

Name:

M Hamed Raza

ID:

F2023065255

Section:

W6 (Mam Kinza Sardar)

SOFTWARE ENGINEERING PRPOJECT

Introduction:

* Project Overview

My project is to create an online app named medico to book online appointments of doctors.

* Problem Statement

Wants to create an online platform to give patients online appointment of their desired doctor to reduce rush in hospitals and also give facilitation to patients.

* Objectives

It will be an online app which will help the user to get the appointment of a particular doctor, it will prevent the patients to stand in rows and wait for the appointment for a long time.

**Tools:**

**. Frontend Development:**

* **Languages**: HTML, CSS, JavaScript
* **Frameworks**: React.js, Vue.js (web), Flutter, React Native (mobile)
* **Design Tools**: Figma, Adobe XD, Sketch
* **Libraries**: Bootstrap, Tailwind CSS, Axios

**2. Backend Development:**

* **Languages**: Node.js, Python (Django/Flask), Java (Spring Boot)
* **Database**: MySQL, PostgreSQL, MongoDB, Firebase
* **Authentication**: JWT, OAuth, Passport.js
* **Payment Integration**: Stripe, PayPal, Jazz Cash, Razorpay

**3. Mobile App Development:**

* **Frameworks**: React Native, Flutter
* **Native Development**: Xcode (iOS), Android Studio (Android)

**4. Cloud & Hosting:**

* **Platforms**: AWS, Google Cloud, Azure, Heroku, DigitalOcean

**5. Security:**

* **SSL/TLS**, **Two-Factor Authentication (2FA)**, **Helmet.js**, **OWASP ZAP**

**6. Testing Tools:**

* **Testing**: Jest, Mocha, Cypress, Appium
* **API Testing**: Postman

**7. Version Control:**

* **Tools**: Git, GitHub, GitLab

**8. Deployment & CI/CD:**

* **Tools**: Docker, Jenkins, CircleCI, GitHub Actions

**9. Notification System:**

* **Services**: Twilio (SMS), SendGrid (Email), Firebase Cloud Messaging (Push)

**10. Analytics:**

* **Tools**: Google Analytics, Firebase Analytics, Mixpanel

This set of tools ensures efficient development, security, scalability, and smooth user experience.

**SOFTWARE MODEL:**

The **Agile Model** is most suited for the **Medico** system due to its iterative, flexible nature, enabling continuous development, frequent releases, and collaboration with stakeholders.

Here are the **reasons** for choosing the **Agile Model** for the **Medico** project:

1. **Iterative Development**: Allows for regular updates and enhancements.
2. **Flexibility**: Can easily accommodate changing requirements (new features, feedback).
3. **Frequent Releases**: Enables early deployment of basic features with continuous updates.
4. **Customer Involvement**: Continuous collaboration with clients and stakeholders.
5. **Faster Time-to-Market**: Quicker development cycles and earlier availability of key features.

These benefits make **Agile** ideal for the **Medico** system.

**FUNCTIONAL REUIREMENTS:**

**1.User Registration:**

* Users must be able to register using their email and phone number.
* A verification code is sent via email/SMS for authentication.

**2. User Authentication:**

* Users must authenticate using the verification code sent to their email/phone.
* After successful authentication, the user can access the app.

**3. Doctor Profiles:**

* The system should display detailed doctor profiles, including qualifications, specialization, and available time slots.
* Doctors should be able to update their availability.

**4. Search for Doctors:**

* Users can search for doctors by location or specialty.
* The app should display doctor profiles based on user location or selected filters.

**5. Appointment Booking:**

* Users can select a doctor and an available time slot for an appointment.
* Users must confirm the appointment before proceeding with payment.

**6. Payment Processing:**

* Users must be able to make payments through various methods (e.g., e-banking, Jazz Cash, PayPal).
* The system should process payments securely and issue a final appointment slip after successful payment.

