**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. These functions are detailed further in Section 4 (Functional Requirements). 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:

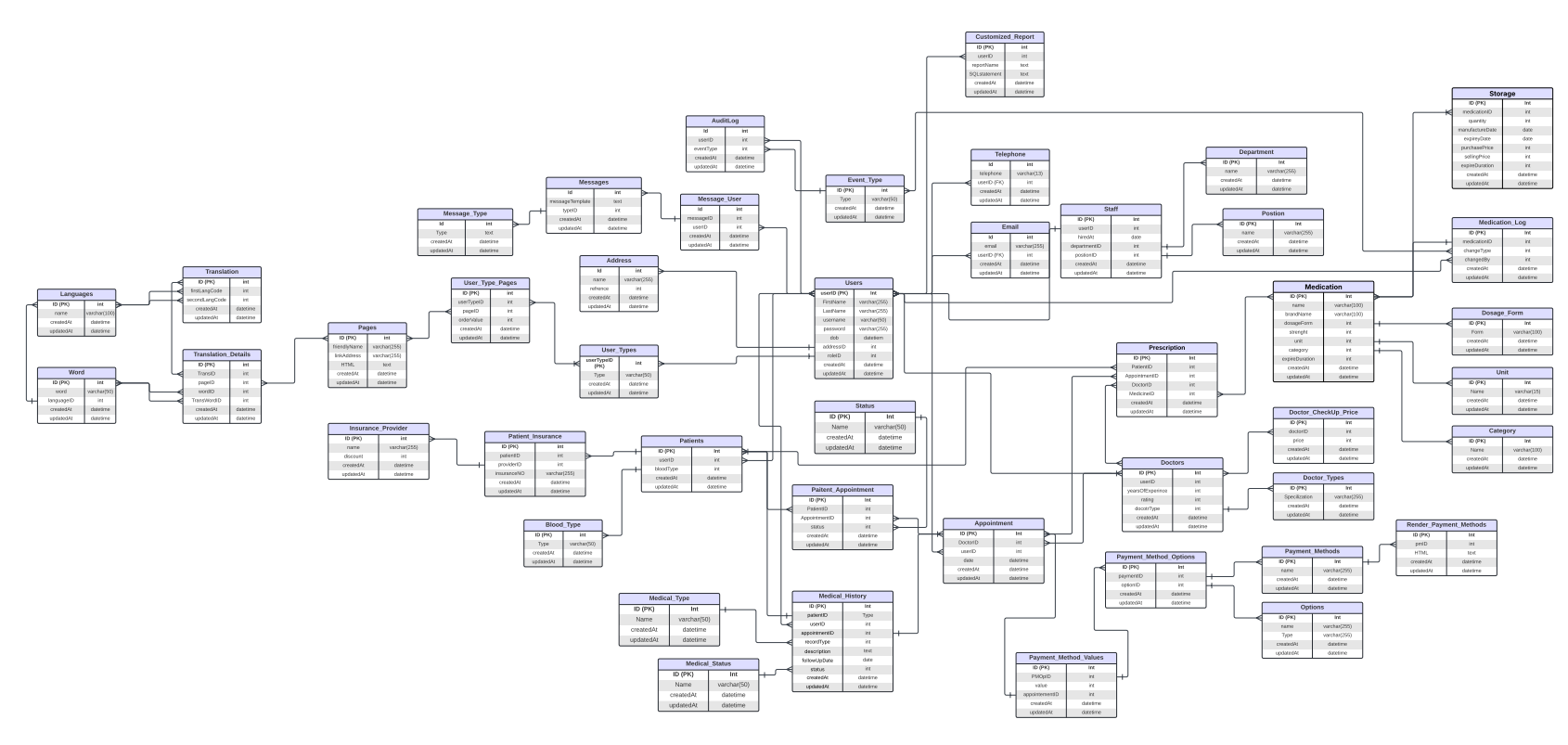
1. **Finances**: Intended for non-profit organizations with limited financial resources.
2. **Regulatory**: Must comply with healthcare data protection legislation, e.g., HIPAA-related.
3. **Internet Dependent**: Should maintain a stable Internet connection for real-time data access.
4. **Scalability**: 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 Data Model and Structure**

This section describes the underlying data structure of the Clinicus Clinic System, including the database schema and the list of tables.

