Software requirement specification document for
project **Clinicus Clinic System**

Ahmed Ismail, Ahmed Khattab, Ibrahim Labib, Islam Abdelmotaleb, Yassin Bedier

Supervised by:
Dr. Ayman Ezzat and Dr. Raghda Essam
March 23, 2025

1 Introduction

1.1 Purpose of this document

The purpose of this document is to define the Software Requirements Specification (SRS) for the Clinicus Clinic System. This document serves as a reference for all involved in the clinic system's development, including any potential stakeholders, as well as the software engineers responsible for constructing it. The intended audience are the doctors and staff who will use the system for convenience and management, as well as the patients involved who will have the information and prescriptions managed.

1.2 Scope of this document

Clinicus Clinic System is a web-application that provides effective patient management within free medical clinics and healthcare NGOs. There are various modules for user registration, management of doctors and other staff, scheduling appointments, keeping patient medical histories, managing medicine inventories, dispensing prescriptions, and organizing storage. The requirements elicitation team consists of administrators, doctors, diligent staff, and software developers. Constraints, such as a fixed timeline for development and compliance with healthcare regulations, were imposed on the entire process. The system is intended to be scalable, secure, and interoperable with any clinic's existing healthcare infrastructure.

1.3 Overview

The Clinicus Clinic System is a web-based application for the streamlining of healthcare service to free clinics and NGOs. Health professionals can use the system to manage patients' records, monitor medical histories, and schedule appointments or manage a medicinal inventory. The web-based system aims to improve accessibility to services, minimize the administrative workload, and maximize the patient's care by providing a smooth digital interface for healthcare professionals and support staff. Some of the main features of the system are:

1. Patient registration and record management
2. Doctor and staff management
3. Appointment scheduling
4. Medical history tracking
5. Medicine inventory management
6. Handling of prescriptions
7. Departmental organization of doctors

1.4 Business Context

The Clinicus Clinic System is designed to complement the mission of those healthcare NGOs and free clinics that serve the essential medical needs of the underserved. Its sponsor organizations aim to enhance access to health services, improve patient record management, and optimal resource allocation. This is indeed in keeping with the larger goal of offering effective, high-quality healthcare delivery with reduced operational waste with respect to stacking and automation of core administrative functions. The system objectives include:

1. Improve patient care through accurate and timely medical history information.
2. Improve clinic workflow with less paperwork.
3. Have a more effective medicine inventory system.
4. Provide clinicians with a friendly system that can be quite reliable.
5. More effective management of transactions in and out of the system.
6. Ease of use for appointments.

2 General Description

2.1 Product Functions

The main functions of the Clinicus Clinic System are presented in this section. Following are the functions provided by the system:

1. **Patient Registration and Management**: It allows health care providers to register new patients into the system and store personal and medical details of the patients and then update them.
2. **Doctor and Staff Management**: It allows the adding, modifying, and management of doctors, nurses, and administrative staff along with their schedules and assigned roles.
3. **Appointment Scheduling**: It allows patients and doctors to book appointments online, reschedule them (only doctors) or cancel them.
4. **Medical History Tracking**: It could store and retrieve the medical records of patients, having data like past diagnosis and treatments provided.
5. **Medicine Inventory Management**: It takes care of all medical supplies and checks the stock levels to reduce deficiencies.
6. **Prescription Handling**: This handles electronic prescriptions by doctors for their patients.

7. **Departmental Organization**: That categorizes all the various departments doctors may specialize in.

2.2 Similar System Information

Standalone in nature, this system can intermingle with the prevailing healthcare management systems for additional functionalities. Some similar systems are:

1. **OpenMRS**: An open-source electronic medical record application used by developing countries.
2. **FreeMED**: This is a free and open-source electronic health record (EHR) and practice management tool.
3. **Bahmni**: An open-source system designed as a clinic information system in low-resource environments.

This system is specially made for free medical clinics and healthcare NGOs and hence stands apart as a more affordable yet easy-to-use solution.

2.3 User Characteristics

The user base will likely prove to be moderately to very slightly tech savvy in terms of using web applications, and training will increase for maximized use.

- **Doctors**: Medical professionals who are licensed or trained to diagnose and treat patients, with the assumption that they would need some accessibility to medical records and prescriptions for their work.
- **Staff**: Includes storage managers, appointment managers and anyone supervising operational tasks.
- **Patients**: A person seeking healthcare services, or would also be using the system to search for and book an appointment besides which it was used to view their medical records.
- **Administrators**: A group made up of IT staff who are responsible for maintaining and securing the system.

2.4 User Problem Statement

Particularly, free clinics and healthcare NGOs face challenges of being unable to manage patient information efficiently because of:

1. **Record-keeping practices**: Paper-basis transfer leading to the loss or mismanagement of patient data.
2. **Lack of integration**: An inability to share records across departments or clinics.
3. **Inefficient scheduling of appointments**: Long waiting periods and crowded facilities.
4. **Ineffective inventory management**: This results in the shortage or overstocking of medications.
5. **Limited resources**: Insufficient funds to afford commercial solutions in healthcare management.

