Indian Institute of Technology Hyderabad (IIT)

Software Requirements Design (SRD) for MASS
(Medical Access Service System)

Submitted by:

Priyanshu Sharma (CS24BTKMU11004)
Sadikshya Pokharel (CS24BTKMU11002)
Surbhi (CS22BTECH11057)
Mane Pooja Vinod CS22BTECH11035

Submitted on:

15th March, 2025

Software Architecture

A. Component and Connector view

The Medical Access Service System (MASS) is designed with a three-tier architecture consisting of the **Presentation Layer, Application Layer**, and **Data Layer.** Below is a breakdown of the components and connectors:

1. Presentation Layer (Frontend):

React Native for mobile applications (cross-platform compatibility for Android and iOS).
Web Interface for desktop users, built using React.js for a responsive and interactive user experience.
Connectors: RESTful APIs to communicate with the backend server.

2. Application Layer (Backend):

1. **Node.js** with Express.js for handling HTTP requests, routing, and business logic.
2. **Authentication Service:** Handles user authentication (login, signup, role-based access control).
3. Appointment Management Service: Manages appointment booking, rescheduling, and cancellation.
4. **Medical Record Service:** Handles the storage, retrieval, and updating of patient medical records.
5. Notification Service: Sends real-time notifications (appointment reminders, prescription updates) via SMS, email, and in-app alerts.
6. **Connectors:** RESTful APIs to interact with the frontend and database.

3. Data Layer:

1. **PostgreSQL:** A relational database for structured data storage (e.g., patient records, doctor profiles, hospital details).
2. **MongoDB:** A NoSQL database for unstructured data (e.g., patient notes, logs).
3. **Connectors:** ORM (Object-Relational Mapping) tools like Sequelize for PostgreSQL and Mongoose for MongoDB to interact with the backend.

4. External Services:

1. **Cloud Hosting:** The system is deployed on cloud platforms like AWS, Google Cloud, or Azure for scalability and high availability.
2. **Third-Party APIs:** Integration with SMS/email services for notifications and payment gateways for future features like online medicine purchases.

B. Explanation of the Architecture

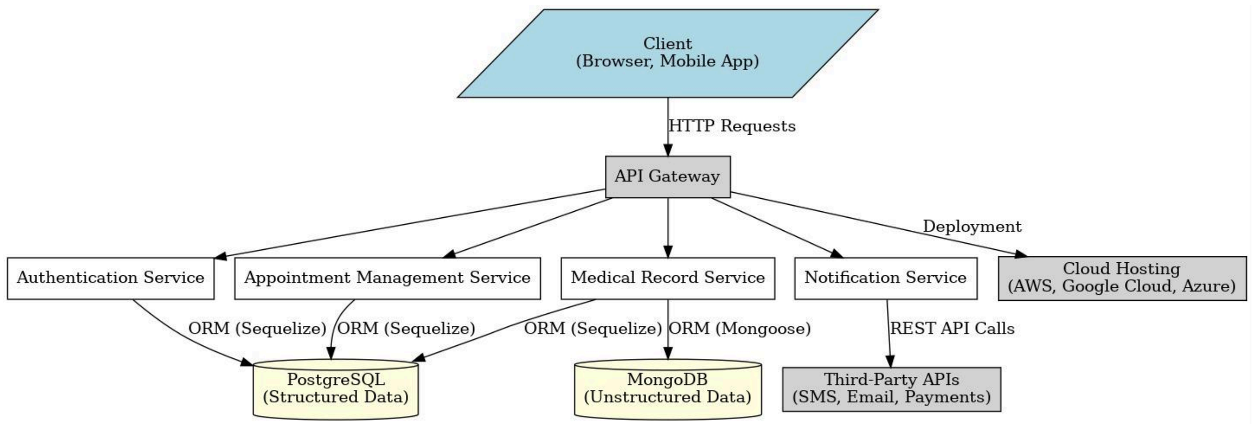


Fig : Architecture for MASS

The three-tier architecture of MASS ensures a clear separation of concerns, making the system modular, scalable, and maintainable. The Presentation Layer handles user interactions and provides a seamless experience across devices. The Application Layer contains the core business logic, including user authentication, appointment management, and medical record handling. The Data Layer ensures secure and efficient storage of structured and unstructured data.

The use of RESTful APIs allows for loose coupling between the frontend and backend, enabling easy updates and scalability. The system is designed to be cloud-based, ensuring high availability and scalability to handle a large number of users. The role-based access control ensures that only authorized users (patients, doctors, administrators) can access specific data, maintaining security and privacy.

C. ATAM Analysis:

1. Scenarios of Interest:

1. **S1:**A patient attempts to book an appointment with a doctor (normal scenario: all servers are up and running under normal load).
2. **S2:** A hospital administrator updates patient medical records or assigns a new doctor to a patient.
3. **S3:** Many patients try to access the system simultaneously to book appointments or view medical records (high load scenario).
4. **S4:** The database, server, or cloud service is temporarily down (system failure scenario).
5. **S5:** A doctor issues a prescription, and the system updates the patient's medical records and sends notifications (real-time update scenario).

2. Requirements/Constraints

Taking the concerns of the stakeholders into account, the requirements are:

1. **Security:** Ensuring the confidentiality and integrity of patient data, especially during data retrieval, updates, and sharing.
2. **Design Simplicity:** Striving for a straightforward and understandable system architecture, minimizing complexity to enhance development, debugging, and maintenance.
3. **Maintainability:** Facilitating easy updates, bug fixes, and overall system enhancements over the software's lifecycle.

4. **Scalability:** Allowing for an increasing number of users, transactions, or data volume.
5. **Availability:** Ensuring that the system is consistently accessible and operational, even during high load or partial system failures.
6. **Response Time:** Focusing on optimizing the system's performance to provide timely and responsive interactions, especially for critical operations like appointment booking and prescription updates.

3. Attribute-Specific Analysis

3.1 Security:

➔ **Architecture 1: Client-server with cloud deployment**

Strengths: The system uses role-based access control (RBAC) to ensure that only authorized users (patients, doctors, administrators) can access specific data. Data is encrypted both in transit and at rest, ensuring confidentiality.