**3.1 Database Schema**

The database schema for the Clinicus Clinic System is represented by the following Entity-Relationship Diagram (ERD):



Key entities in the schema include Users, Patients, Doctors, Staff, Appointments, Medical History, Medications, Prescriptions, Storage, and Audit Logs, among others.

**3.2 List of Tables**

The Clinicus Clinic System database consists of the following tables:

* User
* Log\_User
* UserType
* Admin
* Patient
* AuditLogs
* Storage
* Log\_Storage
* Medication
* Log\_Medication
* Prescription
* Log\_Prescription
* Doctor
* DoctorType
* Staff
* Appointment
* Medical\_History
* Log\_Medical\_History
* User\_type\_pages
* Pages
* Languages
* Translation
* Word
* Translation\_details
* customized\_report
* Files
* Theme
* Message\_user
* Messages
* Payment\_methods
* Payment\_methods\_options
* Options
* Payment\_methods\_value

**4 Functional Requirements**

This section details the functional requirements of the Clinicus Clinic System, describing specific actions the system must perform. These requirements are broken down into granular functions for clarity. The priority ranking indicates the importance of the requirement, with #1 being the highest priority.

**4.1 User Management and Authentication**

**Functional Requirement (*FR-01.1*)**

**Priority & Ranking**: System Depends On It (#1)

|  |  |
| --- | --- |
| **Function Name** | User Registration |
| **Description** | Allows new users (Patient, Doctor, Staff, Admin) to create an account in the system. |
| **Critically** | Essential for system access. |
| **Technical issues** | Implementing secure password hashing and email validation. |
| **Cost and schedule** | 1 week |
| **Risks** | Weak security could lead to unauthorized access. |
| **Dependencies with other requirements** | Needed for all user interactions within the system. |
| **Pre-Condition** | User has internet access. |
| **Post-Condition** | User account created and pending activation (if applicable). |
| **Inputs** | Username, email, password, role, other profile details. |
| **Outputs** | User ID, confirmation message. |

**Functional Requirement (*FR-01.2*)**

**Priority & Ranking**: System Depends On It (#1)

|  |  |
| --- | --- |
| **Function Name** | User Login |
| **Description** | Allows registered users to securely log in to the system using their credentials. |
| **Critically** | Essential for accessing system features. |
| **Technical issues** | Implementing a secure authentication protocol (e.g., session management, tokens). |
| **Cost and schedule** | 1 week |
| **Risks** | Unauthorized access if authentication is weak. |
| **Dependencies with other requirements** | Requires User Registration. |
| **Pre-Condition** | User has valid credentials and internet access. |
| **Post-Condition** | User is authenticated and redirected to their dashboard. |
| **Inputs** | Username/Email, Password. |
| **Outputs** | Authentication token, user profile data. |

**Functional Requirement (*FR-01.3*)**

**Priority & Ranking**: System Depends On It (#1)

|  |  |
| --- | --- |
| **Function Name** | User Logout |
| **Description** | Allows a logged-in user to securely end their session. |
| **Critically** | Important for security and privacy. |
| **Technical issues** | Ensuring proper session termination on the server side. |
| **Cost and schedule** | 0.5 week |
| **Risks** | Session hijacking if not implemented correctly. |
| **Dependencies with other requirements** | Requires User Login. |
| **Pre-Condition** | User is logged in. |
| **Post-Condition** | User session is terminated. |
| **Inputs** | None |
| **Outputs** | Confirmation of logout. |

**Functional Requirement (*FR-01.4*)**

**Priority & Ranking**: Extremely High (#2)

|  |  |
| --- | --- |
| **Function Name** | User Profile Viewing |
| **Description** | Allows a user to view their own profile information. |
| **Critically** | Important for users to verify their details. |
| **Technical issues** | Securely retrieving user data from the database based on their authenticated session. |
| **Cost and schedule** | 0.5 week |
| **Risks** | Exposure of sensitive data if access control is flawed. |
| **Dependencies with other requirements** | Requires User Login. |
| **Pre-Condition** | User is logged in. |
| **Post-Condition** | User profile information is displayed. |
| **Inputs** | User ID (implicit from session). |
| **Outputs** | User profile data. |

