

# HEALTH CARE SYSTEM

## PROJECT REPORT

Submitted in partial fulfillment for the award of the degree  
of

## BACHELOR OF TECHNOLOGY

In Computer science and engineering

Session – 2022-23

Submitted by

**KHUSHBOO SONI (2006480109013)**

**RANJEET KUMAR (2006480109024)**

**SURAJ PRAJAPATI (2006480109027)**

Under the Supervision of

**Mr. Hira Singh Yadav**  
**(HOD CSE)**



Babu Sunder Singh Institute of Technology And Management Lucknow, Uttar Pradesh



**DR. A.P.J. ABDUL KALAM TECHNICAL UNIVERSITY**  
**UTTAR PRADESH, LUCKNOW**



## **Faculty of Computer Science & Engineering**

---

### **DECLARATION**

This is to certify that this thesis work entitled "**HEALTH CARE SYSTEM**" Which is submitted by me in partial fulfilment of there requirement forth degree? completion of Bachelor of Technology (B.Tech), in Computer Science and Engineering from Babu Sunder Singh Institute of Technology & Management, Lucknow unified only my original work and due acknowledgement has been made in text to all other materials used. This thesis work was done under the guidance of **Mr. Manish Singh** and the content of the thesis do not form the basis for the award of any other degree to me or to anybody else from University/Institution. I have given due credit to the original authors and sources for all the words, ideas, diagrams, graphics, computer programs, experiments, results that are not my original contribution. I have used quotation marks to identify sentences and given credit to the original authors and sources.

**Name: Khushboo Soni**

**Ranjeet Kumar**

**Suraj Prajapati**

**Branch: B.tech (CSE)**  
**(Khushboo Soni, Ranjeet Kumar, Suraj Prajapati)**

## **CERTIFICATE**

I hereby certify that the work which is being presented under my supervision by **Mr. M.B.Singh** titled "**Health Care System**" in partial fulfilment of the requirement for the award of the **Bachelor of Technology** and submitted the department of **Computer Science and Engineering** of Babu Sunder Singh Institute of Technology & Management, Lucknow.

The foregoing thesis is hereby accept as a creditable study in the area of Computer Science and Engineering carried out under my guidance and presented the work in a manner satisfactory to permit its acceptance as a prerequisite to the degree for which it has been submitted.

**Signature**

**Mr. Manish Singh**

**Assistant Professor (CSE)**

**Place: Lucknow**

**Date:**

## **ACKNOWLEDGEMENTS**

The successful completion of any task would be incomplete without the mention of people who made it complete. I would like to take this opportunity to express my deep sense of gratitude to all who helped me directly or indirectly during this thesis work. First of all, also would like to express my deepest gratitude to my thesis supervisor, for his enormous help and advice and for providing inspiration which cannot be expressed with words. I would not have accomplished this thesis without his patient care, understanding and encouragement. His advice, encouragement and critics are source of good ideas, inspiration and causes behind the successful completion of this thesis work. The confidence shown on me by him was the biggest source of inspiration for me.

I would like to thank all the faculty members of Babu Sunder Singh Institute of Technology and Management, Lucknow for their direct or indirect support throughout the work. This thesis is the result of one year work and the result cannot possible without support by many peoples.

I would like to express my deep gratitude to **Mr. Hira Singh Yadav** on behalf of the head of department computer science and engineering department of Babu Sunder Singh Institute of Technology & Management, Lucknow, for providing access to the good integration of intellectual properties, technical support and facilities.

Finally I would like to heartily thank my parents without his support nothing is possible for me and also thanks to my friends whose love, patience, cooperation and support helped me complete this work.

**RANJEET KUMAR (2006480109024)**

**KHUSHBOO SONI (2006480109013)**

**SURAJ PRAJAPATI (2006480109027)**

## **ABSTRACT**

Online Doctor appointment is a smart web application, this provides registration and login for both doctors and patients. Doctors can register by giving their necessary details like timings, fee, category, etc. After successful registration, the doctor can log in by giving a username and password. The doctor can view the booking request by patients and if he accepts the patient requests the status will be shown as booking confirmed to the patient. He can also view the feedback given by the patient. The patients must be registered and log in to book a doctor based on the category and the type of problem patient is facing and the location. The search results will show the list of doctors matching patients required criteria and he can select one and send a request the request will be forwarded to admin and admin forward to doctor and if he is available, he will send the confirmation request back to admin the admin update the booking request and says confirmed to the patient. The patient can view the status in the status tab and he will get the mail saying the booking is Confirmed.

**The Application has following modules:-**

- **Admin**
- **Doctor**
- **Patient**

## **TABLE OF CONTENT**

<b>1. INRODUCTION</b>	<b>(8-9)</b>
1.1. Purpose	
1.2. Scope	
1.3. System Overview	
<b>2. LITERATURE SURVEY</b>	<b>(10-12)</b>
2.1. Existing Solution	
2.2. Proposed Solution	
<b>3. SOFTWARE REQUIREMENT &amp; SPECIFICATION</b>	<b>(13-18)</b>
3.1. Introduction	
3.2. Users Perspective	
3.3. Functional Requirements	
3.4. Non-Functional Requirements	
3.5. Hardware Needs	
3.6. Software Needs	
3.7. Constraints	
3.8. Assumptions and Dependencies	
3.9. User Interfaces	
3.10.     Use Cases	
3.11.     System Architecture	
3.12.     Data Requirements	
3.13.     Testing and Validation	

<b>4. ANALYSIS OF SYSTEM</b>	<b>(19-23)</b>
4.1. User Analysis	
4.2. Usability Analysis	
4.3. Performance Analysis	
4.4. Security and Privacy Anal Integration Analysis	
4.5. System Reliability and Availability Analysis	
4.6. Data Analysis	
4.7. Stakeholder Analysis	
4.8. Competitor Analysis	
<b>5. TESTING</b>	<b>(24-27)</b>
5.1. Definition	
5.2. Validation and System Testing	
5.3. Integration Testing	
5.4. Unit Testing	
<b>6. LIST OF FIGURES</b>	<b>(28-35)</b>
<b>7. USE CASE DESCRIPTION</b>	<b>(36-45)</b>
7.1. PATIENT	
7.2. DOCTOR	
7.3. ADMIN	
7.4. Data Dictionary	
7.5. Data Design	
7.6. Appointment	
7.7. Prescription	
7.8. ER diagram	
7.9. Component Level diagram	
<b>8. MODULE WISE CODE</b>	<b>(46-106)</b>
<b>9. RESULT CONCLUSION</b>	<b>(107)</b>

# 1. Introduction

## 1.1 Purpose

The online healthcare system will be used by hospitals, clinics and other medical centres to manage every aspect of their patient's management. This software substitutes the tasks which a hospital staff member would usually perform by allowing patients to schedule their own appointments, to check the result of their laboratory tests and to permit doctors to manage appointments and records of patients. The acquisition, management and timely recovery of large amounts of information constitute a significant part of the operation of every hospital. Typically, this includes: patient personal information and medical history, staff information, the schedule of rooms and offices, staff schedules, scheduling of operation theatre and different waiting lists of facilities. All this information must be effectively managed and cost-effectively in order to make effective use of institutional resources our healthcare system automates hospital management to improve efficiency and to prevent errors. The objective is to standardised data, to consolidate data integrity and reducing data leakage and human errors.

## 1.2 Scope

This web based medical system provides patients, system manager/admin and doctors with the option of scheduling and managing their appointments, managing patient records, managing medical records from their own homes and offices using the latest technology. Our health care system is able to do many tasks which are interdependent, including patients and doctors.

There are four user interfaces for our system: the patient interface, the physician's interface and the admin interface, the staff member interface.

The software has 3 types of users in our system: • Doctor • Administrator • Patient.

## 1.3 System Overview

There Three Main Module in Automated Doctor Consulting System they are:

### a) Admin:

Admin needs to login with username and password and in the admin home screen, he can see the basic functionalities of admin. Admin can add doctors and view doctor. He can also view the users request and he will confirm the users. Admin can view the appointment and view feedback of users. Admin can able to control the whole system. He/she can add, delete, update and modify the system. Admin keeps the system up-to-date.

### b) Doctor:



Doctor need to be registered by giving the necessary details like experience, timing, fees etc. After registering he need to log in and in the home screen he can view the basic functionalities. Doctors are able to see the respective appointments taken Doctor should fairly know about the usage of the system. He can view the users request forwarded from admin and he can accept and he can also view the feedback given by users.

### **c) Patient:**

The patient needs to be registered and log in after logging on he can search for the doctor by giving the location. Base on the doctor availability the admin will confirm the booking request and will send to mail that the booking is confirmed he can also view in the status and he can also give feedback basing the performance of the doctor.

## 2. Literature Survey

### 2.1 Existing Solution

There are numerous existing solutions and interventions within healthcare systems that aim to address various challenges and improve healthcare outcomes. Here are some examples of healthcare solutions:

1. **Electronic Health Records (EHRs):** EHRs are digital versions of patients' medical records that provide a centralised and comprehensive view of their health information. EHRs enhance care coordination, reduce errors, improve communication among healthcare providers, and enable efficient access to patient data.
2. **Telemedicine and Remote Healthcare:** Telemedicine allows patients to receive medical consultations and treatment remotely using technology, such as video conferencing. It improves access to healthcare, particularly for individuals in remote areas or with limited mobility. Telemedicine also enables remote monitoring of patients with chronic conditions, reducing hospital visits and improving disease management.
3. **Preventive Care and Health Promotion Programs:** Healthcare systems emphasise preventive care through various programs and initiatives. These can include vaccinations, cancer screenings, health education campaigns, and lifestyle interventions aimed at reducing the incidence of diseases and promoting healthy behaviour's.
4. **Health Information Exchange (HIE):** HIE systems enable the secure sharing of patient health information among different healthcare organisations. They facilitate seamless communication, improve care coordination, reduce duplication of tests or procedures, and enhance patient safety and continuity of care.
5. **Chronic Disease Management Programs:** Healthcare systems often implement specialised programs for managing chronic diseases. These programs provide comprehensive care, patient education, regular monitoring, and personalised treatment plans to individuals with conditions such as diabetes, heart disease, or asthma. They aim to improve health outcomes, reduce complications, and enhance patients' quality of life.
6. **Quality Improvement Initiatives:** Healthcare systems engage in continuous quality improvement efforts. These initiatives involve implementing evidence-based practices, setting clinical guidelines and standards, conducting performance assessments, and utilising data-driven approaches to enhance patient safety, reduce medical errors, and improve healthcare outcomes.
7. **Health Promotion and Education Campaigns:** Healthcare systems conduct public health campaigns to raise awareness about specific health issues, promote healthy behaviours, and provide education on disease prevention. These campaigns often target topics such as smoking cessation, healthy eating, physical activity, and mental health.
8. **Integrated Care Models:** Integrated care models aim to improve coordination and collaboration among healthcare providers, including primary care physicians, specialists, and other healthcare professionals. These models enhance care transitions, reduce fragmentation, and ensure that patients receive comprehensive and coordinated care across different healthcare settings.
9. **Medical Research and Innovation:** Ongoing medical research leads to advancements in diagnosis, treatment, and medical technology. Research efforts contribute to the development of new therapies, medical devices, and treatment protocols, ultimately improving patient outcomes and expanding the knowledge base within healthcare.

10. **Health Policy and Regulation:** Governments and regulatory bodies play a crucial role in shaping healthcare systems through policies and regulations. These measures ensure patient safety, govern ethical standards, define reimbursement frameworks, and establish guidelines for healthcare delivery, pharmaceuticals, and medical devices.

## 2.2 Proposed Solution

### Proposed Solution for Healthcare:

1. **Telehealth and Remote Care:**
  - Implement and promote Telehealth services to provide remote consultations, diagnosis, and treatment for non-emergency medical conditions.
  - Develop user-friendly Telehealth platforms or mobile applications that facilitate secure video or audio communication between patients and healthcare providers.
  - Ensure integration with electronic health records (EHRs) to maintain comprehensive patient records and enable seamless information exchange.
  -
2. **Health Information Exchange (HIE) and Interoperability:**
  - Establish a robust health information exchange system that enables secure and seamless sharing of patient data among healthcare providers, clinics, hospitals, and laboratories.
  - Adopt standardised data formats and interoperability standards to ensure compatibility and smooth communication between different systems.
  - Facilitate the secure exchange of patient information, including medical history, test results, and treatment plans, to improve care coordination and patient outcomes.
  -
3. **Electronic Health Records (EHRs) and Clinical Decision Support Systems:**
  - Implement comprehensive electronic health record systems that capture and store patient health information in a structured format.
  - Ensure EHRs are user-friendly, accessible, and designed to improve efficiency and accuracy in documenting patient encounters, medication management, and care planning.
  - Integrate clinical decision support systems that provide real-time alerts, reminders, and evidence-based recommendations to healthcare providers, enhancing clinical decision-making and patient safety.
  -
4. **Remote Patient Monitoring and Wearable Technology:**
  - Utilise remote patient monitoring devices and wearable technology to track and monitor vital signs, chronic conditions, and overall wellness.
  - Enable patients to transmit their health data to healthcare providers, allowing for proactive intervention and personalised care plans.
  - Develop algorithms and analytics to analyse remote monitoring data, identify trends, and provide early detection of health issues, enabling timely interventions.
  -
5. **Population Health Management:**
  - Implement population health management strategies to proactively address the health needs of specific populations or communities.
  - Utilise data analytics to identify high-risk groups, assess health trends, and develop targeted interventions for prevention, education, and disease management.
  - Collaborate with community organisations, public health agencies, and social services to address social

determinants of health and promote holistic well-being.