Weaknesses: If the cloud service is compromised, patient data could be exposed. However, this risk is mitigated by using secure cloud providers like AWS or Azure, which offer robust security features.

➔ **Architecture 2: Distributed Microservices with CDN**

Strengths: A distributed architecture can isolate failures, reducing the risk of a single point of failure. However, this architecture is not currently proposed for

MASS.

Weaknesses: Not applicable since MASS is designed as a client-server model with cloud deployment.

3.2 Design Simplicity:

- **Architecture 1: Client-Server with Cloud Deployment**

  Strengths: The client-server model is straightforward, with clear separation between the frontend (React Native), backend (Node.js/Express.js), and database (PostgreSQL/MongoDB). This simplicity makes it easier to develop, debug, and maintain.

  Weaknesses: As the system grows, the monolithic backend could become harder to manage. However, this can be mitigated by modularizing the backend code.

3.3 Maintainability:

- **Architecture 1: Client-Server with Cloud Deployment**

  Strengths: The use of modular components and RESTful APIs ensures that updates and bug fixes can be implemented without disrupting the entire system. The codebase is well-documented, making it easier for developers to understand and modify.

  Weaknesses: If the backend becomes too large, it may require refactoring into microservices, which would increase complexity.

3.4 Scalability:

➔ **Architecture 1: Client-Server with Cloud Deployment**

Strengths: The cloud-based architecture allows for horizontal scaling (adding more servers) and vertical scaling (increasing server capacity). The system can handle a large number of users by leveraging cloud services like AWS or Google Cloud.

Weaknesses: Scaling the database (PostgreSQL) for high read/write operations may require additional optimization, such as using read replicas or sharding.

3.5 Availability:

➔ **Architecture 1: Client-Server with Cloud Deployment**

Strengths: The system is deployed on cloud platforms like AWS or Google Cloud, which offer high availability and redundancy. In case of server failure, the system can automatically switch to backup servers.

Weaknesses: If the cloud provider experiences an outage, the system may become unavailable. However, this risk is mitigated by choosing reliable cloud providers with strong SLAs (Service Level Agreements).

3.6 Response Time:

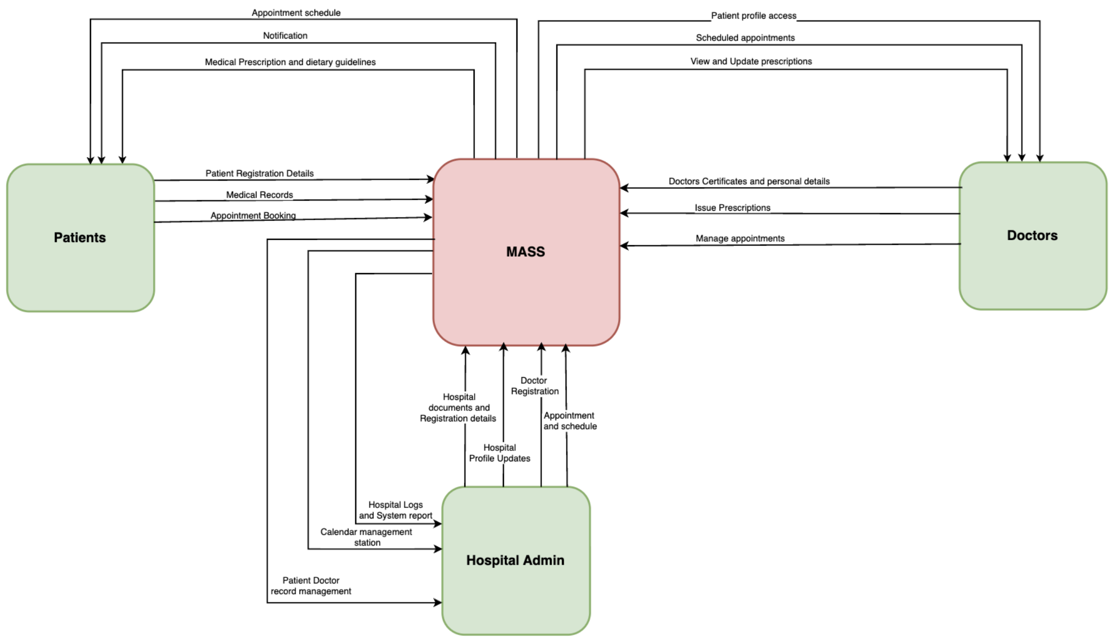
➔ **Architecture 1: Client-Server with Cloud Deployment**

Strengths: The system is optimized for low latency, with response times of < 200ms for most operations under normal load. The use of a NoSQL database (MongoDB) for unstructured data improves performance for certain queries.

Weaknesses: Under high load, response times may increase, especially for operations involving large datasets (e.g., retrieving medical records). However, this can be mitigated by optimizing database queries and using caching mechanisms.

Data Flow Diagrams

A. DFD


Appointment schedule
Notification
Medical Prescription and dietary guidelines
Patient profile access
Scheduled appointments
View and Update prescriptions
Patients
Patient Registration Details
Medical Records
Appointment Booking
MASS
Doctors Certificates and personal details
Issue Prescriptions
Manage appointments
Doctors
Hospital documents and Registration details
Hospital Profile Updates
Doctor Registration
Appointment and schedule
Hospital Logs and System report
Calendar management station
Patient Doctor record management
Hospital Admin