2.5 User Objectives

This system serves the following goals for its users:

1. **Enhance Patient Care**: Through availing accurate patient's information at the appropriate time.
2. **Improve Operational Efficiency**: Digitization of record keeping and streamlining the activities.
3. **Reduce Administrative Burden**: Automating appointment scheduling and inventory tracking.
4. **Data Security**: Sensitive patient information is protected through role-based access and secure databases.
5. **Reporting and Analytics**: Constant updating of actions through audit logs.

2.6 General Constraints

The development and implementation of this system primarily focuses on the following constraints:

1. **Finances**: Intended for non-profit organizations with limited financial resources.
2. **Regulatory**: Must comply with healthcare data protection legislation, e.g., HIPAA-related.
3. **Internet Dependent**: Should maintain a stable Internet connection for real-time data access.
4. **Scalability**: The system should allow multiple clinics and a varying number of users over time.
5. **Security Requirements**: Encryption and secure authentication mechanisms should be used to protect confidential information.

3 Functional Requirements

- These are the functional requirements of the Clinicus Clinic System, and is what the system must do in order to be operational. The tables are ranked in descending order from highest priority to least important.

Table 1: **Functional Requirement (*FR-01*)**
Priority & Ranking: System Depends On It (#1)

Function Name	User Registration, Authentication, and Management
Description	This function allows users (admins, patients, doctors, and staff) to create an account, log in, and access the system securely. It includes role-based authentication to ensure access control. The way in which each user is typed varies. Admins specifically will be allowed to manage users extensively.

Critically	Essential for ensuring only authorized personnel can access sensitive medical data.
Technical issues	Implementation of a secure authentication protocol and database encryption for user credentials to ensure safety.
Cost and schedule	Requires moderate development time, so likely between 2-3 weeks.
Risks	Weak authentication mechanisms could lead to data breaches, and that risks patients and all in the clinic.
Dependencies with other requirements	Needed for all user interactions within the system.
Pre-Condition	Users must have an internet connection and valid credentials.
Post-Condition	Users gain access based on their role permissions.
Inputs	Username, email, and probably user email.
Outputs	Authentication token, user profile data.

Table 2: **Functional Requirement (*FR-02*)**
Priority & Ranking: Extremely High (#2)

Function Name	Medical History Management
Description	Allows doctors to update and review patient medical history, and for patients to view their own records at the clinic. Records can be altered and updated, and can also be deleted if deemed necessary.
Critically	Critical for effective treatment and patient safety, and keeping proper records if they are returning to the clinic.
Technical issues	Ensuring data integrity and implementing data retrieval optimization.
Cost and schedule	3-4 weeks for integration with patient records.
Risks	Unauthorized access or data loss.
Dependencies with other requirements	Requires patient registration and authentication.
Pre-Condition	Doctors must be logged in to manage the system.
Post-Condition	Updated medical history is stored securely.
Inputs	Patient ID, diagnosis, past treatments, and could include allergies.
Outputs	Updated patient medical history.

Table 3: **Functional Requirement (*FR-03*)**
Priority & Ranking: Extremely High (#3)

Function Name	Prescription Management
Description	Enables doctors to issue prescriptions for patients and supply them with medicine for their diagnosis as well as allowing them to update them based on follow ups.
Critically	Essential for medication tracking and regulatory compliance, and for the patient's health.
Technical issues	Integration with medication and storage databases, and the security measures that are associated with it.
Cost and schedule	4-6 weeks for prescription logic and integration.

Risks	Incorrect prescriptions leading to patient harm.
Dependencies with other requirements	Requires patient and doctor modules.
Pre-Condition	Doctor must be logged in and have patient details.
Post-Condition	Prescription is generated and recorded.
Inputs	Patient ID, medication name, dosage, and all details associated with it.
Outputs	Prescription recorded, storage unit notified.

Table 4: **Functional Requirement (*FR-04*)**
Priority & Ranking: High (#4)

Function Name	Audit Logging and Security
Description	Tracks users and storage activities, ensuring compliance and security in the system. It is also a useful means of analyzing past actions by looking at who did what and in what span of time.
Critically	Essential for monitoring unauthorized access and ensuring data integrity, as well as ensuring that the day-to-day operations of the clinic are moving smoothly.
Technical issues	Implementing log storage and retrieval mechanisms, and also making sure the system is functional because if not user actions will not be recorded,
Cost and schedule	2-3 weeks for the backend logging system.
Risks	Insufficient logging can lead to undetected security breaches, and cost related to storage restocking will not be recorded.
Dependencies with other requirements	Works across all system modules, but depends heavily on the ones where inputted actions take place, like storage change and prescriptions issued.
Pre-Condition	System must be operational, because it is active the entire time the system is up.
Post-Condition	Logs are stored and available for review.
Inputs	User actions, timestamps, storage changes, and any modules that are affected by users.
Outputs	Log records all the actions in the system.

Table 5: **Functional Requirement (*FR-05*)**
Priority & Ranking: High (#5)

Function Name	Appointment Scheduling and Management
Description	Enables patients and doctors to book appointments with each other based on time slots, and allows the option for follow ups or cancellation. Furthermore, this involves staff and how they manage appointments as well.
Critically	Essential for managing patient flow and doctor availability.