**7. Appointment Confirmation:**

* Once payment is made, users will receive an appointment confirmation via email/SMS with details (doctor, time, payment receipt).

**8. Appointment Reminders:**

* The system should send reminders to users about their upcoming appointments via email and SMS.

**9. Location-Based Services:**

* The app should suggest nearby doctors based on the user's current location or entered area.

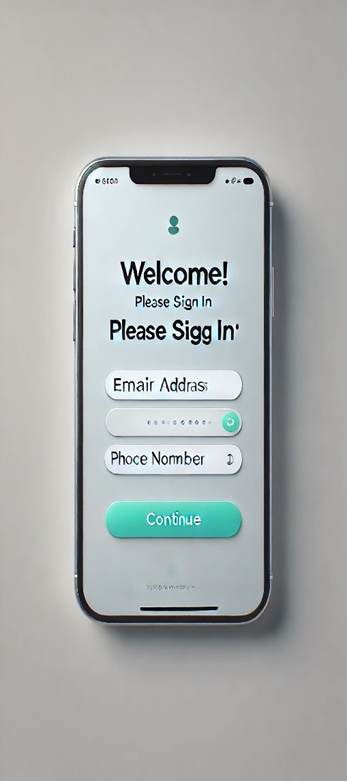
**10. Admin Management:**

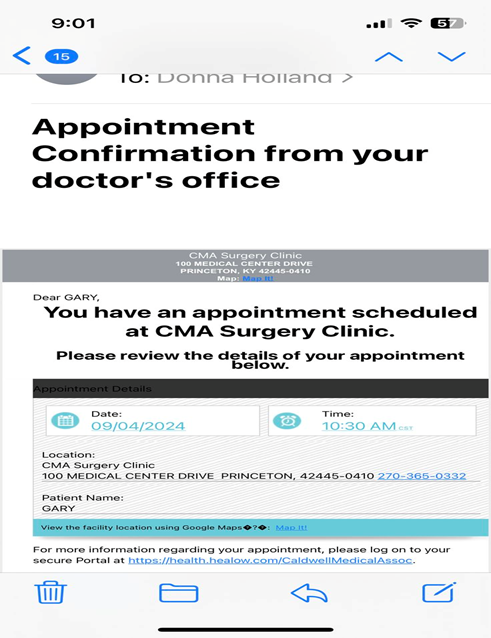
* Admin should be able to manage doctor registrations, profiles, and availability.
* Admin can also oversee user accounts and system operations.