6. Mobile Health Applications and Patient Engagement:

- Develop mobile health applications that empower patients to actively participate in their healthcare journey.
- Provide features such as appointment scheduling, medication reminders, access to personal health records, educational resources, and personalised health recommendations.
- Encourage patient engagement through secure messaging, virtual support groups, and gamification elements to promote adherence to treatment plans and healthy behaviours.

7. Artificial Intelligence (AI) and Machine Learning (ML) in Healthcare:

- Explore the use of AI and ML algorithms to improve diagnostics, predict disease outcomes, and personalise treatment plans.
- Leverage AI-powered chatbots or virtual assistants to provide 24/7 patient support, answer common queries, and triage symptoms.
- Utilise predictive analytics to optimise healthcare resource allocation, staffing, and capacity planning.

8. Health Education and Preventive Care:

- Develop comprehensive health education programs targeting different populations to promote preventive care, healthy lifestyles, and disease management.
- Collaborate with educational institutions, community organisations, and healthcare providers to disseminate accurate and accessible health information.
- Implement preventive care initiatives such as vaccination campaigns, screening programs, and early detection programs for diseases like cancer and chronic conditions.

9. Data Security and Privacy Measures:

- Ensure robust data security protocols to protect patient information from unauthorised access, breaches, and cyber threats.
- Comply with data privacy regulations, such as GDPR or HIPAA, and establish stringent data governance frameworks.
- Educate healthcare professionals and staff about data security best practices and provide regular training to mitigate risks.

10. Collaboration and Interdisciplinary Care:

- Promote collaboration and interdisciplinary care among healthcare providers, including primary care physicians, specialists, nurses, and allied health professionals.
- Facilitate care coordination through shared care plans, multidisciplinary meetings, and secure communication channels.
- Foster a culture of teamwork, mutual respect, and knowledge sharing among healthcare professionals to optimise patient care outcomes.

## 3. Software Requirements & Specification

### 3.1 Introduction

- Provide an overview of the online doctor appointment system and its purpose.
- Describe the intended users of the system (patients, doctors, administrators, etc.).
- Explain the benefits and objectives of implementing the system.

### 3.2 Users Perspective

- Users expect easy access to healthcare services, including the ability to find healthcare providers, schedule appointments, and access their health information conveniently.
- Online platforms and mobile applications can provide 24/7 accessibility, allowing users to book appointments, receive Telehealth consultations, access test results, and manage prescriptions from the comfort of their homes.
- Healthcare systems should have intuitive and user-friendly interfaces that are easy to navigate, regardless of the user's technical proficiency.
- Clear and concise design, with well-organised information and intuitive workflows, can enhance the user experience and minimise confusion or frustration.
- Users value personalised care that takes into account their specific needs, preferences, and medical history.
- The system should allow users to provide relevant information about their health conditions, allergies, medications, and preferences, enabling healthcare providers to deliver tailored care and treatment plans.
- Users expect easy access to their personal health records, including medical history, test results, prescriptions, and treatment plans.
- Providing a comprehensive and up-to-date overview of their health information empowers users to actively participate in their care decisions and facilitates continuity of care across different healthcare providers.

### 3.3 Functional Requirements

Functional requirements for a healthcare system typically revolve around the core functionalities and capabilities that enable the system to effectively manage and support healthcare processes. Here are some common functional requirements for a healthcare system:

1. User Registration and Authentication:
  - Allow users to register and create accounts with appropriate authentication measures, such as username/password, two-factor authentication, or biometric authentication.
2. Patient Management:
  - Enable healthcare providers to manage patient information, including demographics, medical history,

- allergies, medications, and contact details.
  - Support the creation and maintenance of patient profiles, including the ability to update and track changes over time.
3. Appointment Scheduling and Management:
    - Facilitate the scheduling of appointments between patients and healthcare providers, considering availability, specialty, location, and preferred time slots.
    - Allow users to view available appointment slots, book, reschedule, or cancel appointments as needed.
    - Send appointment reminders to patients and healthcare providers through various communication channels.
  4. Electronic Health Records (EHR):
    - Provide a secure and centralised repository for storing and accessing patient health records.
    - Capture and organise essential patient information, including diagnoses, procedures, medications, test results, and treatment plans.
    - Enable healthcare providers to view, update, and share patient records securely, ensuring data integrity and privacy.
  5. Medical Billing and Insurance Management:
    - Support the management of medical billing and insurance-related activities, including generating invoices, processing insurance claims, and tracking payments.
    - Integrate with billing systems and insurance databases to validate coverage, verify eligibility, and streamline the reimbursement process.
  6. Prescription and Medication Management:
    - Allow healthcare providers to prescribe medications electronically, including dosage instructions and refill information.
    - Enable patients to access their medication history, request prescription refills, and receive notifications for medication adherence.
    - Integrate with pharmacy systems to transmit prescriptions and facilitate accurate dispensing of medications.
  7. Laboratory and Test Result Management:
    - Support the ordering, tracking, and management of laboratory tests and diagnostic procedures.
    - Facilitate the electronic submission and retrieval of test requests and results.
    - Enable healthcare providers to review and interpret test results, generate reports, and communicate findings to patients.
  8. Communication and Messaging:
    - Provide secure and reliable communication channels for patients and healthcare providers to exchange messages, ask questions, and share information.
    - Allow users to send and receive messages, attachments, and notifications within the healthcare system.
  9. Telehealth and Remote Consultations:
    - Enable virtual consultations and remote healthcare services through secure video or audio communication.
    - Facilitate the scheduling, documentation, and reimbursement of Telehealth visits.
    - Integrate with other system functionalities, such as appointment scheduling and electronic health records, to ensure seamless Telehealth experiences.
  10. Reporting and Analytics:
    - Generate reports and analytics to support decision-making, performance monitoring, and quality
-

improvement initiatives.

- Provide insights on patient outcomes, resource utilisation, wait times, and other key performance indicators to optimise healthcare delivery.

### 3.4 Non-Functional Requirements

Non-functional requirements for a healthcare system focus on the quality attributes and characteristics of the system that go beyond its basic functionality. These requirements address aspects such as performance, security, usability, reliability, and scalability. Here are some common non-functional requirements for a healthcare system:

1. Performance:
  - Response Time: The system should provide fast response times to ensure a smooth user experience and minimise waiting times.
  - Throughput: The system should be able to handle a high volume of concurrent users and transactions without performance degradation.
  - Scalability: The system should be scalable to accommodate increasing user loads, especially during peak usage periods.
2. Security:
  - Data Privacy: The system should comply with relevant data protection regulations and ensure the privacy and confidentiality of patient information.

Introduction

Users Perspective

Functional Requirements

Non-Functional Requirements

Hardware Needs

Software Needs

Constraints

Assumptions and Dependencies

User Interfaces

Use Cases

System Architecture

Data Requirements

Testing and Validation

- Authentication and Authorisation: The system should have secure user authentication mechanisms and enforce appropriate access controls to protect sensitive data.
- Encryption: Sensitive data transmitted within the system should be encrypted to prevent unauthorised

access or interception.

- **Audit Trail:** The system should maintain an audit trail of user activities and system changes for security monitoring and forensic purposes.

3. **Usability:**

- **User Interface Design:** The system should have a user-friendly and intuitive interface that is easy to navigate, reducing the learning curve for users.
- **Accessibility:** The system should adhere to accessibility standards to ensure access for users with disabilities, including support for assistive technologies.
- **Multilingual Support:** If applicable, the system should support multiple languages to accommodate diverse user populations.

4. **Reliability:**

- **System Availability:** The system should be highly available, with minimal downtime, to ensure users can access it when needed.
- **Fault Tolerance:** The system should be resilient to hardware or software failures, with built-in mechanisms for fault detection, recovery, and redundancy.
- **Backup and Disaster Recovery:** Regular backups should be performed, and a disaster recovery plan should be in place to protect against data loss and ensure system continuity.

5. **Interoperability:**

- **Standards Compliance:** The system should adhere to industry standards and interoperability frameworks to facilitate seamless data exchange and integration with other healthcare systems.
- **Integration Capability:** The system should have the ability to integrate with external systems, such as electronic health record systems, laboratory systems, or billing systems, to enable efficient information sharing and workflow coordination.

6. **Performance:**

- **Regulatory Compliance:** The system should comply with relevant healthcare regulations and standards, such as HIPAA (Health Insurance Portability and Accountability Act) or GDPR (General Data Protection Regulation).
- **Risk Management:** The system should have mechanisms in place to identify, assess, and mitigate potential risks and vulnerabilities, including security risks, privacy breaches, and data integrity issues.
- **Data Integrity:** The system should ensure the accuracy, completeness, and consistency of patient data, with appropriate validation and error-checking mechanisms.

7. **Integration:**

- **Integration with External Devices:** The system should have the capability to integrate with medical devices, wearables, or other IoT (Internet of Things) devices for remote monitoring and data collection.
- **Standards Compliance:** The system should support industry standards, such as HL7 (Health Level Seven) or DICOM (Digital Imaging and Communications in Medicine), to enable interoperability with external systems and devices.

### 3.5 Hardware Needs

Processor : Core i5 processor

Installed memory (RAM) : 8 GB

Hard Disk : 20GB of hard disk space in terminal machines

Operating System : Windows, Linux, IOS

Server : 1TB hard disk space in Server Machine



### 3.6 Software Needs

IDE : Visual Studio  
Framework: .NET Framework  
Front End Language: HTML,CSS, Bootstrap  
Scripting Language : JS and JQuery  
Back End Language: C#  
SQL Database : SQL Server

### 3.7 Constraints

System is wirelessly networked with an encryption.  
System is only accessible within the hospital's website only.  
Database is password protected.  
Should use less RAM and processing power  
Each user should have individual ID and password  
Only administrator can access the whole system.

### 3.8 Assumption and Dependencies

Each user must have a valid user id and password  
Server must be running for the system to function  
Users must log in to the system to access any record.  
Only the Administrator can delete records.

### 3.9 User Interfaces

This section provides a detailed description of all inputs into and outputs from the system. It also gives a description of the hardware, software and communication interfaces and provides basic prototypes of the user interface  
The protocol used shall be HTTP.  
The Port number used will be 80.  
There shall be logical address of the system in IPv4 format.

### 3.10 User Cases

User Registration and Authentication  
Searching for Healthcare Providers  
Booking an Appointment  
Managing Personal Health Records  
Communication and Messaging  
Feedback and Reviews  
Reporting and Analytics  
Prescription Management

### 3.11 System Architecture

Presentation Layer  
Application Layer  
Services Layer  
Data Layer  
Infrastructure Layer  
External Systems and Devices

### 3.12 Data Requirements

1. Patient Data:
  - Personal Information: This includes patient demographics such as name, date of birth, gender, contact details, and address.
  - Medical History: Patient's medical conditions, allergies, past surgeries, medications, immunisations, and family medical history.
  - Vital Signs: Data related to vital signs such as blood pressure, heart rate, temperature, respiratory rate, and oxygen saturation.
  - Laboratory Results: Results of various lab tests, including blood tests, urine tests, imaging reports, pathology reports, and other diagnostic test results.
  - Medical Imaging: Medical images such as X-rays, MRIs, CT scans, ultrasounds, or mammograms.
  - Clinical Notes: Narrative documentation of patient encounters, including progress notes, discharge summaries, consultation notes, and other clinical observations.
2. Healthcare Provider Data:
  - Provider Information: Details of healthcare providers, including their names, specialties, qualifications, contact information, and availability.
  - Schedules and Appointments: Appointment data, including scheduled dates, times, duration, and associated healthcare providers.
  - Treatment Plans: Treatment plans, care instructions, and prescribed medications for patients.
  - Clinical Decision Support: Data related to clinical guidelines, protocols, and best practices to support healthcare providers in making informed decisions.
  - Referral Information: Data related to referrals to other healthcare providers or specialists, including referral notes and reports.
3. Administrative Data:
  - Insurance Information: Details of patient insurance coverage, including insurance providers, policy numbers, and billing information.
  - Billing and Claims: Data related to medical billing, invoicing, insurance claims, and payment transactions.
  - Patient Registration: Data associated with patient registration, including consent forms, identification documents, and administrative records.
  - Scheduling and Resource Management: Data related to appointment scheduling, managing healthcare provider availability, and facility/resource allocation.
4. System and Audit Data:
  - User Management: User accounts, roles, and permissions for system administrators, healthcare providers, and other system users.
  - System Logs: Logs capturing system events, user activities, error messages, and audit trails for security and compliance purposes.
  - System Configurations: Configuration settings, system preferences, and customisation options for the healthcare system.
  - Data Privacy and Security: Data protection measures, encryption keys, access controls, and consent

management.