Technical issues	Implementing a real-time calendar system with conflict checking is difficult and will need to be re-evaluated.
Cost and schedule	3-5 weeks for scheduling logic and user interface.
Risks	Double-booking errors and mismanagement of doctor schedules.
Dependencies with other requirements	Requires user authentication and patient registration.
Pre-Condition	Patient and doctor profiles must exist.
Post-Condition	Appointment is confirmed and stored in the system.
Inputs	Patient ID, doctor ID, date, time slot.
Outputs	Appointment confirmation with a unique reference number.

Table 6: **Functional Requirement (*FR-06*)**
Priority & Ranking: High (#6)

Function Name	Patient Functionality Management
Description	Records actions of patients when they update their own personal details, review their medical records, and interact with doctors. Patients are also allowed to rate doctors.
Critically	Crucial for maintaining accurate patient records and patients actions altogether.
Technical issues	Need for a relational database to store patient details securely.
Cost and schedule	Requires 3-4 weeks of development and database structuring.
Risks	Data loss or corruption could impact patient care.
Dependencies with other requirements	Relies on user authentication and database access, and the medical history class / records table.
Pre-Condition	Patients must be logged in with appropriate permissions.
Post-Condition	Patient records are stored and accessible, and communications to doctors are established.
Inputs	Same inputs as the user to enter the system.
Outputs	Unique patient ID, confirmation of registration.

Table 7: **Functional Requirement (*FR-07*)**
Priority & Ranking: Medium to High (#7)

Function Name	Medication Storage Management
Description	Tracks stock levels, expiration dates, and restocking alerts for medications, to where staff are required to maintain diligence and act on any storage shortage,
Critically	Vital for ensuring medicine availability and safety.
Technical issues	Implementing real-time stock updates.
Cost and schedule	3-5 weeks for database structuring and audit system.
Risks	Stock depletion leading to unavailability of critical medications.
Dependencies with other requirements	Requires prescription, doctor type responsibilities, and staff coordination.

Pre-Condition	Staff must be logged in to manage storage.
Post-Condition	Medication stock is updated.
Inputs	Medicine ID and the stock quantity are required.
Outputs	Updated storage status and updated in the audit logs.

Table 8: **Functional Requirement (*FR-08*)**
Priority & Ranking: Medium to High (#8)

Function Name	Doctor Departmental Management
Description	Organizes and manages different clinic departments, assigning doctors accordingly based on their qualifications and what field they specialized in.
Critically	Helps in streamlining clinic workflow, the proper classification of doctors, and department allocation for storage access.
Technical issues	Hierarchical role-based department management, since everything descends from the doctor class and different parts of the clinic have to be allocated to different doctor types.
Cost and schedule	3-4 weeks for role-based access and department structuring.
Risks	Misallocation of storage access, so if medication that is not legally accessible to a doctor is not accounted for, an inexperienced doctor could accidently recommend a dangerous diagnosis.
Dependencies with other requirements	Requires doctor management and attributes from its class.
Pre-Condition	Doctors must be assigned to departments to exist and they need the qualifications for it too.
Post-Condition	Department assignments are updated and they are now under that specialization.
Inputs	Doctor type name, assigned doctors.
Outputs	Updated department structure.

4 Interface Requirements

This section describes how the Clinicus Clinic System interfaces with users, hardware, software, and other external systems. It defines the expected user interactions, graphical and command-line interfaces, APIs, and hardware/software communications.

4.1 User Interfaces

The system has a user-friendly GUI designed for different user roles: patients, doctors, staff, and administrators.

Login Screen	Provides user authentication using email and password. Includes role-based access control (RBAC).
Dashboard	Displays personalized views based on the user role. Patients see appointments, prescriptions, and medical history, while doctors see assigned patients, schedules, request medicine, and offer prescriptions.
Appointment Scheduling	Allows patients to book appointments, view availability, and receive confirmations. Doctors and staff can manage appointments. Admins can manipulate audit logs and manage all previous types of users.
Patient Management	Admin can register new patients, update records, and assign doctors. Doctors can access and update patient history.
Medical History	Doctors can add, view, and update patient medical history. Patients can only view their records.
Prescription Handling	Doctors can generate and manage prescriptions for patients, which connects to the medicine and storage classes.
Medical Storage	Staff can track stock levels on medicine and manage when the clinic needs to order more.
Audit Logs	Admins can track all user actions for security and compliance purposes, and changes in storage are recorded here.

4.2 Software Interfaces

The Clinicus Clinic System integrates with various software platforms to extend its functionality. The system's database is managed using MySQL, providing structured data storage and fast retrieval of patient records, appointments, and prescriptions. All data and information is stored on an online phpMyAdmin database that manages everything. User credentials and actions of any type are safely stored on it and only high ranking members of the software engineer team have access to it.

