, deployment, prep presentation	Apr 14th	May 10th	All team members

Team Responsibilities

Team Member	Completed Work	Current Task	Future Task
Brandon	- Report Documentation - PQC Base & Continued Research - PQC Reference Research - Determine connection with database and PQC environment	- Finalize research on PQC design - Continue PQC reference research - Design key exchange	- Design & implement PQC
Brittany	-Report Documentation -Frontend Layout	-Research and Design -Report Documentation -Frontend webpage development	- Database - Frontend -Testing
Christoper	- Report Documentation - Research & Design	- Research and Design - Report Documentation	- Database - Front End - Q/A Testing
Riley	- Report Documentation - Database design - Research Database Implementation	- Research and Design - Report Documentation - Implement Database to web page - Test CRUD for Database to endpoint	- Testing - Database - Frontend

Section 13: References

***Electronic Medical Record Systems | Digital Healthcare Research*. (n.d.-b).**
https://digital.ahrq.gov/electronic-medical-record-systems

***Medical Record*. (2006). Sciencedirect. Retrieved February 6, 2025, from**
https://www.sciencedirect.com/topics/medicine-and-dentistry/medical

-record.

SaberiKamarposhti, Morteza, et al. "Post-quantum healthcare: A roadmap for cybersecurity resilience in medical data". *Heliyon*, vol. 10, no. 10. 30 May 2024.
https://www.cell.com/heliyon/fulltext/S2405-8440(24)07437-1

Bos, Joppe, et al. "CRYSTALS – Kyber: a CCA-secure module-lattice-based KEM". *IEEE*, pp. 353-367. Apr. 2018.
https://ieeexplore.ieee.org/document/8406610

Bruce Schneier, et al. "Threefish". AND "The Skein Hash Function Family". 2008-2010.
https://www.schneier.com/academic/skein/threefish/
https://www.schneier.com/wp-content/uploads/2008/10/skein.pdf