INTRODUCTION

CHAPTER 1
INTRODUCTION

1.1 Introduction

In the rapidly evolving landscape of healthcare, hospitals face significant challenges in managing complex operations while ensuring high-quality Patient care. The MedSync Healthcare Platform is a state-of-the-art, web-based Healthcare Information System (HIS) developed to address these challenges for medium to large hospitals. Built using a robust technology stack comprising PHP for backend processing, MySQL for efficient data storage, and a responsive frontend crafted with HTML, CSS, and JavaScript, MedSync aims to streamline critical hospital workflows. The platform automates essential processes, including appointment scheduling, live token tracking, billing, prescription management, Patient admissions, discharges, and resource allocation.

MedSync incorporates advanced features such as role-based access control (RBAC) to provide tailored functionalities for Administrators, Doctors, Staff, and Patients, ensuring secure and efficient operations. The system leverages PHPMailer for robust email notifications, delivering real-time updates for appointments, billing, and discharge processes, and dompdf for generating professional PDF documents such as bills, prescriptions, and discharge summaries. By offering a responsive and user-friendly interface accessible across devices, MedSync enhances operational efficiency, reduces manual errors, and improves Patient satisfaction. The platform's real-time dashboards and color-coded resource tracking enable hospital Staff to make informed decisions, while secure authentication mechanisms, including OTP-based registration and CSRF protection, safeguard sensitive data. This introduction outlines the purpose, scope, and significance of MedSync in transforming hospital management.

1.2 Problem Statement

Traditional hospital management systems often rely on manual or semi-automated processes, leading to inefficiencies that hinder operational performance and Patient care. Common issues include prolonged appointment scheduling delays, errors in billing due to manual calculations, and mismanagement of critical resources such as beds, medicines, and blood inventory. These inefficiencies result in extended Patient wait times, reduced Staff productivity, and diminished Patient satisfaction. Additionally, many existing systems lack real-time updates, causing miscommunication between departments and delays in processes like Patient discharges.

Security vulnerabilities, such as inadequate user authentication and lack of role-based access control, expose sensitive Patient data to risks like unauthorized access or data breaches. The absence of automated notifications and standardized documentation further complicates coordination among Administrators, Doctors, and Staff. For Patients, the lack of transparency

in appointment queue status and billing details leads to frustration and distrust. MedSync addresses these challenges by providing a comprehensive, secure, and automated platform that integrates all hospital operations into a single system, reducing errors, improving resource utilization, and enhancing the overall Patient experience.

1.3 Scope and Relevance of the Project

The MedSync Healthcare Platform is designed to modernize hospital operations by providing a scalable and secure solution tailored to the needs of medium to large hospitals. The project's scope includes the development of a web-based system that supports four distinct user roles—Administrators, Doctors, Staff, and Patients—with specific functionalities to streamline workflows. Key features include:

- **User Management**: Secure OTP-based registration, role-based access control, and robust session management to ensure data security and tailored user experiences.
- **Appointment and Token System**: Real-time appointment booking, digital token generation, and live queue status updates to reduce scheduling delays and improve Patient flow.
- **Healthcare Services**: Digital prescription management, secure access to medical records, and lab result tracking to enhance clinical efficiency.
- **Resource and Inventory Management**: Color-coded bed tracking and inventory monitoring for medicines and blood to prevent double bookings and ensure availability.

The relevance of MedSync lies in its ability to address the growing demand for efficient, Patient-centered healthcare systems. By reducing appointment scheduling delays by an estimated 40%, minimizing billing errors by 25%, and optimizing resource utilization, MedSync enhances hospital efficiency and Patient satisfaction. The platform's secure architecture, including password hashing, CSRF protection, and audit logging, ensures compliance with data privacy standards. In an era where hospitals are under pressure to deliver high-quality care while managing increasing Patient volumes, MedSync offers a scalable solution that supports operational planning, reduces manual paperwork by 35%, and fosters seamless communication across departments.

1.4 Objectives

The MedSync Healthcare Platform is developed with the following objectives to address the identified challenges and enhance hospital operations:

1. **Automate Key Hospital Processes**: Implement automated workflows for appointment scheduling, billing, Patient admissions, and discharges to reduce manual errors, improve operational efficiency by 40%, and streamline coordination among hospital Staff.

2. **Enhance Patient Experience**: Provide real-time updates through live token tracking, enabling Patients to monitor appointment statuses and access bills, lab results, and discharge summaries, thereby improving satisfaction and transparency.
3. **Optimize Resource Utilization**: Develop a color-coded interface for real-time tracking of beds (available, occupied, reserved, cleaning) and inventory (medicines, blood) to prevent double bookings and ensure efficient allocation.
4. **Ensure Robust Security**: Implement secure user authentication with OTP-based registration, password hashing using password_hash() and password_verify(), CSRF protection, and session management with a 30-minute timeout to safeguard sensitive Patient and hospital data.
5. **Support Scalability and Maintainability**: Design a modular system using PHP, MySQL, HTML, CSS, and JavaScript, with prepared statements for database interactions and a clear database structure to ensure scalability and ease of future enhancements.
6. **Provide Actionable Insights**: Develop real-time dashboards and comprehensive reports on revenue, bed occupancy, and hospital performance, exportable as PDFs, to support data-driven decision-making and operational planning.

By achieving these objectives, MedSync aims to transform hospital management by integrating advanced technology with user-centric design, ultimately improving Patient care and operational efficiency.

SYSTEM ANALYSIS

CHAPTER 2
SYSTEM ANALYSIS

2.1 Introduction

System analysis is a critical phase in the development of the MedSync Healthcare Platform, aimed at understanding the requirements, identifying inefficiencies in existing systems, and proposing a robust solution to meet the needs of medium to large hospitals. This chapter evaluates the current hospital management systems, outlines the proposed MedSync platform, and assesses its feasibility to ensure it addresses operational challenges effectively. By analyzing the existing system's limitations and defining the advantages of the proposed system, this chapter establishes a foundation for designing a secure, efficient, and scalable Healthcare Information System (HIS). The analysis also includes a feasibility study to evaluate the technical, operational, and economic viability of the project, along with the software engineering paradigm applied to ensure structured development.

2.2 Existing System

Many hospitals, particularly medium to large ones, rely on a combination of manual processes and fragmented digital systems for managing operations such as appointment scheduling, billing, patient admissions, discharges, and resource allocation.

The existing systems often include:

- **Manual Record-Keeping**: Patient records, prescriptions, and billing details are frequently maintained in physical registers or basic spreadsheets.
- **Basic Appointment Systems**: Appointment scheduling is typically handled through phone calls or in-person bookings, with limited real-time visibility of doctor availability.
- **Fragmented Software Solutions**: Some hospitals use standalone software for specific tasks, but these systems lack integration, which causes data silos.
- **Manual Queue Management**: Tokens are often physical paper slips, requiring patients to wait on-site without real-time status updates.
- **Inefficient Discharge Process**: The discharge workflow involves the physical movement of files between nursing stations, the pharmacy, and the billing desk.
- **Limited Automation**: Key processes like inventory tracking and bed availability updates are often semi-automated or entirely manual.

2.2.1 Limitations of Existing System

The reliance on these manual and fragmented systems leads to significant operational challenges:

- **Data Inefficiency**: Manual record-keeping is prone to errors, data loss, and time-consuming retrieval processes.
- **Poor Patient Experience**: The lack of real-time visibility in scheduling results in long patient wait times and conflicts. Patients also have limited or no direct access to their medical records, appointment statuses, or bills, resulting in poor transparency and reduced satisfaction.
- **Operational Delays**: The manual discharge process is slow and prone to loss of information. This, along with manual inventory tracking, leads to delays in updating bed availability and errors in medication dispensing.
- **Security Weaknesses**: Existing systems often lack robust security measures, such as role-based access control (RBAC), password hashing, or protection against modern vulnerabilities like SQL injection and Cross-Site Scripting (XSS).
- **Financial & Resource Impact**: These inefficiencies contribute directly to appointment delays (estimated at 30–40% longer than necessary), billing errors (approximately 20% error rate), and resource mismanagement, such as double-booked beds or untracked inventory.

2.3 Proposed System

The MedSync Healthcare Platform is a comprehensive, web-based HIS designed to address the shortcomings of existing systems by integrating all hospital operations into a single, secure, and user-friendly platform.

The proposed system offers the following key features:

- **Integrated User Management**: Supports four user roles (administrators, doctors, staff, patients) with secure OTP-based registration via PHPMailer and an integrated Google Sign-In option, role-based access control (RBAC), and session management.
- **Real-Time Appointment and Token System**: Enables patients to book appointments based on doctor availability, with digital token generation and live status updates via AJAX (refreshing every **30 seconds for patients** and **10 seconds for staff**).
- A "Search Doctor" feature allows filtering by name, specialty, and time slots, with email confirmations sent via PHPMailer.
- **Healthcare Services**: Facilitates digital prescription management (with doctor and hospital details), secure access to patient medical records, lab **ordering by doctors**, and lab **result entry/uploading by staff**, with viewing access for both.
- **Resource and Inventory Management**: Provides a color-coded interface (green: available, red: occupied, purple: reserved, yellow: cleaning) for bed tracking and monitors medicine and blood inventory.

- **Notifications and Reporting**: Sends automated email notifications for appointments using PHPMailer.
- Generates comprehensive reports on revenue, bed occupancy, and hospital performance, exportable as PDFs.
- **Security Enhancements**: Implements **Google reCAPTCHA** on login/registration, password hashing (PASSWORD_BCRYPT), CSRF token protection, prepared statements (to prevent SQL injection), and administrator-level **IP blocking**.
- **Integrated Patient Feedback:** Allows patients to submit star ratings (overall and doctor-specific) and written comments for completed appointments, providing actionable data for administrative review.

The proposed system aims to reduce appointment scheduling delays by 40%, billing errors by 25%, and manual paperwork by 35%, while improving patient satisfaction through transparent, real-time updates and secure access to medical information.

2.3.1 Advantages of the Proposed System

The MedSync Healthcare Platform offers several advantages over existing systems, making it a transformative solution for hospital management:

- **Enhanced Efficiency**: Automation of appointment scheduling, discharge processes, and resource tracking reduces manual intervention, streamlining operations and saving time for staff and doctors.
- **Improved Patient Experience**: Live token tracking, email notifications, and access to bills, prescriptions, and lab results empower patients with transparency and reduce wait times.
- **Quality Assurance:** Gathers direct patient feedback on appointments and doctors, enabling continuous quality improvement and performance tracking.
- **Resource Optimization**: Real-time, color-coded tracking of beds and inventory ensures efficient allocation, preventing double bookings and stock shortages.
- **Robust Security**: Features like OTP-based registration, password hashing, CSRF protection, and audit logging safeguard sensitive data, ensuring compliance with healthcare privacy standards.
- **Seamless Communication**: Automated notifications and standardized PDF documents (bills, discharge summaries) improve coordination among departments and with patients.
- **Scalability and Maintainability**: The modular design using PHP, MySQL, and a responsive frontend supports future enhancements and scalability for growing hospital needs.
- **Data-Driven Decision-Making**: Real-time dashboards and exportable reports provide actionable insights into hospital performance, revenue, and resource utilization.

2.4 Feasibility Study

The feasibility study evaluates the practicality of developing and implementing the MedSync Healthcare Platform across technical, operational, and economic dimensions.

2.4.1 Technical Feasibility

The MedSync platform is technically feasible due to the availability of mature and widely supported technologies:

- **Technology Stack**: PHP (backend), MySQL (database), HTML, CSS, JavaScript (frontend), PHPMailer (email notifications), and dompdf (PDF generation) are well-documented, open-source tools with extensive community support.
- **Hardware Requirements**: The system can be hosted on standard servers (e.g., XAMPP for development) with moderate specifications (e.g., 8GB RAM, 2.4 GHz processor, 500GB storage), which are readily available in most hospital IT infrastructures.
- **Development Expertise**: The project leverages commonly used programming languages and frameworks, requiring developers with proficiency in PHP, MySQL, and web development, which are widely available skill sets.
- **Scalability**: The modular architecture and MySQL's relational database structure support scalability for increased user loads and data volumes.
- **Security**: Implementation of prepared statements, password hashing, CSRF tokens, and session management ensures robust protection against common vulnerabilities like SQL injection.

2.4.2 Operational Feasibility

The MedSync platform is operationally feasible as it aligns with the workflows of medium to large hospitals:

- **User Adoption**: The intuitive, role-based dashboards (for administrators, doctors, staff, patients) and responsive design ensure ease of use across devices, facilitating adoption by users with varying technical expertise.
- **Integration with Existing Processes**: The system supports existing hospital workflows (e.g., appointment scheduling, discharge processes) while introducing automation to reduce manual tasks, minimizing disruption during implementation.
- **Training Requirements**: Staff, doctors, and administrators can be trained on the system's functionalities through short workshops, as the interface is designed to be user-friendly with features like color-coded bed tracking and AJAX-based real-time updates.
- **Maintenance**: The modular design and use of prepared statements simplify debugging and updates, while audit logs aid in tracking system issues, ensuring operational sustainability.

- **Stakeholder Support**: The platform addresses the needs of all stakeholders (patients, doctors, staff, administrators), increasing the likelihood of acceptance and successful deployment.

2.4.3 Economic Feasibility

The economic feasibility of MedSync is supported by its cost-effective development and potential for significant return on investment:

- **Development Costs**: The use of open-source technologies (PHP, MySQL, PHPMailer, dompdf) and XAMPP for development eliminates licensing fees.
- Development costs primarily involve developer salaries and server setup, which are manageable for a project of this scale.
- **Operational Savings**: Automation of appointment scheduling, billing, and discharge processes is expected to reduce manual paperwork by 35%, saving staff time and reducing errors (e.g., 25% reduction in billing errors).
- These efficiencies translate to cost savings in administrative overhead.
- **Revenue Impact**: Improved patient satisfaction through reduced wait times and transparent communication can increase patient retention and attract new patients, boosting hospital revenue.
- **Maintenance Costs**: The system's modular design and use of standard technologies minimize maintenance costs, with updates and bug fixes manageable by a small IT team.
- **Long-Term Benefits**: The platform's scalability ensures it can handle future growth without significant additional investment, while features like report generation support strategic financial planning.

The economic benefits of reduced errors, improved efficiency, and enhanced patient satisfaction outweigh the initial development and implementation costs, making MedSync economically viable.

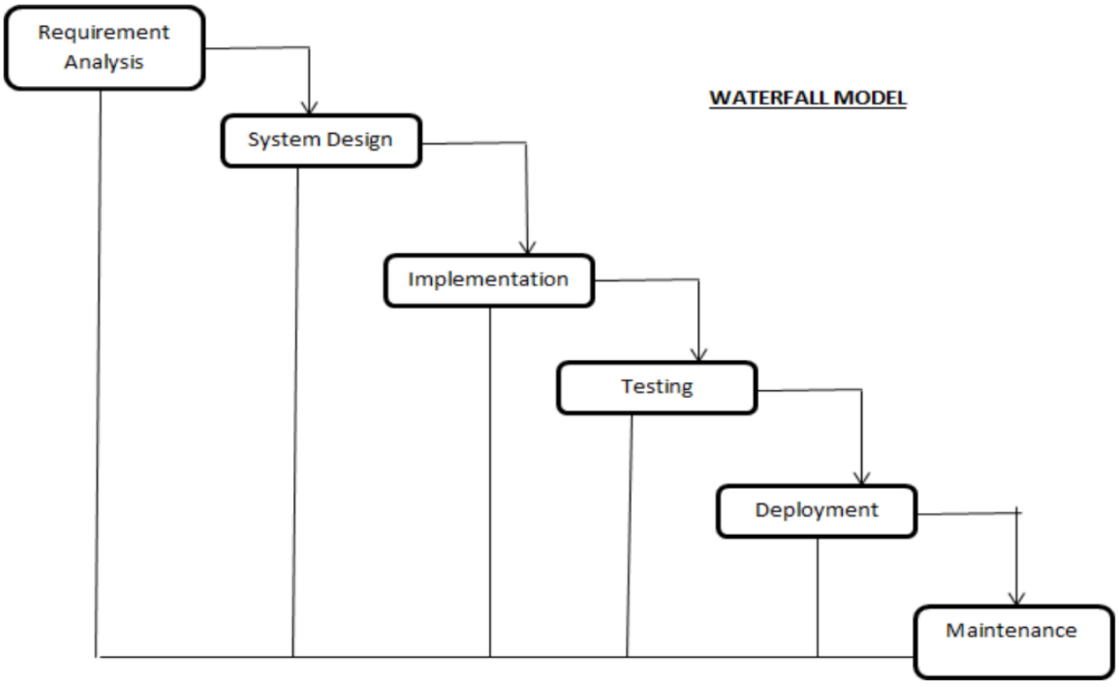
2.5 Software Engineering Paradigm Applied

The development of the MedSync Healthcare Platform follows the **Waterfall Model** as the primary software engineering paradigm, supplemented by iterative prototyping for user interface design. The Waterfall Model is suitable for this project due to its structured approach, which aligns with the well-defined requirements provided in the project plan.

The development process is divided into sequential phases:

1. **Requirement Analysis**: Gathering and documenting requirements from the project plan, including user roles, core functionalities, and security measures, as outlined in the provided documents.
2. **System Design**: Designing the database schema (e.g., users, appointments, discharge_clearance tables), process flows (e.g., Data Flow Diagrams), and object-oriented designs (e.g., UML diagrams like Use Case, Class, and Sequence Diagrams).
3. **Implementation**: Coding the system using PHP, MySQL, HTML, CSS, and JavaScript, with a focus on modularity and security (e.g., prepared statements, CSRF tokens).
4. **Testing**: Conducting unit testing, integration testing, and system testing to ensure functionality, security, and performance meet requirements.
5. **Deployment**: Deploying the system on a hospital server (e.g., using XAMPP) and training staff for operational use.
6. **Maintenance**: Providing ongoing support for bug fixes, updates, and future enhancements.

The following illustration is a representation of the different phases of the Waterfall Model :



SYSTEM DESIGN

CHAPTER 3:
SYSTEM DESIGN

3.1 Introduction

System design is the process of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements. For the MedSync Healthcare Platform, the design phase translates the requirements identified in the system analysis into a blueprint for constructing the software. This chapter outlines the comprehensive design of the MedSync platform, covering database structure, process flows, object-oriented models, and input/output design.

The primary goal of this design is to create a secure, scalable, and maintainable Healthcare Information System (HIS) that is both efficient for hospital staff and user-friendly for patients. The design emphasizes a modular architecture, which allows for future enhancements and scalability to accommodate the growing needs of medium to large hospitals. Key design considerations include robust data management through a relational database, clear process modeling using Data Flow Diagrams (DFDs), and detailed object-oriented design with UML diagrams to represent the system's static structure and dynamic behavior.

3.2 Database Design

The database is the cornerstone of the MedSync platform, designed to store and manage all patient, staff, and operational data securely and efficiently. A relational database model using MySQL was chosen for its robustness, scalability, and widespread support. The design prioritizes data integrity, performance, and clarity. Key features of the database design include:

- **Normalization:** The schema is normalized to reduce data redundancy and improve data integrity. For instance, user roles are stored in a dedicated roles table and linked via foreign keys, rather than using an ENUM type in the users table.
- **Data Integrity:** Foreign key constraints are used extensively to maintain relationships between tables, ensuring that, for example, an appointment cannot exist without a valid patient and doctor. Cascading rules for updates and deletions are defined to maintain consistency.
- **Performance:** Indexes are created on frequently queried columns, such as user IDs, names, and dates, to accelerate data retrieval operations, which is crucial for features like real-time token tracking and searching for users.
- **Clarity and Convention:**.Naming conventions for tables and columns are consistent to improve readability of the code.