### 3.13 Testing and Validation

- Verification of Regulatory Compliance: Ensure the healthcare system meets relevant regulatory standards and requirements, such as HIPAA, GDPR, or local healthcare regulations.
- 
- User Validation: Involve end-users and stakeholders in validating the system's functionalities, usability, and overall user experience.
- 
- Documentation Review: Verify that the system documentation, including user manuals, installation guides, and system specifications, accurately reflects the implemented functionalities and features.
- 
- Use a bug tracking system to log and track identified issues, assign them to the development team, and monitor their resolution.
- 
- Generate detailed bug reports with steps to reproduce, expected outcomes, and actual results for effective communication and collaboration among team members.

## 4. Analysis of System

### 4.1 User Analysis

User analysis is a crucial step in understanding the needs, preferences, and behaviours of the end-users of a healthcare system. By conducting user analysis, you can gain insights into their requirements and design a system that meets their expectations effectively. Here are some key aspects of user analysis in the context of a healthcare system:

Develop user personas based on research and data to represent different types of users, such as patients, healthcare providers, administrators, or insurance representatives.

Define their demographics, goals, motivations, behaviours, and pain points to understand their specific needs and challenges within the healthcare system.

Establish feedback channels, such as user forums, feedback forms, or support tickets, to encourage users to provide ongoing feedback and report issues.

Analyse user feedback and support requests to identify recurring patterns, common pain points, or feature requests. Utilise user feedback to prioritise enhancements, bug fixes, and new features in future iterations of the healthcare system.

## 4.2 Usability Analysis

Usability analysis is an essential part of evaluating the effectiveness, efficiency, and satisfaction of a healthcare system's user interface. It focuses on assessing how easily and efficiently users can accomplish their goals within the system and how satisfied they are with their overall user experience. Here are key aspects of conducting a usability analysis for a healthcare system

1. Usability Goals and Metrics:
  - Define specific usability goals and metrics that align with the objectives of the healthcare system. Examples include task completion rate, time on task, error rate, user satisfaction ratings, or perceived ease of use.
  - Establish benchmarks or industry standards to compare the system's usability performance against.
2. User Testing:
  - Recruit representative users who match the intended user demographics and characteristics of the healthcare system.
  - Design test scenarios that reflect typical user tasks and workflows within the system.
  - Observe users as they interact with the system, noting their actions, difficulties, and feedback.
  - Collect qualitative and quantitative data on task completion rates, time taken to complete tasks, errors encountered, and user feedback.

## 4.3 Performance Analysis

- Throughput: Assess the number of transactions or requests processed by the system within a given time frame.
- Scalability: Evaluate the system's ability to handle increased workload or user concurrency by adding resources or scaling horizontally.
- Resource Utilisation: Monitor the usage of system resources such as CPU, memory, disk space, and network bandwidth to identify bottlenecks or potential performance issues.
- Availability: Determine the system's uptime and reliability by measuring the percentage of time the system is accessible and operational.
- Simulate realistic user loads by generating concurrent requests or transactions to stress the system and evaluate its performance under normal and peak load conditions.
- Measure response times, throughput, and resource utilisation during load testing to identify performance degradation or capacity limitations.
- Analyse system behaviour and performance metrics to determine if the system meets the required performance criteria.

## 4.4 Security and Privacy Analysis

1. Threat Modelling:
  - Identify potential threats and risks that the healthcare system may face, such as unauthorised access, data breaches, or malicious attacks.
  - Analyse the system's architecture, data flow, and components to determine potential vulnerabilities and attack vectors.
  - Prioritise threats based on their impact and likelihood to focus on the most critical security risks.
2. Authentication and Authorisation:

- Assess the system's authentication mechanisms to ensure secure user access and prevent unauthorised logins.
  - Evaluate the effectiveness of password policies, multi-factor authentication, and session management.
  - Verify the system's authorisation controls to ensure that users have appropriate access privileges based on their roles and responsibilities.
3. Data Privacy and Encryption:
- Ensure compliance with relevant privacy regulations, such as HIPAA (Health Insurance Portability and Accountability Act) or GDPR (General Data Protection Regulation).
  - Evaluate data handling and storage practices to ensure sensitive patient information is properly encrypted and protected.
  - Assess the system's mechanisms for data anonymisation and pseudonymization to protect patient privacy.

## 4.5 System Reliability and Availability Analysis

1. Fault Tolerance and Redundancy:
  - Assess the system's architecture and components to identify single points of failure.
  - Implement fault-tolerant measures such as redundancy, failover mechanisms, and backup systems to ensure system availability in case of component failures.
  - Evaluate the effectiveness of redundancy strategies, such as data replication, load balancing, or clustering, to minimise downtime and maintain service availability.
2. Disaster Recovery Planning:
  - Develop a comprehensive disaster recovery plan to address potential system failures or disasters.
  - Define backup and recovery procedures for critical data, applications, and system configurations.
  - Test and validate the effectiveness of the disaster recovery plan through periodic drills and simulations.
3. Monitoring and Alerting:
  - Implement monitoring tools to continuously monitor the health and performance of system components, network connectivity, and critical processes.
  - Set up alerts and notifications to proactively identify system failures, performance degradation, or abnormal behaviour.
  - Monitor key performance indicators (KPIs), such as response times, error rates, or resource utilisation, to detect anomalies and trigger appropriate actions.
4. High Availability Infrastructure:
  - Ensure that the underlying infrastructure, including servers, network equipment, and data center's, is designed for high availability.
  - Implement redundant power supplies, backup generators, uninterruptible power supplies (UPS), and redundant network connections to minimise downtime.
5. Scalability and Load Balancing:
  - Evaluate the system's scalability to handle increased user loads and growing data volumes.
  - Implement load balancing mechanisms to distribute incoming traffic and workload across multiple servers or resources to prevent overload and maintain performance.

## 4.6 Data Analysis

Data analysis in the context of a healthcare system involves examining and interpreting the data generated within the system to derive meaningful insights and support informed decision-making. It encompasses various techniques and methodologies to extract knowledge, identify patterns, and make predictions from healthcare data. Here are key aspects of data analysis in a healthcare system:

1. Data Collection and Integration:
  - Identify the relevant data sources within the healthcare system, such as electronic health records (EHRs), medical imaging data, laboratory results, patient demographics, and billing information.
  - Integrate data from multiple sources, ensuring data quality, standardisation, and compatibility for analysis.
2. Data Cleaning and Preprocessing:
  - Clean and preprocess the data to handle missing values, outliers, and inconsistencies.
  - Perform data transformations, normalisation, and aggregation to prepare the data for analysis.
3. Descriptive Analysis:
  - Conduct descriptive analysis to summarise and describe the characteristics of the healthcare data.
  - Calculate basic statistics, such as mean, median, mode, standard deviation, or frequency distributions, to gain insights into the data distribution.
4. Exploratory Data Analysis (EDA):
  - Perform exploratory data analysis to discover patterns, relationships, or trends in the data.
  - Utilise visualisation techniques, such as charts, graphs, or heat maps, to visually explore the data and identify patterns or correlations.
5. Diagnostic Analysis:
  - Apply diagnostic analysis techniques to identify the causes or factors contributing to specific healthcare outcomes or conditions.
  - Utilise statistical tests, regression analysis, or classification algorithms to determine the relationship between variables and outcomes.
6. Predictive Modeling:
  - Develop predictive models using machine learning or statistical techniques to forecast outcomes or make predictions in healthcare.
  - Utilise techniques such as regression, classification, time series analysis, or clustering to build models based on historical data.
7. Data Mining:
  - Apply data mining techniques to discover hidden patterns, associations, or anomalies in the healthcare data.
  - Utilise methods such as association rules, sequential pattern mining, or anomaly detection to extract valuable insights from large datasets.
8. Predictive Analytics:
  - Utilise predictive analytics to forecast healthcare outcomes, predict disease progression, or identify high-risk patients.
  - Combine historical data, clinical knowledge, and machine learning algorithms to generate predictive models and make informed predictions.

## 9. Outcome Evaluation and Reporting:

- Evaluate the performance and accuracy of predictive models or interventions using appropriate evaluation metrics.
- Generate reports and visualisations to communicate the findings and insights derived from the data analysis.
- Provide actionable recommendations to healthcare professionals based on the data analysis results.

## 10. Privacy and Ethical Considerations:

- Ensure compliance with privacy regulations, such as HIPAA or GDPR, and adhere to ethical guidelines when handling and analysing sensitive healthcare data.
- Anonymise or de-identify patient data to protect privacy and maintain confidentiality.

## 4.7 Stakeholder Analysis

- Identify all relevant stakeholders who are directly or indirectly impacted by the healthcare system.
- Include healthcare professionals, patients, administrators, policymakers, insurance providers, regulatory bodies, IT staff, and other individuals or groups involved in healthcare delivery.
- Assess the importance and influence of each stakeholder on the healthcare system.
- Consider factors such as their level of interest, power, expertise, resources, and potential impact on decision-making.
- Determine the level of influence each stakeholder has on the healthcare system and its decision-making processes.
- Assess their level of support, potential conflicts of interest, and alignment with the system's objectives.

## 4.8 Competitor Analysis

- Identify direct competitors who offer similar healthcare services or solutions within the same geographic region.
- Consider both established competitors and emerging players in the healthcare industry.
- Collect information about each competitor, including their mission, vision, organisational structure, size, market presence, and target market segments.
- Research their range of services, healthcare specialties, technological capabilities, and unique selling propositions.
- Evaluate the range and quality of healthcare services offered by competitors.
- Compare the breadth and depth of their services, specialty areas, treatment options, and patient experience.

## 5. Testing

### 5.1 Definition

Testing is a systematic process of evaluating a system, software application, or product to ensure that it meets the specified requirements, functions correctly, and delivers the expected results. It involves executing various test cases, scenarios, or procedures to identify defects, errors, or deviations from the intended behavior. The primary goal of testing is to uncover defects or bugs and provide feedback to improve the quality, reliability, and performance of the system being tested.

Testing can be performed at different stages of the software development lifecycle, including requirements analysis, design, coding, and post-development. It encompasses a range of activities, techniques, and methodologies to validate and verify the system's functionality, performance, security, usability, and compatibility.

Key aspects of testing include:

1. **Test Planning:** Defining the overall testing approach, test objectives, test strategy, and test plan. This involves identifying test requirements, resources, and timelines.
2. **Test Design:** Creating test cases, test scenarios, and test scripts based on requirements, specifications, and use cases. Designing test data and test environments.
3. **Test Execution:** Running the tests according to the defined test cases or test scripts. Observing the system's



behaviour and comparing the actual results with the expected results.

4. **Defect Identification and Reporting:** Documenting and reporting any defects or discrepancies found during testing. Providing detailed information about the defect to help developers reproduce and fix it.
5. **Test Coverage:** Ensuring that a sufficient range of test cases or scenarios are executed to cover all aspects of the system's functionality, including positive and negative test cases.
6. **Regression Testing:** Repeating tests on modified or newly added functionalities to ensure that changes do not introduce new defects or have unintended consequences on existing functionalities.
7. **Performance Testing:** Assessing the system's performance, scalability, and responsiveness under different workload conditions. This includes load testing, stress testing, and performance profiling.
8. **Security Testing:** Evaluating the system's vulnerability to security breaches, data leaks, or unauthorised access. Identifying and addressing potential security risks.

## 5.2 Validation and System Testing

11. Validation testing is a process of evaluating a system or software application during or at the end of the development process to ensure that it satisfies the specified requirements and meets the intended user needs. The main objective of validation testing is to assess whether the system, as a whole, is fit for its intended purpose and performs effectively in its intended environment. It involves activities such as requirements validation, user acceptance testing, and functional testing.