5 Performance Requirements

The Clinicus Clinic System, which has been built using PHP, HTML, and a MySQL database that has been managed via phpMyAdmin, should be fast and memory efficient enough for smooth operation. The user dashboard should load within a maximum two - second limit and any retrieval of a patient's record, appointment, and prescription should also take not more than five seconds. Query execution for medical history must not exceed 1–2 seconds so doctors and staff can access critical information without. The MySQL database must handle over 1000 queries per second with transactions completing in less than 100ms under normal conditions. To ensure

performance, indexing and query optimization techniques must be applied, avoiding any performance bottlenecks. The system should efficiently serve a maximum of 100 concurrent users without performance degradation. PHP execution time per script should not exceed 5 seconds-which would prevent slowdowns in the system-and OPcache should be enabled for caching mechanisms to improve performance in PHP scripts. The system logs have to be rotated automatically every seven days so that excessive memory utilization is avoided while the log for security purposes does not exceed 5MB daily. This will keep the system fast as well as responsive and manage the increase of user loads adeptly.

6 Design Constraints

The Clinicus Clinic System must adhere to several design constraints to ensure compatibility, scalability, and maintainability. The system is built using PHP and HTML, with MySQL as the database, managed via phpMyAdmin, which imposes limitations on database performance, query optimization, and storage capacity. The design must comply with web development best practices, ensuring secure coding standards to protect patient data.

6.1 Standards Compliance

The system must follow HIPAA (Health Insurance Portability and Accountability Act) guidelines for data privacy and security where applicable. Password management should implement hashing for stored passwords.

6.2 Hardware Limitations

Since the system is intended to be deployed on a web server, it must operate efficiently on standard hosting environments with at least 2 CPU cores, 4GB RAM, and 50GB SSD storage. The system should be optimized for shared hosting environments while allowing scalability for dedicated or cloud-based hosting solutions.

6.3 Development and Deployment Constraints

The system will use PHP 8.0 or later and MySQL 5.7 or later, limiting compatibility with outdated versions. It must also be designed for cross-browser compatibility, ensuring full functionality on Google Chrome, Mozilla Firefox, Microsoft Edge, and Safari.

6.4 Scalability and Maintainability

The system must be modular, allowing new features to be integrated without disrupting existing functionalities. Code documentation should be provided for easy maintenance, and automated backups must be implemented for database recovery in case of system failure. The appointment scheduling and patient record modules should be optimized for large-scale data handling, ensuring no significant delays as the database grows.

7 Non-functional Requirements

The Clinicus Clinic System must meet several non-functional attributes to ensure security, reliability, maintainability, and efficiency. These attributes define the system's overall quality, usability, and long-term performance.

7.1 Security (*NF-R1*) | **Priority:** High

The medical records are sensitive information, so security becomes a major priority. The system should have role-based access control (RBAC) which allows admins, patients, doctors, and staff members to access only portions to which they are allowed. Data traffic must be encrypted towards anything, while patient information must be secured in hashed and salted passwords used for authentication. Periodically, security audits and vulnerability testing should be done to avoid risk such as SQL injection attacks.

7.2 Reliability & Recovery (*NF-R2*) | **Priority:** High

The uptime of the system must be maintained at 99.9%, so that it will always be available to users. Backups of the database must be carried out automatically every now and then so that data would never be lost. The system should have a failover mechanism in case of server unavailability. Error handling and logging mechanisms should enable tracking of issues to facilitate a speedy resolution.

7.3 Maintainability (NF-R3) | **Priority:** Medium

The system shall be laid out in a modular programming style to incorporate possible future improvements and modifications without disturbing the working of any currently existing functionalities. Appropriate code documentation must be kept as part of software maintenance activities. Using well-structured procedural PHP will help maintain the system.

7.4 Portability (NF-R4) | **Priority:** Medium

The system should be capable of being deployable on various hosting environments, including cloud and on-premise servers. The system should run efficiently on Windows, Linux, and macOS-based environments. The front-end should be responsive and cross-browser compatible, working on desktops and laptops.

7.5 Extensibility (NF-R5) | **Priority:** Medium

The framework should soon enable any further improvements, like AI-powered diagnostics and automatic prescription handling. The structure of the database should be sufficiently flexible to easily extend the existing modules, relationships, and features without requiring large-scale restructuring.

7.6 Reusability (NF-R6) | **Priority:** Low

Reusable modules have been developed for core components like user authentication, appointment scheduling, and management of medical history. Thus, such modules can easily find integration into other healthcare-related applications or systems developed for current programs.

7.7 Resource Utilization (NF-R7) | **Priority:** High

The system should be optimized for minimum CPU and memory usage to run smoothly on its standard hosting environments with 4GB RAM and a dual-core processor. The database should use efficient indexing and optimized queries to handle large volumes of patient records and appointments without any degree of slowdown.

7.8 UI Accessibility & Serviceability (NF-R8) | **Priority:** Medium

Administrative dashboards should allow easy management of users and troubleshooting. The system has to contain logging and alerting mechanisms that indicate probable failures or unauthorized access attempts or that provide warning on performance issues. The user interface should also be simple and easy to use for those lacking in advanced computer knowledge.