**Functional Requirement (*FR-01.5*)**

**Priority & Ranking**: Extremely High (#2)

|  |  |
| --- | --- |
| **Function Name** | User Profile Updating |
| **Description** | Allows a user to update their own profile information (e.g., address, phone number). |
| **Critically** | Important for maintaining accurate user data. |
| **Technical issues** | Implementing data validation and secure database updates. |
| **Cost and schedule** | 1 week |
| **Risks** | Data corruption or unauthorized changes if not secure. |
| **Dependencies with other requirements** | Requires User Login. |
| **Pre-Condition** | User is logged in and viewing their profile. |
| **Post-Condition** | User profile information is updated in the database. |
| **Inputs** | User ID (implicit from session), updated profile details. |
| **Outputs** | Confirmation of update. |

**4.2 Medical History Management**

**Functional Requirement (*FR-02.1*)**

**Priority & Ranking**: Extremely High (#2)

|  |  |
| --- | --- |
| **Function Name** | Doctor: View Patient Medical History |
| **Description** | Allows a doctor to view the complete medical history of a specific patient. |
| **Critically** | Critical for diagnosis and treatment. |
| **Technical issues** | Securely retrieving and displaying potentially large and complex medical records. |
| **Cost and schedule** | 2 weeks. |
| **Risks** | Viewing incorrect or incomplete history leading to misdiagnosis. |
| **Dependencies with other requirements** | Requires Doctor Login, Patient data. |
| **Pre-Condition** | Doctor is logged in and has selected a patient. |
| **Post-Condition** | Patient's medical history is displayed. |
| **Inputs** | Patient ID |
| **Outputs** | Patient medical history data. |

**Functional Requirement (*FR-02.2*)**

**Priority & Ranking**: Extremely High (#2)

|  |  |
| --- | --- |
| **Function Name** | Doctor: Add Entry to Medical History |
| **Description** | Allows a doctor to add a new entry (diagnosis, treatment, notes) to a patient's medical history. |
| **Critically** | Essential for keeping records up-to-date. |
| **Technical issues** | Ensuring data integrity and associating the entry with the correct patient and doctor. |
| **Cost and schedule** | 2 weeks. |
| **Risks** | Adding incorrect or fabricated information. |
| **Dependencies with other requirements** | Requires Doctor Login, Patient data. |
| **Pre-Condition** | Doctor is logged in and viewing patient's history. |
| **Post-Condition** | New entry is added to the patient's medical history. |
| **Inputs** | Patient ID, Doctor ID, Diagnosis, Treatment, Date, Notes. |
| **Outputs** | Confirmation of addition. |

**Functional Requirement (*FR-02.3*)**

**Priority & Ranking**: Extremely High (#2)

|  |  |
| --- | --- |
| **Function Name** | Doctor: Update Entry in Medical History |
| **Description** | Allows a doctor to modify an existing entry in a patient's medical history. |
| **Critically** | Important for correcting errors or adding details to previous entries. |
| **Technical issues** | Implementing secure updates while potentially maintaining an audit trail of changes. |
| **Cost and schedule** | 2 weeks. |
| **Risks** | Altering history maliciously or incorrectly. |
| **Dependencies with other requirements** | Requires Doctor Login, Patient data. |
| **Pre-Condition** | Doctor is logged in and viewing patient's history. |
| **Post-Condition** | Existing entry is updated in the database. |
| **Inputs** | Entry ID, updated Diagnosis, Treatment, Date, Notes. |
| **Outputs** | Confirmation of update. |

**Functional Requirement (*FR-02.3*)**

**Priority & Ranking**: Extremely High (#2)

|  |  |
| --- | --- |
| **Function Name** | Doctor: Delete Entry in Medical History |
| **Description** | Allows a doctor to remove an entry from a patient's medical history (with appropriate permissions/logging). |
| **Critically** | Necessary for removing erroneous entries, but requires careful handling. |
| **Technical issues** | Implementing soft deletes or strict access control and logging for deletions. |
| **Cost and schedule** | 2 weeks. |
| **Risks** | Accidental or malicious deletion of critical history. |
| **Dependencies with other requirements** | Requires Doctor Login, Patient data. |
| **Pre-Condition** | Doctor is logged in and viewing patient's history. |
| **Post-Condition** | Entry is removed or marked as deleted. |
| **Inputs** | Entry ID |
| **Outputs** | Confirmation of deletion. |