B. Most Abstract Input (mai):

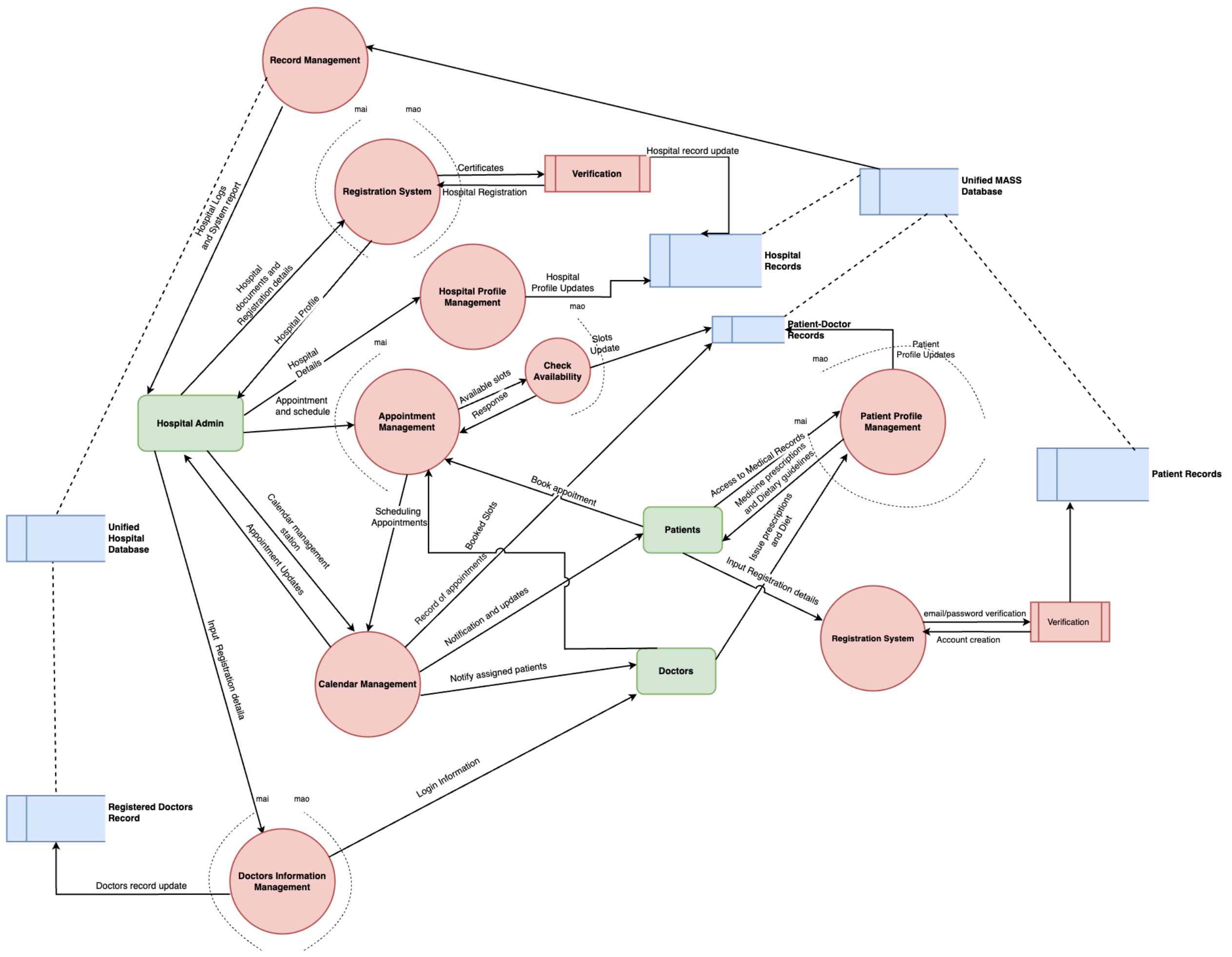
The highest-level inputs that initiate the system's processes are:

1. **User Registration & Authentication Data**
    a. This includes login credentials (email, password), user role (patient, doctor, admin), and identity verification.
    b. It enables access to the system and defines user permissions.
2. **Hospital & Doctor Registration Data**
    a. Information about hospitals, doctors, and medical staff registration.
    b. Required for setting up profiles and linking medical services.
3. **Appointment Requests & Patient Data**
    a. Patients submitting appointment requests.
    b. Includes symptoms, medical history, and consultation needs.

C. Most Abstract Output (mao):

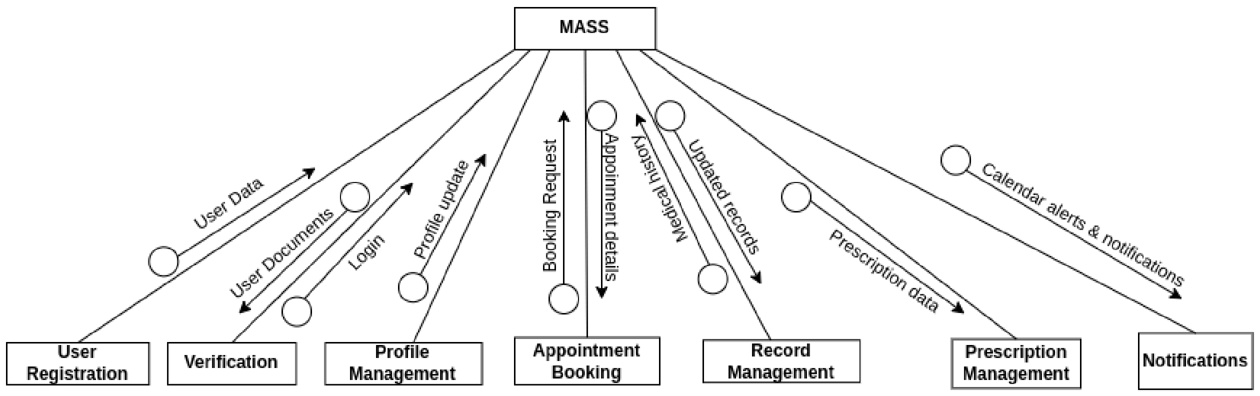
The highest-level outputs that summarize system functions are:

1. **Medical Records & Patient-Doctor Interaction History**
   a. Generated from consultations, lab reports, and prescriptions.
   b. Ensures continuity of care and historical medical tracking.
2. **Appointment Confirmation & Scheduling Details**
   a. Sent to patients and doctors upon successful scheduling.
   b. Includes time, location (physical/virtual), and reminders.
3. **Hospital & Doctor Availability Updates**
   a. Reflects real-time availability based on appointments, working hours, and holidays.