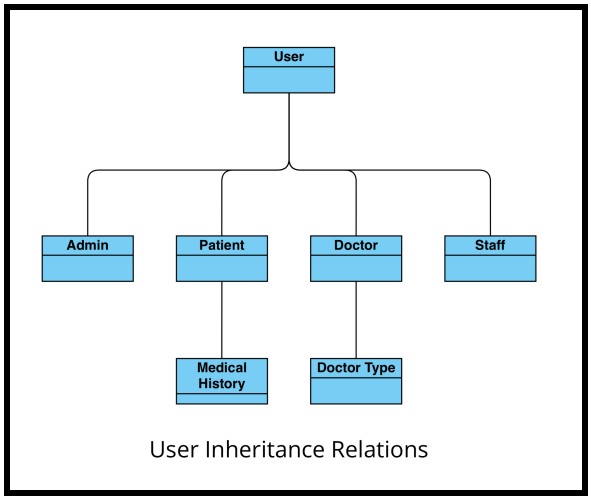
7.9 User Activity Logging (NF-R9) | **Priority:** High

The system shall log user activity to monitor and audit access to sensitive data, and this extends on security since it will keep track of all user actions. Proper user activity logging will also be critical for analyzing user interactions, commonly used medication and doctor performance.

8 Preliminary Object-Oriented Domain Analysis

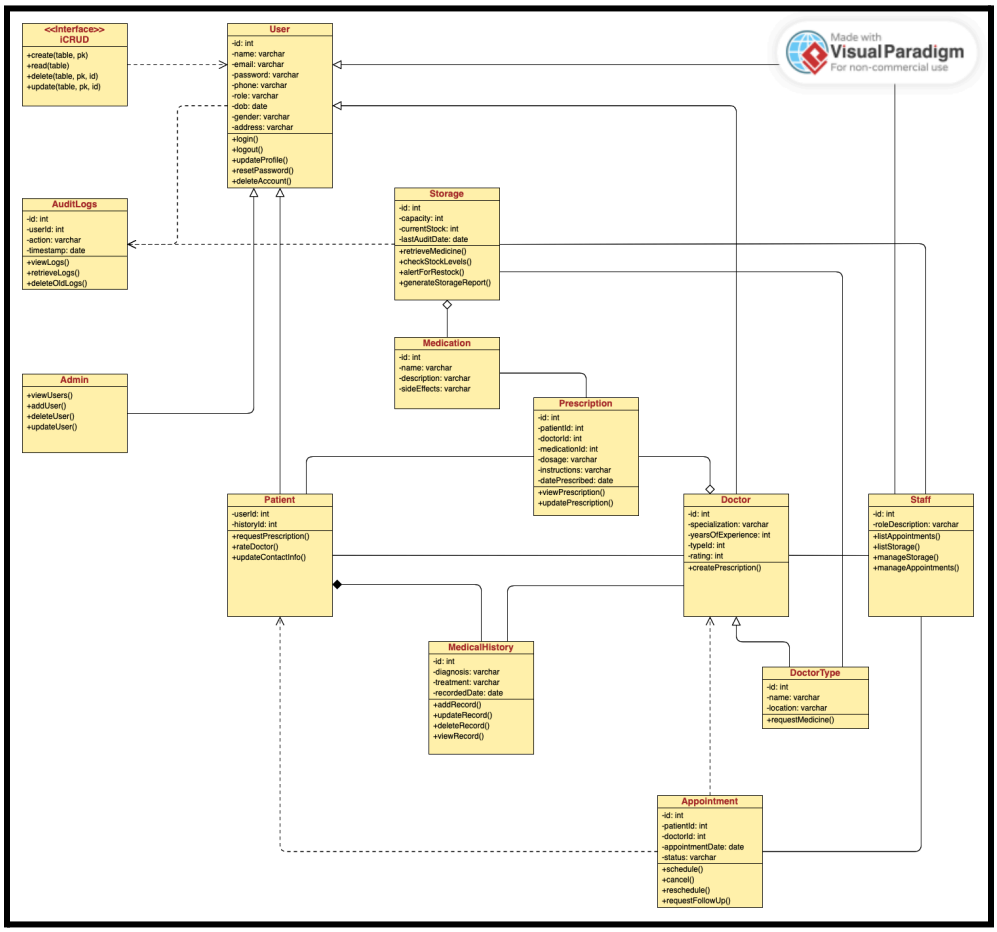
This section presents a list of the fundamental objects that must be modeled within the system to satisfy its requirements. The purpose is to provide an alternative, ”structural” view on the requirements stated above and how they might be satisfied in the system. A primitive class diagram to be delivered.

8.1 Inheritance Relationships



User Inheritance Relations

8.2 Class Descriptions



8.2.1 User

Class Name	User (Abstract)

Superclass	None
Subclasses	Patient, Doctor, Staff
Purpose	Represents any user interacting with the system.
Collaborations	AuditLogs (records user actions) Appointment (patients book, doctors schedule) Prescription (doctors issue)
Attributes	id (int, primary key) name (varchar) password (varchar) email (varchar) phone (varchar) role (varchar) dob (date) gender (varchar) address (varchar)
Operations	login() logout() updateProfile() resetPassword() deleteAccount()
Constraints	Username must be unique. Passwords must be a certain length. Email format validation required. All user identification fields must be filled.

8.2.2 Admin

Class Name	Admin (Abstract)
Superclass	User
Subclasses	None
Purpose	Allows administrators or high ranking software engineers to manage logs and users.
Collaborations	User (is a type of user classification)
Attributes	None
Operations	viewUsers() updateUsers() createUsers() deleteUsers()
Constraints	Admins only have access to managing users and logs, but no action that relates to the day-to-day operations of the clinic.