**4.3 Prescription Management**

**Functional Requirement (*FR-03.1*)**

**Priority & Ranking**: High (#3)

|  |  |
| --- | --- |
| **Function Name** | Doctor: Issue New Prescription |
| **Description** | Allows a doctor to create and issue a new prescription for a selected patient. |
| **Critically** | Essential for patient treatment and medication management. |
| **Technical issues** | Integrating with medication and patient data, generating a unique prescription record. |
| **Cost and schedule** | 3 weeks |
| **Risks** | Incorrect medication or dosage prescribed. |
| **Dependencies with other requirements** | Requires Doctor Login, Patient data, Medication data. |
| **Pre-Condition** | Doctor is logged in and has selected a patient. |
| **Post-Condition** | New prescription record is created and linked. |
| **Inputs** | Patient ID, Doctor ID, Medication ID, Dosage, Instructions. |
| **Outputs** | Prescription ID, confirmation. |

**Functional Requirement (*FR-03.2*)**

**Priority & Ranking**: High (#3)

|  |  |
| --- | --- |
| **Function Name** | Doctor: View Patient Prescriptions |
| **Description** | Allows a doctor to view all prescriptions issued for a specific patient. |
| **Critically** | Important for tracking patient medication history and effectiveness. |
| **Technical issues** | Securely retrieving and displaying a list of prescriptions for a patient. |
| **Cost and schedule** | 1 week |
| **Risks** | Viewing prescriptions for the wrong patient. |
| **Dependencies with other requirements** | Requires Doctor Login, Patient data, Medication data. |
| **Pre-Condition** | Doctor is logged in and has selected a patient. |
| **Post-Condition** | List of patient prescriptions is displayed. |
| **Inputs** | Patient ID |
| **Outputs** | List of prescription data. |

**4.4 Audit Logging and Security**

**Functional Requirement (*FR-04.1*)**

**Priority & Ranking**: High (#3)

|  |  |
| --- | --- |
| **Function Name** | System: Log User Actions |
| **Description** | Automatically records significant actions performed by any user in the system. |
| **Critically** | Essential for security, compliance, and troubleshooting. |
| **Technical issues** | Designing a comprehensive logging mechanism that captures necessary details without impacting performance. |
| **Cost and schedule** | 2-3 weeks |
| **Risks** | Missing critical logs, making audits impossible. |
| **Dependencies with other requirements** | All system functions. |
| **Pre-Condition** | System is operational. |
| **Post-Condition** | User action is recorded in the audit log. |
| **Inputs** | User ID, Action performed, Timestamp, Relevant data IDs. |
| **Outputs** | Log entry created. |

**Functional Requirement (*FR-04.2*)**

**Priority & Ranking**: High (#3)

|  |  |
| --- | --- |
| **Function Name** | Admin: View Audit Logs |
| **Description** | Allows an administrator to view the recorded audit logs. |
| **Critically** | Essential for monitoring system activity and identifying suspicious behavior. |
| **Technical issues** | Implementing a user interface to display potentially large volumes of log data with filtering/sorting options. |
| **Cost and schedule** | 1.5 weeks |
| **Risks** | Inability to view logs when needed, hindering security investigations. |
| **Dependencies with other requirements** | Requires Admin Login, Audit Log data. |
| **Pre-Condition** | Admin is logged in. |
| **Post-Condition** | Audit logs are displayed. |
| **Inputs** | Filtering/sorting criteria (optional). |
| **Outputs** | List of audit log entries. |

**4.5 Appointment Scheduling and Management**

**Functional Requirement (*FR-05.1*)**

**Priority & Ranking**: High (#3)

|  |  |
| --- | --- |
| **Function Name** | Patient: View Doctor Availability |
| **Description** | Allows a patient to view the available time slots for doctors. |
| **Critically** | Essential for patients to book appointments. |
| **Technical issues** | Implementing a calendar view that shows available slots based on doctor schedules and existing appointments. |
| **Cost and schedule** | 2 weeks |
| **Risks** | Displaying incorrect availability, leading to booking errors. |
| **Dependencies with other requirements** | Requires Patient Login, Doctor data, Appointment data. |
| **Pre-Condition** | Patient is logged in. |
| **Post-Condition** | Available time slots are displayed. |
| **Inputs** | Doctor ID, Date range. |
| **Outputs** | List of available time slots. |