Key aspects of validation testing include:

- Verifying that the system meets the user's requirements and expectations.
- Ensuring that the system functions correctly and performs the intended tasks.
- Assessing whether the system meets the quality and performance standards.
- Validating that the system operates effectively in the intended environment.
- Checking compliance with regulatory or industry standards.
- Obtaining user feedback and approval for the system.

3. Validation testing is often conducted by stakeholders, end-users, or independent testers who assess the system's functionality, usability, and overall fitness for use.

4. **System Testing:**

5. System testing is a phase in the software testing process that focuses on testing the entire system as an integrated unit to verify that it functions correctly and meets the specified requirements. It involves testing the system as a whole rather than individual components or modules. The objective of system testing is to identify defects or issues that may arise due to the interactions between different components or modules within the system.

Key aspects of system testing include:

- Verifying that the system components work together seamlessly and as expected.
  - Ensuring that the system meets the specified functional and non-functional requirements.
  - Testing the system's interfaces, interactions, and data flows.
  - Assessing the system's reliability, performance, security, and scalability.
  - Identifying and fixing defects or issues in the system.
  - Validating that the system meets the overall quality and performance standards.
6. System testing is typically performed after unit testing and integration testing, and it may involve different types of testing such as functional testing, integration testing, performance testing, security testing, and usability testing.

## 5.3 Integration Testing

Key aspects of integration testing include:

1. **Integration Scope:** Determine the scope of integration testing, including the specific modules or components that need to be tested together. This can involve identifying dependencies, interfaces, and communication channels between the components.
2. **Test Environment Setup:** Set up the necessary test environment, including configuring hardware, software, databases, and other dependencies required for the integration testing.
3. **Integration Test Plan:** Develop a test plan that outlines the integration testing approach, test objectives, test scenarios, and test cases. This plan should define the sequence of integration, the order in which components are integrated, and any specific data or inputs required for testing.
4. **Integration Strategy:** Define the integration strategy, which can include top-down, bottom-up, or a combination of both approaches. Top-down integration involves testing higher-level modules first and gradually integrating lower-level modules. Bottom-up integration starts with testing lower-level modules and progressively integrates higher-level modules.
5. **Stub and Driver Development:** Create stubs or drivers as necessary to simulate the behaviour of components that are not yet available or fully implemented. Stubs provide fake responses to the component being tested, while drivers enable the execution of tests by providing inputs and invoking the necessary functions.

## 5.4 Unit Testing

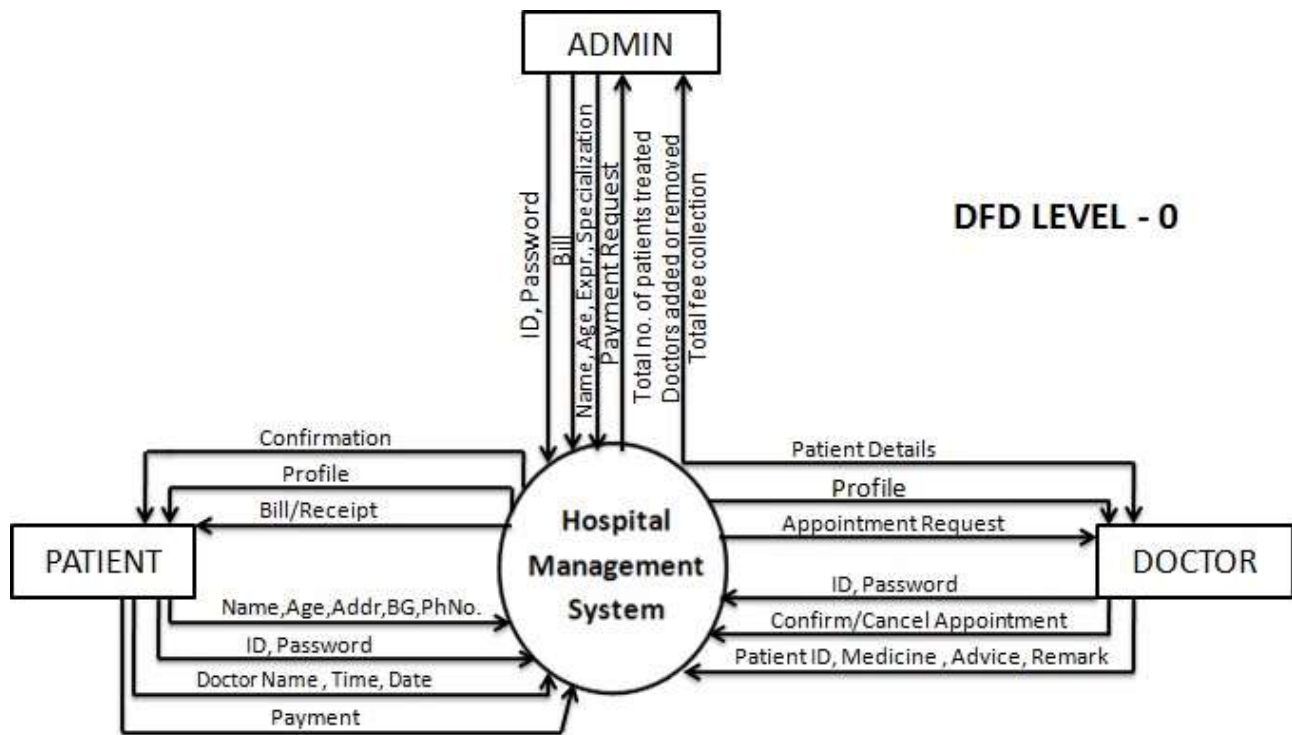
The key aspects of unit testing include:

1. **Test Scope:** Determine the scope of the unit testing, including the specific units or components that need to be tested. Each unit should be tested independently, without dependencies on other units.
2. **Test Framework Setup:** Set up the necessary test environment and frameworks for unit testing. This typically involves selecting a unit testing framework (e.g., JUnit, NUnit) and configuring it to run the tests.
3. **Test Plan and Test Cases:** Develop a test plan that outlines the unit testing approach, test objectives, and test cases. Each test case should cover a specific aspect or behaviour of the unit being tested.
4. **Test Data Preparation:** Prepare test data and inputs required for unit testing. This can involve creating sample data, mock objects, or stubs to simulate the behaviour of dependencies or external systems.
5. **Test Execution:** Execute the unit tests by running the test cases against the individual units. Each unit test should exercise the functionality of the unit and validate its outputs or behaviour.

6. **Assertions and Assertions:** Use assertions or assertions within the unit tests to compare the expected outputs or behaviours with the actual results. These assertions help determine whether the unit is functioning correctly.