8.2.3 Patient

Class Name	Patient (Abstract)
Superclass	User
Subclasses	MedicalHistory
Purpose	Represents a registered patient receiving medical services.
Collaborations	User (descends from it to exist) Appointment (books appointments) Prescription (receives prescriptions) MedicalHistory (stores patient history) Doctor (goes to the doctor for a checkup)
Attributes	userId (int, primary key, foreign key from User) historyId (int, foreign key)
Operations	requestPrescription() rateDoctor() updateContactInfo()
Constraints	Must have a valid medical history record.

8.2.4 Doctor

Class Name	Doctor (Abstract)
Superclass	User
Subclasses	DoctorType
Purpose	Represents a medical professional treating patients and offering prescriptions.
Collaborations	User (descends from it to exist) DoctorType (assigned to a doctor specialization) Appointment (manages patient appointments) Prescription (writes prescriptions for patients) MedicalHistory (updates patient history) Staff (assist doctors and refers patients to them) Patient (interact with them and diagnose them)
Attributes	id (int, primary key, foreign key from User) specialization (varchar) yearsOfExperience (int) typeId (int, foreign key) rating (int)
Operations	createPrescription()
Constraints	Must belong to a doctor type.

8.2.5 Staff

Class Name	Staff (Concrete)

Superclass	User
Subclasses	None
Purpose	Represents administrative or medical staff, and also those responsible for storage management.
Collaborations	User (descends from it to exist) Storage (they are responsible for managing medicine stocks) Appointment (staff that manages appointments at the front desk)
Attributes	id (int, primary key, foreign key from User) roleDescription (varchar)
Operations	listAppointments() listStorage() manageStorage() manageAppointments()
Constraints	Besides mandatory user requirements, there are none notable.

8.2.6 Appointment

Class Name	Appointment (Concrete)
Superclass	None
Subclasses	None
Purpose	Represents an appointment between a patient and a doctor.
Collaborations	Patient (books appointment) Doctor (conducts appointment) Staff (manages when patients enter the door)
Attributes	id (int, primary key) patientId (int, foreign key) doctorId (int, foreign key) appointmentDate (date) status (varchar: "Scheduled", "Completed", "Canceled")
Operations	schedule() cancel() reschedule()
Constraints	Cannot be booked for a past date.

8.2.7 Medical History

Class Name	MedicalHistory (Concrete)
Superclass	Patient
Subclasses	None
Purpose	Stores medical history of a patient.
Collaborations	Patient (linked to medical records)

	Doctor (updates medical history)
Attributes	id (int, primary key) diagnosis (varchar) treatment (varchar) recordedDate (date)
Operations	addRecord() updateRecord() deleteRecord() viewRecord()
Constraints	Cannot be deleted if referenced by Prescription.

8.2.8 Medication

Class Name	Medication (Concrete)
Superclass	None
Subclasses	None
Purpose	Represents a medication prescribed to a patient.
Collaborations	Prescription (includes medication) Storage (must be stored in storage, and is accessed from there)
Attributes	id (int, primary key) name (varchar) description (varchar) sideEffects (varchar)
Operations	None
Constraints	Cannot be deleted if referenced by Prescription.

8.2.9 Doctor Type

Class Name	DoctorType (Concrete)
Superclass	Doctor
Subclasses	None
Purpose	Represents a department in the clinic doctors are assigned to.
Collaborations	Doctor (assigned to a department) Storage (only qualified doctors can prescribe specific medicines from the storage)
Attributes	id (int, primary key) name (varchar) location (varchar)
Operations	requestMedicine()
Constraints	Must have at least one doctor and there must be at least two doctor types.

8.2.10 Prescription

Class Name	Prescription (Concrete)
Superclass	None
Subclasses	None
Purpose	Represents a doctor's prescription / diagnosis for a patient and the medication they must take.
Collaborations	Doctor (issues prescription) Patient (receives prescription) Medication (included in prescription)
Attributes	id (int, primary key) patientId (int, foreign key) doctorId (int, foreign key) medicationId (int) dosage (varchar) instructions (varchar) datePrescribed (date)
Operations	viewPrescription() updatePrescription()
Constraints	Can only be issued by Doctors.

8.2.11 Audit Logs

Class Name	AuditLogs (Concrete)
Superclass	None
Subclasses	None
Purpose	Logs system activity from both users and medication.
Collaborations	Users (all user action is stored here and it is frequently checked by admins) Storage (all medicine stock changes are recorded here)
Attributes	id (int, primary key) userId (int, foreign key) action (varchar) timestamp (date)
Operations	viewLogs() retrieveLogs() deleteOldLogs(0
Constraints	Only manageable in its entirety by an admin.