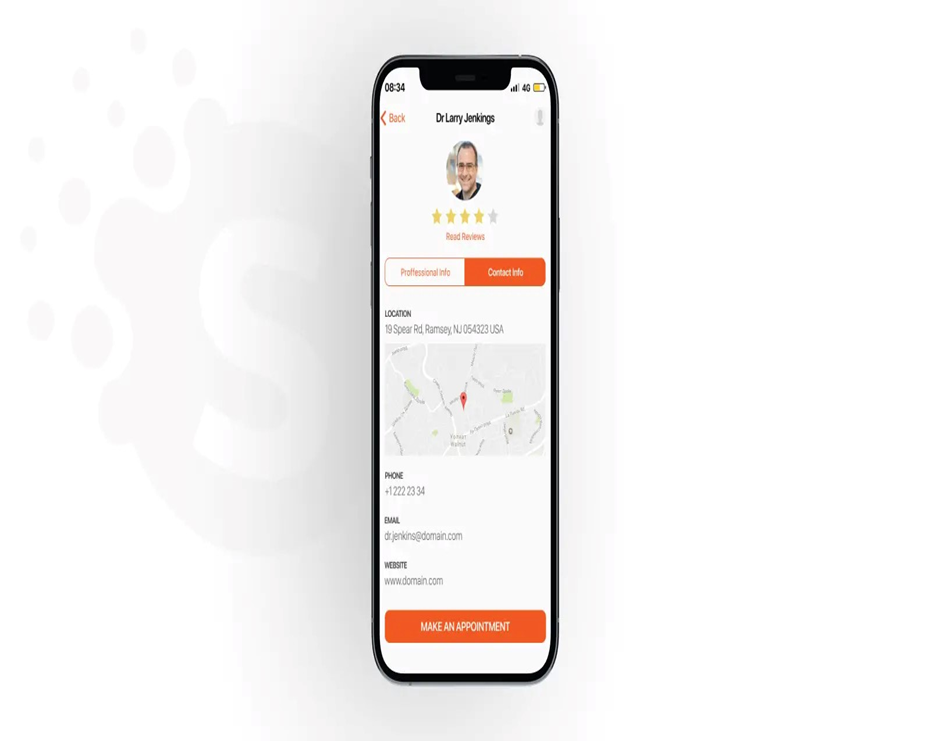
**NON-FUNCTIONAL REQUIREMENTS:**

1. **Data Security**: Encrypt user data, secure communication, two-factor authentication.
2. **Performance and Scalability**: Handle high traffic, scalable architecture.
3. **Reliability and Availability**: 99.9% uptime, regular backups.
4. **Usability**: Simple, intuitive UI, responsive design.
5. **Compatibility**: Cross-platform support (Android, iOS, web).
6. **Maintainability**: Modular system, error logging, quick issue resolution.
7. **Security**: Protection from unauthorized access, role-based access control.
8. **Localization**: Multi-language support, regional payment methods.
9. **Backup and Recovery**: Regular data backups, disaster recovery plan.
10. **Legal and Compliance**: Compliance with GDPR, HIPAA regulations.