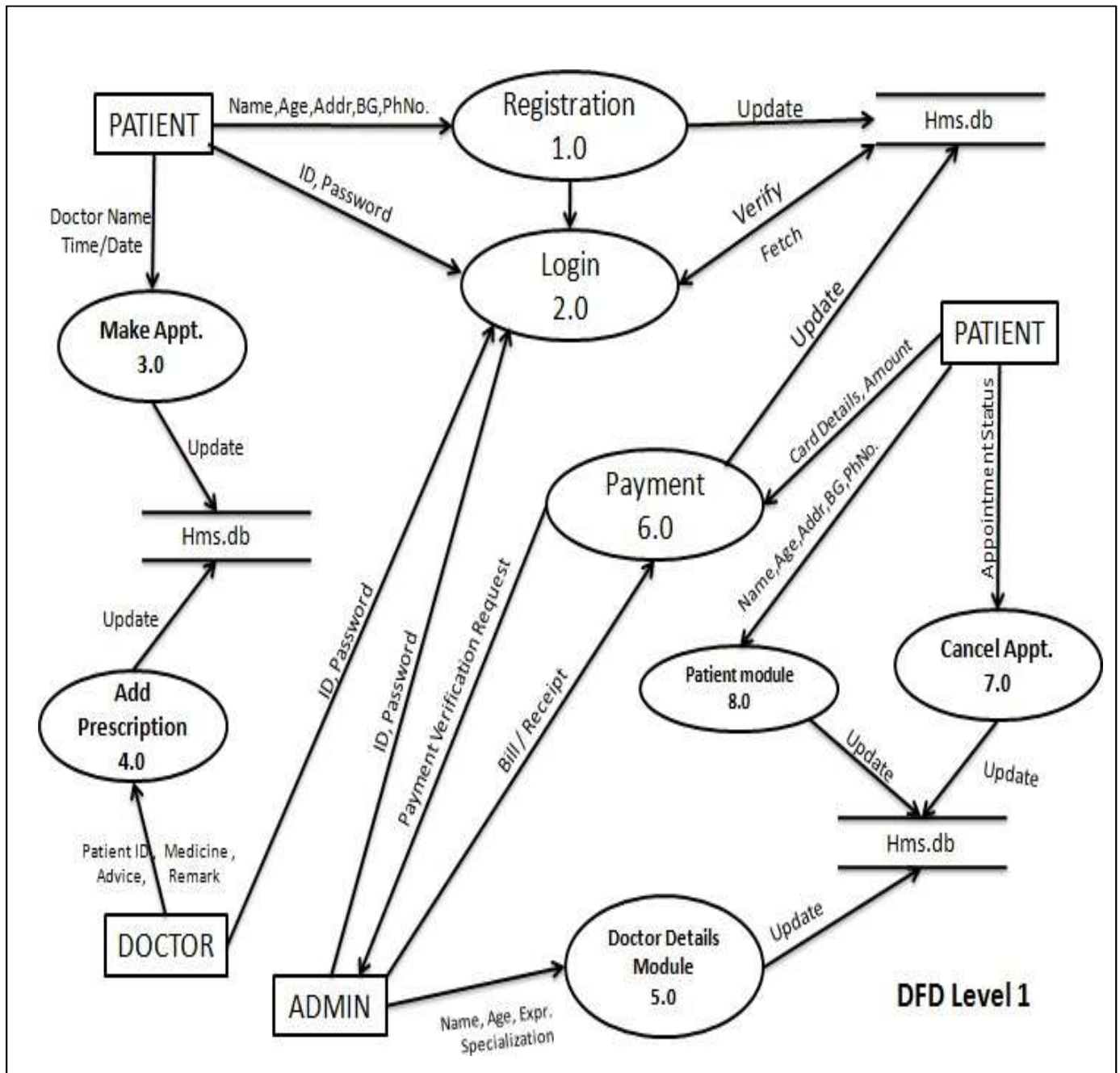
## **6- List Of Figure**

### **1. Context level diagram**



**Context Level Diagram**

## 2. DFD Level 1



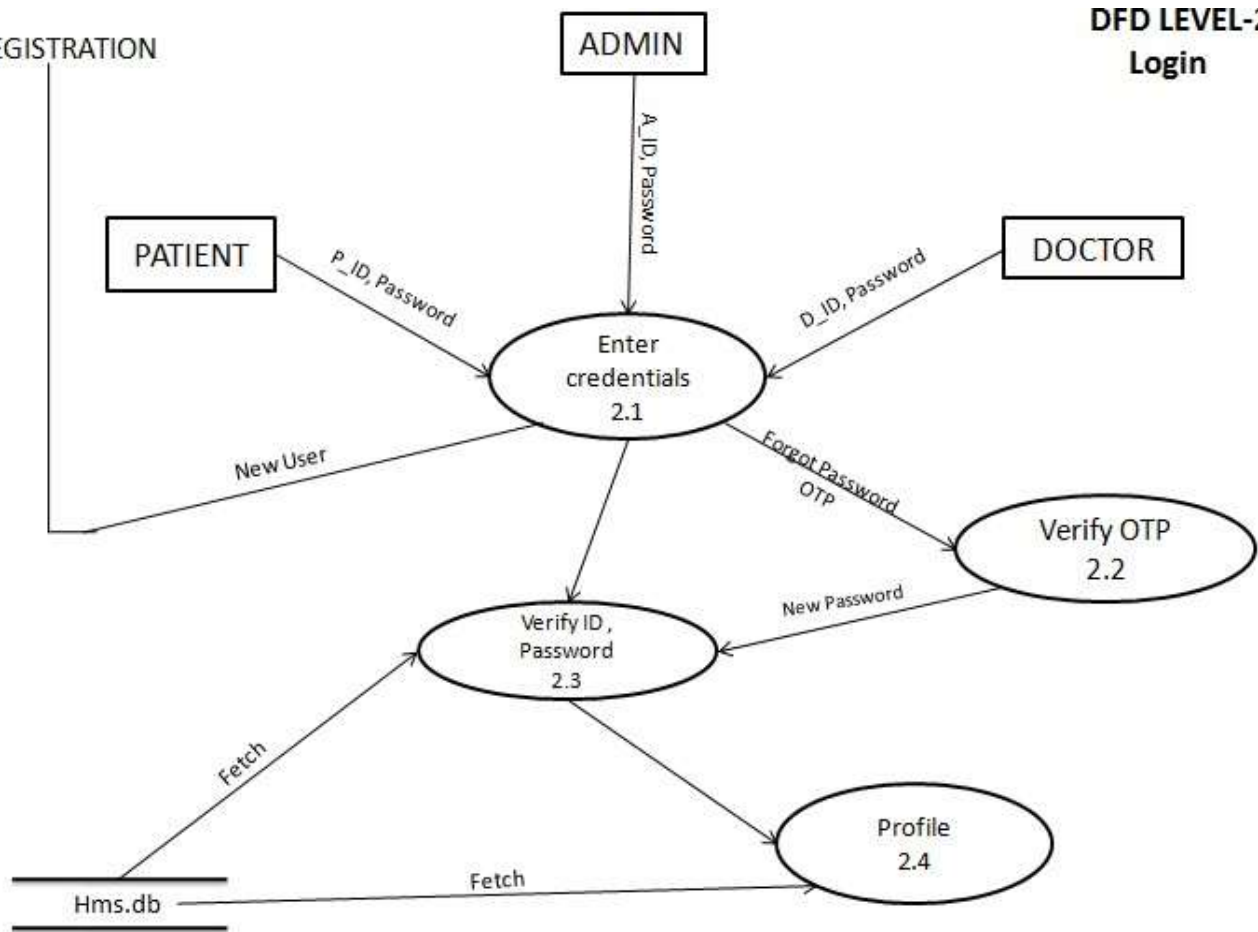
## DFD Level 1

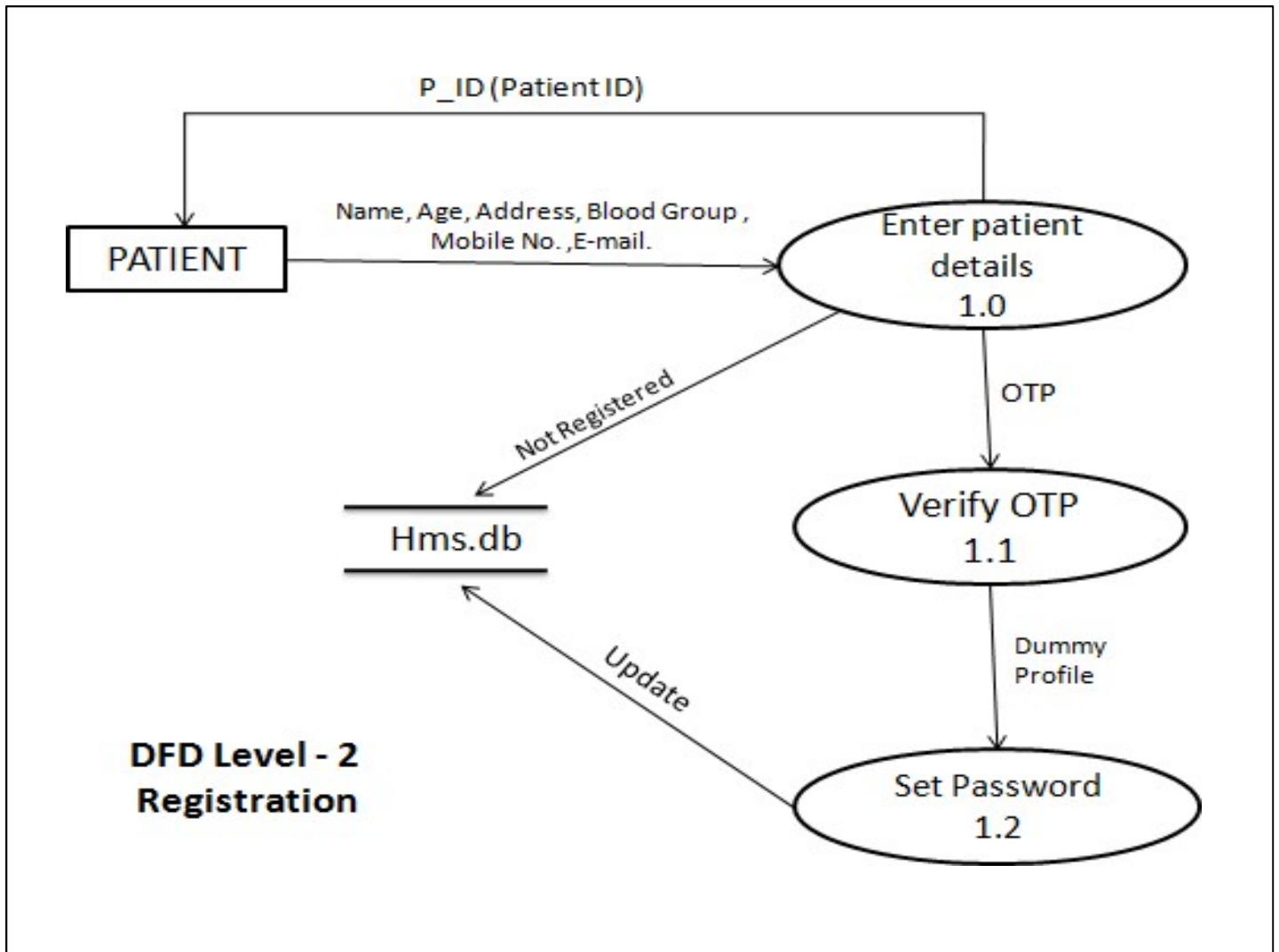
**Figure 2.2 Level 1 DFD**

## 3. DFD Level 2

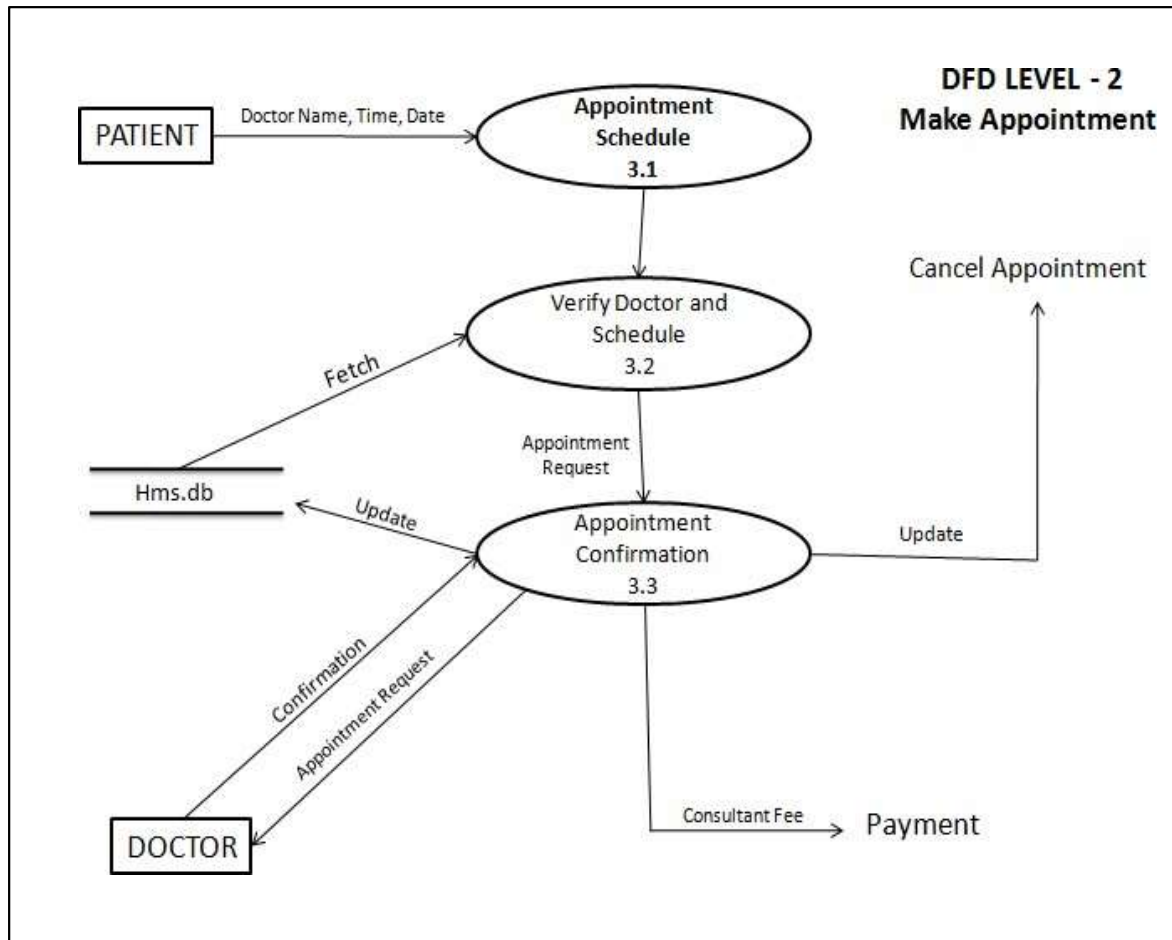