8.2.12 Storage

Class Name	Storage (Concrete)

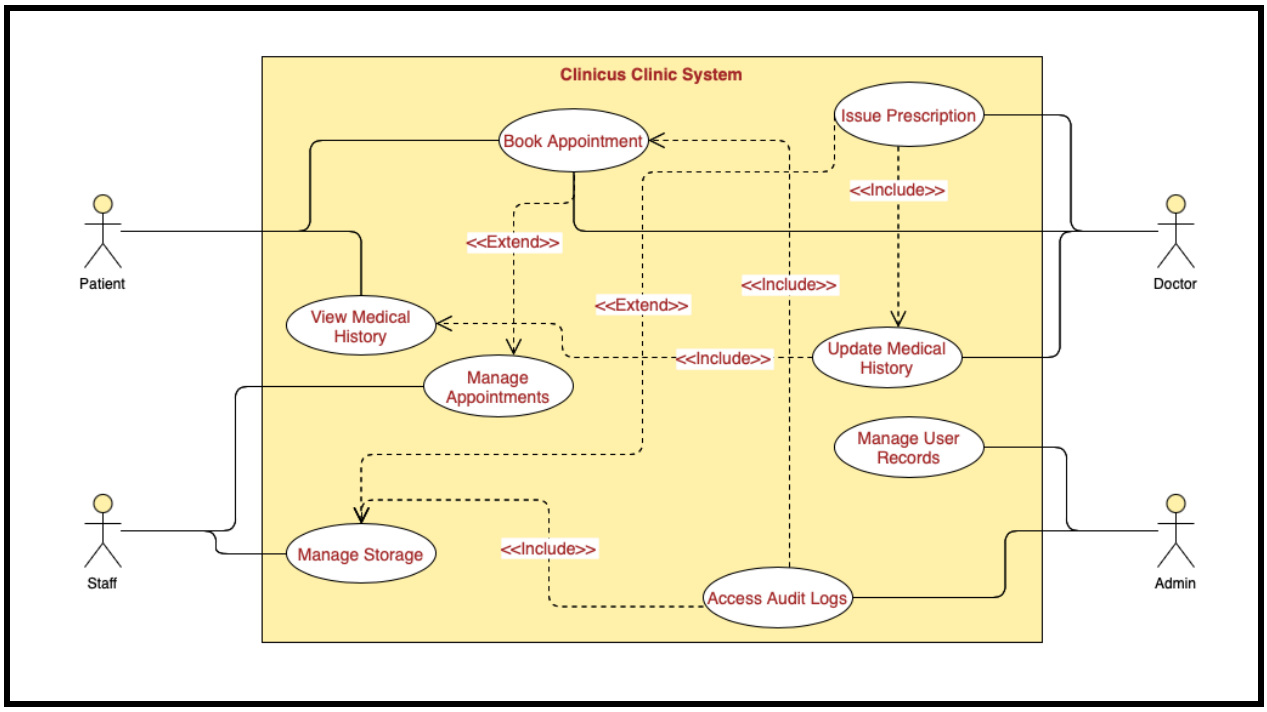
Superclass	None
Subclasses	None
Purpose	Management of medicine stocks.
Collaborations	AuditLogs (all changes in stock are record in the audit logs) Staff (frequently restock it and are responsible for its management) Medication (aggregates from storage and is all stored there) DoctorType (specialized doctors can only request medicine based on their qualifications)
Attributes	id (int, primary key) capacity (int, foreign key) currentStock (varchar) lastAuditDate (date)
Operations	retrieveMedicine() checkStockLevels() alertForRestock() generateStroageReport()
Constraints	Only manageable by staff, and must be restocked consistently.

8.2.13 <<Interface>> iCRUD

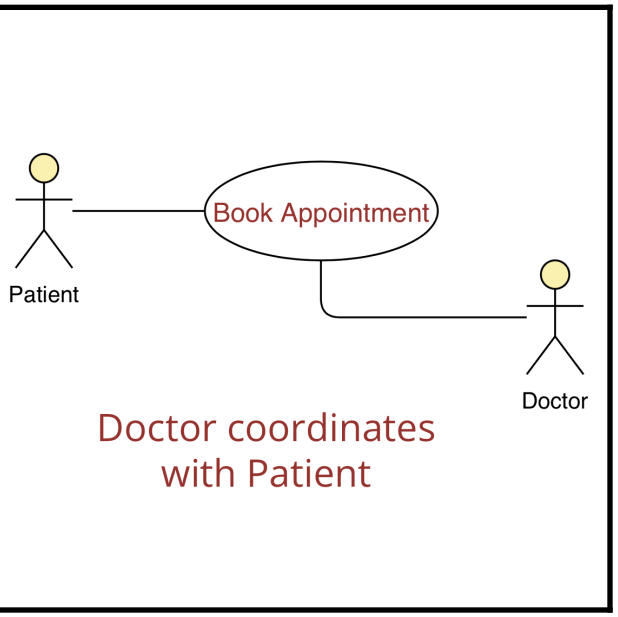
Class Name	Interface Class (Abstract)
Superclass	None
Subclasses	None
Purpose	What is responsible for polymorphism and CRUD operations.
Collaborations	User (users are responsible for CRUD operations done through the system)
Attributes	None
Operations	create(table, pk) read(table) delete(table, pk, id) update(table, pk, id)
Constraints	The system will not operate without an interface class, so the system is constrained if this is not operable.