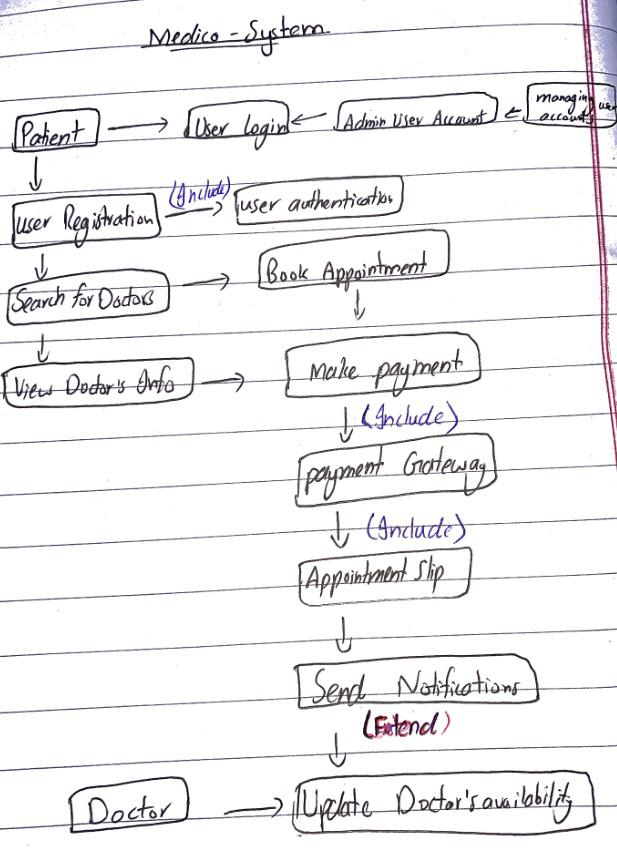
**PROTOTYPES**

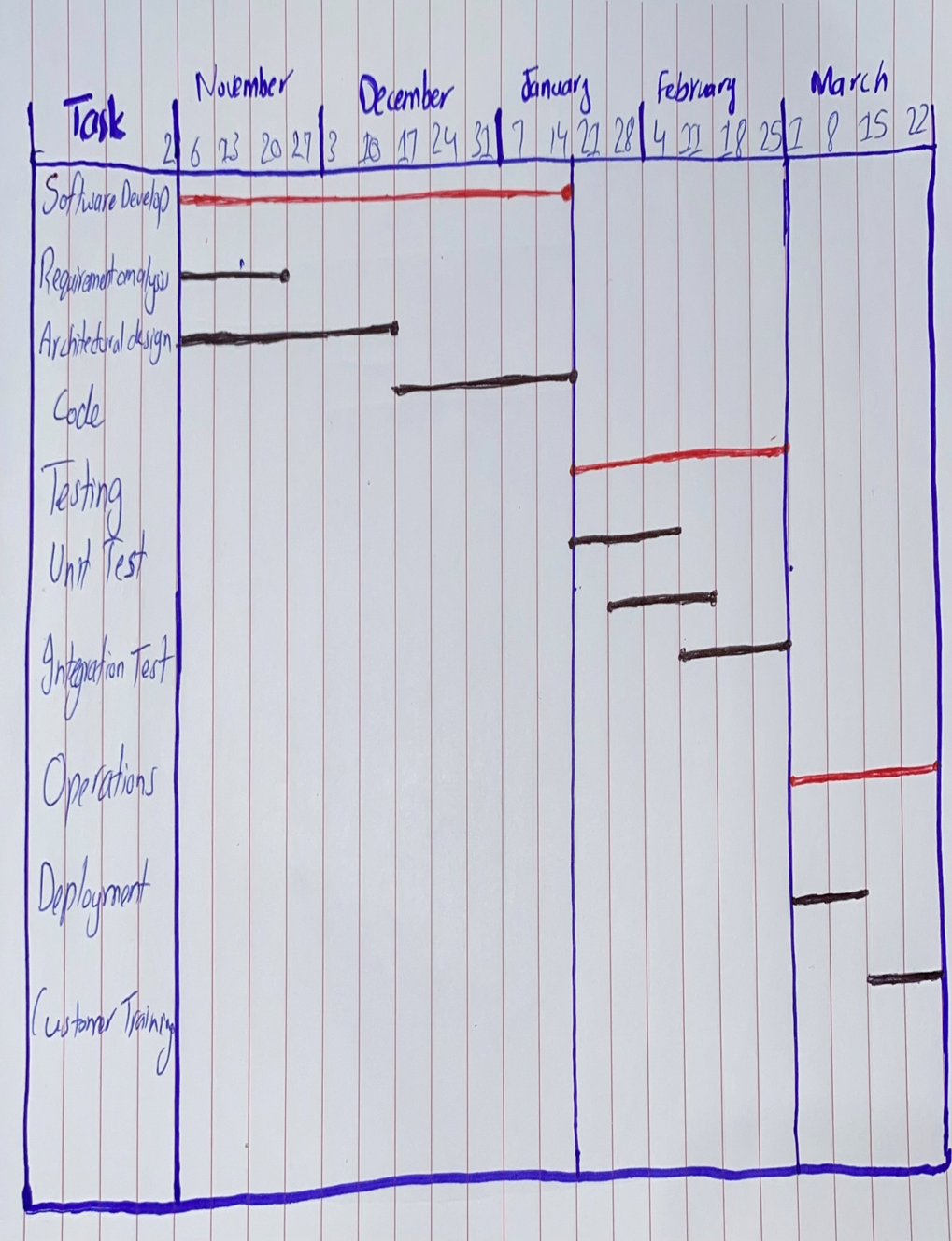




**USE CASE DIAGRAM**



**GANTT CHART**

****

**Architecture type:**

For the **Medico** online appointment booking app, the **Client-Server Architecture** would be the most suitable choice. Here's why:

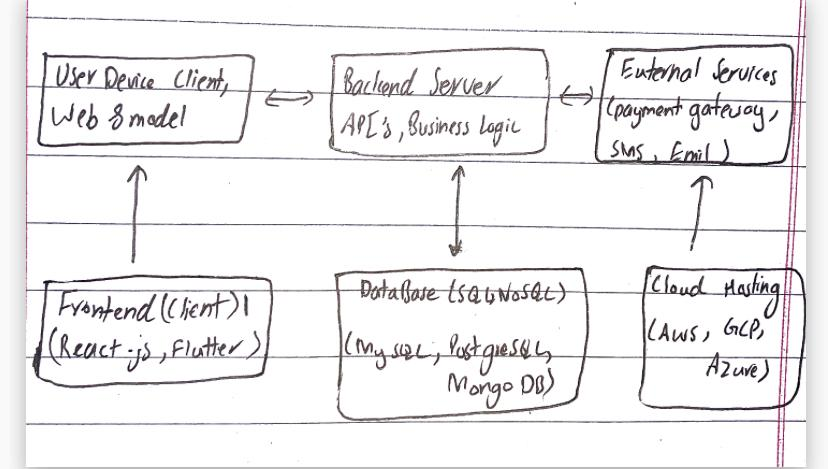
**Client-Server Architecture:**

* **Frontend (Client-side)**: The app will interact with the users (patients, doctors, admins) through a mobile/web interface, making it the client-side.

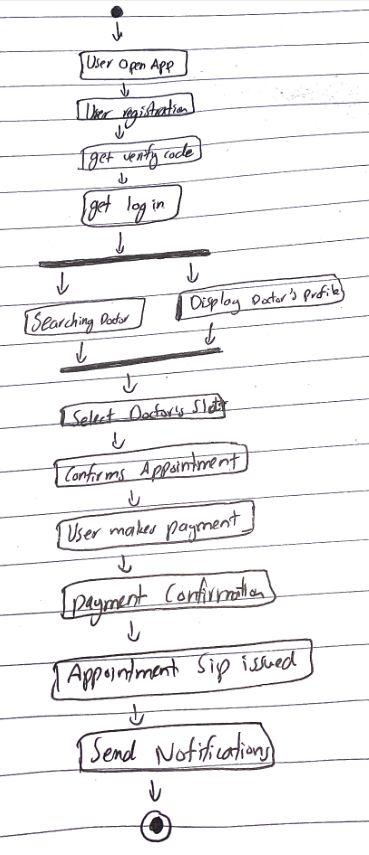
### **Backend (Server-side)**: The server will handle the business logic, manage data (appointments, doctor profiles, user accounts, etc.), and integrate with external systems like the payment gateway.

### **Why Client-Server Architecture?**

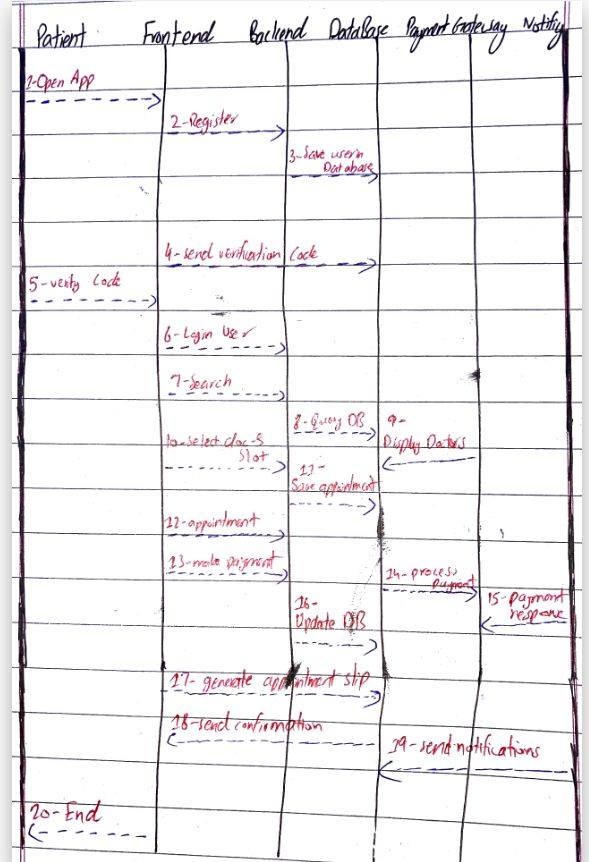
* **Separation of Concerns**: It separates the user interface (frontend) and the business logic (backend), ensuring better maintainability and scalability.
* **Security**: The backend server can ensure secure communication, handle data storage, and protect sensitive user information like payment details and medical data.
* **Scalability**: It allows for easy scaling. The server can be upgraded to handle more requests as the number of users grows.
* **Interoperability**: The client can interact with the server using APIs, making it easy to update or modify either side without affecting the other.