**DFD LEVEL-2  
Login**

REGISTRATION



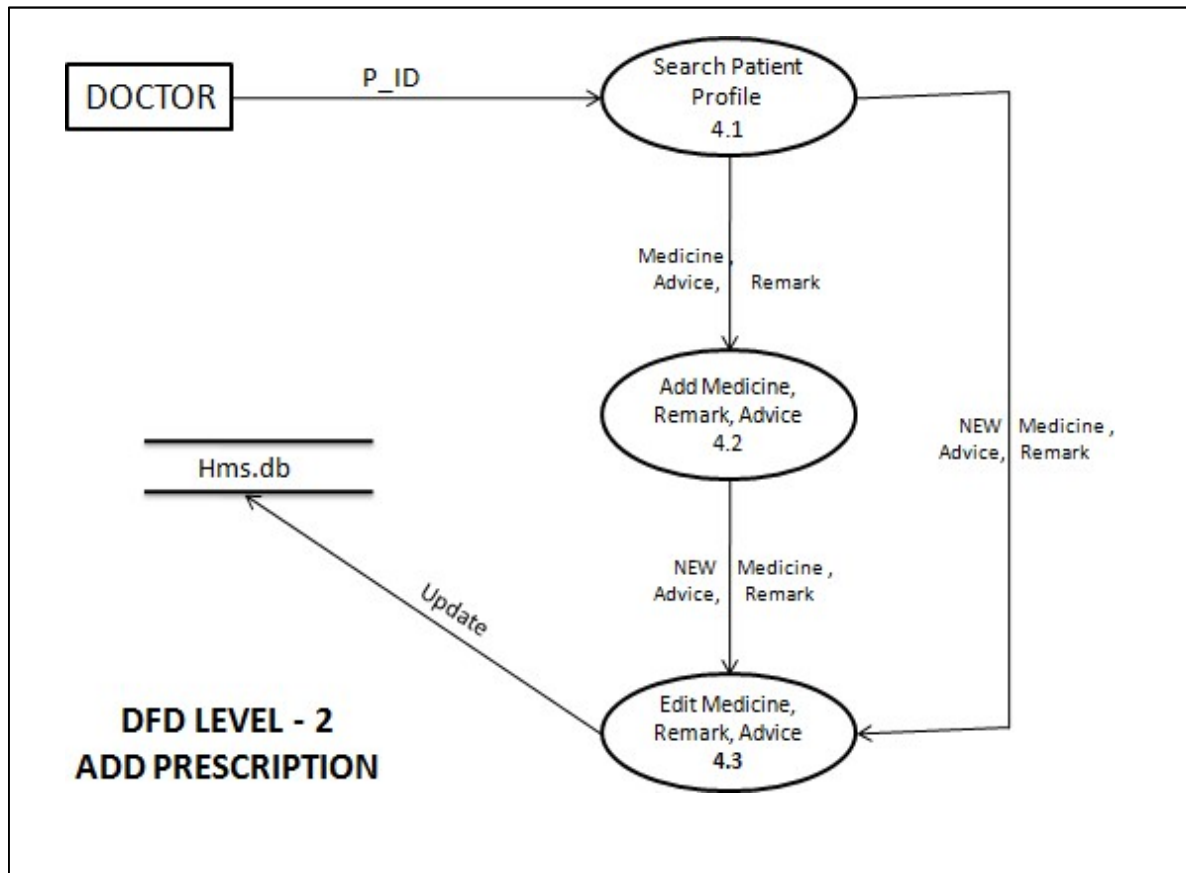


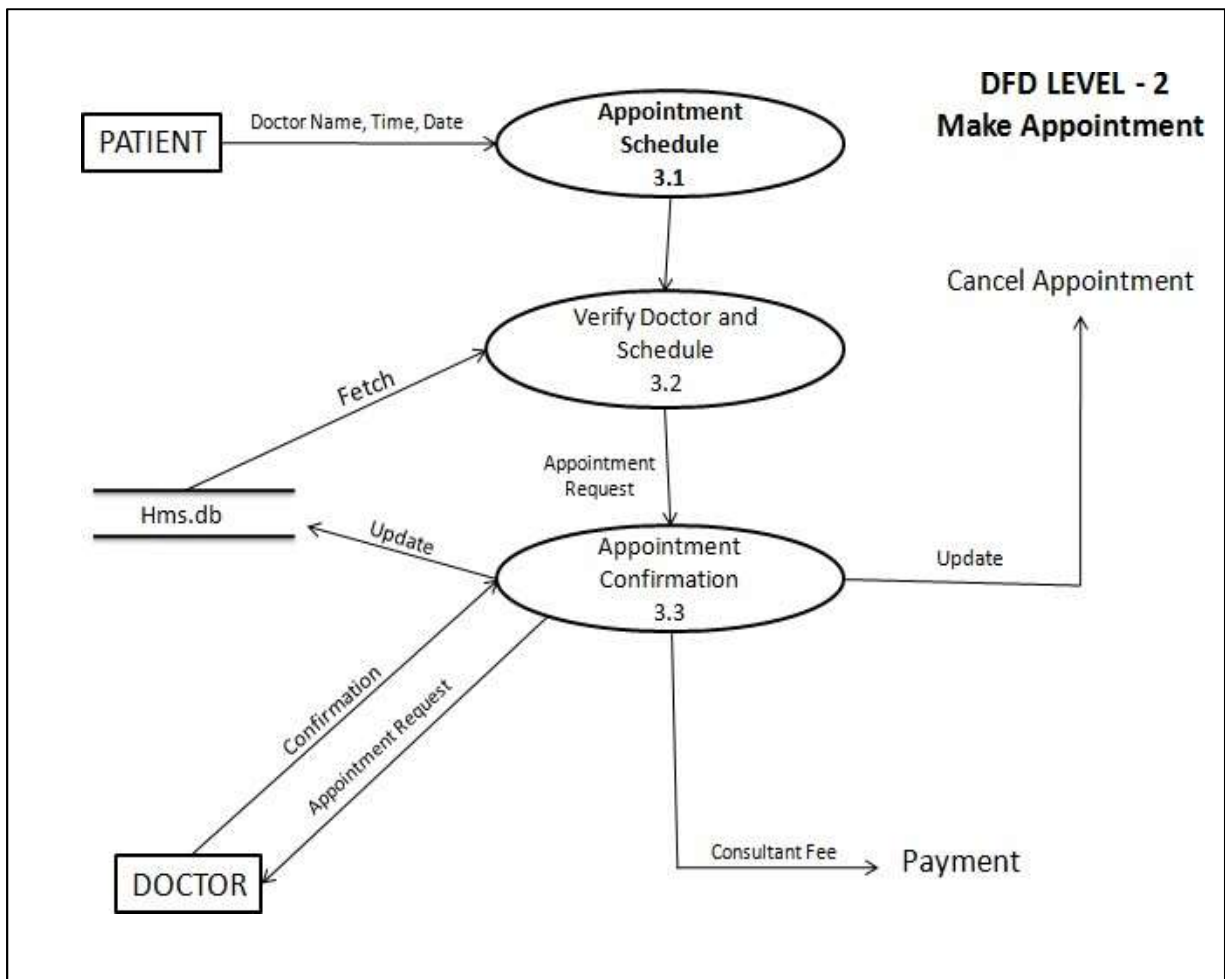
**FIGURE 2.3 LEVEL – 2 Registration**



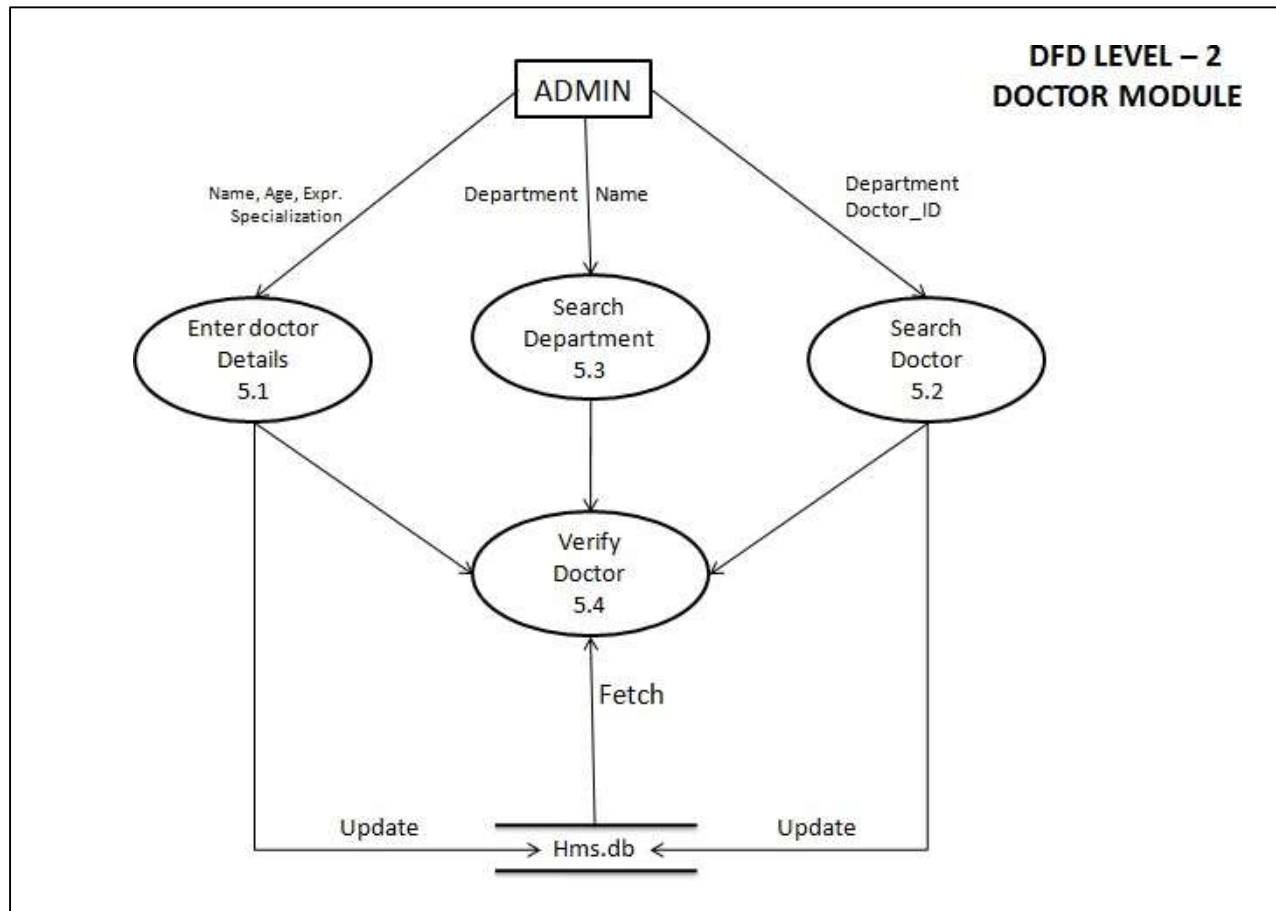
**FIGURE 2.4 LEVEL – 2 Login**







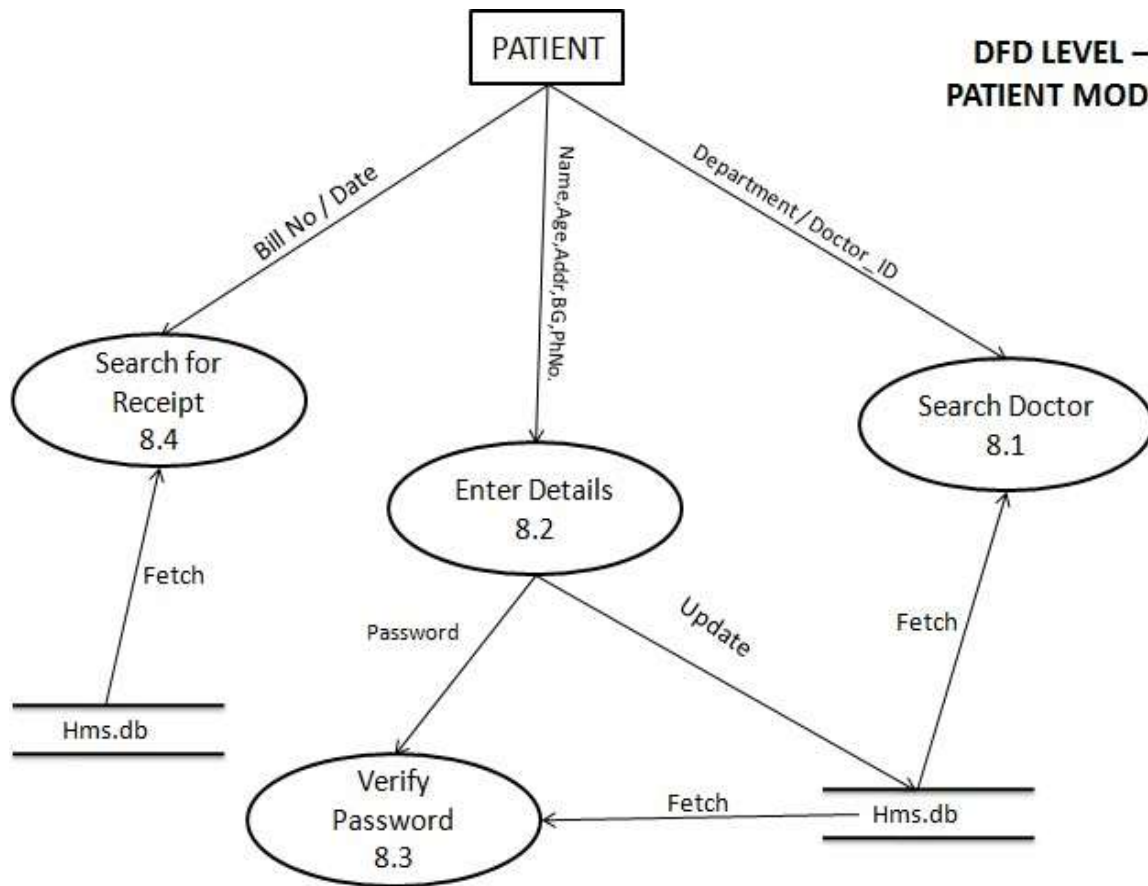
**FIGURE 2.6 LEVEL – 2 Add Description**