Normalization

Normalization is the process of organizing the fields and tables of a relational database to minimize redundancy and dependency. It divides larger tables into smaller tables and links them using relationships. The main goal is to isolate data so that additions, deletions, and modifications of a field can be made in just one table and then propagated through the rest of the database via the defined relationships.

First Normal Form (1NF) A relation is in 1NF if it contains an atomic value. It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attributes. In the MedSync system, this ensures that every column in a table (like `users` or `medicines`) holds a single value (e.g., a single phone number rather than a comma-separated list).

Second Normal Form (2NF) A relation is in 2NF if it is in 1NF and all non-key attributes are fully functionally dependent on the primary key. This step removes data that applies to multiple rows of a table and places them in separate tables. For example, in MedSync, patient details are separated from appointment details to ensure that patient data doesn't depend on the appointment ID.

Third Normal Form (3NF) A relation is in 3NF if it is in 2NF and has no transitive dependency for non-prime attributes. This means that non-key attributes must depend only on the primary key and not on other non-key attributes. In MedSync, this is achieved by ensuring, for instance, that a `ward_name` depends only on the `ward_id` and is not stored redundantly in the `accommodations` table.

3.2.1 Entity Relationship Model

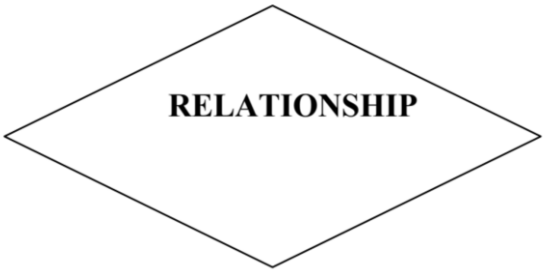
The Entity-Relationship (E-R) model for the MedSync platform defines the main entities, their attributes, and the relationships between them. The central entity is users, which is linked to various other tables like appointments, prescriptions, and admissions. The roles table defines the access level for each user. The accommodations table is a unified entity for both beds and private rooms, linked to wards when the type is 'bed'. The relational schema illustrates the connections between tables. For example, the admissions table links a patient_id and doctor_id (both from the users table) to an accommodation_id.

Components of the Entity-Relationship (E-R) Diagram
The E-R diagram uses specific symbols to represent the database structure visually:

- **Entities:** Represented by rectangles. An entity is an object or concept about which data is stored (e.g., `User`, `Doctor`, `Appointment`).

- **Attributes:** Represented by ovals. These are the properties or details describing an entity (e.g., `username`, `email`, `specialty`).

- **Relationships:** Represented by diamond shapes. These define how entities interact with each other (e.g., A Patient *Books* an Appointment).



- **Connecting Lines:** Solid lines that connect attributes to entities and entities to relationships.

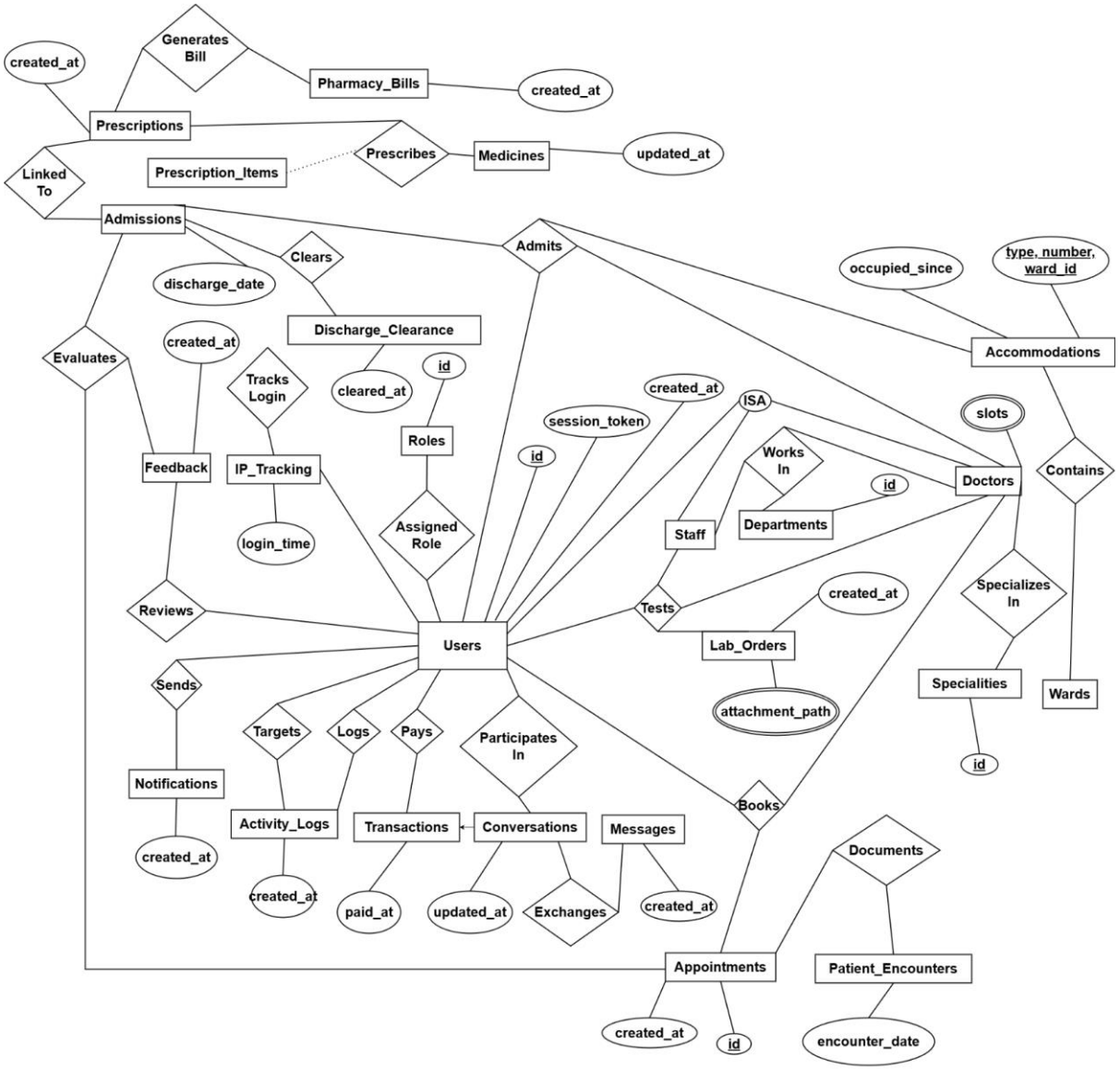

Generates Bill
created_at
Pharmacy_Bills
created_at
Prescriptions
Linked To
Prescription_Items
Prescribes
Medicines
updated_at
Admissions
Clears
discharge_date
Admits
occupied_since
type, number, ward_id
Discharge_Clearance
Accommodations
Evaluates
created_at
Tracks Login
cleared_at
id
Roles
created_at
ISA
session_token
slots
Feedback
IP_Tracking
id
Works In
Doctors
Contains
login_time
Assigned Role
Staff
Departments
id
Reviews
Tests
created_at
Specializes In
Users
Lab_Orders
Sends
Specialities
Wards
attachment_path
Targets
Logs
Pays
Participates In
Notifications
Books
Activity_Logs
Transactions
Conversations
Messages
Documents
created_at
created_at
paid_at
updated_at
Exchanges
created_at
Appointments
Patient_Encounters
created_at
id
encounter_date

Table Design - MedSync Database

Core & User Management

Table 1: roles

Stores the different user roles available in the system.

Column	Type	Constraints	Description
id	int(11)	Primary Key, AUTO_INCREMENT	Unique identifier for the role.
role_name	varchar(50)	Unique, NOT NULL	The name of the role (e.g., 'user', 'doctor').

Table 2: role_counters

A utility table to generate sequential IDs based on roles.

Column	Type	Constraints	Description
role_prefix	char(1)	Primary Key	The prefix for the role (e.g., 'D' for Doctor).
last_id	int(10) unsigned	NOT NULL, DEFAULT 0	The last used ID number for that role.

Table 3: users

The central table for storing all user information, regardless of role.

Column	Type	Constraints	Description
id	int(11)	Primary Key, AUTO_INCREMENT	Unique identifier for the user.
display_user_id	varchar(20)	Unique, NOT NULL	A user-facing unique ID.
username	varchar(50)	Unique, NOT NULL	The user's login name.
email	varchar(100)	Unique, NOT NULL	The user's email address.
password	varchar(255)	NOT NULL	Hashed password for the user.
role_id	int(11)	Foreign Key to roles.id, NOT NULL, DEFAULT 1	Links to the user's role.
name	varchar(100)		User's full name.
gender	enum('Male','Female','Other')		User's gender.
profile_picture	VARCHAR(255)	DEFAULT 'default.png'	Path to the user's profile picture.
phone	varchar(25)		User's phone number.

date_of_birth	date		User's date of birth.
notify_appointments	tinyint(1)	NOT NULL, DEFAULT 1	Notification preference for appointments.
notify_billing	tinyint(1)	NOT NULL, DEFAULT 1	Notification preference for billing.
notify_labs	tinyint(1)	NOT NULL, DEFAULT 1	Notification preference for lab results.
notify_prescriptions	tinyint(1)	NOT NULL, DEFAULT 1	Notification preference for prescriptions.
is_active	tinyint(1)	NOT NULL, DEFAULT 1	1 for active, 0 for inactive.
session_token	varchar(255)		Stores the user's session token.
created_at	timestamp	NOT NULL, DEFAULT CURRENT_TIMESTAMP	Timestamp of user creation.

Table 4: doctors

Stores additional information specific to users with the 'doctor' role.

Column	Type	Constraints	Description
id	int(11)	Primary Key, AUTO_INCREMENT	Unique identifier for the doctor record.
user_id	int(11)	Unique, Foreign Key to users.id, NOT NULL	Links to the main user entry.
specialty_id	int(11)	Foreign Key to specialities.id	Links to the doctor's medical specialty.
qualifications	varchar(255)		e.g., MBBS, MD
department_id	int(11)	Foreign Key to departments.id	The department the doctor belongs to.
is_available	tinyint(1)	NOT NULL, DEFAULT 1	1 = Available, 0 = On Leave
slots	longtext		JSON array of available time slots.
office_floor	VARCHAR(50)	DEFAULT 'Ground Floor'	Doctor's office floor.
office_room_number	VARCHAR(50)	DEFAULT 'N/A'	Doctor's office room number.

Table 5: staff

Stores information for non-doctor hospital staff.

Column	Type	Constraints	Description
id	int(11)	Primary Key, AUTO_INCREMENT	Unique identifier for the staff record.
user_id	int(11)	Unique, Foreign Key to users.id, NOT NULL	Links to the main user entry.
shift	enum('day','night','off')	NOT NULL, DEFAULT 'day'	The staff member's current shift.
assigned_department_id	int(11)	Foreign Key to departments.id	The department the staff is assigned to.

Hospital Operations & Resources

Table 6: departments

Stores hospital departments.

Column	Type	Constraints	Description
id	int(11)	Primary Key, AUTO_INCREMENT	Unique identifier for the department.
name	varchar(100)	Unique, NOT NULL	Name of the department.
head_of_department_id	int(11)	Foreign Key to users.id	The user ID of the department head.
is_active	tinyint(1)	NOT NULL, DEFAULT 1	1 for active, 0 for inactive.

Table 7: specialities

Stores medical specialties for doctors.

Column	Type	Constraints	Description
id	int(11)	Primary Key, AUTO_INCREMENT	Unique identifier for the specialty.
name	varchar(100)	Unique, NOT NULL	Name of the specialty.
description	text		A brief description of the specialty.

Table 8: wards

Stores hospital wards.

Column	Type	Constraints	Description
id	INT(11)	Primary Key, AUTO_INCREMENT	Unique identifier for the ward.
name	VARCHAR(100)	Unique, NOT NULL	e.g., General Ward, ICU, Pediatric Ward
capacity	INT(11)	NOT NULL, DEFAULT 0	The total capacity of the ward.
description	text		A brief description of the ward.
is_active	TINYINT(1)	NOT NULL, DEFAULT 1	1 for active, 0 for inactive.

Table 9: accommodations

A unified table for all patient accommodations like beds and rooms.

Column	Type	Constraints	Description
id	INT(11)	Primary Key, AUTO_INCREMENT	Unique ID for the accommodation.
type	enum('bed', 'room')	NOT NULL	The type of accommodation.

number	VARCHAR(50)	NOT NULL	Bed number or room number.
ward_id	INT(11)	Foreign Key to wards.id	The ward this accommodation is in.
status	enum('available', 'occupied', 'reserved', 'cleaning')	NOT NULL, DEFAULT 'available'	Current status of the accommodation.
patient_id	INT(11)	Foreign Key to users.id	The patient currently occupying it.
doctor_id	INT(11)	Foreign Key to users.id	The doctor assigned to the patient.
occupied_since	DATETIME		Timestamp when occupation started.
reserved_since	DATETIME		Timestamp when reservation started.
price_per_day	decimal(10, 2)	NOT NULL, DEFAULT 0.00	Cost of the accommodation per day.
last_updated	timestamp	NOT NULL, DEFAULT CURRENT_TIMESTAMP	Timestamp of the last update.
Composite Key	(type, number, ward_id)	Unique	Ensures no duplicate accommodations in a ward.

Patient-Related Tables

Table 10: admissions

Tracks patient admissions and discharges.

Column	Type	Constraints	Description
id	int(11)	Primary Key, AUTO_INCREMENT	Unique identifier for the admission.
patient_id	int(11)	Foreign Key to users.id, NOT NULL	The admitted patient.
doctor_id	int(11)	Foreign Key to users.id	The admitting or primary doctor.
accommodation_id	int(11)	Foreign Key to accommodations.id	The assigned bed or room.
admission_date	datetime	NOT NULL	The date and time of admission.
discharge_date	datetime		The date and time of discharge.

Table 11: appointments

Manages patient appointments with doctors.

Column	Type	Constraints	Description
id	int(11)	Primary Key, AUTO_INCREMENT	Unique identifier for the appointment.
user_id	int(11)	Foreign Key to users.id, NOT NULL	The patient who booked the appointment.
doctor_id	int(11)	Foreign Key to users.id, NOT NULL	The doctor for the appointment.
appointment_date	datetime	NOT NULL	The date and time of the appointment.
slot_start_time	time		The start time of the appointment slot.
slot_end_time	time		The end time of the appointment slot.
token_number	int(11)		A sequential token for the appointment.
token_status	enum('waiting','in_consultation','completed','skipped')	NOT NULL, DEFAULT 'waiting'	The current status of the patient's token.
consultation_start_time	DATETIME		Timestamp for when the consultation actually began.
status	enum('scheduled','co	NOT NULL,	Status of the

	mpleted','cancelled')	DEFAULT 'scheduled'	appointment.
created_at	timestamp	NOT NULL, DEFAULT CURRENT_TIMESTAMP	Timestamp of appointment creation.

Table 12: patient_encounters

Stores details from a specific patient-doctor encounter (SOAP notes).

Column	Type	Constraints	Description
id	int(11)	Primary Key, AUTO_INCREMENT	Unique identifier for the encounter.
appointment_id	int(11)	Foreign Key to appointments.id, NOT NULL	Links to the parent appointment.
patient_id	int(11)	NOT NULL	The ID of the patient.
doctor_id	int(11)	NOT NULL	The ID of the doctor.
encounter_date	timestamp	NOT NULL, DEFAULT CURRENT_TIMESTAMP	The date and time of the encounter.
chief_complaint	text		The patient's main reason for the visit.

vitals	json		JSON object storing vitals (e.g., BP, temp).
soap_subjective	text		SOAP Note: Subjective.
soap_objective	text		SOAP Note: Objective.
soap_assessment	text		SOAP Note: Assessment.
soap_plan	text		SOAP Note: Plan.
diagnosis_icd10	varchar(255)		ICD-10 Diagnosis code(s).

Table 13: prescriptions

Stores prescription headers.

Column	Type	Constraints	Description
id	int(11)	Primary Key, AUTO_INCREMENT	Unique identifier for the prescription.
patient_id	int(11)	Foreign Key to users.id, NOT NULL	The patient receiving the prescription.
doctor_id	int(11)	Foreign Key to users.id, NOT NULL	The prescribing doctor.

admission_id	INT(11)	Foreign Key to admissions.id	Links the prescription to a hospital admission.
encounter_id	INT(11)		Links to a specific patient encounter.
prescription_date	date	NOT NULL	The date of the prescription.
status	enum('pending','dispensed','partial','cancelled')	NOT NULL, DEFAULT 'pending'	The current status of the prescription.
notes	text		Additional notes from the doctor.
created_at	timestamp	NOT NULL, DEFAULT CURRENT_TIMESTAMP	Timestamp of prescription creation.

Table 14: prescription_items

Stores the individual medicine items for each prescription.

Column	Type	Constraints	Description
id	int(11)	Primary Key, AUTO_INCREMENT	Unique identifier for the item.
prescription_id	int(11)	Foreign Key to prescriptions.id, NOT NULL	Links to the parent prescription.

medicine_id	int(11)	Foreign Key to medicines.id, NOT NULL	The prescribed medicine.
dosage	varchar(100)	NOT NULL	The prescribed dosage (e.g., "500mg").
frequency	varchar(100)	NOT NULL	How often to take it (e.g., "Twice a day").
quantity_prescribed	int(11)	NOT NULL, DEFAULT 1	The amount of medicine prescribed.
quantity_dispensed	int(11)	NOT NULL, DEFAULT 0	The amount of medicine dispensed.
is_dispensed	tinyint(1)	NOT NULL, DEFAULT 0	1 if the pharmacy has dispensed this item.

Table 15: lab_orders

Stores patient lab test orders and results.

Column	Type	Constraints	Description
id	int(11)	Primary Key, AUTO_INCREMENT	Unique identifier for the lab order.
patient_id	int(11)	Foreign Key to users.id, NOT NULL	The patient who was tested.
doctor_id	int(11)	Foreign Key to users.id	The doctor who ordered the test.

staff_id	int(11)	Foreign Key to users.id	Staff member who processed the result.
encounter_id	INT(11)		Links to a specific patient encounter.
ordered_at	timestamp	NOT NULL, DEFAULT CURRENT_TIMESTAMP	When the test was ordered.
test_name	varchar(255)	NOT NULL	The name of the test performed.
test_date	date		The date the test was conducted.
status	enum('ordered','pending','processing','completed')	NOT NULL, DEFAULT 'ordered'	The status of the lab order.
result_details	text		The detailed results of the test.
cost	decimal(10,2)	NOT NULL, DEFAULT 0.00	The cost of the lab test.
attachment_path	varchar(255)		Path to an uploaded file (e.g., PDF).
created_at	timestamp	NOT NULL, DEFAULT CURRENT_TIMESTAMP	Timestamp of result entry.

Table 16: discharge_clearance

Tracks clearance steps for the automated discharge process.

Column	Type	Constraints	Description
id	INT(11)	Primary Key, AUTO_INCREMENT	Unique ID for the clearance record.
admission_id	INT(11)	Foreign Key to admissions.id, NOT NULL	The admission being cleared.
created_at	TIMESTAMP	NOT NULL, DEFAULT CURRENT_TIMESTAMP	Timestamp of creation.
clearance_step	ENUM('nursing', 'pharmacy', 'billing')	NOT NULL	The department that needs to give clearance.
is_cleared	TINYINT(1)	NOT NULL, DEFAULT 0	1 if the step is cleared.
cleared_by_user_id	INT(11)	Foreign Key to users.id	The user who cleared the step.
cleared_at	TIMESTAMP		Timestamp of when clearance was given.
notes	TEXT		Any notes related to the clearance.
discharge_date	**DATE**		**Date of discharge, for summary PDF.**

summary_text	**TEXT**		**Narrative summary for discharge PDF.**
doctor_id	INT(11)	Foreign Key to users.id	Doctor who authored the summary.
Composite Key	(admission_id, clearance_step)	Unique	Ensures each clearance step is unique for an admission.