**Functional Requirement (*FR-05.2*)**

**Priority & Ranking**: High (#3)

|  |  |
| --- | --- |
| **Function Name** | Patient: Book Appointment |
| **Description** | Allows a patient to book an appointment with a selected doctor at an available time slot. |
| **Critically** | Core functionality for patient access to care. |
| **Technical issues** | Creating a new appointment record and updating the doctor's schedule and availability. |
| **Cost and schedule** | 2.5 weeks |
| **Risks** | Double-booking appointments. |
| **Dependencies with other requirements** | Requires Patient Login, Doctor data, Appointment data. |
| **Pre-Condition** | Patient is logged in and viewing availability. |
| **Post-Condition** | New appointment record is created. |
| **Inputs** | Patient ID, Doctor ID, Selected Date, Selected Time Slot. |
| **Outputs** | Appointment confirmation, Appointment ID. |

**Functional Requirement (*FR-05.3*)**

**Priority & Ranking**: High (#3)

|  |  |
| --- | --- |
| **Function Name** | Patient: Cancel Appointment |
| **Description** | Allows a patient to cancel an appointment they have booked. |
| **Critically** | Provides flexibility for patients to manage their schedule. |
| **Technical issues** | Updating the appointment status to 'Canceled' and potentially making the time slot available again. |
| **Cost and schedule** | 1 week |
| **Risks** | Appointment not being correctly canceled or time slot not freed up. |
| **Dependencies with other requirements** | Requires Patient Login, Appointment data. |
| **Pre-Condition** | Patient is logged in and viewing their appointments. |
| **Post-Condition** | Appointment status is updated to 'Canceled'. |
| **Inputs** | Appointment ID |
| **Outputs** | Confirmation of cancellation. |

**Functional Requirement (*FR-05.4*)**

**Priority & Ranking**: High (#5)

|  |  |
| --- | --- |
| **Function Name** | Staff: Manage Appointments |
| **Description** | Allows staff members to book, reschedule, or cancel appointments for patients and doctors. |
| **Critically** | Core functionality for administrative appointment management. |
| **Technical issues** | Performing appointment CRUD operations (Create, Read, Update, Delete) with appropriate validation. |
| **Cost and schedule** | 2 weeks |
| **Risks** | Creating conflicting appointments or incorrectly modifying records. |
| **Dependencies with other requirements** | Requires Staff Login, Patient data, Doctor data, Appointment data. |
| **Pre-Condition** | Staff is logged in. |
| **Post-Condition** | Appointment records are updated. |
| **Inputs** | Patient ID, Doctor ID, Date, Time Slot, Appointment ID (for update/cancel). |
| **Outputs** | Confirmation of action. |

**4.6 Patient Functions**

**Functional Requirement (*FR-06.1*)**

**Priority & Ranking**: High (#3)

|  |  |
| --- | --- |
| **Function Name** | Patient: Update Personal Details |
| **Description** | Allows a patient to update their personal information (e.g., address, contact). |
| **Critically** | Important for maintaining accurate patient contact information. |
| **Technical issues** | Implementing secure updates to the patient's user profile data. |
| **Cost and schedule** | 1 week |
| **Risks** | Updating incorrect patient's details. |
| **Dependencies with other requirements** | Requires Patient Login, User data. |
| **Pre-Condition** | Patient is logged in. |
| **Post-Condition** | Patient's user profile is updated. |
| **Inputs** | Patient ID (implicit), updated details. |
| **Outputs** | Confirmation of update. |

**Functional Requirement (*FR-06.2*)**

**Priority & Ranking**: Moderate (#4)

|  |  |
| --- | --- |
| **Function Name** | Patient: Rate Doctor |
| **Description** | Allows a patient to provide a rating or feedback for a doctor they have seen. |
| **Critically** | Useful for clinic performance evaluation and doctor feedback. |
| **Technical issues** | Storing ratings and associating them with the correct doctor and patient (optional). |
| **Cost and schedule** | 1.5 weeks |
| **Risks** | Biased or unfair ratings, or inability to link rating to visit. |
| **Dependencies with other requirements** | Requires Patient Login, Doctor data. |
| **Pre-Condition** | Patient is logged in after an appointment. |
| **Post-Condition** | Doctor's rating is recorded. |
| **Inputs** | Patient ID, Doctor ID, Rating, Comments. |
| **Outputs** | Confirmation of rating submission. |