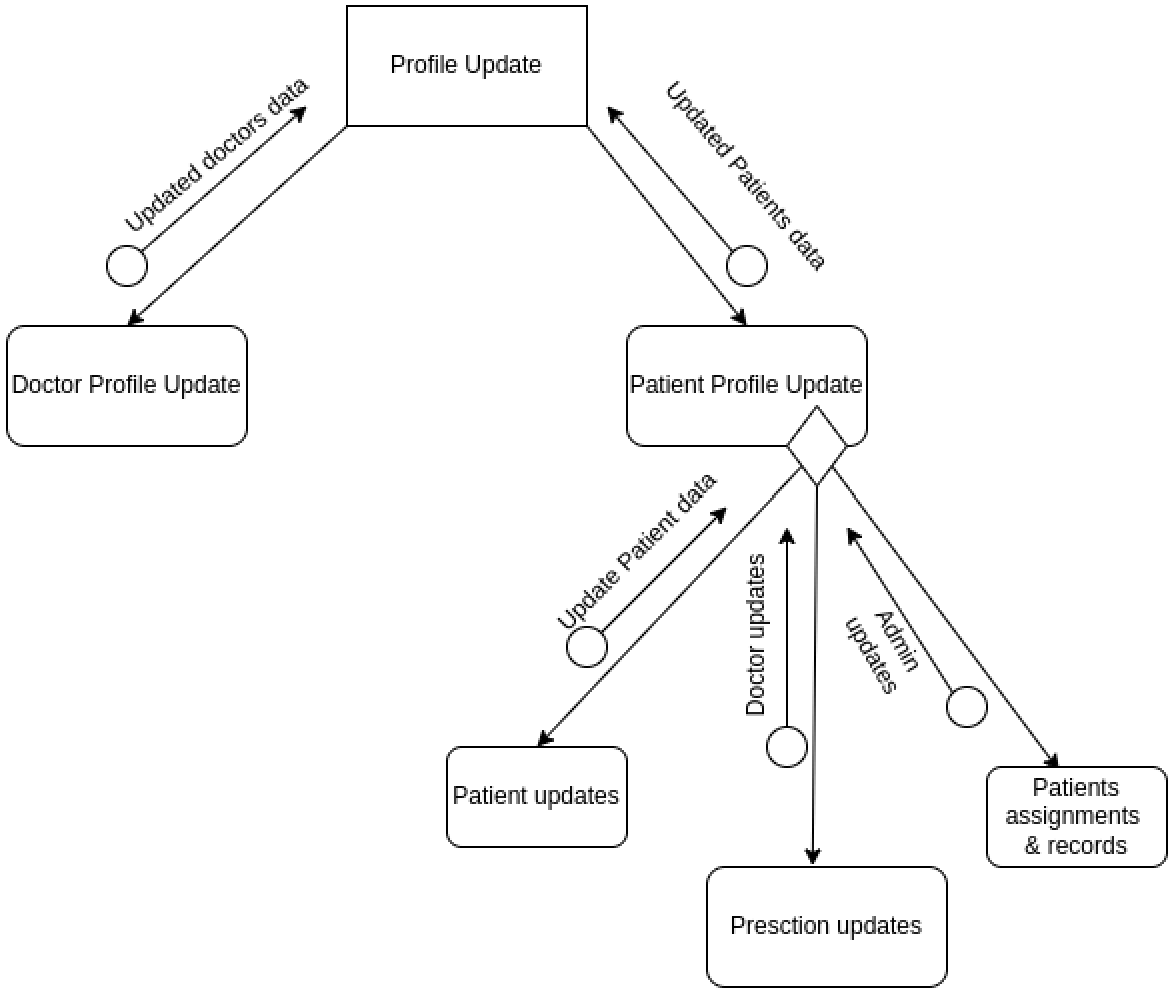
Structure Charts

1. First level factored modules

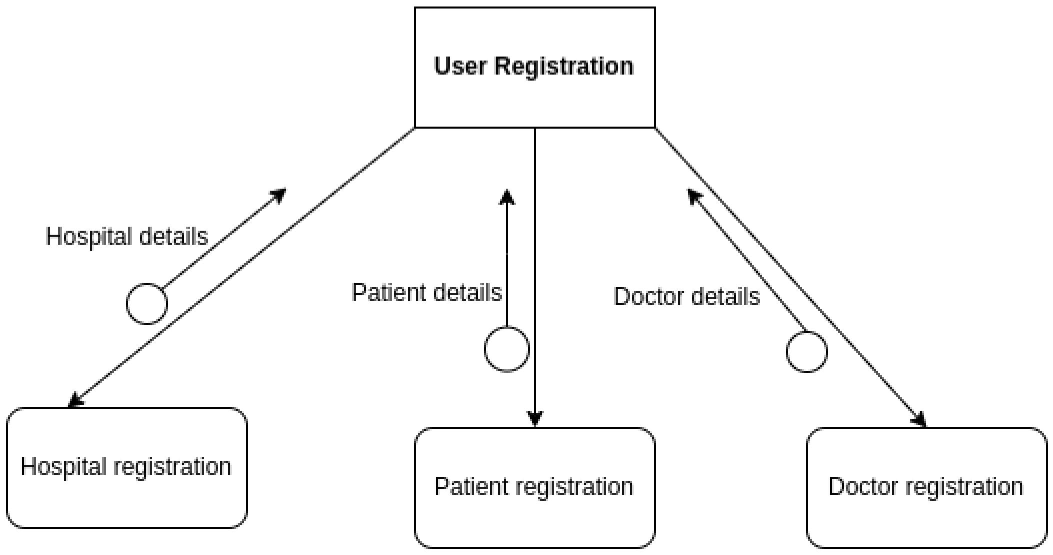


2. Factored input modules

a. Profile Updation module

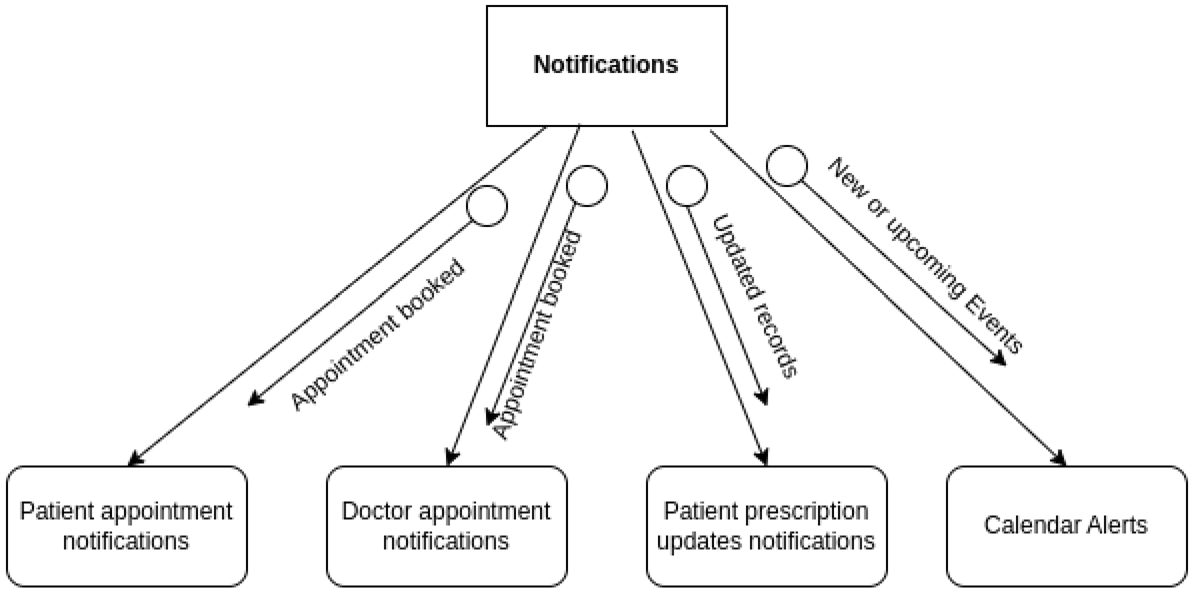


1. User Registration module

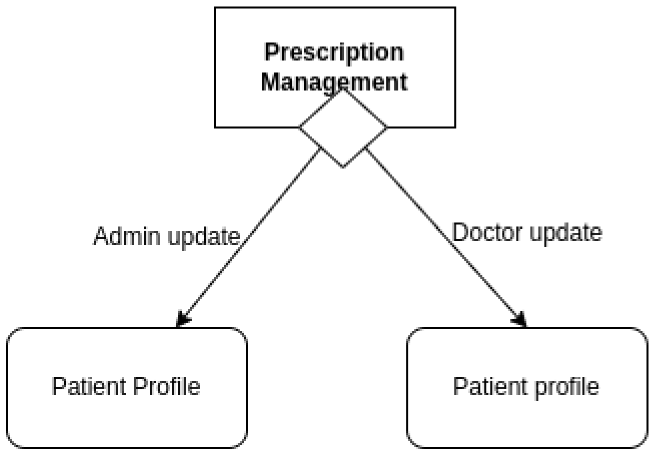