Inventory & Financials

Table 17: medicines

Manages the medicine inventory.

Column	Type	Constraints	Description
id	INT(11)	Primary Key, AUTO_INCREMENT	Unique identifier for the medicine.
name	VARCHAR(255)	Unique, NOT NULL	The name of the medicine.
description	TEXT		A description of the medicine.
quantity	INT(11)	NOT NULL, DEFAULT 0	The current stock quantity.
unit_price	DECIMAL(10, 2)	NOT NULL, DEFAULT 0.00	The price per unit.
low_stock_threshold	INT(11)	NOT NULL, DEFAULT 10	Threshold for low stock alerts.
created_at	TIMESTAMP	NOT NULL, DEFAULT CURRENT_TIMESTAMP	Timestamp of creation.
updated_at	TIMESTAMP	NOT NULL, DEFAULT CURRENT_TIMESTAMP	Timestamp of last update.

Table 18: blood_inventory

Manages the blood bank inventory.

Column	Type	Constraints	Description
id	INT(11)	Primary Key, AUTO_INCREMENT	Unique ID.
blood_group	ENUM('A+', 'A-', 'B+', 'B-', 'AB+', 'AB-', 'O+', 'O-')	Unique, NOT NULL	The blood group type.
quantity_ml	INT(11)	NOT NULL, DEFAULT 0	Quantity in milliliters.
low_stock_threshold_ml	INT(11)	NOT NULL, DEFAULT 5000	Low stock threshold in ml.
last_updated	TIMESTAMP	NOT NULL, DEFAULT CURRENT_TIMESTAMP	Timestamp of the last update.

Table 19: transactions

Logs all financial transactions.

Column	Type	Constraints	Description
id	int(11)	Primary Key, AUTO_INCREMENT	Unique transaction ID.
user_id	int(11)	Foreign Key to users.id, NOT NULL	The user associated with the transaction.

admission_id	INT(11)	Foreign Key to admissions.id	Links the transaction to a hospital admission.
description	varchar(255)	NOT NULL	A description of the transaction.
amount	decimal(10,2)	NOT NULL	The transaction amount.
type	enum('payment','refund')	NOT NULL, DEFAULT 'payment'	The type of transaction.
status	enum('pending','paid')	NOT NULL, DEFAULT 'pending'	The payment status.
payment_mode	**enum('unpaid','cash','card','online', 'upi')**	**NOT NULL, DEFAULT 'unpaid'**	**The mode of payment.**
created_at	timestamp	NOT NULL, DEFAULT CURRENT_TIMESTAMP	Timestamp of creation.
paid_at	timestamp		Timestamp when payment was completed.

Table 20: pharmacy_bills

Links prescriptions to financial transactions, creating a bill for dispensed medicines.

Column	Type	Constraints	Description
id	INT(11)	Primary Key, AUTO_INCREMENT	Unique ID for the pharmacy bill.

prescription_id	INT(11)	Unique, Foreign Key to prescriptions.id, NOT NULL	The prescription being billed.
transaction_id	INT(11)	Foreign Key to transactions.id, NOT NULL	The associated record in the transactions table.
created_by_staff_id	INT(11)	Foreign Key to users.id, NOT NULL	The staff member who generated the bill.
total_amount	DECIMAL(10, 2)	NOT NULL	The total cost of the prescribed items.
created_at	TIMESTAMP	NOT NULL, DEFAULT CURRENT_TIMESTAMP	Timestamp of bill creation.

Communication & System

Table 21: conversations

Defines a communication channel between two users.

Column	Type	Constraints	Description
id	int(11)	Primary Key, AUTO_INCREMENT	Unique ID for the conversation.
user_one_id	int(11)	Foreign Key to users.id, NOT NULL	The first user in the conversation.
user_two_id	int(11)	Foreign Key to users.id, NOT NULL	The second user in the conversation.
created_at	timestamp	NOT NULL, DEFAULT CURRENT_TIMESTAMP	Timestamp of creation.
updated_at	timestamp	NOT NULL, DEFAULT CURRENT_TIMESTAMP	Timestamp of the last message.
Composite Key	(user_one_id, user_two_id)	Unique	Ensures only one conversation exists between two users.

Table 22: messages

Stores individual messages within a conversation.

Column	Type	Constraints	Description
id	int(11)	Primary Key, AUTO_INCREMENT	Unique message ID.
conversation_id	int(11)	Foreign Key to conversations.id, NOT NULL	The conversation this message belongs to.
sender_id	int(11)	Foreign Key to users.id, NOT NULL	The user who sent the message.
receiver_id	int(11)	Foreign Key to users.id, NOT NULL	The user who received the message.
message_text	text	NOT NULL	The content of the message.
is_read	tinyint(1)	NOT NULL, DEFAULT 0	1 if the receiver has read the message.
created_at	timestamp	NOT NULL, DEFAULT CURRENT_TIMESTAMP	Timestamp of message creation.

Table 23: notifications

Stores system-generated notifications for users.

Column	Type	Constraints	Description
id	int(11)	Primary Key, AUTO_INCREMENT	Unique notification ID.
sender_id	int(11)	Foreign Key to users.id, NOT NULL	The user who initiated the notification.
recipient_role	varchar(50)		The role of the recipient (for role-based notifications).
recipient_user_id	int(11)	Foreign Key to users.id	The specific user to be notified.
message	text	NOT NULL	The notification message.
is_read	tinyint(1)	NOT NULL, DEFAULT 0	1 if the notification has been read.
created_at	timestamp	NOT NULL, DEFAULT CURRENT_TIMESTAMP	Timestamp of creation.

Table 24: callback_requests

Stores requests from users for a callback.

Column	Type	Constraints	Description
id	int(11)	Primary Key, AUTO_INCREMENT	Unique ID for the request.
name	varchar(100)	NOT NULL	Name of the person requesting a callback.
phone	varchar(20)	NOT NULL	Phone number for the callback.
created_at	timestamp	NOT NULL, DEFAULT CURRENT_TIMESTAMP	Timestamp of the request.
is_contacted	tinyint(1)	NOT NULL, DEFAULT 0	1 if the user has been contacted.

System & Utility Tables

Table 25: activity_logs

Logs user actions for auditing purposes.

Column	Type	Constraints	Description
id	int(11)	Primary Key, AUTO_INCREMENT	Unique log entry ID.
user_id	int(11)	Foreign Key to users.id, NOT NULL	The user who performed the action.
action	varchar(255)	NOT NULL	e.g., 'create_user', 'update_medicine'.
target_user_id	int(11)	Foreign Key to users.id,	The user who was affected by the action.
details	text		A human-readable description of the action.
created_at	timestamp	NOT NULL, DEFAULT CURRENT_TIMESTAMP	Timestamp of the logged action.

Table 26: system_settings

Stores system-wide settings and configurations.

Column	Type	Constraints	Description
setting_key	VARCHAR(255)	Primary Key	The name of the setting.
setting_value	TEXT	NOT NULL	The value of the setting.

Table 27: ip_tracking

Stores login attempts and IP addresses for security.

Column	Type	Constraints	Description
id	int(11)	Primary Key, AUTO_INCREMENT	Unique identifier for the log.
user_id	int(11)	Foreign Key to users.id, NOT NULL	The user who logged in.
ip_address	varchar(45)	NOT NULL	The IP address used for login.
login_time	timestamp	NOT NULL, DEFAULT CURRENT_TIMESTAMP	The timestamp of the login.
name	varchar(255)		The name of the user (for easy reference).

Table 28: ip_blocks

Manages blocked IP addresses.

Column	Type	Constraints	Description
id	int(11)	Primary Key, AUTO_INCREMENT	Unique identifier for the block.
ip_address	varchar(45)	Unique, NOT NULL	The IP address to block.
reason	varchar(255)		The reason for blocking the IP.
created_at	timestamp	NOT NULL, DEFAULT CURRENT_TIMESTAMP	Timestamp when the block was created.

Patient Experience

Table 29: feedback

Stores patient feedback regarding appointments and hospital stays.

Column	Type	Constraints	Description
id	INT(11)	Primary Key, AUTO_INCREMENT	Unique ID for the feedback entry.
patient_id	INT(11)	Foreign Key to users.id, NOT NULL	The patient providing the feedback.
appointment_id	INT(11)	Foreign Key to appointments.id,	Links feedback to a specific appointment.
admission_id	INT(11)	Foreign Key to admissions.id,	Links feedback to a hospital stay.
overall_rating	TINYINT(1)	NOT NULL	Overall experience rating from 1 to 5.
doctor_rating	TINYINT(1)		Specific rating for the doctor (1-5).
nursing_rating	TINYINT(1)		Specific rating for the nursing staff (1-5).
staff_rating	TINYINT(1)		Specific rating for other hospital staff (1-5).

cleanliness_rating	TINYINT(1)		Rating for hospital cleanliness (1-5).
comments	TEXT		Detailed comments or suggestions.
feedback_type	ENUM('Suggestion', 'Complaint', 'Praise')	NOT NULL, DEFAULT 'Suggestion'	The category of the feedback.
is_anonymous	TINYINT(1)	NOT NULL, DEFAULT 0	1 if the feedback is submitted anonymously.
created_at	TIMESTAMP	NOT NULL, DEFAULT CURRENT_TIMESTAMP	Timestamp of when feedback was submitted.

3.3 Process Design - Dataflow Diagrams

Process design for the MedSync platform is modeled using Data Flow Diagrams (DFDs) to visually represent the flow of information and the transformations that occur as data moves through the system. DFDs are essential for understanding how different processes interact with data stores and external entities (users). They provide a clear, high-level view of the system's operations without detailing the implementation logic.

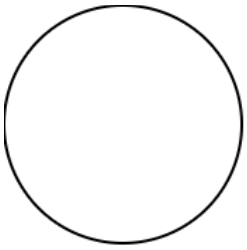
The DFDs for MedSync are structured in levels, starting from a context-level diagram (Level 0) that shows the entire system as a single process, and breaking down into more detailed diagrams (Level 1, Level 2, etc.) for specific functions like appointment booking and patient discharge.

Data Flow Diagram Symbols The following standard symbols are used in the Data Flow Diagrams to represent the system's components:

- **External Entity (Rectangle):** Represents a source or destination of data (e.g., Patient, Administrator).

- **Data Flow (Arrow):** Represents the movement of data between entities, processes, and data stores.

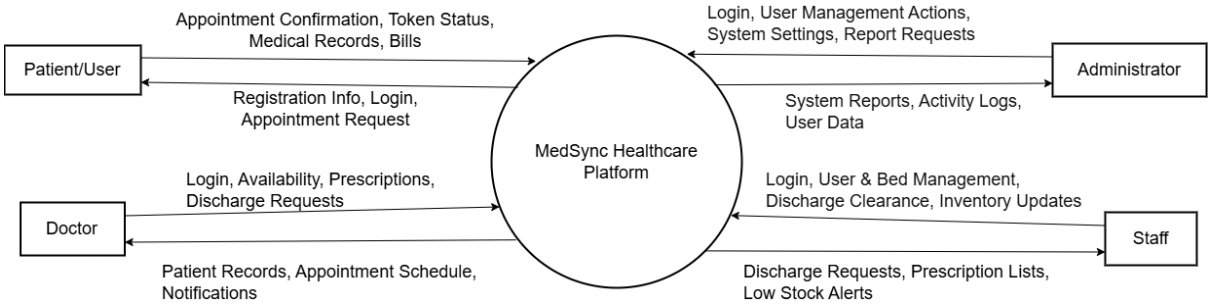
- **Process (Circle/Rounded Rectangle):** Represents a task or function that transforms incoming data into outgoing data.



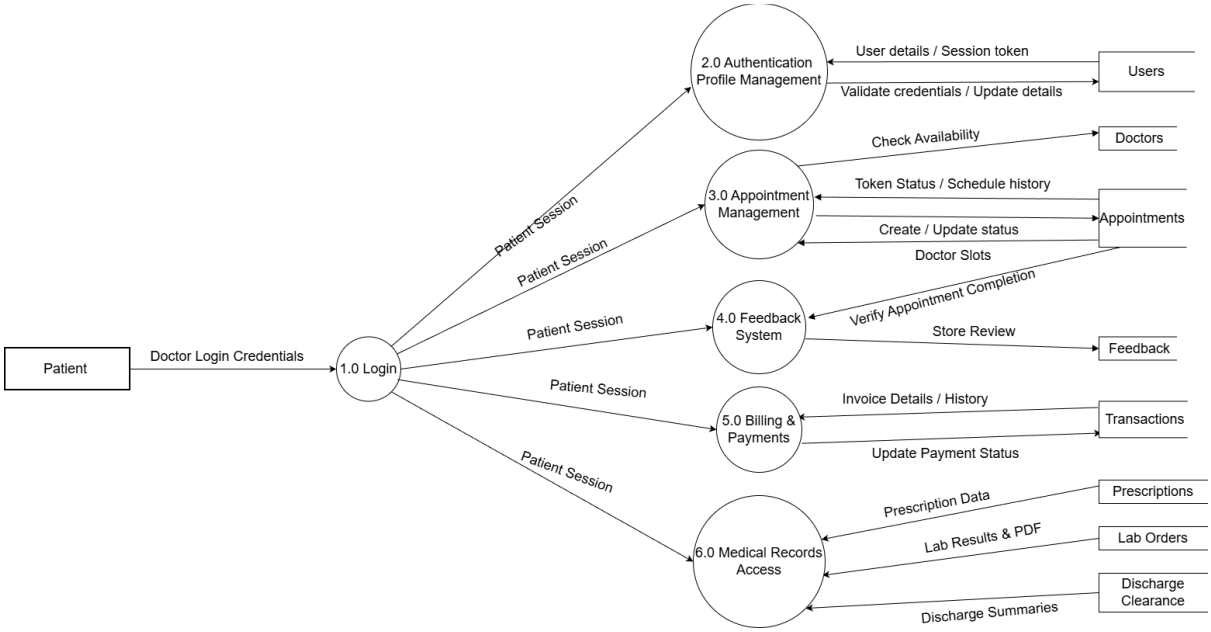
- **Data Store (Open Rectangle):** Represents a repository where data is stored for later use (e.g., Database tables).

3.3.1 Context-Level Diagram (DFD Level 0)

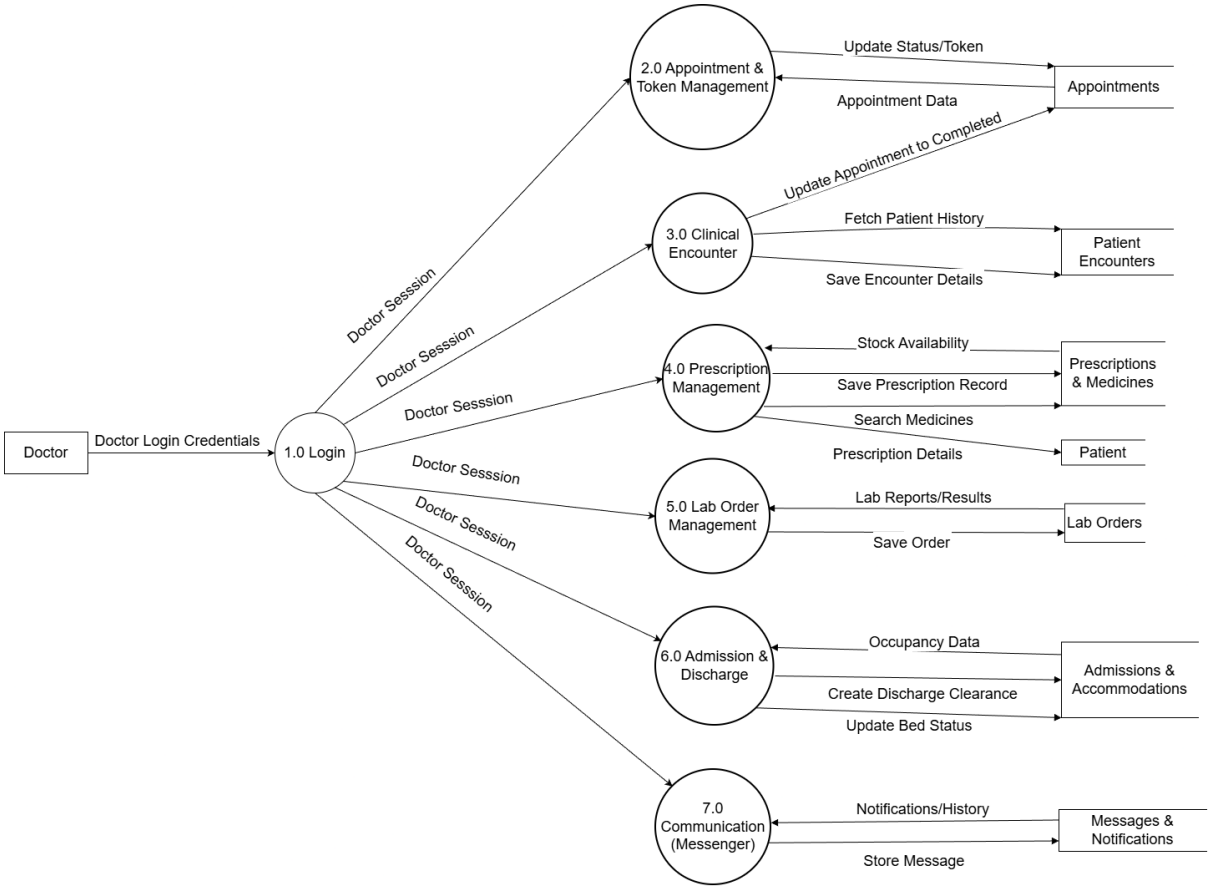
The context-level diagram illustrates the interaction between the MedSync system and its external entities: **Patient**, **Doctor**, **Staff**, and **Administrator**. It shows the main data inputs and outputs for each entity. For example, a Patient provides login credentials and appointment requests, and in return, receives appointment confirmations and live token status.