**4.7 Medication Storage Management Functions**

**Functional Requirement (*FR-07.1*)**

**Priority & Ranking**: Medium (#5)

|  |  |
| --- | --- |
| **Function Name** | Staff: View Medication Stock Levels |
| **Description** | Allows staff to view the current stock levels of all medications in storage. |
| **Critically** | Essential for monitoring inventory and preventing shortages. |
| **Technical issues** | Efficiently querying and displaying medication stock data. |
| **Cost and schedule** | 2 weeks |
| **Risks** | Viewing incorrect stock levels. |
| **Dependencies with other requirements** | Requires Staff Login, Storage data, Medication data. |
| **Pre-Condition** | Staff is logged in. |
| **Post-Condition** | Medication stock levels are displayed. |
| **Inputs** | Filtering/sorting criteria (optional). |
| **Outputs** | List of medications with stock levels. |

**5 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.

**5.1 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. |

**5.2 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.

**6 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.

**7 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.

**7.1 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.

**7.2 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.

**7.3 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.

**7.4 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.

**8 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.

**8.1 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.

**8.2 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.

**8.3 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.

**8.4 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.

**8.5 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.

**8.6 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.

**8.7 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.

**8.8 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.

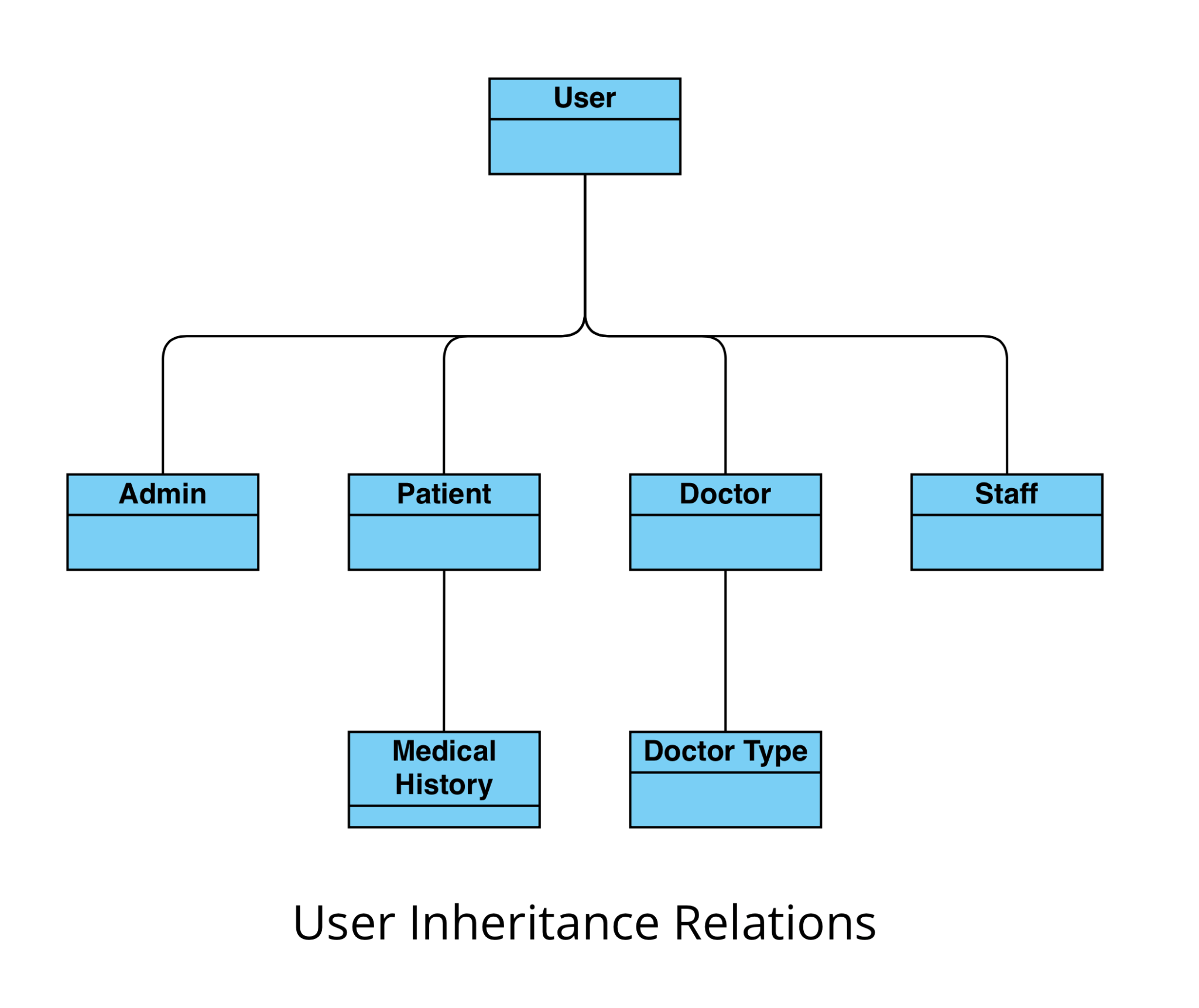
**8.9 User Activity Logging (NF-R9) | Priority:** High