3. Factored output modules

a. Notifications module

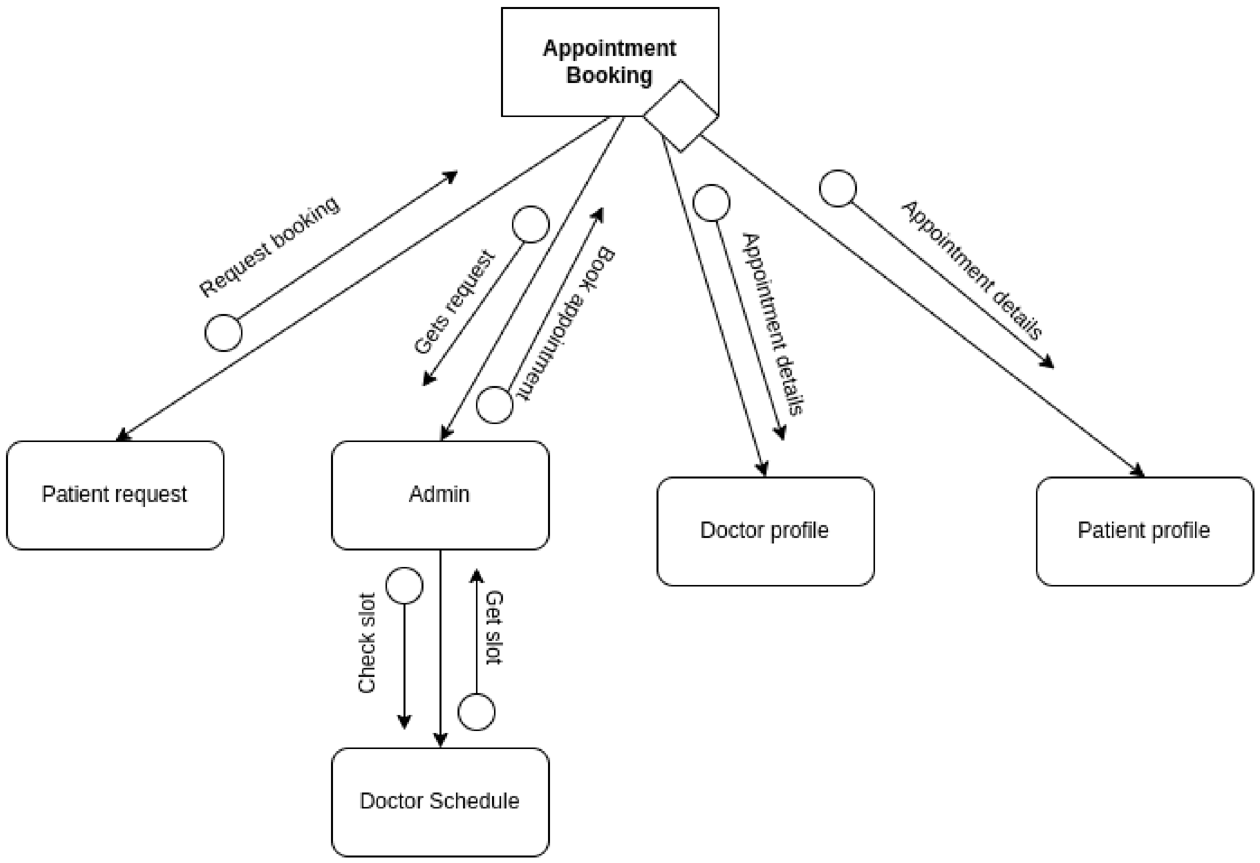

Notifications
Appointment booked
Appointment booked
Updated records
New or upcoming Events
Patient appointment notifications
Doctor appointment notifications
Patient prescription updates notifications
Calendar Alerts