9 Operational Scenarios (Use-Case Diagram)

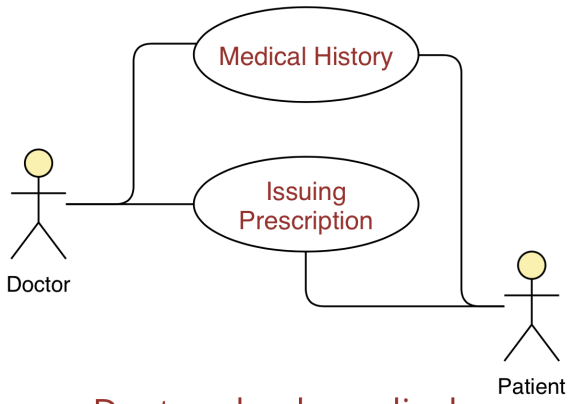
Use-Case Diagram



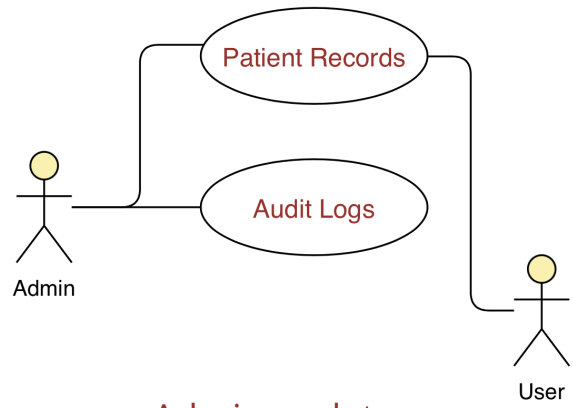
9.1 Scenario 1

Use-Case: Patient Booking an Appointment	
Actor: Patient	
Steps: 1. The patient logs into the system. 2. They navigate to the "Appointments" section. 3. They select a doctor and an available time slot. 4. The system confirms the appointment and updates records. 5. The patient receives a confirmation message.	

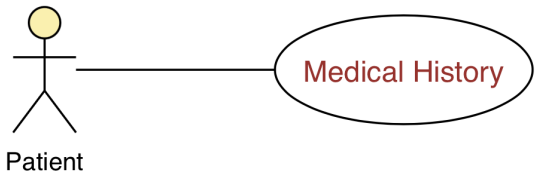
9.2 Scenario 2

Use-Case: Doctor Issuing a Prescription	
Actor: Doctor	Doctor check medical history and prescribes Patient
Steps: 1. The doctor logs into the system. 2. They access the patient's medical history. 3. The doctor prescribes medication and dosage. 4. The system records the prescription and links it to the patient. 5. The patient is notified of the new prescription.	

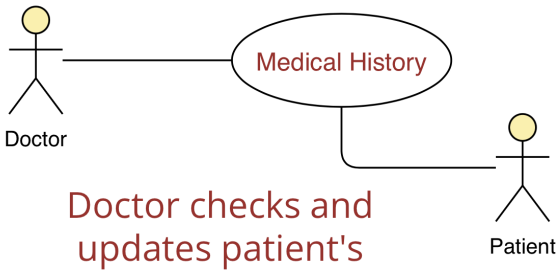
9.3 Scenario 3

Use-Case: Admin Managing Patient Records	
Actor: Admin	Admin updates patient records, and audit logs react
Steps: 1. An admin logs into the system. 2. They navigate to the "Patient Records" section. 3. They update or verify patient information. 4. Changes are saved, and an audit log is recorded.	

9.4 Scenario 4

Use-Case: Patient Viewing Medical History	
Actor: Patient	Patient checks their own medical history
Steps: 1. The patient logs into the system. 2. They navigate to the "Medical History" section. 3. The system retrieves the patient's records from the database. 4. The patient views past diagnoses, treatments, and prescriptions.	

9.5 Scenario 5

Use-Case: Doctor Updating Medical History	
Actor: Doctor	Doctor checks and updates patient's medical history
Steps: 1. The doctor logs into the system. 2. They search for a specific patient's record. 3. The system retrieves the patient's medical history. 4. The doctor adds a new diagnosis and treatment details. 5. The system updates the database and saves changes.	

10 Preliminary Schedule Adjusted

Project Phases and Timeline

Phase	Start Date	End Date	Tasks
Requirements Analysis	March 7	March 18	Finalize system requirements and use cases

System Design	March 18	March 19	Create system architecture and database schema
Development Phase 1	March 19	March 23	First part of development up to seven of the classes having CRUD implementation, and dashboards.
Development Phase 2	March 23	May 10	Implement all of the classes and CRUD operations to where the full system can be tested.
Testing and Debugging	May 10	May 20	Perform unit and integration testing
Final Review & Adjustments	May 20	June 14	Gather feedback and make final modifications

11 Preliminary Budget Adjusted

- At the current moment, a budget has not been considered or even hypothetically (especially considering we created Clinicus ourselves), so this section will be left empty.

12 References

- MSA CS Software requirement specification document, www.overleaf.com/latex/templates/msa-cs-software-requirement-specification-document
- "DHTMLX - Hospital Management Dashboard." Dhtmlx.com, 2025, dhtmlx.com/docs/products/demoApps/dhtmlxHospital/#patients.
- PHP & MySQL Best Practices for Secure Web Applications
- phpMyAdmin, www.phpmyadmin.net
- Visual Paradigm Online, www.visual-paradigm.com