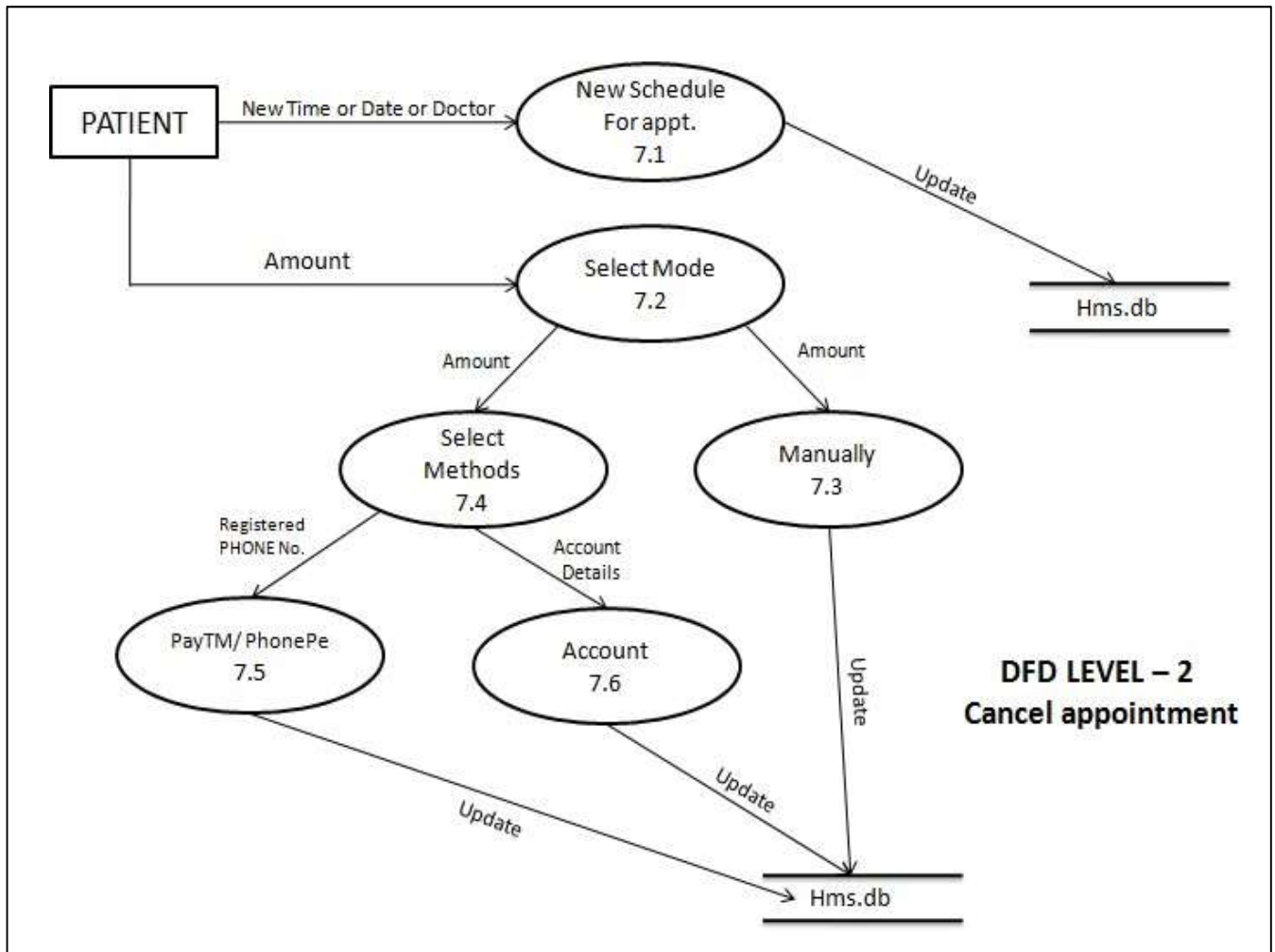


**FIGURE 2.7 LEVEL – 2 Doctor Module**

**FIGURE 2.8 LEVEL – 2 Payment**

**DFD LEVEL – 2  
PATIENT MODULE**

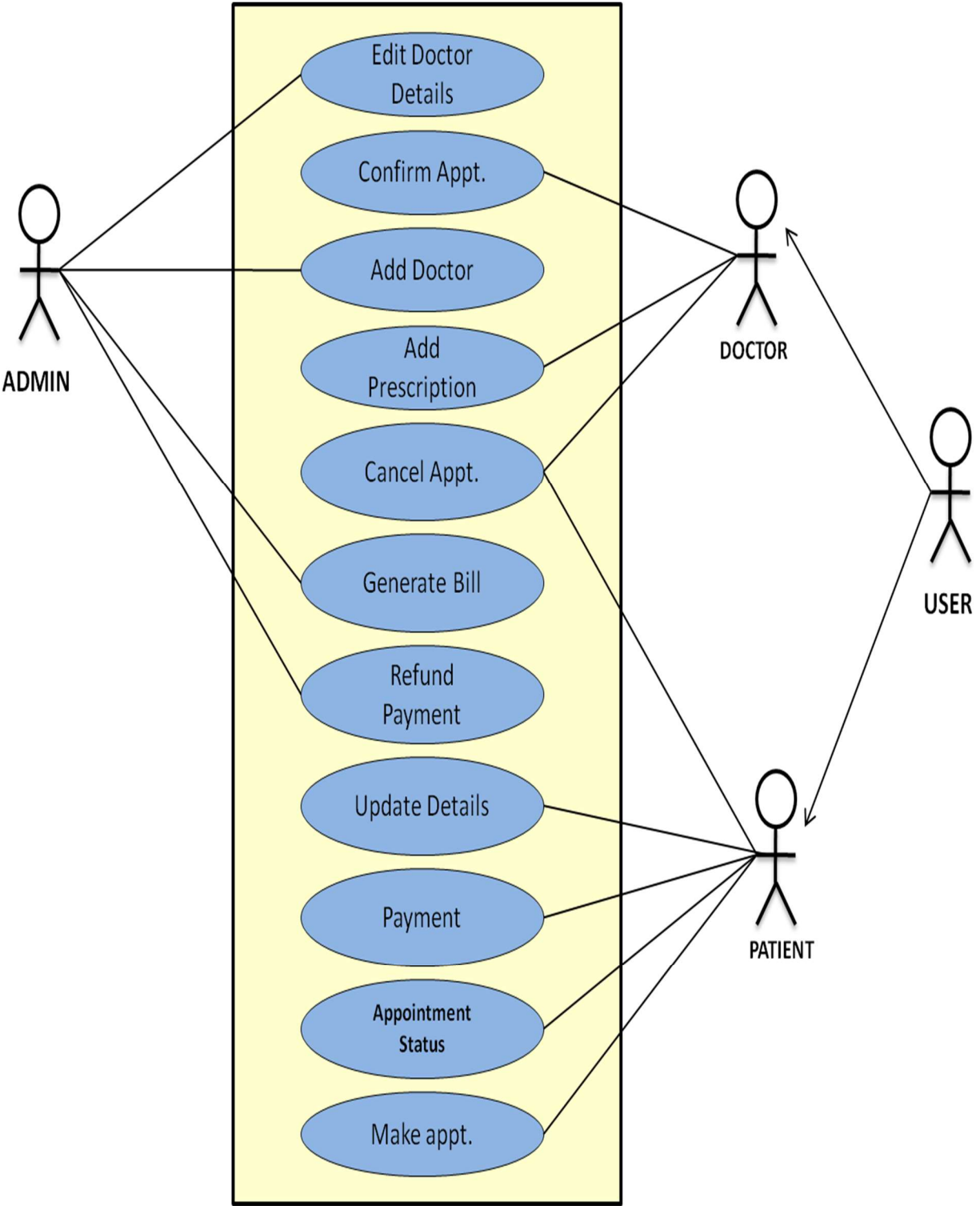




**FIGURE 2.9 LEVEL – 2 Cancel Appointment**

**FIGURE 2.10 LEVEL – 2 Patient Module**

4. Use Case Diagram



## 7. Use Case Description

### (1) PATIENT

#### REGISTRATION

DESCRIPTION - The new patient can register themselves and add their details like name, age, gender, blood group etc. The patient entry will be made in the hms database.

PRE-CONDITION – The patient must be a new patient, If necessary fields left by user then prompt user to fill the necessary fields.

#### MAIN FLOW OF EVENTS

1. Patient selects sign up in login module.
2. A registration form get displayed
3. Patient fills the required details.

POST CONDITIONS - Patient record is added to healthcare database.

#### UPDATION

DESCRIPTION-The patient should be enabled to update his/her details and the changes should reflect in healthcare database.

PRE-CONDITION – The patient must be a registered patient, The patient cannot update details after treatment starts.

#### MAIN FLOW OF EVENTS

1. Patient logs in to the system.
2. Patient view his record
3. Patient selects update details.
4. Now patient may change the necessary fields.
5. Pop of update details.

POST CONDITION - The record of patient is updated in healthcare database.

## **APPOINTMENT**

DESCRIPTION - It shows users a list of available doctors, timings, dates and enables patients to select the most suitable appointment date and doctor. The patient may also the cancel the appointment.

PRE-CONDITION - The patient must be a registered patient, Patient can fix only one appointment for a particular department.

### MAIN FLOW OF EVENT

1. Patient first logs in to system.
2. View his/her record.
3. Create a new appointment or cancel the appointment..

POST CONDITIONS - patient details are displayed and a new appointment is fix or a existing appointment is cancelled. The hms database is updated.

## **PAYMENT**

DESCRIPTION – It enables user to pay the consultant fee of Doctor online.

PRE-CONDITION - The patient must be a registered patient, If Patient don't wants to pay online he/she can pay by cash also.

### MAIN FLOW OF EVENT

1. Patient first logs in to system.
2. View his/her record.
3. Appointment confirmed by the Doctor then go for Payment.

POST CONDITIONS – A Reciept will be displayed. The hms database is updated

## **(2) DOCTOR**

DESCRIPTION- The doctor view patient record/ update his details and add description of the treatment given to patient.

PRE-CONDITION – The doctor must be a registered doctor, System does not allow the doctor to modify the qualification, hospital managed details.



### MAIN FLOW OF EVENTS

1. Doctor logs in to the system.
2. Doctor may select view patient.
  - 2.1. Patient record is displayed with treatment history.
3. Doctor add description of patient treatment.
4. Doctor may select appointment details
  - 4.1. Appointment Requests is displayed with schedule.
5. Doctor confirm or cancel appointment.

POST CONDITION – The patient and doctor 's database are updated.

### **(3) ADMIN**

DESCRIPTION - The admin add doctor, update doctor details and verify payment and generate Bill/Receipt for the same.

### MAIN FLOW OF EVENTS

1. Admin logs in the system.
2. Admin may add doctor new doctor.
  - 2.1. admin fills the doctor's details.
3. Admin view Doctor record.
  - 3.1. Admin enters the doctor id in the system.
  - 3.2. Doctor details are displayed, Admin can update details.
4. Admin Verify the payment submitted by the Patient.
  - 4.1. Generate Bill/Receipt and confirmation message for the same.

PRE –CONDITION - Admin must first log in with his/her credentials.

POST CONDITION - The hms database is updated.

## 5. Data Dictionary

1.	legal_character	[a-z   A-Z]
2.	Digit	[0-9]
3.	special_ch	[@   \$   #   +   -]
4.	Blood	[A   B   AB   O]

1.	Name	first_name + (middle_name) + last_name
2.	first_name	{legal_character}*
3.	middle_name	{legal_character}*
4.	last_name	{legal_character}*
5.	P_ID	{legal_character + digit}*
6.	D_ID	{legal_character + digit}*
7.	A_ID	{legal_character + digit}*
8.	Password	{legal_character + digit + special_ch}*
9.	Address	House_no + (Street) + City
10.	House_no	{legal_character + digit}*
11.	Street	{legal_character}*
12.	City	{legal_character}*
13.	Mobile No.	{ digit }*
14.	Blood_Group	{Blood + special_ch}*
15.	Specialization	{legal_character}*
16.	Consultant Fee	{ digit }*

17.	Medicine	{legal_character + digit}*
18.	Advice	{legal_character + digit}*
19.	Remark	{legal_character + digit}*

**Table 4.1 Data Dictionary**

## 6. Data Design

<b>S N O.</b>	<b>C O L U M N A M E</b>	<b>D A T A T Y P E</b>	<b>CONSTRAINTS</b>	<b>DESCRIPTION</b>
1.	P_ID	Varchar(50)	Primary Key	Contains Unique Id
2.	Name	Varchar(50)	-	Contains Name
3.	DOB	Varchar(50)	-	Contains Date Of Birth
4.	Gender	Varchar(50)	-	Contains Gender

5.	Blood Group	Varchar(50)	-	Contains Blood Group
6.	Email ID	Varchar(50)	-	Contains Email Id
7.	Address	Varchar(50)	-	Contains Address
8.	Mobile No.	Integer	-	Contains Mobile No.
9.	CGHS/Private	Varchar(50)	-	Contains Category

**Table 4.2 Patient**

<b>S N O .</b>	<b>C O L U M N N A M E</b>	<b>D A T A T Y P E</b>	<b>CONSTRAI NTS</b>	<b>DESCRIPTION</b>
1.	P_ID	Varchar(50)	Primary Key	Contains Unique Id Patient
2.	Specialization	Varchar(50)	-	Contains Name of theDepartment in which Patient wants to visit
3.	Doctor's Name	Varchar(50)	-	Contains Doctor Name Patient Wants To Visit
4	Consultant	Integer	-	Contains Consultant Fee Of Doctor

.	Fee			
5.	Date	Date	-	Contains Date For The Appointment
6.	Time	Time	-	Contains Time For The Appointment

**Table 4.3 Appointment**

<b>S N O .</b>	<b>C O L U M N N A M E</b>	<b>D A T A T Y P E</b>	<b>CONSTRAI NTS</b>	<b>DESCRIPTION</b>
1.	D_ID	Varchar(50)	Primary Key	Contains unique ID
2.	Age	Integer	-	Contains age
3.	Gender	Varchar(50)	-	Contains gender
4.	Specialization	Varchar(50)	-	Contains specialization