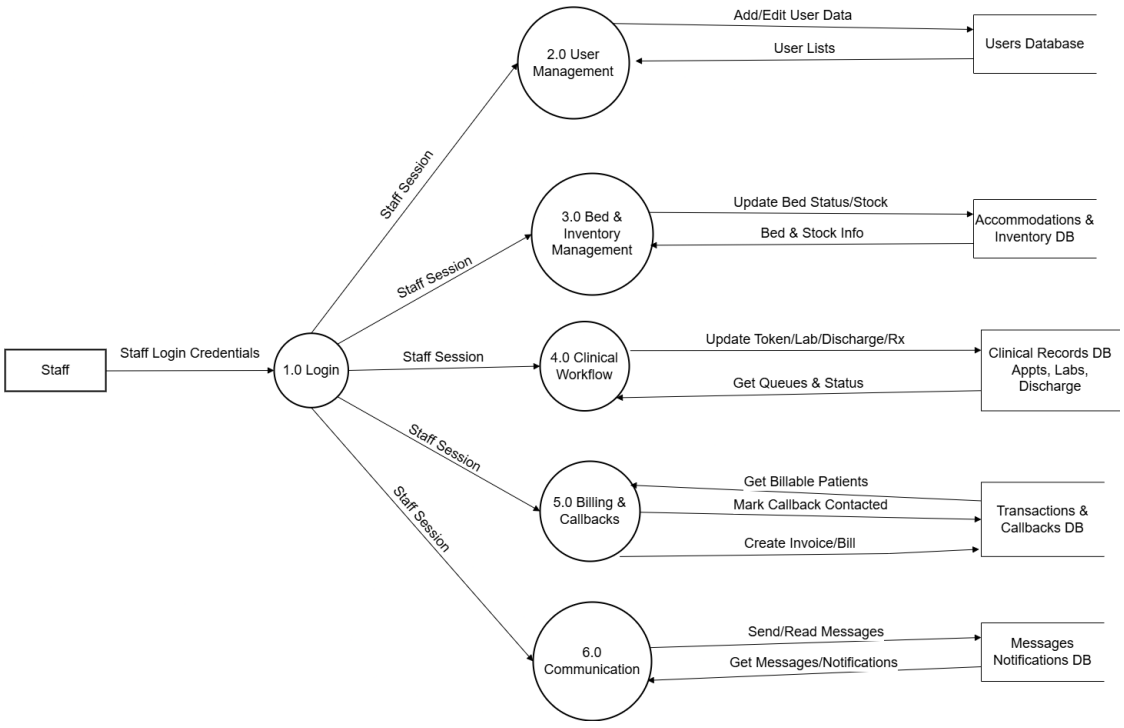
3.3.2 Level 1 DFD for Patient



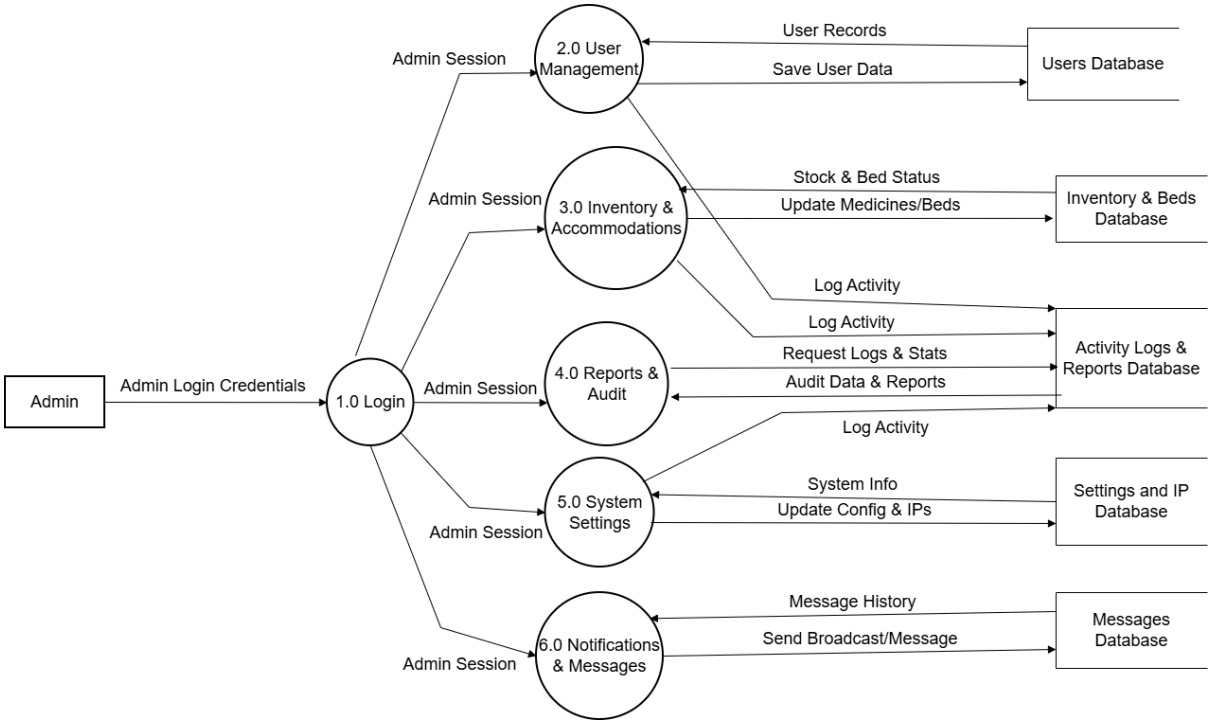
3.3.3 Level 1 DFD for Doctor



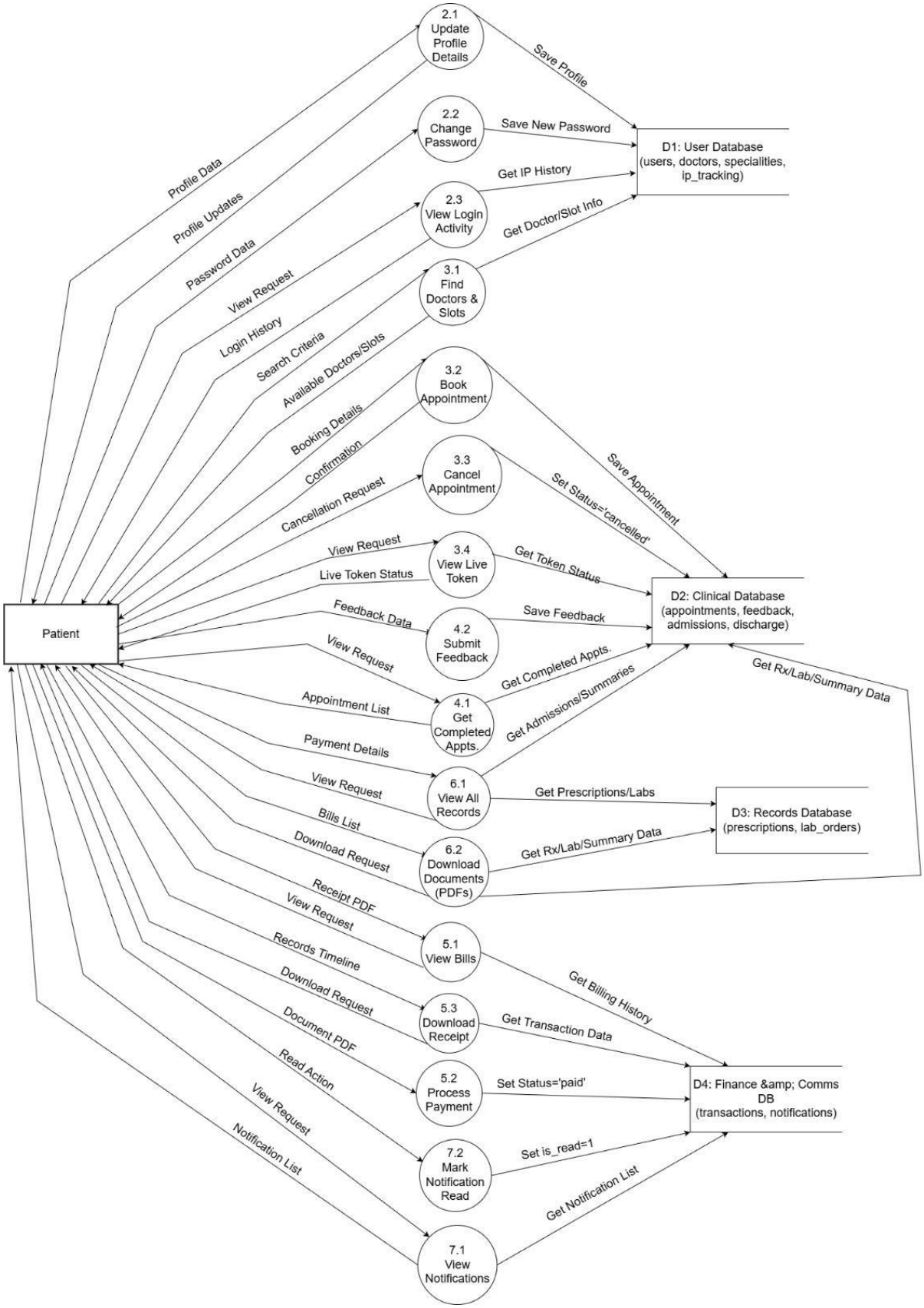
3.3.4 Level 1 DFD for Staff



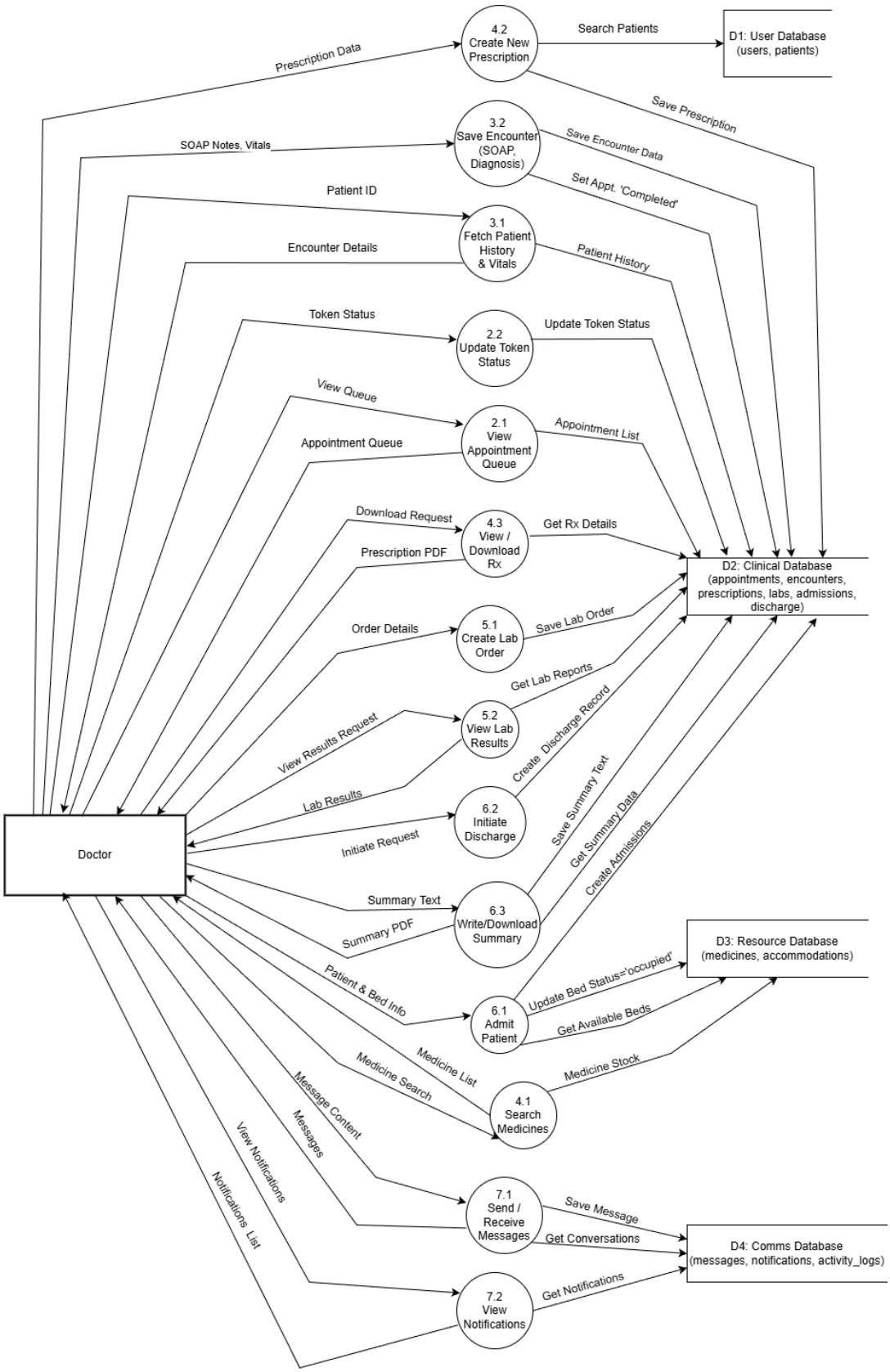
3.3.5 Level 1 DFD for Admin



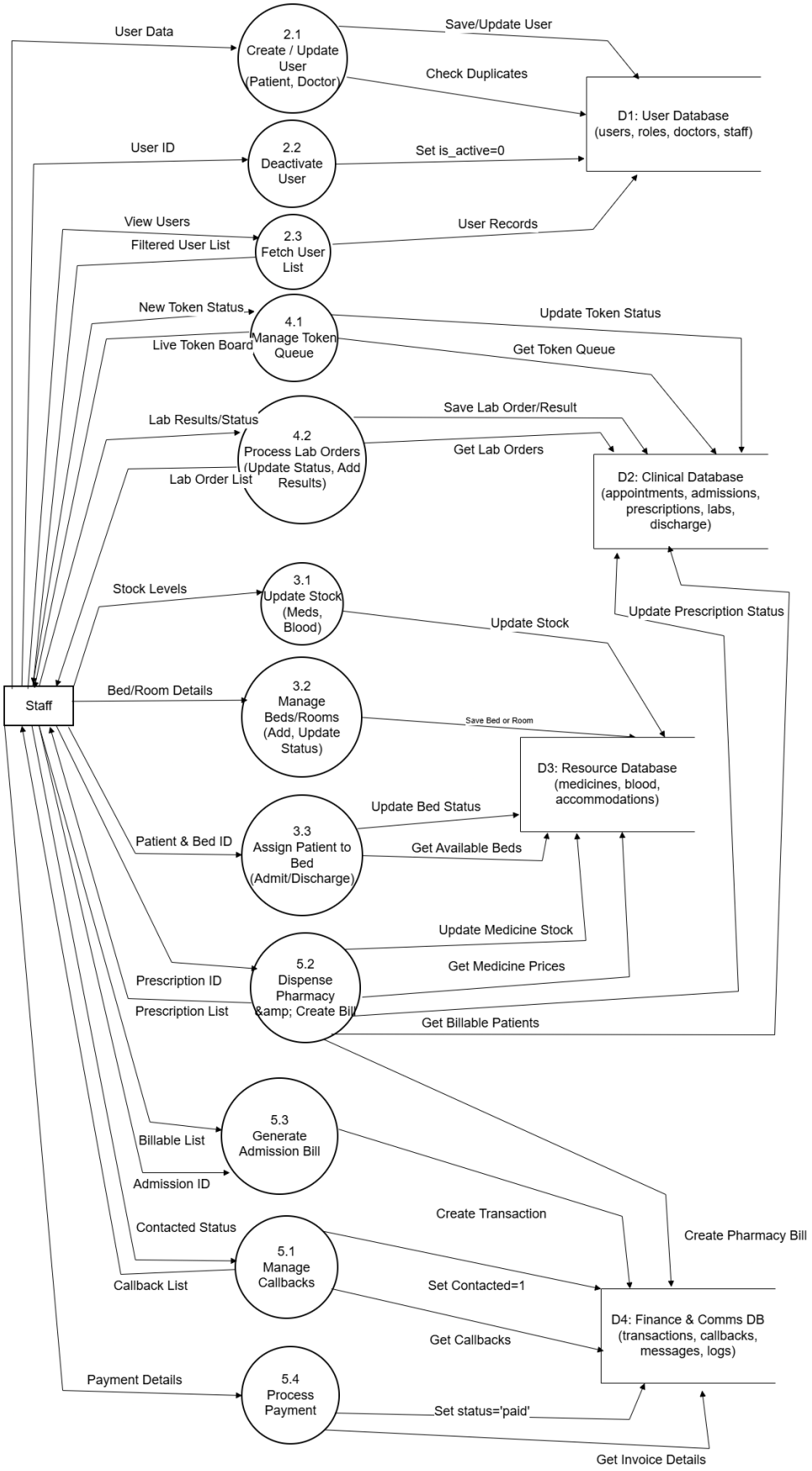
3.3.6 Level 2 DFD for Patient



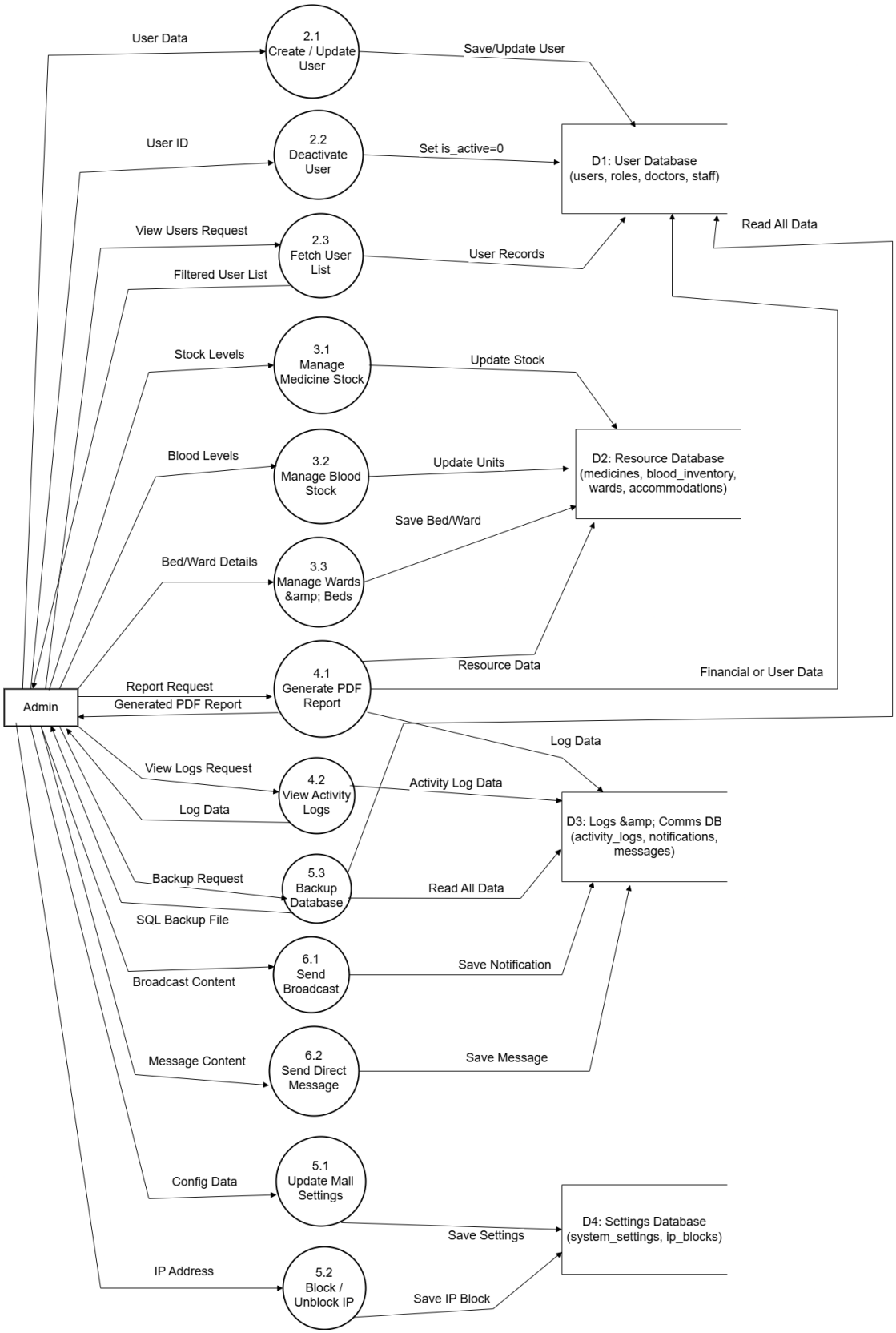
3.3.7 Level 2 DFD for Doctor



3.3.8 Level 2 DFD for Staff



3.3.9 Level 2 DFD for Admin



3.4 Object Oriented Design - UML Diagrams

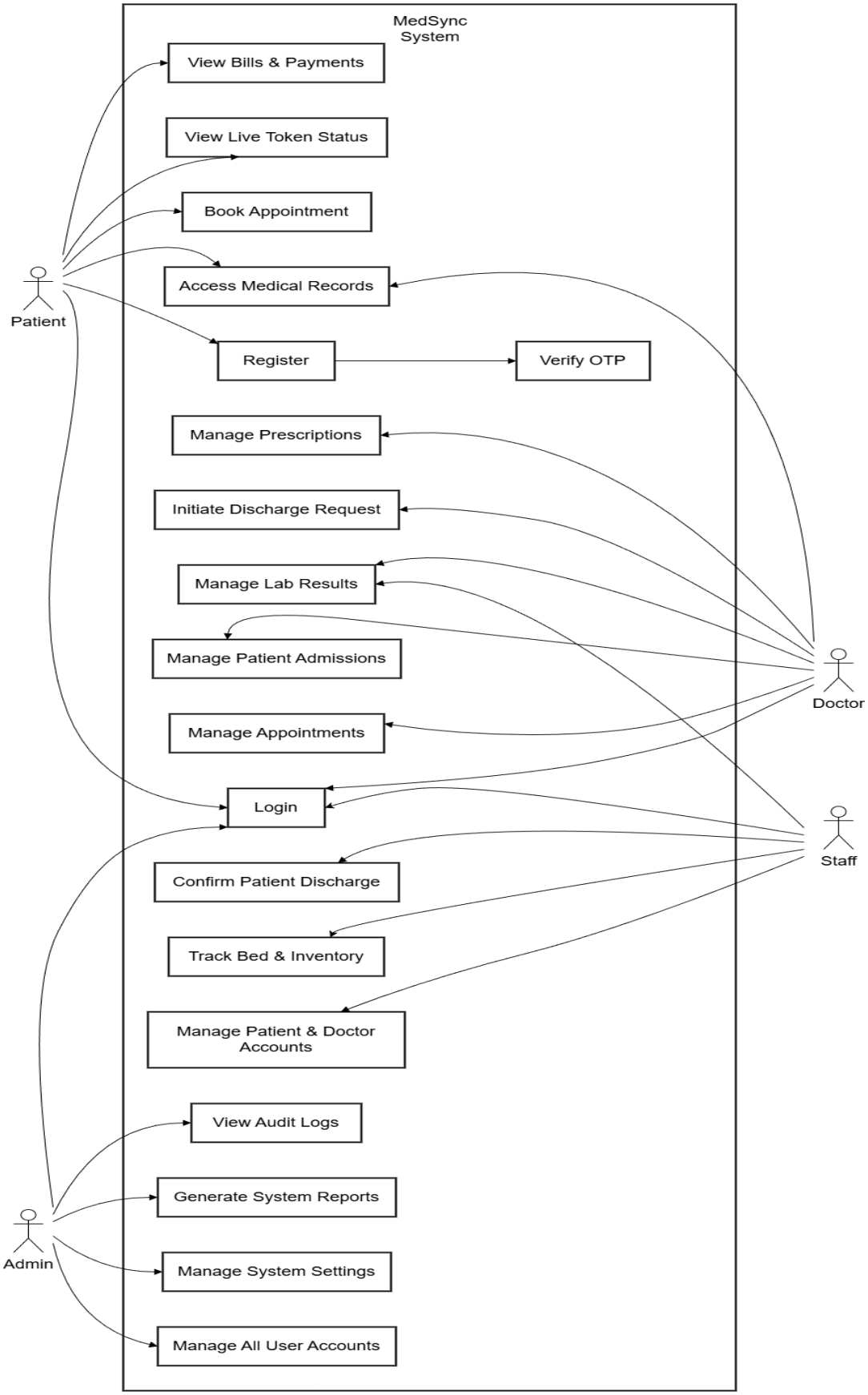
Object-Oriented Design (OOD) is a crucial methodology used in the MedSync platform to model the system as a collection of interacting objects. The Unified Modeling Language (UML) is employed to create a set of diagrams that visually represent the system's architecture, behavior, and interactions. This approach helps in understanding the system's structure, identifying responsibilities of different components, and ensuring a modular and maintainable design. For MedSync, key UML diagrams, including Use Case, Class, and Sequence diagrams, are used to provide a comprehensive view of the system's design.

3.4.1 Use Case Diagram

The Use Case diagram captures the functional requirements of the MedSync platform from the perspective of its users (actors). It illustrates the interactions between the actors (Patient, Doctor, Staff, Administrator) and the system to achieve specific goals. This diagram is fundamental for defining the scope of the system and ensuring that all user requirements are met.

Key use cases include:

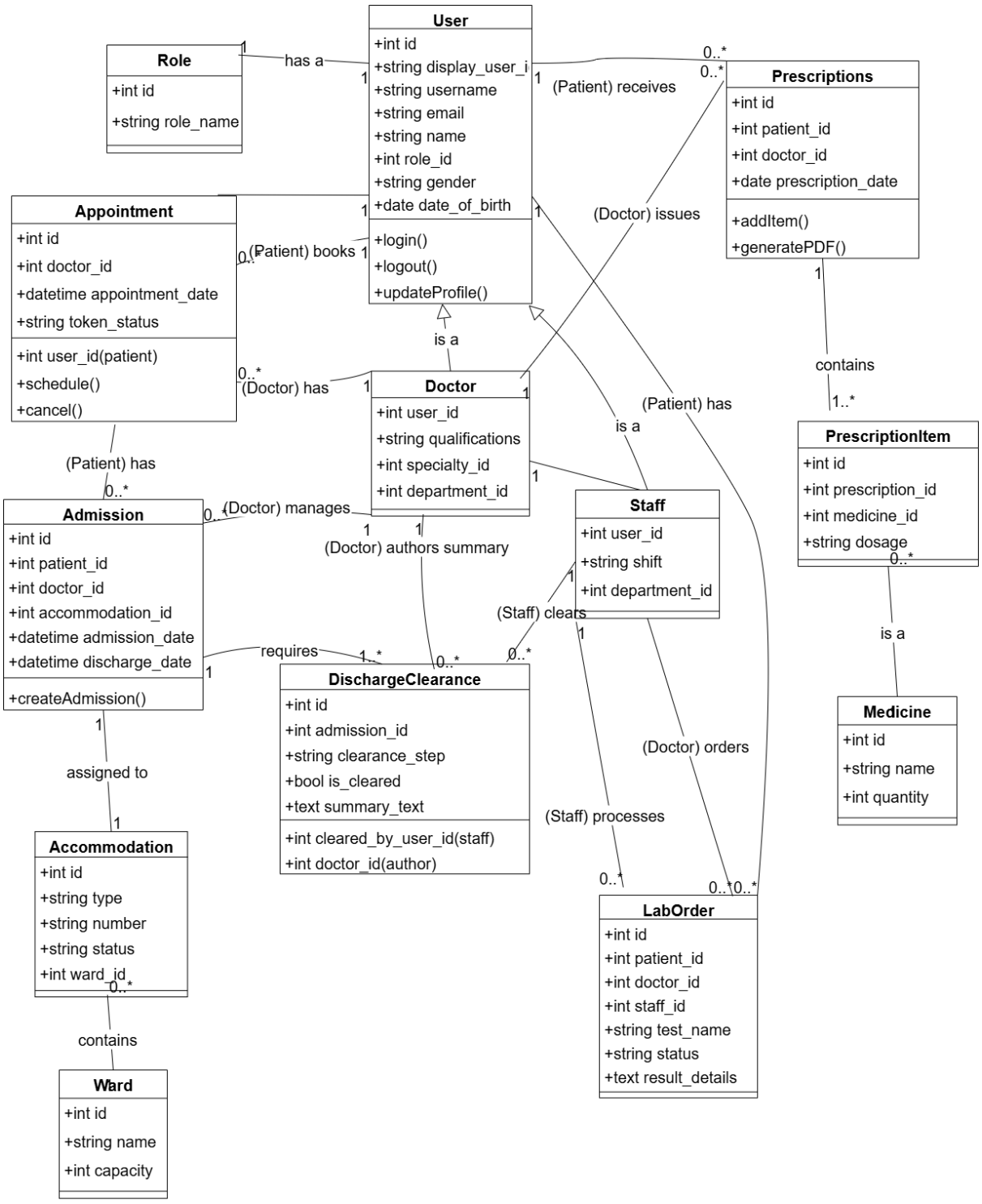
- **Manage Account:** Performed by all actors to handle their profiles and security settings.
- **Book Appointment:** Performed by the Patient.
- **Manage Schedule:** Performed by the Doctor.
- **Initiate Discharge:** Performed by the Doctor.
- **Process Discharge:** Performed by Staff, involving multiple clearance steps.
- **Manage Users:** Performed by Administrators and Staff to add or edit user accounts.


MedSync System
View Bills & Payments
View Live Token Status
Book Appointment
Access Medical Records
Register
Verify OTP
Manage Prescriptions
Initiate Discharge Request
Manage Lab Results
Manage Patient Admissions
Manage Appointments
Login
Confirm Patient Discharge
Track Bed & Inventory
Manage Patient & Doctor Accounts
View Audit Logs
Generate System Reports
Manage System Settings
Manage All User Accounts
Patient
Doctor
Staff
Admin

3.4.2 Class Diagram

The Class diagram provides a static view of the system's structure by showing its classes, their attributes, methods, and the relationships between them. For MedSync, the Class diagram defines key classes such as