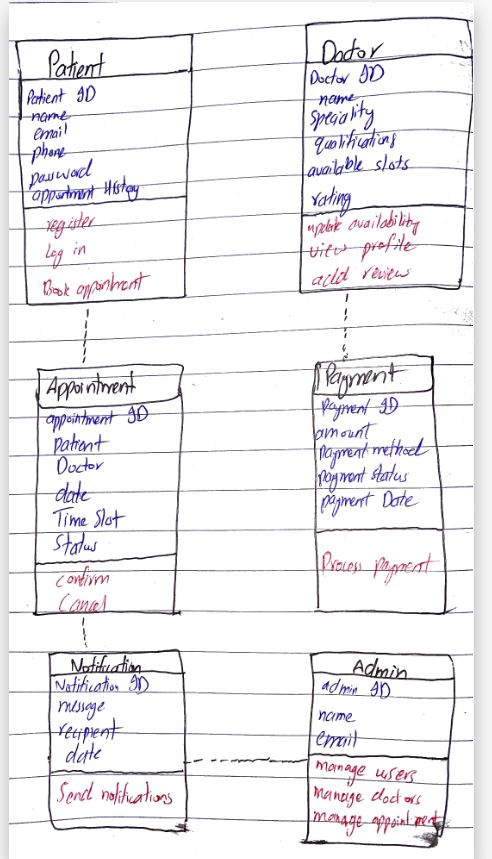
**Activity Diagram**

****

**Sequence Diagram**

****

**Class Diagram**



**Relationship among Classes**

Patient ↔ Appointment: One-to-Many Association.

Doctor ↔ Appointment: One-to-Many Association.

Patient ↔ Payment: One-to-One Association.

Appointment ↔ Doctor: One-to-Many Association.

Payment ↔ Payment Gateway: Dependency.

Notification ↔ Patient/Doctor: Dependency.

Patient ↔ Appointment: Aggregation.

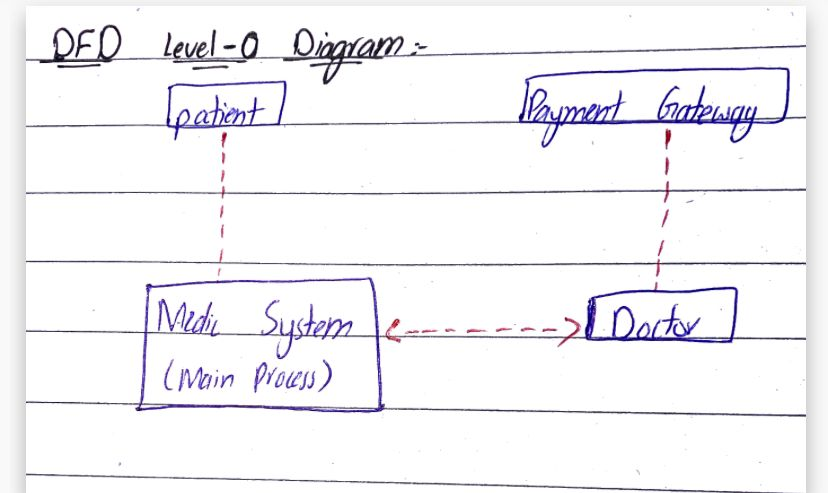
Doctor ↔ User (if you introduce a User superclass): Generalization (Inheritance).