b. Prescription Management module

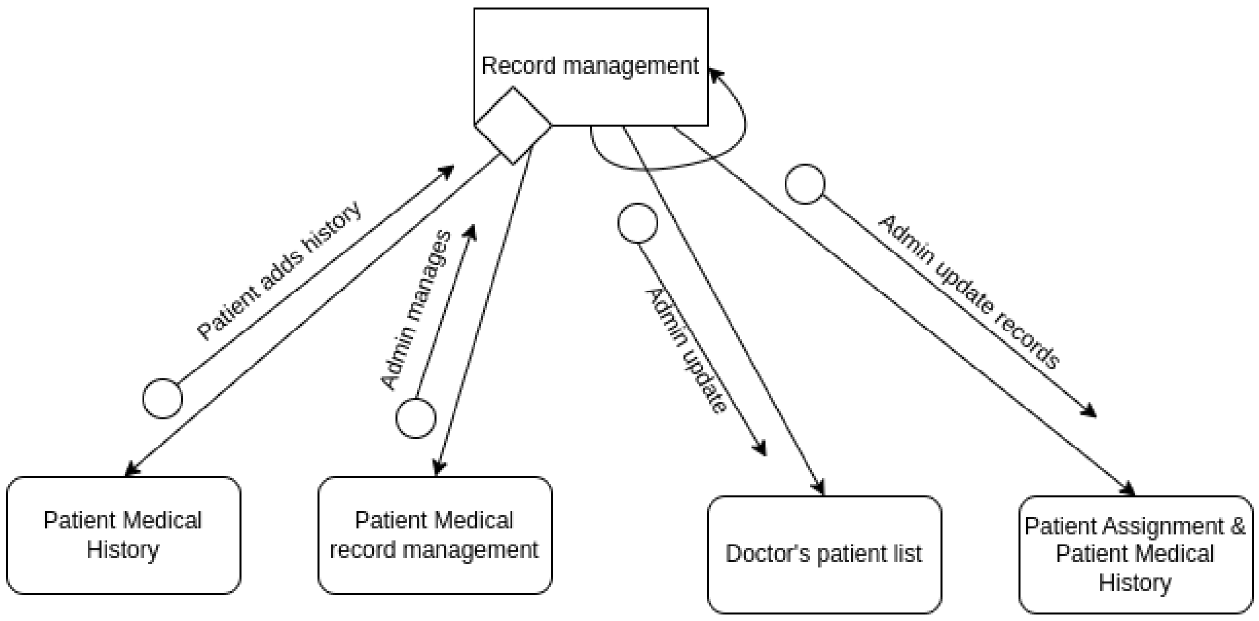


4. Factored transform modules

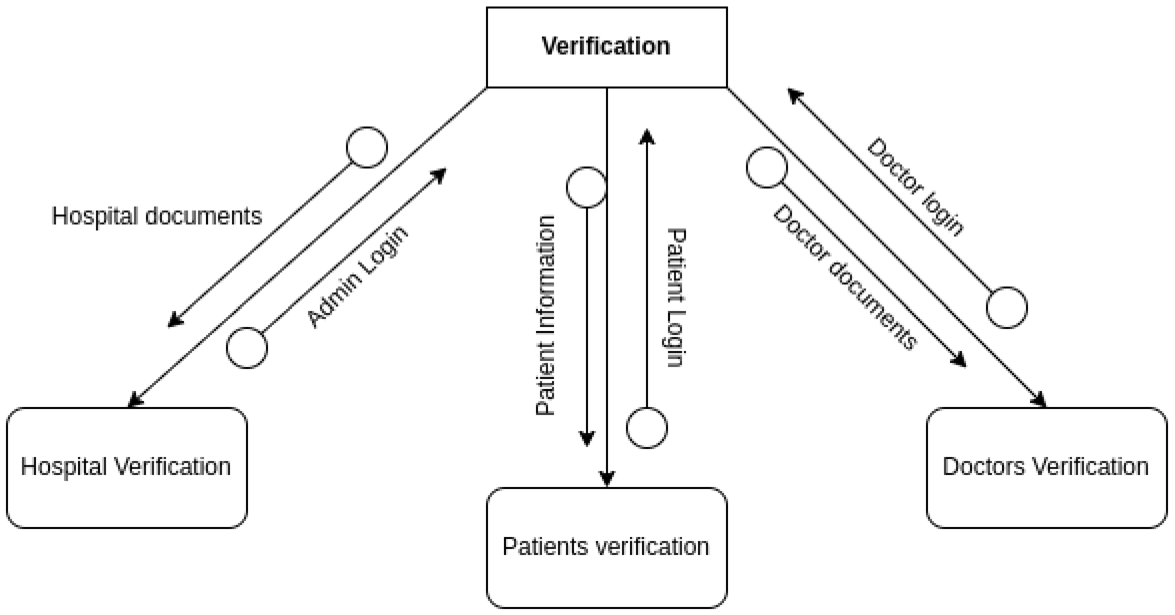
a. Appointment Booking module



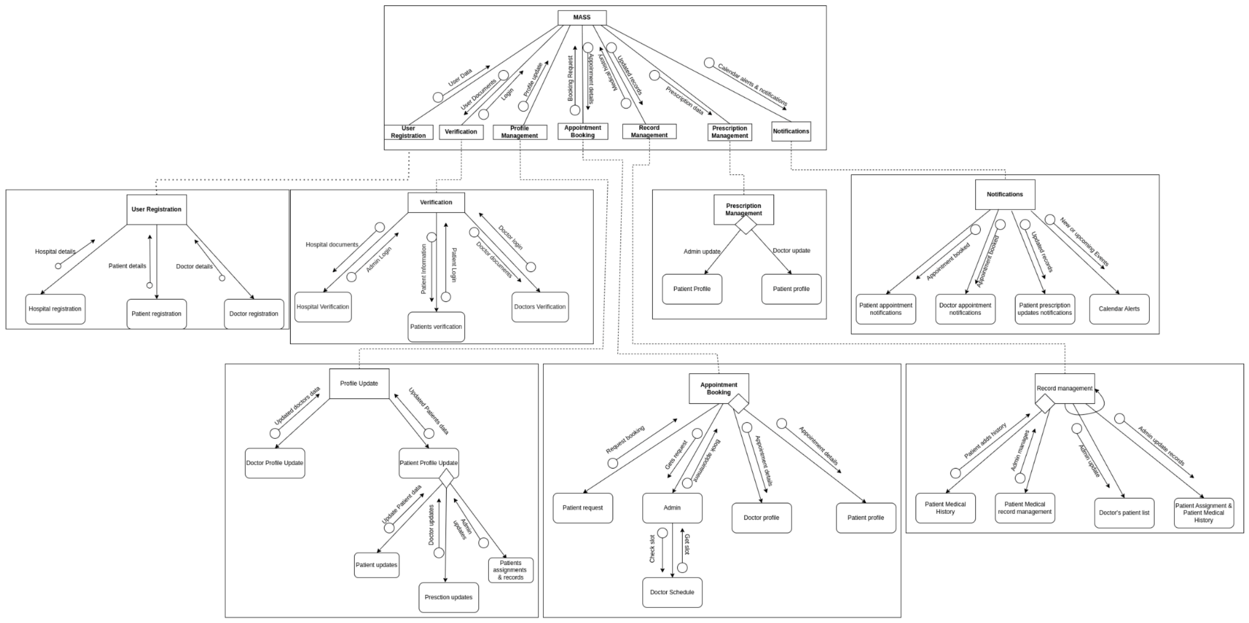
b. Record Management module



c. Verification module



5. Final structure chart showing all the modules

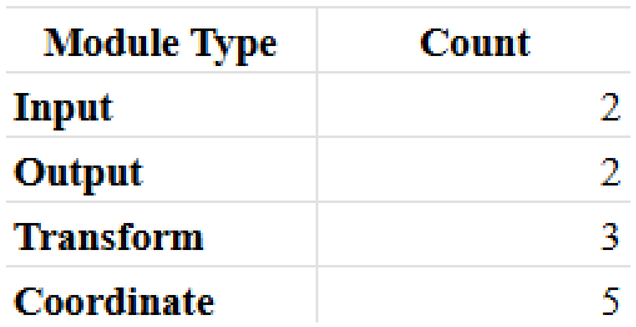


Design Analysis

1. List of Modules with Type, Cohesion, and Description

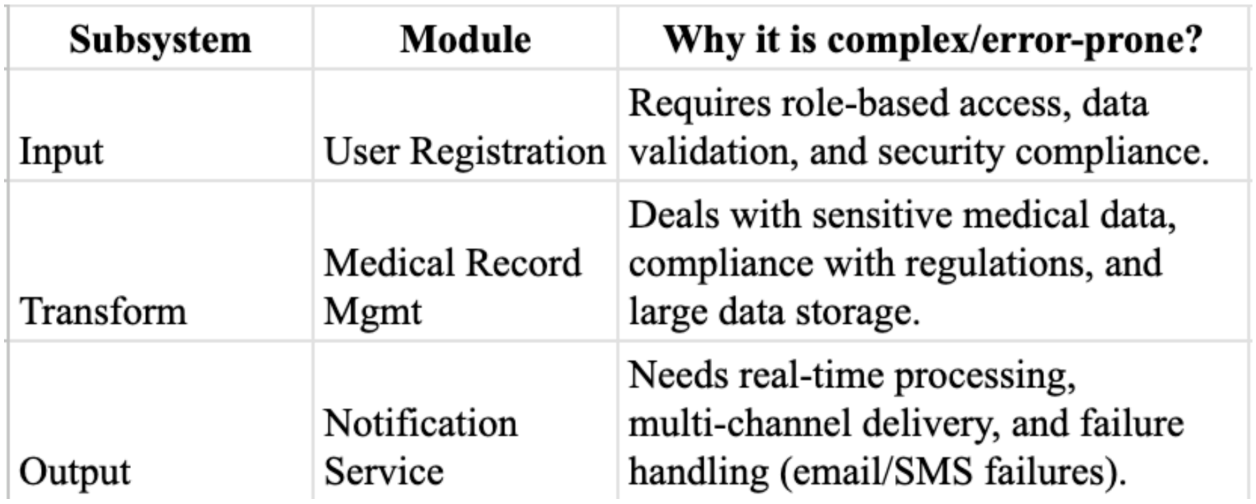
Module Name	Type	Cohesion Type	Description
User Registration	Input	Functional	Handles new user registration (patients, doctors, admins).
Verification	Transform	Functional	Verifies user credentials (patients' documents, doctors' licenses).
Profile Updation	Input	Functional	Updates user profile details such as contact and specialization.
Appointment Booking	Transform	Sequential	Manages appointment requests, checks availability, and updates status.
Medical Record Management	Transform	Sequential	Adds, updates, and retrieves medical records for patients.
Prescription Management	Output	Sequential	Manages patient prescriptions and allows updates.
Notification	Output	Temporal	Sends notifications (appointments, prescriptions, updates) via SMS, email, or app.
Calendar Management	Coordinate	Functional	Updates doctors' schedules and manages available time slots.
Frontend (React Native)	Input	Functional	Provides the interface for mobile and desktop users.
Backend (Node.js, Express.js)	Coordinate	Functional	Handles business logic, request routing, and API communication.
PostgreSQL Database	Data Storage	Functional	Stores structured data like user details, medical records, and appointments.
MongoDB Database	Data Storage	Functional	Stores unstructured data like logs and patient notes.
Authentication Service	Coordinate	Functional	Manages user authentication and role-based access control.
REST API (Frontend-Backend)	Coordinate	Functional	Enables communication between frontend and backend.
REST API (Backend-Database)	Coordinate	Functional	Facilitates interaction between backend and databases.
Cloud Hosting (AWS/GCP)	External	Functional	Ensures scalability, availability, and cloud-based deployment.