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.

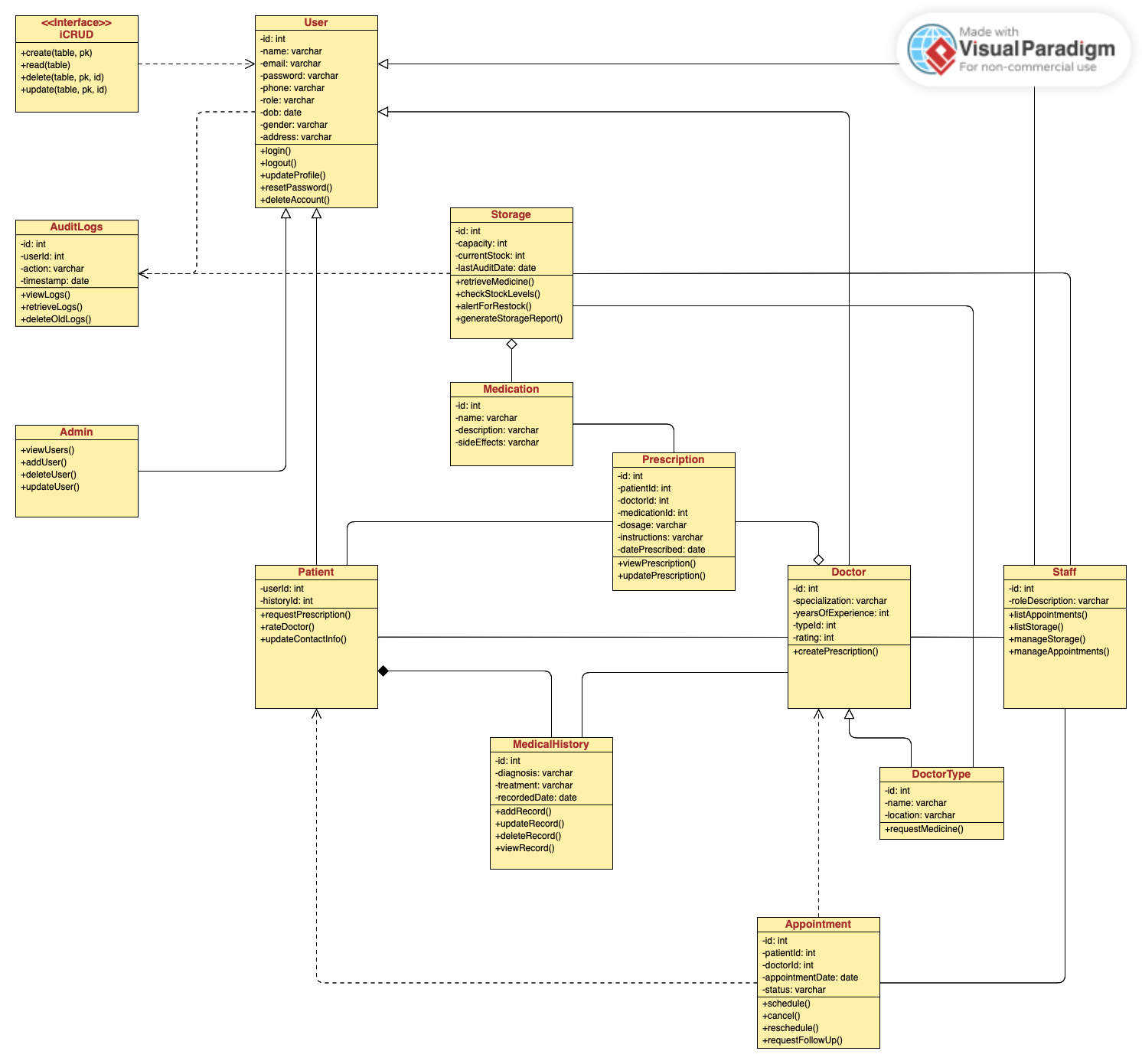
**9 Preliminary Object-Oriented Domain Analysis**