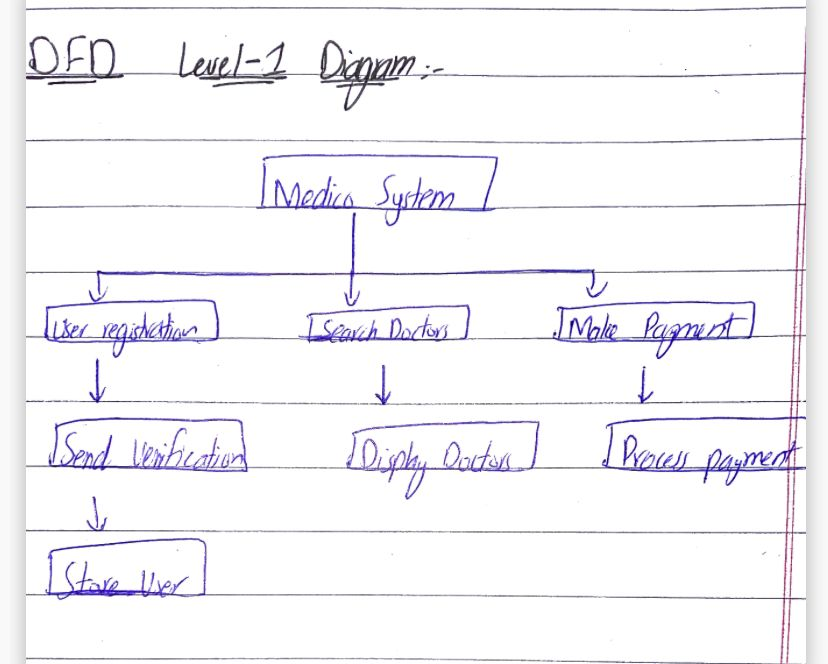
**Conclusion:**

**One-to-many association**: A **Patient** can book multiple appointments over time, but each **Appointment** is linked to a specific **Patient**.

* Example: A patient can book several appointments with different doctors or even the same doctor at different times.

**DFD level-0 Diagram**

****

**D** **FD level-1 Diagram **

**Testing Techniques**

**1. Functional Testing:**

* **Purpose**: Verify the system's core functionality works (e.g., registration, appointment booking, payment).
* **Technique**: **Black-box Testing** (testing without knowing the internal structure).

**2. Usability Testing:**

* **Purpose**: Ensure the app is user-friendly and intuitive.
* **Techniques**:
  + **Heuristic Evaluation** (expert review).
  + **User Acceptance Testing (UAT)** (real-user testing).

**3. Security Testing:**

* **Purpose**: Ensure user data is secure and protected.
* **Techniques**:
  + **Penetration Testing** (simulate attacks).
  + **Static & Dynamic Code Analysis** (analyze code for vulnerabilities).

**4. Performance Testing:**

* **Purpose**: Test system performance under load and stress.
* **Techniques**:
  + **Load Testing** (normal user load).
  + **Stress Testing** (extreme load to check limits).
  + **Scalability Testing** (system’s ability to handle growth).

**5. Integration Testing:**

* **Purpose**: Verify interactions between system components (frontend, backend, payment gateway).
* **Techniques**:
  + **Top-Down Testing** (test high-level to low-level).
  + **Bottom-Up Testing** (test low-level to high-level).

**6. Regression Testing:**

* **Purpose**: Ensure new changes don't affect existing features.
* **Technique**: **Automated Regression Testing** (automate critical features tests).

**7. Compatibility Testing:**

* **Purpose**: Ensure the app works across devices, platforms, and browsers.
* **Techniques**:
  + **Cross-Browser Testing** (different browsers).
  + **Cross-Device Testing** (various devices and screen sizes).

**8. System Testing:**

* **Purpose**: Validate the entire system’s behavior.
* **Technique**: **End-to-End Testing** (test the full user journey).