User, Doctor, Patient, Appointment, Prescription, and Accommodation. It details how these classes are related (e.g., a Doctor is a specialized type of User, and a Patient can have multiple Appointments). This diagram serves as a blueprint for the implementation phase, ensuring a well-structured and logical codebase.

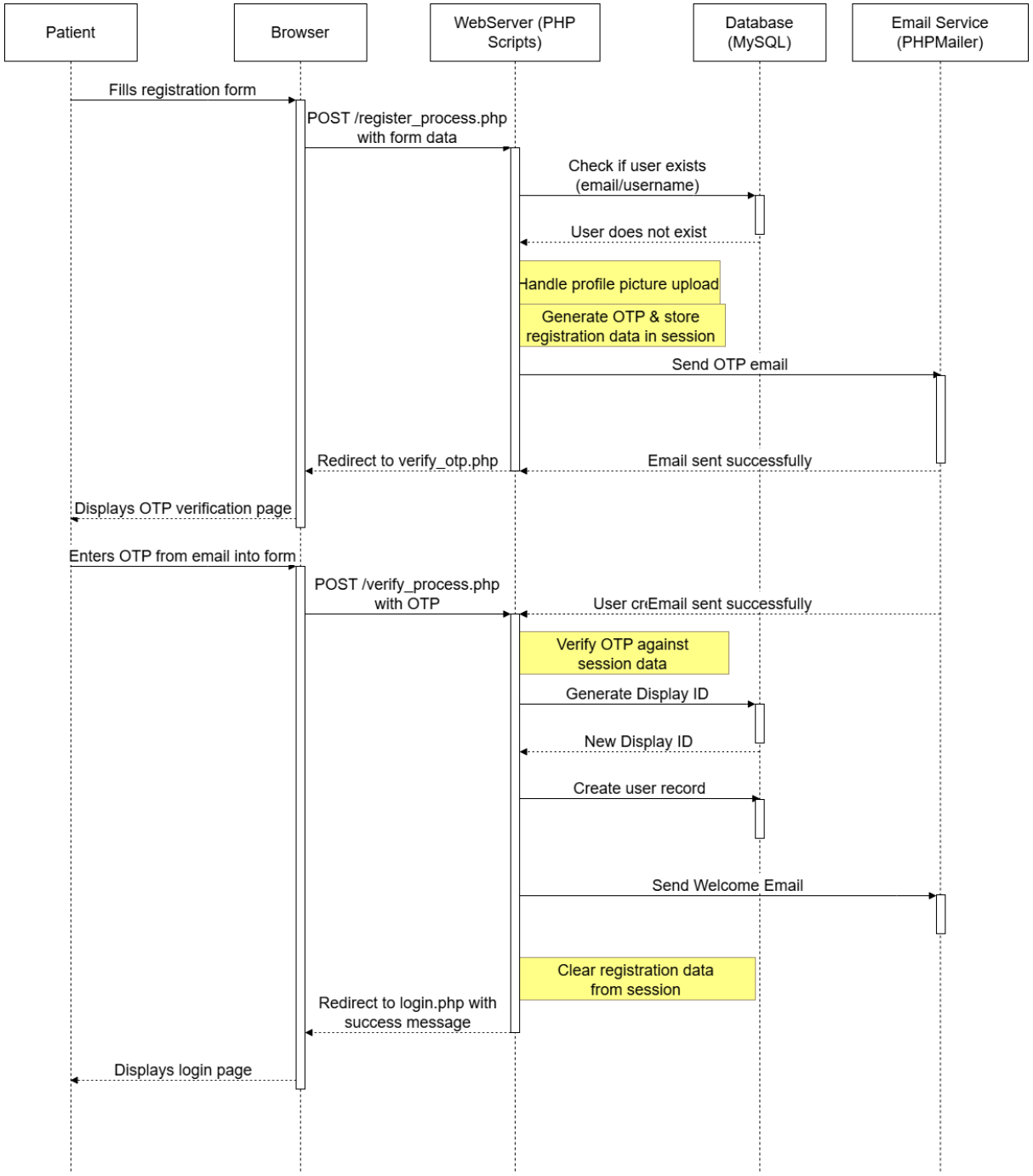

User
+int id
+string display_user_i
+string username
+string email
+string name
+int role_id
+string gender
+date date_of_birth
+login()
+logout()
+updateProfile()
Role
+int id
+string role_name
has a
Prescriptions
+int id
+int patient_id
+int doctor_id
+date prescription_date
+addItem()
+generatePDF()
(Patient) receives
(Doctor) issues
Appointment
+int id
+int doctor_id
+datetime appointment_date
+string token_status
+int user_id(patient)
+schedule()
+cancel()
(Patient) books
(Doctor) has
is a
Doctor
+int user_id
+string qualifications
+int specialty_id
+int department_id
contains
PrescriptionItem
+int id
+int prescription_id
+int medicine_id
+string dosage
(Patient) has
Admission
+int id
+int patient_id
+int doctor_id
+int accommodation_id
+datetime admission_date
+datetime discharge_date
+createAdmission()
(Doctor) manages
(Doctor) authors summary
Staff
+int user_id
+string shift
+int department_id
(Patient) has
is a
(Staff) clears
requires
DischargeClearance
+int id
+int admission_id
+string clearance_step
+bool is_cleared
+text summary_text
+int cleared_by_user_id(staff)
+int doctor_id(author)
Medicine
+int id
+string name
+int quantity
(Doctor) orders
(Staff) processes
assigned to
Accommodation
+int id
+string type
+string number
+string status
+int ward_id
contains
Ward
+int id
+string name
+int capacity
LabOrder
+int id
+int patient_id
+int doctor_id
+int staff_id
+string test_name
+string status
+text result_details

3.4.3 Sequence Diagram

Sequence diagrams model the interactions between objects in a sequential order, showing how different parts of the system collaborate to complete a function. They are used to visualize the dynamic behavior of the system for specific scenarios.

For MedSync, sequence diagrams are created for critical processes like:

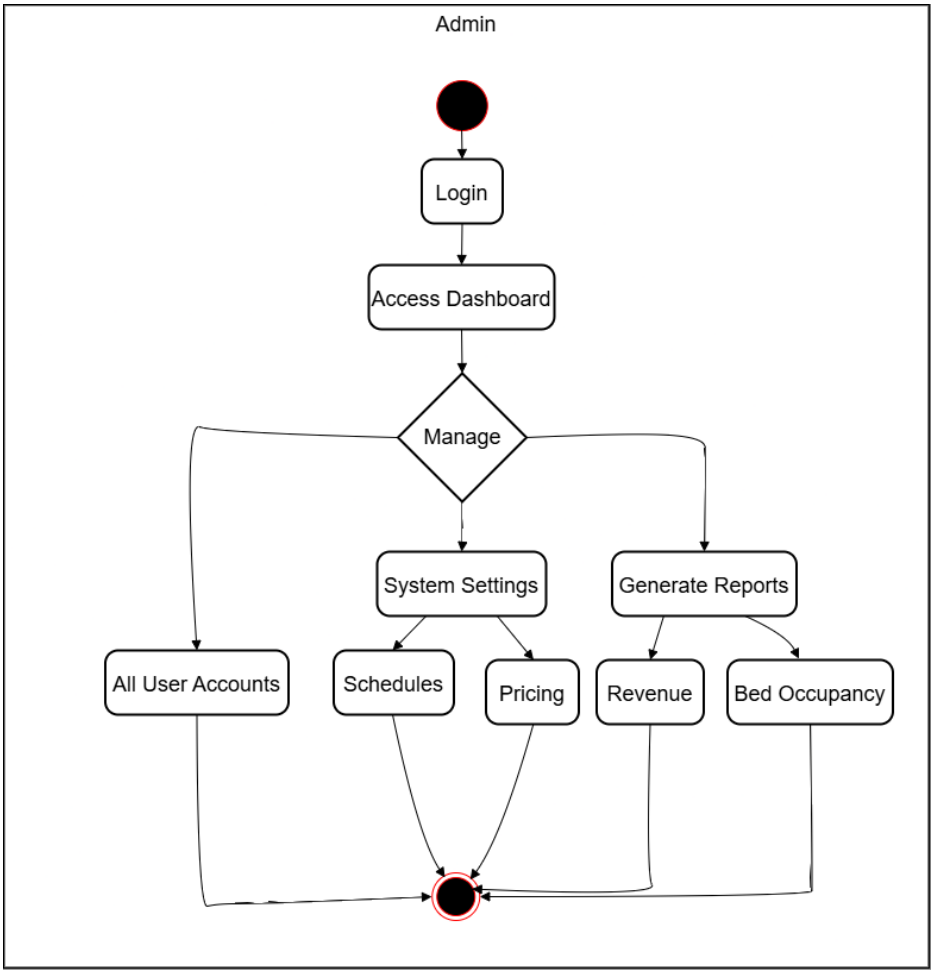
- **User Registration:** Showing the sequence of interactions between the user, the registration form, the server, the database, and the email service (PHPMailer) for OTP verification.
- **Appointment Booking:** Illustrating the steps involved when a Patient books an appointment, including checking the Doctor's availability, creating the appointment record, and sending a confirmation.
- **Discharge Process:** Detailing the messages passed between the Doctor, Staff, System, and Database objects to handle the multi-step discharge and clearance workflow.