2. Count of Module Types



Module Type	Count
Input	2
Output	2
Transform	3
Coordinate	5

3. Most Complex or Error-Prone Modules



Subsystem	Module	Why it is complex/error-prone?
Input	User Registration	Requires role-based access, data validation, and security compliance.
Transform	Medical Record Mgmt	Deals with sensitive medical data, compliance with regulations, and large data storage.
Output	Notification Service	Needs real-time processing, multi-channel delivery, and failure handling (email/SMS failures).

4. Top 3 modules - in terms of fan out and fan in

1. Fan out

1) Appointment Booking
   Likely depends on User Registration (to verify patient), Profile Management (to fetch patient/doctor info), Calendar Management (to check availability), Medical Record Management (to retrieve patient history), and Prescription Management.
   Estimated fan-out: 4+

2) Medical Record Management

Likely depends on User Registration (to verify patient), Profile Management(to fetch patient/doctor info), and Appointment Booking(to log records).
Estimated fan-out: 3+

3) Prescription Management
Likely depends on User Registration, Medical Record Management (to manage prescription history and updates) and Appointment Booking . (to link prescriptions with appointments)
Estimated fan-out: 3+

2. Fan in

1) User Registration
Depended on by Verification, Profile Management, Appointment booking, Prescription Management, Medical record management
Estimated fan-in: 5+

2) Medical Record Management
Depended on by Appointment Booking, Prescription Management.
Estimated fan-in: 2+

3) Profile Management
Depended on by Appointment Booking, medical record management
Estimated fan-in: 2+

Detailed Design Specification

1. User Registration

```
class UserRegistration:
    def __init__(self, user_id, name, email, role):
        self.user_id = user_id
        self.name = name
        self.email = email
        self.role = role  # "patient", "doctor", "admin"

    def register_user(self, details):
        """Registers a new user (patient, doctor, or hospital)."""
        pass
    def update_user_info(self, user_id, updated_details):
        """Updates user registration details."""
        pass
```

2. Verification

```
class Verification:
    def __init__(self, user_id, verification_status):
        self.user_id = user_id
        self.verification_status = verification_status  # "pending", "verified", "rejected"

    def verify_patient(self, patient_id, documents):
        """Verifies patient identity using submitted documents."""
        pass

    def verify_doctor(self, doctor_id, license_details):
        """Verifies doctor credentials before registration approval."""
        pass
```

3. Profile Management

```
class Profile:
    def __init__(self, user_id, profile_type, details):
        self.user_id = user_id
        self.profile_type = profile_type  # "patient", "doctor"
        self.details = details

    def update_profile(self, updates):
        """Updates user profile information (contact details, specialization, etc.)."""
        pass
```

4. Appointment Booking

```
class Appointment:
    def __init__(self, appointment_id, patient_id, doctor_id, date, status):
        self.appointment_id = appointment_id
        self.patient_id = patient_id
        self.doctor_id = doctor_id
        self.date = date
        self.status = status  # "pending", "confirmed", "canceled"
    def request_appointment(self, patient_id, doctor_id, date):
        """Allows a patient to request an appointment with a doctor."""
        pass

    def check_availability(self, doctor_id, date):
        """Checks doctor's availability for an appointment."""
        pass

    def update_status(self, appointment_id, new_status):
        """Updates the status of an appointment."""
        pass
```

5. Record Management

```
class MedicalRecord:
            def __init__(self, record_id, patient_id, doctor_id, diagnosis, prescriptions, notes):
                    self.record_id = record_id
                    self.patient_id = patient_id
                    self.doctor_id = doctor_id
                    self.diagnosis = diagnosis
                    self.prescriptions = prescriptions
                    self.notes = notes
             def add_record(self, patient_id, doctor_id, diagnosis, prescriptions, notes):
                    """Creates a new medical record for a patient."""
                    pass
             def update_record(self, record_id, new_details):
                    """Updates an existing medical record."""
                    Pass
```

6. Prescription Management

```
class Prescription:
    def __init__(self, prescription_id, patient_id, doctor_id, medicines, instructions):
        self.prescription_id = prescription_id
        self.patient_id = patient_id
        self.doctor_id = doctor_id
        self.medicines = medicines
        self.instructions = instructions

    def add_prescription(self, patient_id, doctor_id, medicines, instructions):
        """Creates a new prescription for a patient."""
        pass
    def update_prescription(self, prescription_id, new_medicines, new_instructions):
        """Updates an existing prescription."""
        Pass
```

7. Notification

```
class Notification:
    def __init__(self, notification_id, user_id, message, timestamp):
        self.notification_id = notification_id
        self.user_id = user_id
        self.message = message
        self.timestamp = timestamp

    def send_notification(self, user_id, message):
        """Sends a notification to a user."""
        Pass
```

8. Calendar Management

```
class Calendar:
    def __init__(self, doctor_id, schedule):
        self.doctor_id = doctor_id
        self.schedule = schedule  # Dictionary of available time slots

    def update_schedule(self, doctor_id, new_schedule):
        """Updates the doctor's available time slots."""
        pass
```