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.

**9.1 Inheritance Relationships**



**9.2 Class Descriptions**



**9.2.1 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. |

**9.2.2 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. |

**9.2.3 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. |

**9.2.4 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. |

**9.2.5 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. |

**9.2.6 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. |

**9.2.7 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. |

**9.2.8 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. |

**9.2.9 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. |

**9.2.10 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. |

**9.2.11 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. |

**9.2.12 Storage**

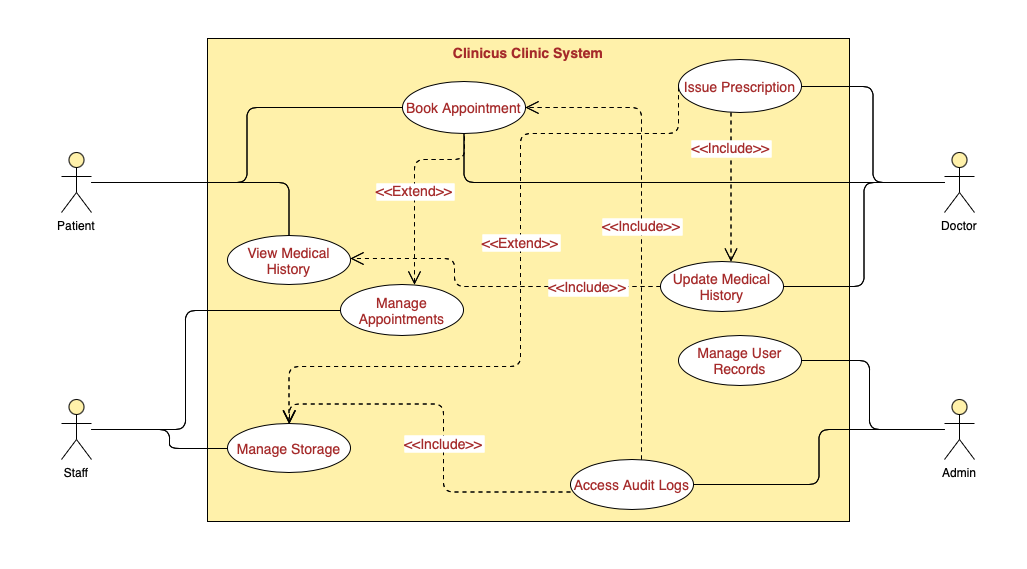
|  |  |
| --- | --- |
| **Class Name** | Storage (Concrete) |
| **Superclass** | None |
| **Subclasses** | None |
| **Purpose** | Management of medicine stocks. |
| **Collaborations** | AuditLogs (all changes in stock are record in the audit logs)  Staff (frequently restock it and are responsible for 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. |

**9.2.13 <<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. |

**10 Operational Scenarios (Use-Case Diagram)**

**Use-Case Diagram**



**10.1 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. |

**10.2 Scenario 2**

|  |  |
| --- | --- |
| **Use-Case**: Doctor Issuing a Prescription |  |
| **Actor**: Doctor |
| **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. |

**10.3 Scenario 3**

|  |  |
| --- | --- |
| **Use-Case**: Admin Managing Patient Records |  |
| **Actor**: Admin |
| **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. |

**10.4 Scenario 4**

|  |  |
| --- | --- |
| **Use-Case**: Patient Viewing Medical History |  |
| **Actor**: Patient |
| **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. |

**10.5 Scenario 5**

|  |  |
| --- | --- |
| **Use-Case**: Doctor Updating Medical History |  |
| **Actor**: Doctor |
| **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. |

**11 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 |

**12 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.

**13 References**

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