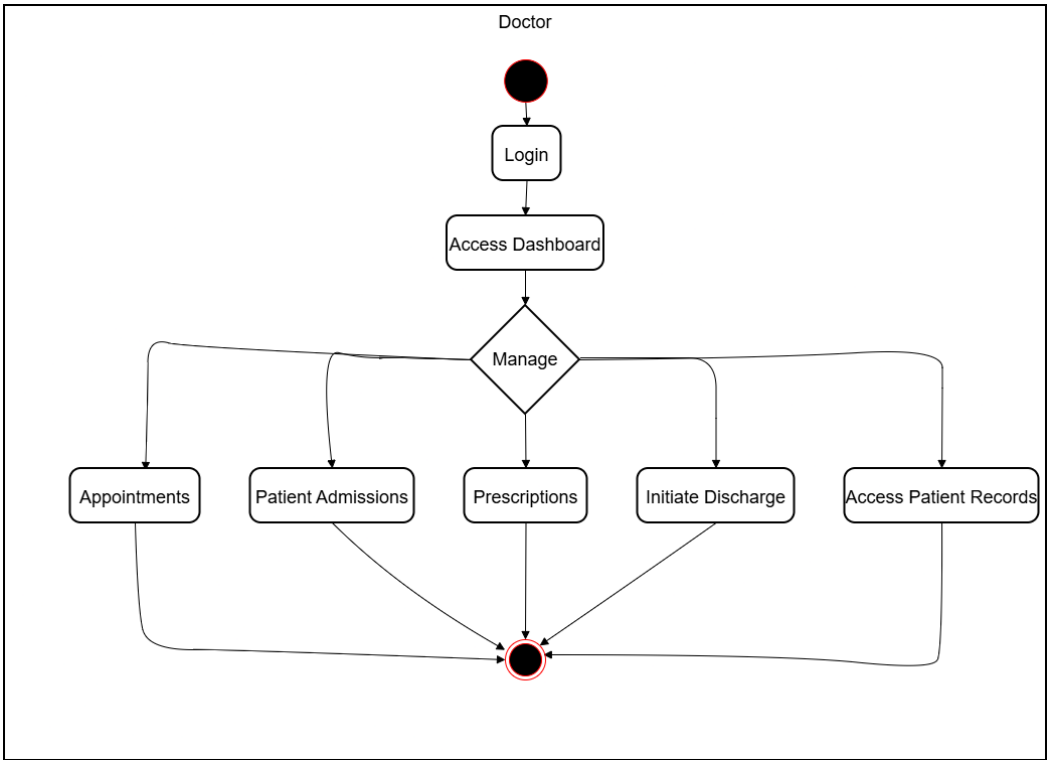

Patient
Browser
WebServer (PHP Scripts)
Database (MySQL)
Email Service (PHPMailer)
Fills registration form
POST /register_process.php with form data
Check if user exists (email/username)
User does not exist
Handle profile picture upload
Generate OTP & store registration data in session
Send OTP email
Redirect to verify_otp.php
Email sent successfully
Displays OTP verification page
Enters OTP from email into form
POST /verify_process.php with OTP
User crEmail sent successfully
Verify OTP against session data
Generate Display ID
New Display ID
Create user record
Send Welcome Email
Clear registration data from session
Redirect to login.php with success message
Displays login page

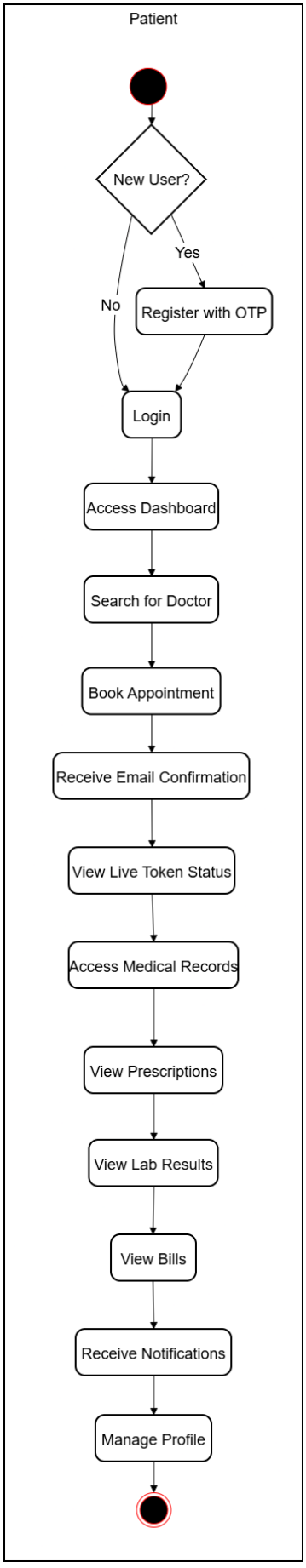
3.4.4 Activity Diagram

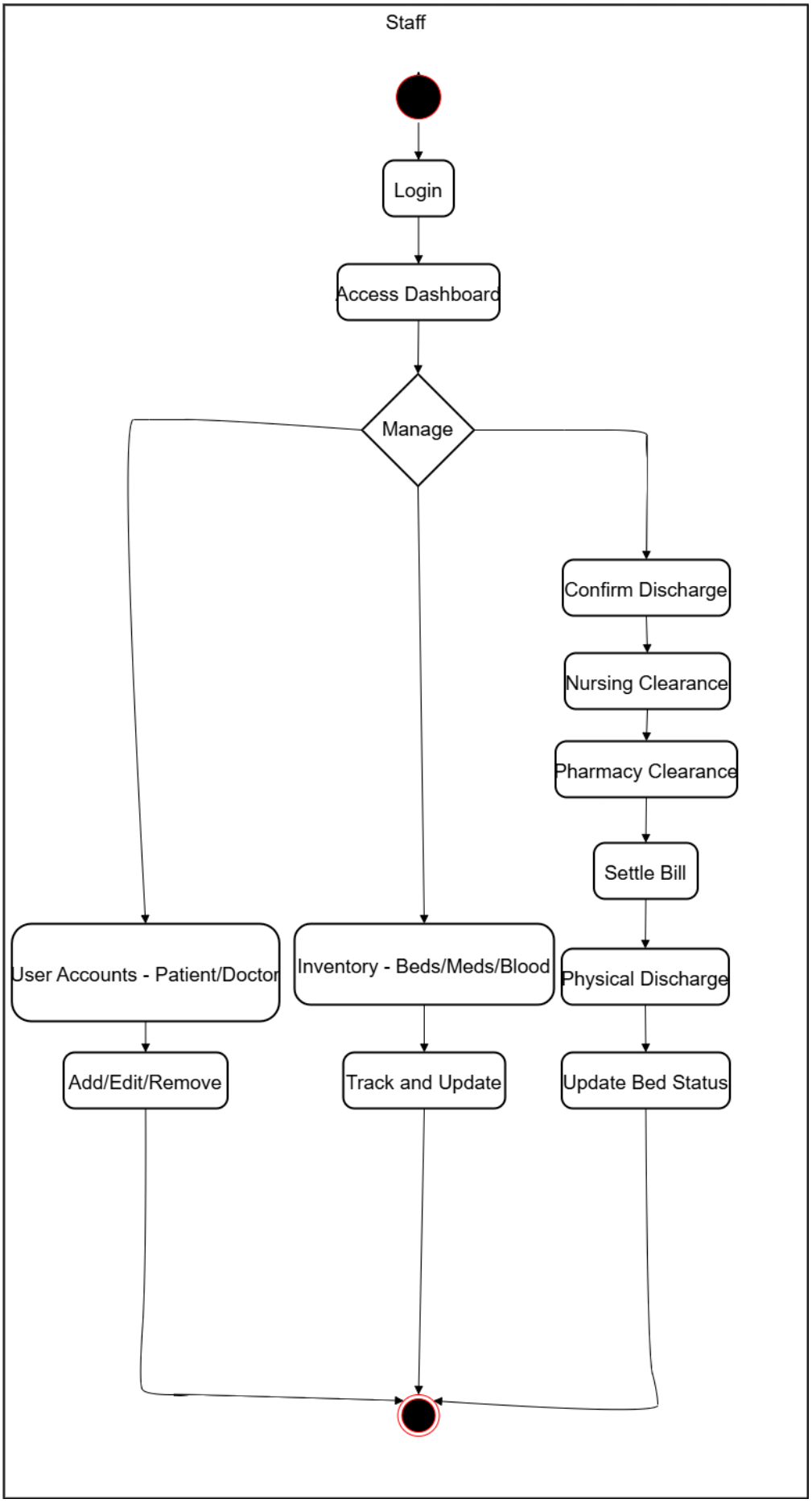
Activity diagrams are used to model the dynamic aspects of the system. They illustrate the flow of control from one activity to another. For the MedSync platform, an activity diagram is crucial for visualizing complex workflows that involve multiple actors and system components.

The diagram below models the **Automated Patient Discharge Process**. This workflow begins when a Doctor initiates a discharge and involves sequential clearances from various staff departments before the patient is finally discharged. The diagram clearly shows the responsibilities of the **Doctor** and **Staff**, the automated actions performed by the **MedSync System**, and the conditions that control the flow.


Admin
Login
Access Dashboard
Manage
System Settings
Generate Reports
All User Accounts
Schedules
Pricing
Revenue
Bed Occupancy


Doctor
Login
Access Dashboard
Manage
Appointments
Patient Admissions
Prescriptions
Initiate Discharge
Access Patient Records


Patient
New User?
Yes
No
Register with OTP
Login
Access Dashboard
Search for Doctor
Book Appointment
Receive Email Confirmation
View Live Token Status
Access Medical Records
View Prescriptions
View Lab Results
View Bills
Receive Notifications
Manage Profile


Staff
Login
Access Dashboard
Manage
Confirm Discharge
Nursing Clearance
Pharmacy Clearance
Settle Bill
User Accounts - Patient/Doctor
Inventory - Beds/Meds/Blood
Physical Discharge
Add/Edit/Remove
Track and Update
Update Bed Status

3.5 Input Design

Input design focuses on creating user-friendly and efficient interfaces for data entry. For the MedSync platform, the design prioritizes clarity, simplicity, and validation to minimize user errors and ensure high-quality data. All input forms, from user registration to prescription creation, are designed to be intuitive and responsive across various devices.

Key principles of input design in MedSync include:

- **User-Friendly Interfaces:** Forms are designed with clear, floating labels, logical grouping of fields, and placeholders to guide the user. For example, the registration form includes a password strength meter to help users create secure passwords.
- **Data Validation:** Both client-side (using JavaScript) and server-side (using PHP) validation are implemented to ensure data accuracy. This includes checks for required fields, correct data formats (e.g., email, phone number), and real-time availability checks for usernames and emails to prevent duplicates.
- **Role-Specific Inputs:** Input forms are tailored to the specific roles of the users. For instance, the "Add User" form in the administrator's dashboard includes fields for assigning roles and departments, which are not present in the patient registration form.
- **Secure Data Entry:** All forms include CSRF token protection to prevent cross-site request forgery attacks. Password fields use visibility toggles to allow users to verify their input.

3.6 Output Design

Output design is concerned with presenting information to users in a clear, meaningful, and accessible format. In MedSync, outputs are designed to provide real-time data, comprehensive reports, and standardized documents that are easy to understand and use.

Key aspects of MedSync's output design include:

- **Real-Time Dashboards:** Role-based dashboards present key information at a glance. The administrator dashboard, for example, displays statistics on user roles and low-stock alerts, while the patient dashboard shows upcoming appointments and recent activity.
- **Standardized Reports and Documents:** The system uses the **dompdf** library to generate professional PDF documents for prescriptions, bills, lab orders, and discharge summaries. These documents have a consistent and professional layout, including the hospital's branding.
- **Data Visualization:** The reporting feature includes charts and graphs to visualize data trends for revenue, patient statistics, and resource utilization, helping administrators make informed decisions.
- **Responsive Tables:** Data is presented in responsive tables that are easily viewable on both desktop and mobile devices, ensuring accessibility for all users.

SYSTEM ENVIRONMENT

CHAPTER 4
SYSTEM ENVIRONMENT

4.1 Introduction

This chapter details the system environment required for the successful deployment and operation of the MedSync Healthcare Platform. It outlines the specific software and hardware prerequisites for both the server-side infrastructure and the client-side user access. Furthermore, this chapter specifies the tools, platforms, and technologies that were integral to the development and functionality of the system, ensuring that the platform runs efficiently, securely, and reliably. A well-defined system environment is crucial for maintainability, scalability, and seamless integration into a hospital's existing IT infrastructure.

4.2 Software Requirements Specification

To ensure optimal performance and compatibility, the following software components are required:

- **Server-Side:**
    - **Operating System:** A 64-bit version of a modern operating system. The platform is developed and tested on Windows and Linux (e.g., Ubuntu 20.04 LTS or later), but it is compatible with any OS that supports the Apache server.
    - **Web Server:** Apache 2.4 or a compatible web server (e.g., Nginx).
    - **Database Management System:** MySQL 8.0 or MariaDB 10.4 or higher.
    - **Scripting Language:** PHP version 8.0 or higher, with necessary extensions enabled (e.g., mysqli, openssl, pdo, mbstring).
    - **Dependency Manager:** Composer 2.0 or higher (for installing PHP libraries).
- **Client-Side:**
    - **Web Browser:** A modern, up-to-date web browser that supports HTML5, CSS3, and JavaScript (ES6+).
        - Google Chrome (latest version)
        - Mozilla Firefox (latest version)
        - Microsoft Edge (latest version)
        - Apple Safari (latest version)

4.3 Hardware Requirements

While the system is lightweight, the following specifications are recommended for a production environment handling a medium-sized hospital's traffic.

- **Server-Side (Recommended):**
  - **Processor:** 2.0 GHz multi-core processor (4+ cores recommended).
  - **RAM:** 4 GB or more.
  - **Storage:** 50 GB or more of SSD storage (for faster database I/O).
- **Client-Side (Minimum):**
  - Any standard computer, tablet, or mobile device capable of running a modern web browser.

4.4 Technology and Tools Used

The MedSync platform was built using a combination of open-source tools and established technologies to create a robust and scalable system.

- **Front-End Tools:**
  - **HTML:** For the structure and semantic content of the web pages.
  - **CSS:** For styling, layout, and responsive design (using Flexbox and Grid).
  - **JavaScript:** For client-side interactivity, form validation, and dynamic content updates using the **Fetch API (AJAX)**.
- **Back-End Tools:**
  - **PHP:** A server-side scripting language used for all backend processing, including user authentication, data manipulation, and API business logic.
  - **MySQL:** A relational database management system used to store all application data, including user profiles, appointments, medical records, and inventory.
- **Development Environment:**
  - **XAMPP:** A cross-platform web server solution stack used during development, bundling Apache, MySQL, and PHP.
  - **Visual Studio Code:** The primary code editor used for development.
- **External Libraries & APIs:**
  - **vlucas/phpdotenv:** A PHP library used to securely load environment variables (like database credentials and API keys) from a `.env` file, keeping them separate from the codebase.
  - **PHPMailer:** A PHP library used for sending automated emails for OTP verification, welcome messages, appointment confirmations, and other notifications.
  - **dompdf:** A PHP library used to convert HTML and CSS into PDF documents, enabling dynamic generation of invoices, lab reports, and discharge summaries.

- **kreait/firebase-php:** A PHP library for Google Firebase. It is used on the backend to securely verify the identity tokens provided by the Google Sign-In feature.
- **Google reCAPTCHA:** A Google API used on the registration page to protect the system from spam and automated bot signups.
- **Chart.js:** A JavaScript library used to render interactive, animated charts and graphs on the Admin and Staff dashboards for data visualization.

SYSTEM IMPLEMENTATION

CHAPTER 5
SYSTEM IMPLEMENTATION

5.1 Introduction

This chapter outlines the conversion of the MedSync Healthcare Platform's design into a functional web application. The implementation was guided by the designs and requirements established in previous chapters. The core technologies employed were **PHP** for backend logic, **MySQL** for the database, and **HTML, CSS, and JavaScript** for the user interface. The entire development process was conducted within a **XAMPP server environment**. The primary goal of this phase was to build a secure, modular, and maintainable codebase.

5.2 Coding

The platform's code is organized into modules based on user roles and core functionalities to ensure maintainability.

- **Directory Structure:** The project is logically divided into role-specific directories (/admin, /doctor, /staff, /user) and functional directories (/login, /register). Each role-based folder contains its own dashboard.php for the user interface, a backend logic file named api.php, styles.css, and script.js.
- **Core Configuration (config.php):** This central file handles the database connection, session initialization, CSRF token generation, and establishes a custom error handler to log all PHP errors to a log.txt file for secure debugging.
- **Backend Logic (PHP):**
    - **Role-Based Access Control (RBAC):** Every dashboard's main PHP file includes session validation to ensure only authenticated and authorized users can access the content.
    - **Database Interaction:** All SQL queries use **prepared statements** to prevent SQL injection vulnerabilities.
    - **API Endpoints:** Backend api.php files handle AJAX requests for dynamic data loading (e.g., user lists, appointments) without full page reloads.
- **Frontend Development (HTML, CSS, JS):**
    - The user interface is built with standard web technologies for a responsive and interactive experience, including features like a dark mode theme and dynamic content updates.

5.2.1 Sample Codes

1. Secure Login Processing (login/login_process.php)

This snippet showcases the secure authentication process, including CSRF validation, use of prepared statements to check against username, email, or User ID, and password verification with password_verify().

PHP

```php
<?php
// ... (Initial configuration and security checks) ...

// --- Database Authentication ---
$conn = getDbConnection();

// Query to check user by username, email, or display_user_id
$sql = "SELECT
            u.id, u.username, u.password, r.role_name AS role,
            u.display_user_id, u.is_active
        FROM users u
        JOIN roles r ON u.role_id = r.id
        WHERE (u.username = ? OR u.email = ? OR u.display_user_id = ?)
        LIMIT 1";

$stmt = $conn->prepare($sql);
$stmt->bind_param("sss", $login_identifier, $login_identifier, $login_identifier);
$stmt->execute();
$result = $stmt->get_result();

if ($result->num_rows === 1) {
    $user = $result->fetch_assoc();

    // --- Verify Password ---
    if (password_verify($password, $user['password'])) {
        // --- Check if the account is active ---
        if ($user['is_active'] == 0) {
            // ... (Handle inactive account) ...
        }

        // --- Login Successful: Create Session ---
        session_regenerate_id(true); // Prevent session fixation
        $_SESSION['user_id'] = $user['id'];
        $_SESSION['username'] = $user['username'];
```

```
        $_SESSION['role'] = $user['role'];
        // ... (Redirect to the appropriate dashboard) ...
    }
}
?>
```

2. User Registration and OTP Email (register/register_process.php)

This code handles form submission, stores data in the session, and calls the centralized send_mail() function. This is a superior design as it keeps the email logic separate and reusable.

PHP

```
<?php
// ... (Includes, security checks, file handling) ...

// --- OTP Generation and Session Storage ---
$otp = random_int(100000, 999999);
$hashed_password = password_hash($password, PASSWORD_BCRYPT);

$_SESSION['registration_data'] = [
    'username' => $username,
    'email' => $email,
    'password' => $hashed_password,
    'otp' => $otp,
    'timestamp' => time(),
    // ... (other form data) ...
];

// --- Email OTP using Centralized Function ---
require_once __DIR__ . '/../mail/send_mail.php';
require_once __DIR__ . '/../mail/templates.php';

try {
    $subject = 'Your Verification Code for MedSync';
    $body = getOtpEmailTemplate($username, $otp, $_SERVER['REMOTE_ADDR']);

    // Call the centralized mail function
    if (send_mail('MedSync', $email, $subject, $body)) {
        $conn->close();
        header("Location: verify_otp.php");
        exit();
```

```
    } else {
        throw new Exception("Could not send OTP email.");
    }
} catch (Exception $e) {
    // ... (Error handling and cleanup) ...
}?>
```

5.2.2 Code Validation and Optimization

- **Code Validation:**
  - **Client-Side:** Real-time form validation is implemented using JavaScript. For example, `register/check_availability.php` is called via AJAX to check if a username or email is already taken before the user submits the registration form.
  - **Server-Side:** All incoming data is rigorously re-validated on the server to ensure data integrity and security, including checks for required fields, data formats (e.g., `FILTER_VALIDATE_EMAIL`), and file uploads.
  - All POST requests are protected with CSRF tokens, which are generated in `config.php` and validated in each processing file.
- **Optimization:**
  - **Database:** The MySQL schema uses indexes on frequently queried columns (e.g., `username`, `email`, `display_user_id`) and foreign keys to enhance query speed and ensure relational integrity.
  - **Frontend:** AJAX is used extensively to dynamically load content on dashboards (e.g., live tokens, user lists), providing a faster and more responsive user experience without full page reloads.
  - **Backend:** A modular code structure (e.g., central `send_mail.php`, `config.php`) reduces redundancy and improves maintainability.

5.3 Debugging

A custom error handler in `config.php` redirects all PHP errors to a central `log.txt` file located in the corresponding directory. This prevents the exposure of sensitive server information to users and provides developers with a centralized log for debugging. An example log entry from `log.txt`:

`[07-Nov-2025 14:22:01 Europe/Berlin] PHP Fatal error: Uncaught Error: Call to undefined function getDbConnection() in C:\xampp\htdocs\medsync\admin\api.php:12`

5.4 Unit Testing

Unit testing involves the verification of the smallest testable parts of an application, such as functions, classes, and procedures, to ensure they operate correctly in isolation. In the development of MedSync, unit testing was performed manually during the coding phase to validate the logic of critical backend functions before they were integrated into the frontend interfaces.

The primary objective was to ensure that core utilities—such as database connections, ID generation, and email dispatching—returned the expected results under various input conditions.

5.4.1 Test Plan (Unit Level)

The unit testing plan for MedSync focused on "White Box Testing," where the internal structure and logic of the code were known and tested.

1. **Scope:** Testing key PHP functions in config.php, api.php, and mail/send_mail.php.
2. **Methodology:**
    - **Database Connectivity:** Verifying the global connection function handles credentials and errors correctly.
    - **Logic Validation:** Testing algorithm-based functions (e.g., sequential ID generation) with boundary values.
    - **Security Checks:** Testing input sanitization functions against SQL injection strings and XSS payloads.
    - **External Services:** Verifying the integration with PHPMailer and Google Firebase isolated from the main UI.
3. **Tools:** Manual execution of PHP scripts and browser developer tools (Network tab) to inspect JSON responses.

5.4.2 Test Cases

The following tables outline specific unit tests conducted on core system functions.

Test Case 1: Database Connection Establishment

- **Unit Tested:** getDbConnection() function in config.php.
- **Description:** Verifies that the application can connect to the MySQL database using environment variables.

Step	Input / Condition	Expected Output	Actual Output	Status
1	Valid DB_HOST, DB_USER, DB_PASS in .env	Return valid mysqli object; no errors logged.	Connection Object Returned	**Pass**
2	Invalid Password in .env	Script termination; Error logged to log.txt.	Error Logged	**Pass**

Test Case 2: Sequential User ID Generation

- **Unit Tested:** generateDisplayId() function in admin/api.php.
- **Description:** Verifies the logic that locks the counter table and generates unique IDs (e.g., D0001, P0005) based on user roles.

Step	Input / Condition	Expected Output	Actual Output	Status
1	Role = 'doctor', Last ID in DB = 5	Function returns string "D0006".	"D0006"	**Pass**
2	Role = 'user', Last ID in DB = 99	Function returns string "U0100" (padding check).	"U0100"	**Pass**
3	Role = 'invalid_role'	Throw Exception "Invalid role specified".	Exception Caught	**Pass**

Test Case 3: Email Dispatch System

- **Unit Tested:** send_mail() function in mail/send_mail.php.
- **Description:** Tests the PHPMailer integration for sending OTPs and notifications.

Step	Input / Condition	Expected Output	Actual Output	Status
1	Valid Recipient Email, Valid Subject/Body	Function returns true; Email received in inbox.	true, Email Received	**Pass**
2	Invalid SMTP Credentials in system_settings	Function returns false; Error logged to error_log.	false, Error Logged	**Pass**

Test Case 4: Password Security

- **Unit Tested:** password_verify() implementation in login/login_process.php[4].
- **Description:** Verifies that the login logic correctly matches plain text input against the BCRYPT hash stored in the database.

Step	Input / Condition	Expected Output	Actual Output	Status
1	Input: "Password123" (Matches DB Hash)	Function returns true.	true	**Pass**
2	Input: "WrongPass" (Does not match DB Hash)	Function returns false.	false	**Pass**

Test Case 5: Input Sanitization

- **Unit Tested:** sanitize_search_input() in staff/api.php[5].

- **Description:** Ensures search bars strip harmful characters to prevent SQL Injection and XSS.

Step	Input / Condition	Expected Output	Actual Output	Status
1	Input: John Doe	Return: John Doe	John Doe	**Pass**
2	Input: <script>alert(1)</script>	Return: alert(1) (Tags stripped)	alert(1)	**Pass**
3	Input: UNION SELECT * FROM users	Return string with keywords removed.	String sanitized	**Pass**

SYSTEM TESTING

CHAPTER 6
SYSTEM TESTING

6.1 Introduction

System testing is a critical phase in the Software Development Life Cycle (SDLC) that validates the entire application against its specified requirements. The primary goal of testing the MedSync platform was to identify and rectify defects, ensure all modules work together cohesively, and verify that the system is secure, reliable, and user-friendly.

This chapter details the testing strategy, which was executed at multiple levels: **Unit Testing** to validate individual functions, **Integration Testing** to verify the interaction between different modules, and **System Testing** to assess the application as a whole.

6.2 Integration Testing

Integration testing focused on verifying the data flow and interaction between different modules of the MedSync platform.

Integration testing is designed to verify that separate modules work together as expected. There are several approaches to this:

- **Big Bang Integration:** All modules are integrated simultaneously.
- **Top-Down Integration:** Testing starts from the main control module and moves downward to lower-level modules.
- **Bottom-Up Integration:** Testing begins with lower-level modules and moves upward.

For the MedSync platform, a mixed approach was utilized, focusing on critical workflows to ensure data flows correctly between the user interfaces and the database.

- **Registration to Login:** A test was conducted to ensure a new user created via `register_process.php` and verified via `verify_process.php` could immediately log in through `login/login_process.php` and be redirected to the correct user dashboard.
- **Doctor-to-Patient Data Flow:** A doctor account was used to create a new prescription (`doctor/api.php add_prescription`). The corresponding patient account was then checked to confirm that the new prescription appeared correctly in their "Medical Records" and "Prescriptions" page (`user/api.php get_medical_records`).
- **Discharge Workflow Integration:** This was the most complex integration test:
    1. A **Doctor** initiated a discharge (`doctor/api.php initiate_discharge`).
    2. The request immediately appeared in the **Staff** dashboard's "Discharge Clearances" queue (`staff/api.php fetch=discharge_requests`).

3. The **Staff** account processed all required clearances (Nursing, Pharmacy, Billing).
4. The **Doctor** was then able to write and save the discharge summary (doctor/api.php save_discharge_summary).
5. Finally, the **Patient** could log in and successfully view and download the PDF summary (user/api.php get_discharge_summaries, download_discharge_summary).

- **Admin-to-System Communication:** A broadcast notification sent from the Admin dashboard (admin/api.php sendNotification) was verified to appear in the notification panels of all other active roles (Doctor, Staff, Patient).

6.3 System Testing

System Testing is a type of software testing performed on a complete, integrated system to evaluate its compliance with specified requirements. It is purely **Black-box testing**, meaning it tests the external behavior of the system without looking at the internal code structure. It validates both functional and non-functional requirements to ensure the product meets the specified needs. System testing was performed on the complete, integrated application to validate its compliance with all requirements and to test its non-functional aspects.

- **Security Testing:**
  - **Role-Based Access Control (RBAC):** Manually tested by logging in as a user and attempting to access URLs for other dashboards (e.g., /admin/dashboard.php). The system correctly enforced security checks and redirected the user to their own dashboard.
  - **CSRF Protection:** Form submissions (e.g., login, profile update) were tested using browser developer tools to manually remove or alter the csrf_token. All tampered requests were correctly rejected by the server.
  - **IP Blocking:** An IP address was added to the ip_blocks table via the Admin panel. Access to the entire website from that IP was successfully denied, as validated by the check in config.php.
- **Usability Testing:**
  - The application was tested by users on different devices (desktop, tablet, mobile) to ensure the responsive design was intuitive.
  - Features like real-time username validation (register/script.js) and the dark mode theme toggle (doctor/script.js) were tested for ease of use.
- **Error Handling:**
  - The custom error handler in config.php was tested by intentionally introducing a PHP error. The system successfully suppressed the user-facing error and correctly wrote the detailed error message to the log.txt file, as intended.

6.3.1 Test Plan & Test Cases

A structured test plan was created to validate the system's functionality, security, and usability across all user roles. The following table provides a sample of the test cases executed.

Test Case ID	Feature	Test Scenario	Expected Result
TC-REG-01	User Registration	Successful registration with valid data.	User is redirected to the OTP verification page and receives an email.
TC-REG-02	User Registration	Attempt registration with a duplicate email.	The system displays an error message: "Username or email is already taken."
TC-REG-03	User Registration	Enter mismatched passwords.	An error message is shown: "Passwords do not match."
TC-REG-04	OTP Verification	Enter an incorrect OTP.	The page reloads with an error: "Invalid OTP. Please try again."
TC-SEC-01	Security (RBAC)	Log in as user and attempt to access /admin/dashboard.php.	User is redirected back to their own dashboard (/user/dashboard.php).
TC-SEC-02	Security (CSRF)	Submit the login form with an invalid csrf_token.	The request is rejected, and the user is returned to

			the login page with an error.
TC-INT-01	Integration (Discharge)	A Doctor initiates a discharge for an admitted patient.	The request appears in the Staff "Discharge Clearances" queue.
TC-FUNC-01	Admin (Backup)	Admin clicks "Download Database Backup".	A .sql file of the database is successfully downloaded.

SYSTEM MAINTENANCE

CHAPTER 7
SYSTEM MAINTENANCE

7.1 Introduction

System maintenance is a significant process that continues well beyond the implementation of the MedSync Healthcare Platform. The goal of maintenance is to ensure that the system continues to perform well, remains secure, and develops to cater to the dynamic needs of the hospital and its users. This chapter describes how to maintain the MedSync platform, including different types of maintenance activities that will ensure the long-term viability, reliability, and performance of the system. A structured approach to maintenance will help minimize downtime, speed up problem resolution, and make it easy to adapt the system to future needs.

7.2 Maintenance

The maintenance plan for the MedSync platform can be grouped into four key areas, each addressing another aspect of the system's life cycle.

7.2.1 Corrective Maintenance

This involves identifying, isolating, and rectifying faults and errors discovered by users or system administrators after deployment. Corrective maintenance is reactive and focuses on fixing bugs that were not found during the testing phase.

- **Bug Tracking:** A formal process will be established for users to report issues. Each reported bug will be logged, prioritized based on its severity and impact on hospital operations, and assigned to the development team for resolution.
- **Patch Releases:** Fixes for critical bugs will be deployed through scheduled patch releases to minimize disruption to users.

7.2.2 Adaptive Maintenance

This type of maintenance is concerned with modifying the system to cope with changes in its external environment. As technology evolves, the MedSync platform will need to adapt to remain compatible and functional.

- **Software Updates:** The system will be updated to ensure compatibility with new versions of its core technologies, including PHP, MySQL, Apache, and modern web browsers.
- **Hardware Changes:** Adjustments may be required if the hospital upgrades its server hardware or network infrastructure.
- **Policy and Regulation Changes:** The platform may need to be modified to comply with new healthcare regulations or data privacy laws.

7.2.3 Perfective Maintenance

Perfective maintenance focuses on improving the system's performance and usability by implementing new features or enhancing existing ones based on user feedback. This is a proactive approach to making the system better over time.

- **Feature Requests:** A channel **is available** for administrators, doctors, and staff to request new functionalities. As outlined in the CONTRIBUTING.md file, these suggestions are managed and discussed via the project's GitHub Issues.
- **Performance Optimization:** This includes refactoring code for better efficiency, optimizing database queries to reduce response times, and improving the performance of features like the live token tracking system.
- **Usability Enhancements:** The user interface and user experience will be refined based on feedback to make the platform more intuitive and user-friendly.

7.2.4 Preventive Maintenance

Preventive maintenance involves activities aimed at preventing future problems before they occur. This includes making changes to increase the system's reliability and maintainability.

- **Code Refactoring:** Periodically reviewing and restructuring the existing code without changing its external behavior to improve readability and reduce complexity.
- **Database Optimization:** Regularly performing tasks such as index rebuilding and query analysis to ensure the database remains efficient as data volume grows.
- **Database Backups:** The system includes an **on-demand backup feature** implemented in the Administrator dashboard. This utility allows an administrator to generate and download a complete SQL backup of the entire database, preventing data loss in case of system failure.
- **Security Monitoring & Auditing:** The platform includes several built-in features for proactive security and auditing, supplementing regular external audits:
    - **Automated Activity Logging:** A system-wide log_activity function is implemented across all user roles (admin/api.php, doctor/api.php, staff/api.php, user/api.php) to create a detailed audit trail for critical actions, such as user creation, profile updates, and security events.
    - **IP Address Management:** The Administrator dashboard features a panel to track all login IP addresses, label them, and manually block suspicious IPs.
    - **Automated IP Blocking:** The system's core config.php file actively checks every incoming connection against the ip_blocks table and denies access to any blocked IP.
    - **Secure Session Management:** The application enforces secure, httponly session cookies with samesite attributes, along with session timeouts and user-agent checking to mitigate session hijacking risks.
    - **File Access Control:** The server configuration (.htaccess) explicitly denies direct web access to sensitive files, including configuration files (config.php), environment variables (.env), and the vendor directory.

FUTURE ENHANCEMENT AND SCOPE OF FURTHER DEVELOPMENT

CHAPTER 8

FUTURE ENHANCEMENT AND SCOPE OF FURTHER DEVELOPMENT

8.1 Introduction

The MedSync Healthcare Platform, in its current state, provides a robust and comprehensive solution for streamlining hospital operations and enhancing patient care. It successfully automates key workflows, improves resource management, and ensures secure data handling. However, the field of healthcare technology is constantly evolving. This chapter explores the merits and limitations of the current system and outlines a roadmap for future enhancements that can further expand its capabilities, ensuring that MedSync remains a cutting-edge Healthcare Information System (HIS).

8.2 Merits of the System

The current MedSync platform has successfully achieved its primary objectives, offering significant advantages over traditional hospital management systems:

- **Operational Efficiency:** The automation of appointment scheduling, billing, and the multi-step discharge process has significantly reduced manual paperwork and administrative overhead.
- **Enhanced Patient Experience:** Features like live token tracking, automated email notifications via PHPMailer, and easy access to medical records and bills provide patients with unprecedented transparency and convenience.
- **Improved Resource Management:** The real-time, color-coded dashboard for bed and inventory management allows staff to make quick and informed decisions, optimizing the use of hospital resources.
- **Robust Security:** The implementation of modern security practices, including OTP-based authentication, password hashing, prepared statements against SQL injection, and CSRF protection, ensures the integrity and confidentiality of sensitive patient data.
- **Data-Driven Insights:** The reporting module, with PDF generation via dompdf, and integrated analytics dashboards (using Chart.js) provide administrators with valuable data on financial performance, user roles, and resource utilization, enabling better strategic planning.
- **Integrated Communication:** The platform features a secure, real-time "Messenger" for two-way communication between administrators, doctors, and staff, enhancing internal coordination and removing the need for external chat applications.

8.3 Limitations of the System

While the current system is highly functional, it has certain limitations that present opportunities for future development:

- **Web-Only Access:** The platform is currently accessible only through a web browser. It lacks dedicated mobile applications for patients and doctors, which could offer a more convenient and feature-rich experience with push notifications.
- **Limited Healthcare System Integration:** While the system successfully integrates with third-party services for authentication (Google), security (reCAPTCHA), and communication (PHPMailer, Chatbot), it does not yet integrate with external *healthcare systems*. Data from third-party laboratories, external pharmacies, or insurance providers still requires manual entry.
- **No Telemedicine Capabilities:** The platform does not support video consultations, a feature that has become increasingly important in modern healthcare delivery.
- **Manual Schedule Management:** Doctor schedules and availability are managed manually by the administration team via the dashboard. Future versions could allow doctors to manage their own schedules.

8.4 Future Enhancement of the System

To address the current limitations and expand the platform's capabilities, the following enhancements are proposed for future development:

- **Dedicated Mobile Applications:**
    - Develop native mobile apps for both iOS and Android platforms.
    - **Patient App:** Would allow users to book appointments, view live tokens, receive push notifications, access records, and make payments directly from their smartphones.
    - **Doctor/Staff App:** Would enable medical professionals to manage appointments, view patient records, approve discharge steps, and communicate securely on the go.
- **Telemedicine Integration:**
    - Integrate a secure, HIPAA-compliant video conferencing module (e.g., using WebRTC) to allow doctors to conduct remote consultations with patients.
    - This feature would be linked to the appointment scheduling system, allowing patients to book and attend virtual appointments seamlessly.
- **Expand Advanced Analytics and AI-Powered Insights:**
    - **Expand** the existing analytics dashboards (currently using Chart.js) to include more sophisticated and customizable reports.
    - Incorporate predictive analytics to forecast patient admission rates, predict high-demand periods for certain specialties, and manage inventory more effectively.

    - Use machine learning models to identify patterns in patient data that could assist in early diagnosis or population health management.
- **Integration with External Systems (API Development):**
    - Develop a secure API to allow for integration with external laboratory and pharmacy systems, enabling the automatic fetching of lab results and real-time prescription fulfillment updates.
    - Integrate with payment gateways (like Stripe or Razorpay) to automate online bill payments, rather than just logging them.
    - Create modules for insurance claim processing, automating the submission and tracking of claims to various insurance providers.
- **Enhanced Patient Portal:**
    - Add features like a personal health diary for patients to track symptoms or vital signs.
    - Include a module for patient education, providing articles and resources related to their conditions.
    - Expand the existing internal "Messenger" to allow for secure, non-urgent communication between patients and their assigned care providers.

By implementing these enhancements, the MedSync Healthcare Platform can evolve into an even more powerful and indispensable tool for modern hospital management, further improving patient outcomes and operational excellence.

CONCLUSION

CHAPTER 9
CONCLUSION

The MedSync Healthcare Platform project successfully achieved its objective of developing a comprehensive, secure, and efficient Healthcare Information System (HIS) for medium to large hospitals. By leveraging a robust technology stack of PHP, MySQL, and modern frontend technologies, the platform effectively addresses the critical inefficiencies found in traditional hospital management systems.

Throughout its development, the project has delivered a fully functional application that automates key workflows, including real-time appointment scheduling, a multi-step discharge process, and integrated inventory management. The implementation of role-based access control provides a tailored and secure experience for administrators, doctors, staff, and patients. This is further hardened by modern security practices evident in the code, such as **password hashing**, **prepared statements** to prevent SQL injection, and **CSRF token protection** on all forms.

The system's technical success is highlighted by its integration of diverse third-party libraries and services. This includes **PHPMailer** for automated email notifications, **dompdf** for dynamic PDF generation of reports and summaries, **Firebase Authentication** for secure Google Sign-In, **Google reCAPTCHA** for spam prevention, and **Chart.js** for data visualization in administrative dashboards. Furthermore, the platform features a custom-built, real-time **internal messenger** for staff and doctors, enhancing clinical coordination.

The successful completion of the MedSync platform demonstrates the profound impact that well-designed technology can have on the healthcare sector. The system is poised to reduce operational costs, minimize human error, and most importantly, enhance the quality of patient care. By providing a scalable and user-friendly solution, MedSync stands as a testament to the power of technology in creating a more synchronized and patient-centric healthcare environment.

BIBLIOGRAPHY

CHAPTER 10

BIBLIOGRAPHY

Third-Party Libraries, Services, and Documentation:

1. Chart.js. (2025). *Open source HTML5 Charts for your website*. Retrieved from https://www.chartjs.org
2. dompdf. (n.d.). *dompdf HTML to PDF converter*. GitHub. Retrieved from https://github.com/dompdf/dompdf
3. Dotenv, P. (n.d.). *vlucas/phpdotenv*. GitHub. Retrieved from https://github.com/vlucas/phpdotenv
4. Font Awesome. (2025). *The iconic SVG, font, and CSS toolkit*. Retrieved from https://fontawesome.com
5. Google. (2025). *Firebase Authentication*. Google. Retrieved from https://firebase.google.com/docs/auth
6. Google. (2025). *Google Fonts*. Google. Retrieved from https://fonts.google.com
7. Google. (2025). *reCAPTCHA v2*. Google. Retrieved from https://developers.google.com/recaptcha
8. Kreait, J. (n.d.). *Firebase PHP JWT*. GitHub. Retrieved from https://github.com/kreait/firebase-php
9. MySQL Documentation. (2025). *Oracle Corporation*. Retrieved from https://dev.mysql.com/doc/
10. PHP Manual. (2025). *The PHP Group*. Retrieved from https://www.php.net/manual/en/
11. PHPMailer. (n.d.). *A full-featured email creation and transfer class for PHP*. GitHub. Retrieved from https://github.com/PHPMailer/PHPMailer
12. W3Schools Online Web Tutorials. (n.d.). Retrieved from https://www.w3schools.com

APPENDIX

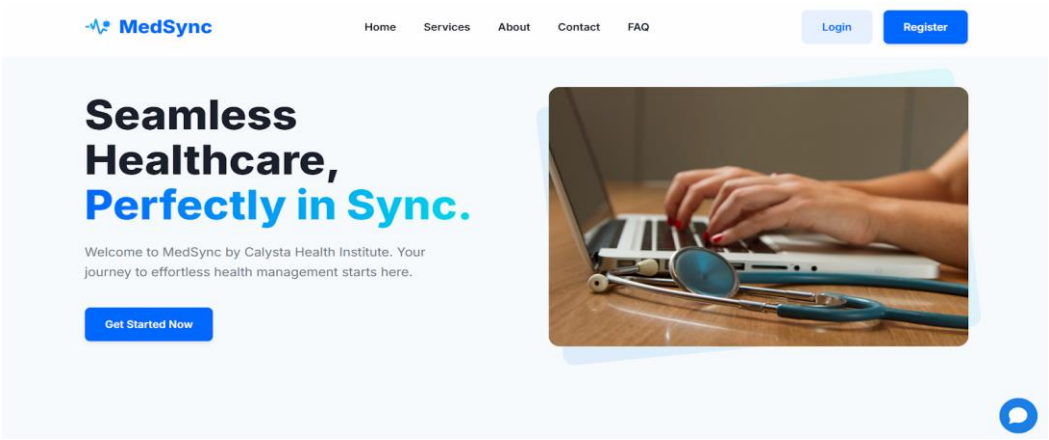
APPENDIX

Screenshots

This section provides a visual overview of the key user interfaces within the MedSync Healthcare Platform.

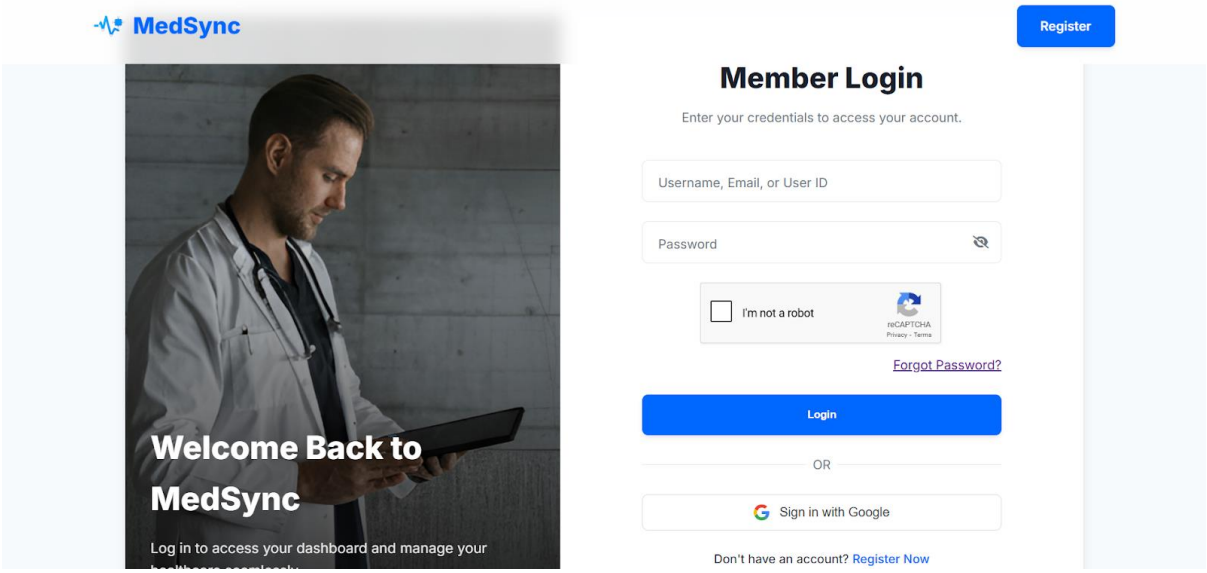
1. Home Page

The main landing page for the MedSync platform, providing navigation for new and existing users. It serves as the public face of the Calysta Health Institute and includes a "Request a Call Back" feature.



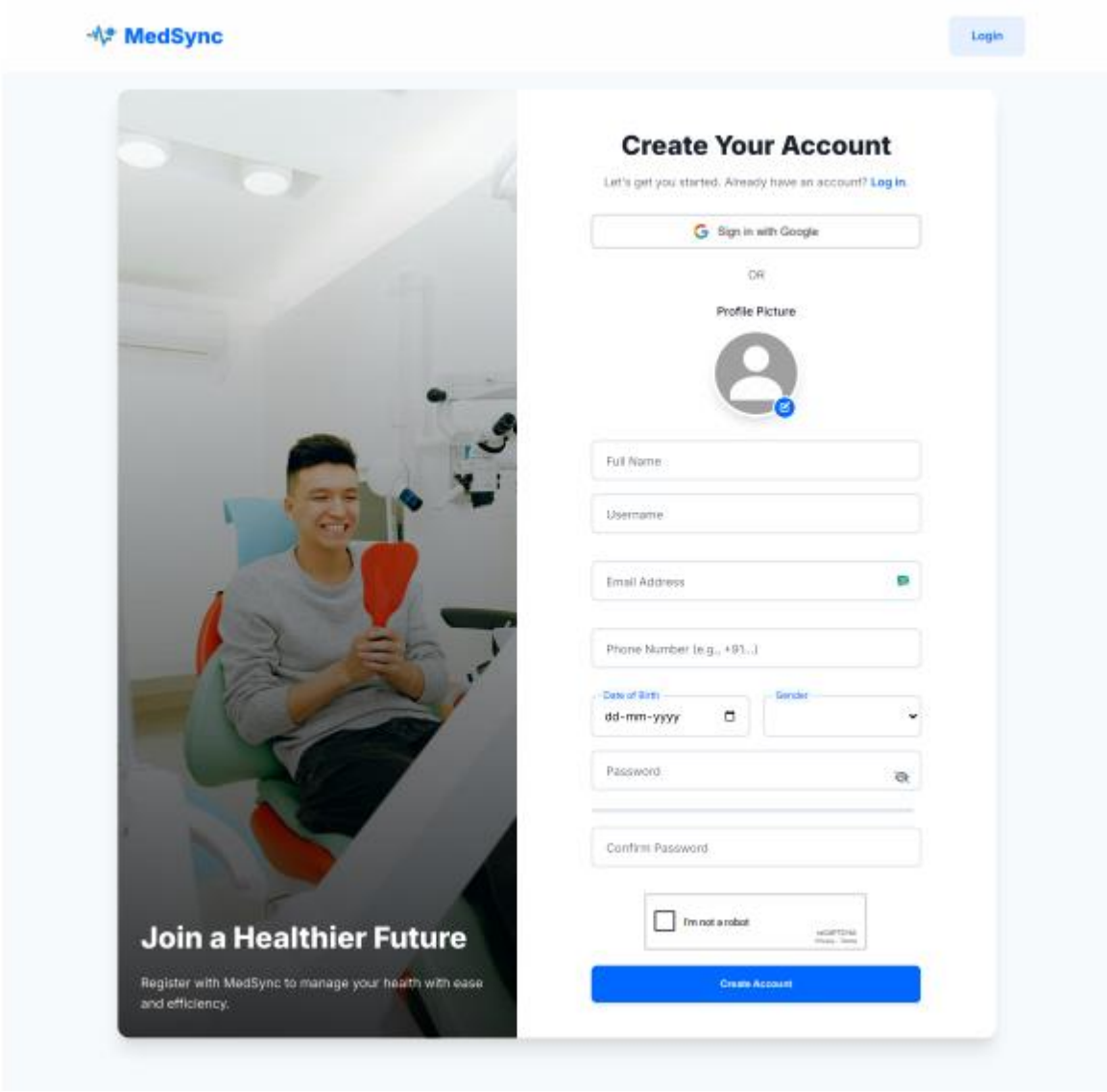
2. Login Page

The secure entry point for all users. It features fields for username/email/User ID, password, a "Forgot Password?" link, and an option to sign in with Google.



3. Patient Registration Page

A user-friendly form for new patients to create an account, featuring profile picture upload, real-time username/email validation, Google Sign-In, and Google reCAPTCHA security.

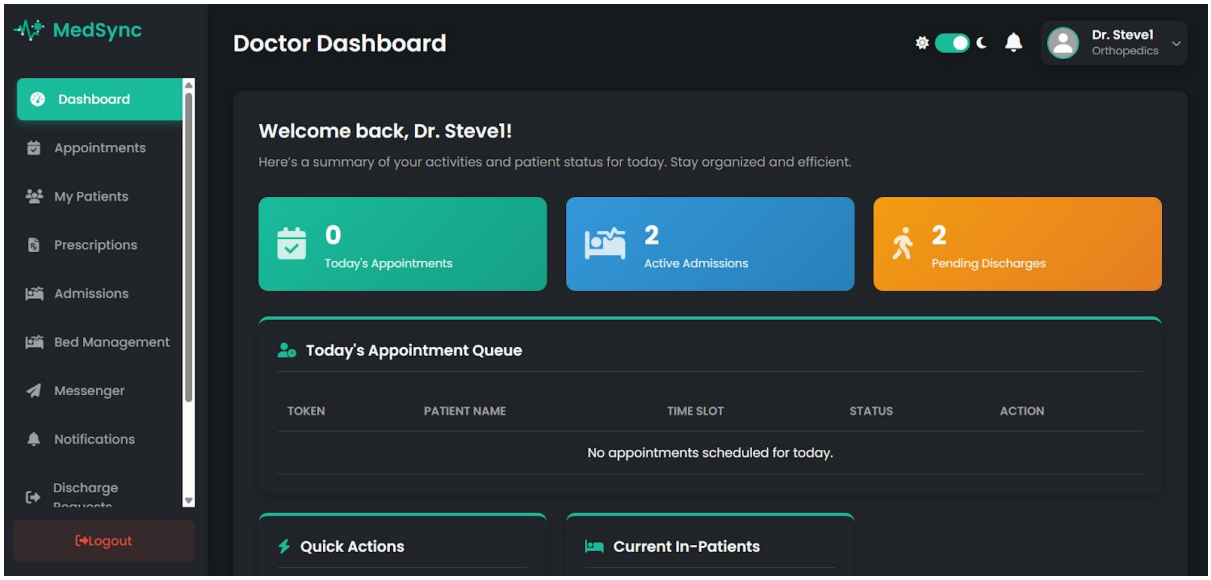


4. Administrator Dashboard

The central hub for administrators, providing an overview of system statistics (Total Users, Active Doctors), low inventory alerts, a User Roles Distribution chart (using Chart.js), and quick access to management functions like user management, activity logs, and system settings.

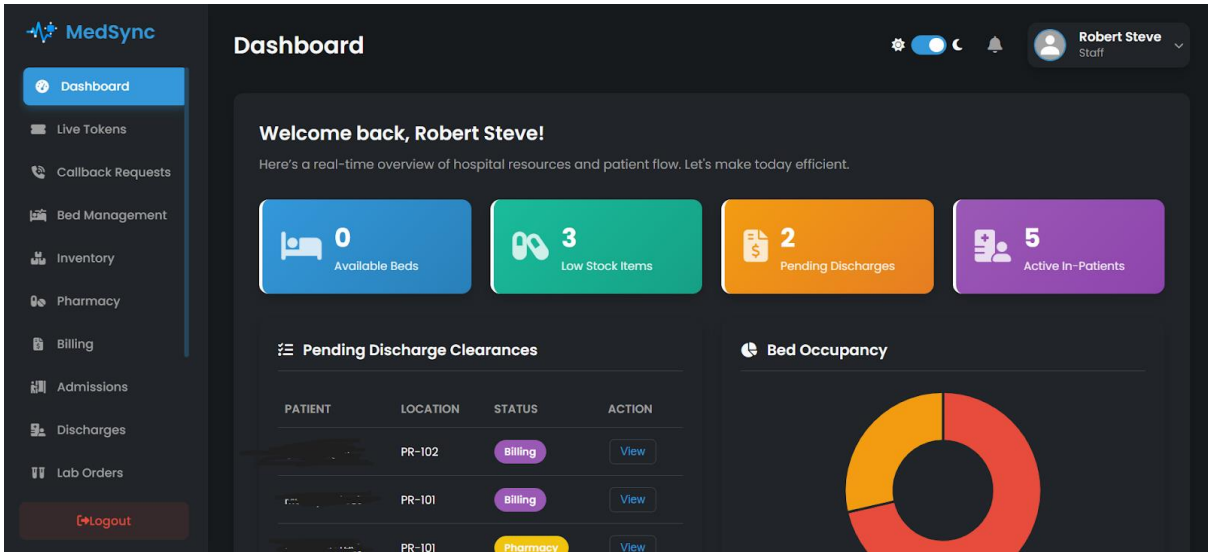
5. Doctor Dashboard

A specialized interface for doctors to manage their workflow. It shows key statistics like "Today's Appointments," "Active Admissions," and "Pending Discharges," along with a queue of today's patients. It also provides quick actions to admit patients, create prescriptions, and initiate discharges.



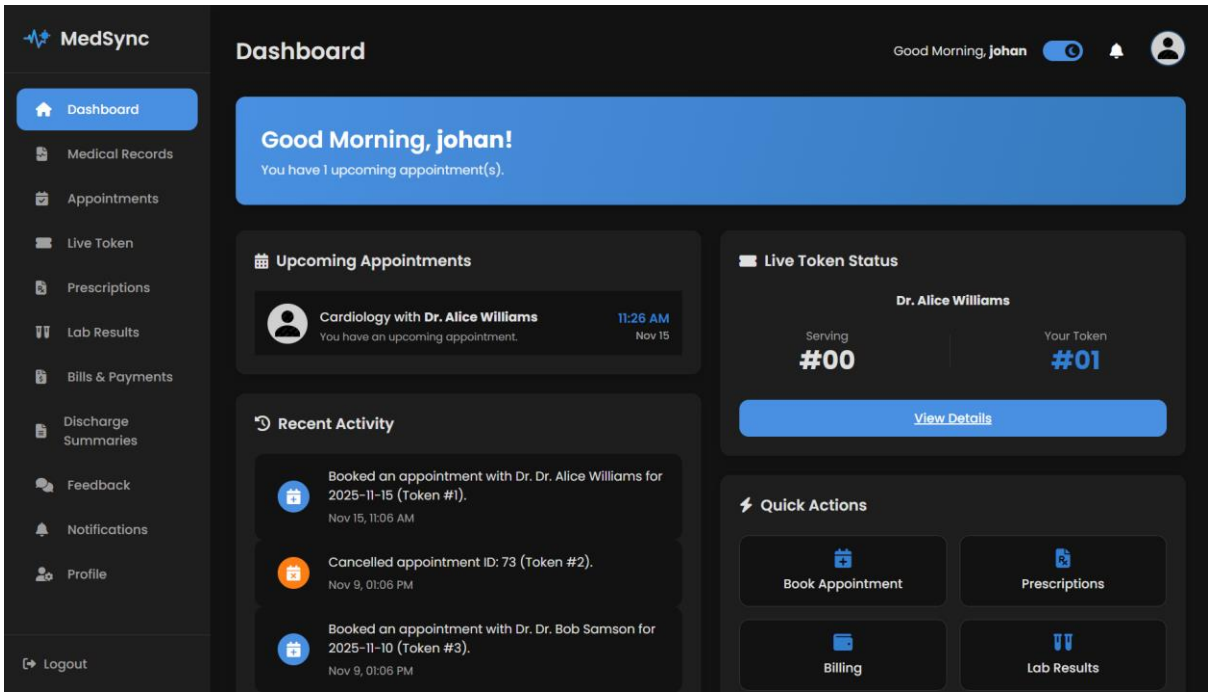
6. Staff Dashboard

An operational dashboard for hospital staff to manage bed allocation ("Available Beds"), inventory ("Low Stock Items"), and patient flow ("Pending Discharges," "Active In-Patients"). It also features a "Pending Discharge Clearances" queue for processing the multi-step discharge workflow.



7. Patient Dashboard

A personal portal for patients to manage their health. Key features include viewing "Upcoming Appointments," checking the "Live Token Status" for current appointments, reviewing "Recent Activity," and using "Quick Actions" to book appointments or view medical records.



GLOSSARY

GLOSSARY

- **AJAX (Asynchronous JavaScript and JSON):** A set of web development techniques used to create asynchronous web applications. In MedSync, it is used extensively via the fetch API to send and receive data in **JSON** format, updating parts of a web page (like the live token status or dashboards) without reloading the whole page.
- **API (Application Programming Interface):** A set of rules and protocols for building and interacting with software. In MedSync, the api.php files in each user directory (e.g., admin/api.php) act as the backend API that the frontend JavaScript communicates with.
- **Chart.js:** A JavaScript library used to create interactive, animated charts. In MedSync, it is used on the Admin and Staff dashboards to visualize data like User Role Distribution and Bed Occupancy.
- **Composer:** A dependency management tool for PHP. It is used in MedSync to install and manage all third-party libraries, such as PHPMailer, dompdf, and Firebase-PHP.
- **CSRF (Cross-Site Request Forgery):** A type of security attack that tricks a user into submitting a malicious request. MedSync prevents this by generating a unique csrf_token in the session and validating it on all form submissions.
- **dompdf:** A PHP library used to convert HTML and CSS into PDF documents. MedSync uses this to dynamically generate PDF invoices, lab reports, and discharge summaries for users to download.
- **.env (Environment File):** A text file used to store configuration variables outside of the main codebase, such as database passwords and API keys. The MedSync config.php file uses the vlucas/phpdotenv library to load these variables securely.
- **Firebase:** A platform by Google that provides various backend services. MedSync uses **Firebase Authentication** to securely manage the "Sign in with Google" feature and verify user identity tokens.
- **HIS (Healthcare Information System):** A comprehensive, integrated information system designed to manage the administrative, financial, and clinical aspects of a hospital.
- **JSON (JavaScript Object Notation):** A lightweight format for storing and transporting data. It is the primary data format used by MedSync's APIs to communicate between the frontend JavaScript and the backend PHP.
- **MySQL:** An open-source relational database management system used to store and manage all data for the MedSync platform.
- **OTP (One-Time Password):** A password that is valid for only one login session or transaction. Used in MedSync for secure user registration and password resets.
- **PHP (Hypertext Preprocessor):** A server-side scripting language used for the backend logic of the MedSync application.
- **PHPMailer:** A popular PHP library used to send emails from a web server. MedSync utilizes this for sending OTPs, welcome emails, and other system notifications.

- **Prepared Statements:** A feature of database systems used to execute the same or similar database statements repeatedly with high efficiency and security against SQL injection.
- **RBAC (Role-Based Access Control):** A security model that restricts system access to authorized users based on their roles. MedSync implements this by dividing users into Patients, Doctors, Staff, and Administrators, each with a unique dashboard and permissions.
- **reCAPTCHA:** A Google service used to distinguish between human users and automated bots. MedSync uses it on the registration page to prevent spam and abuse.
- **SQL Injection:** A cyberattack technique where malicious SQL code is inserted into queries. MedSync prevents this by using prepared statements for all database queries.
- **UI (User Interface):** The graphical layout of an application through which a user interacts.
- **UX (User Experience):** The overall experience of a person using a product or service, especially in terms of how easy or pleasing it is to use.
- **XAMPP:** A free and open-source cross-platform web server solution stack package, used as the development environment for MedSync.