5 .	Experience	Varchar(50)	-	Contains experience of the doctor (In months)
6 .	Language	Varchar(50)	-	Contains in how many languages doctor can speak.
7 .	Mobile No.	Integer	-	Contains mobile number
8 .	Email ID	Varchar(50)	-	Contains Email Id
9 .	Schedule	Varchar(50)	-	Contains day and time for which the doctor is available

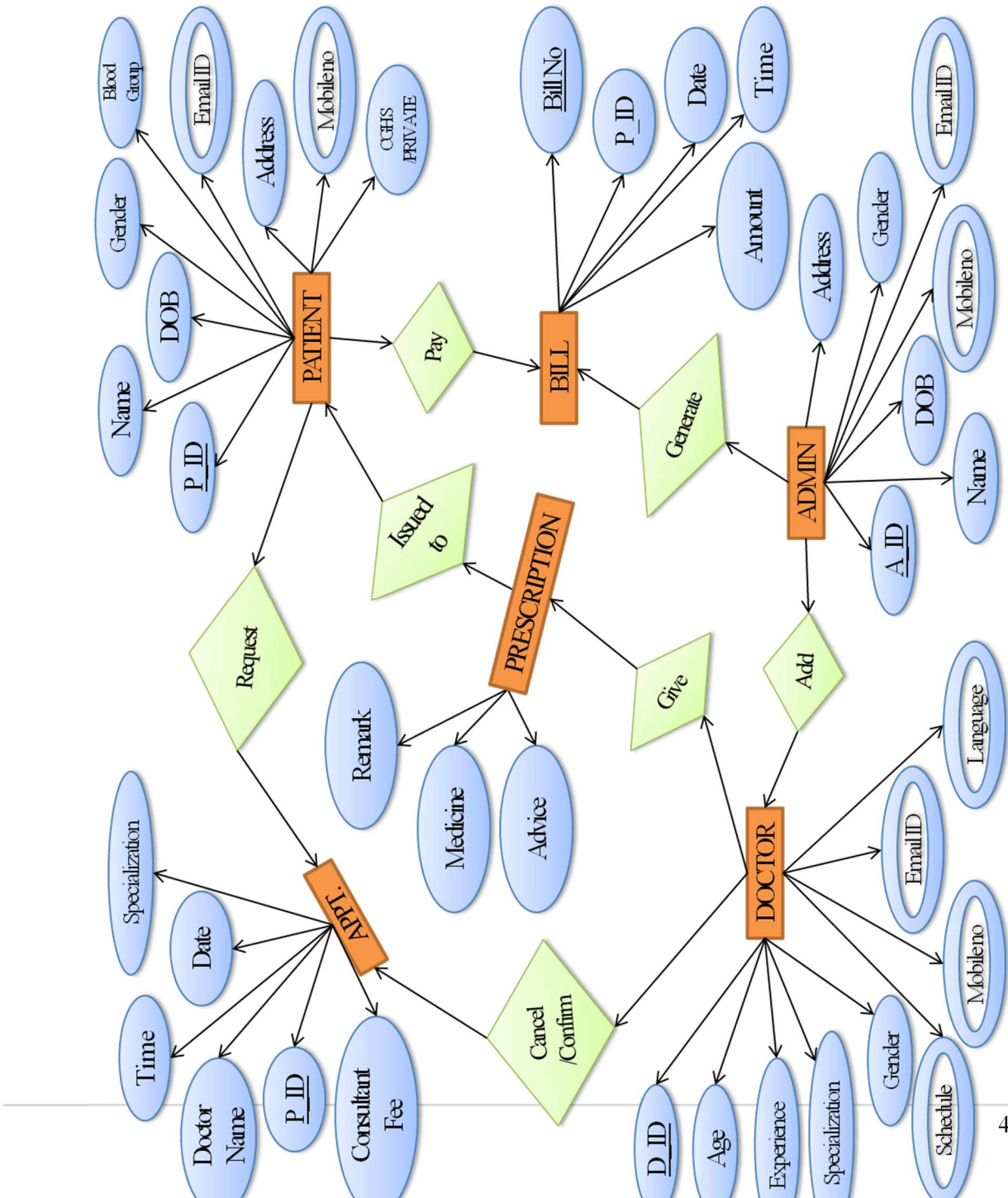
**Table 4.4 Doctor**

S N O .	C O L U M N A M E	D A T A T Y P E	C O N S T R A I N T S	D E S C R I P T I O N
1 .	D_ID	Varchar(50)	-	Contains unique ID

2 .	P_ID	Varchar( 50)	Primary Key	Contains unique ID
3 .	Medicine	Varchar( 50)		Contains name of the medicine.
4 .	Remark	Varchar( 50)		Contains Remark given by the doctor for the patient.
5 .	Advice	Varchar( 50)		Contains any advice for the patient.

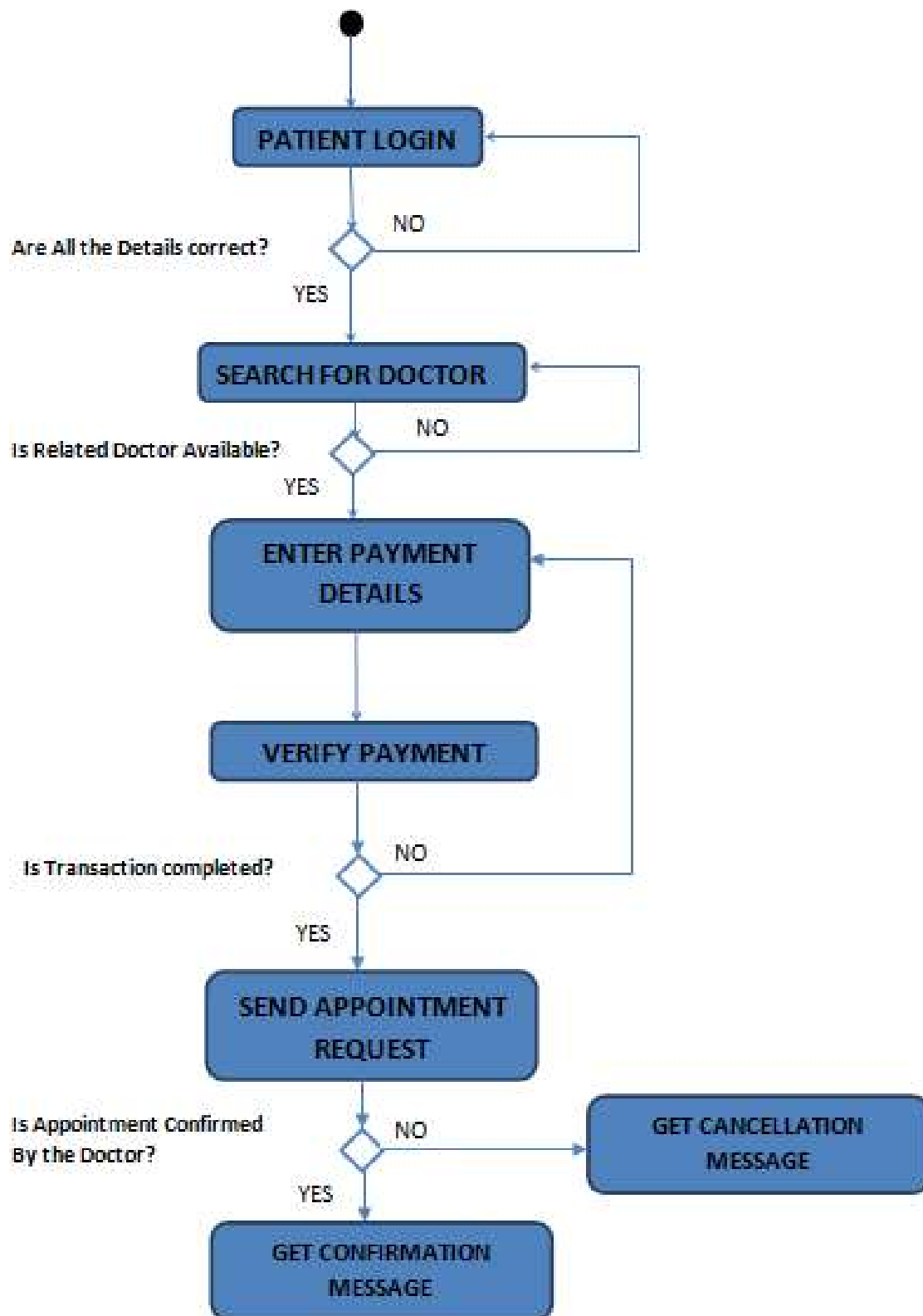
**Table 4.5 Prescription**

7. ER diagram



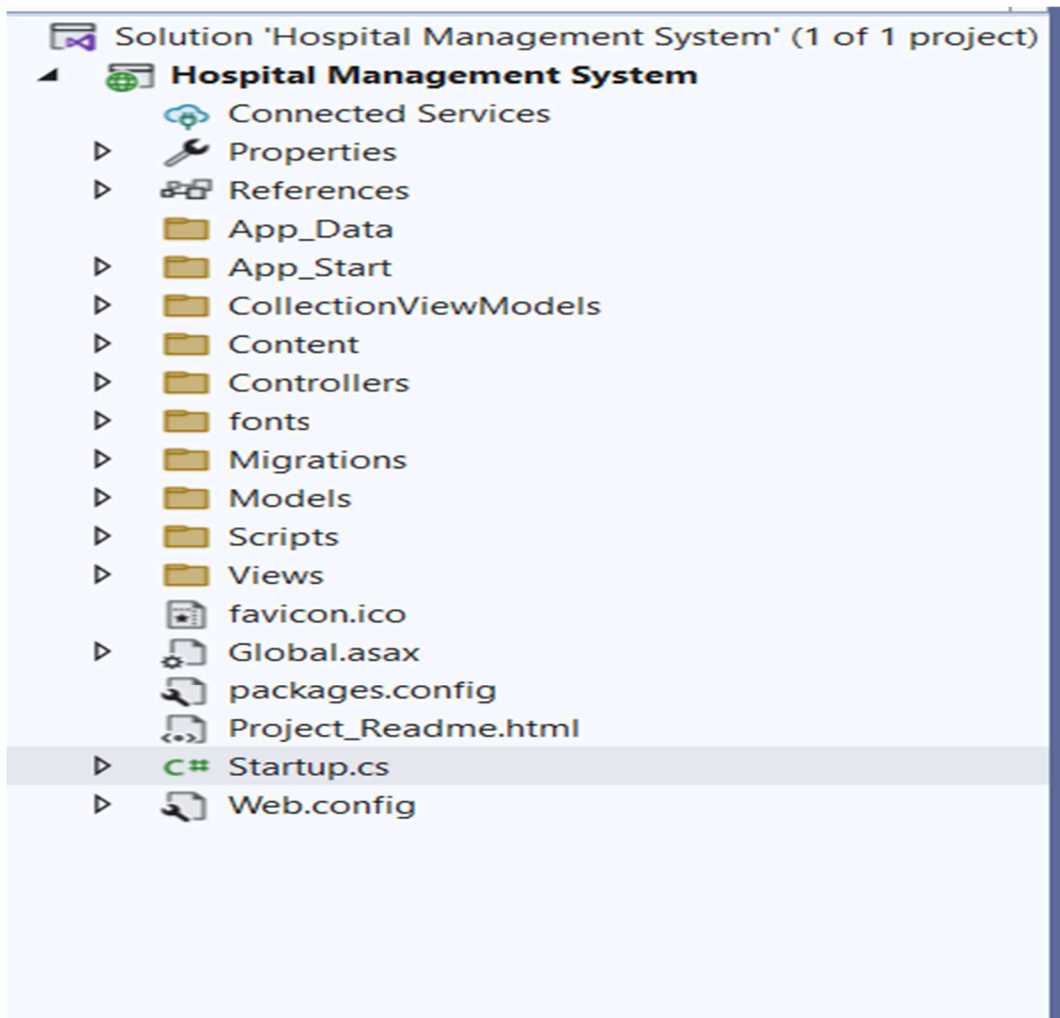


**8. Component Level diagram**



## 8. Module Wise Code

## Project Architecture :



## Account Controller :

```
using System;
using System.Globalization;
using System.Linq;
using System.Security.Claims;
using System.Threading.Tasks;
using System.Web;
using System.Web.Mvc;
using Microsoft.AspNet.Identity;
using Microsoft.AspNet.Identity.Owin;
using Microsoft.Owin.Security;
using Hospital_Management_System.Models;

namespace Hospital_Management_System.Controllers
{
    [Authorize]
    public class AccountController : Controller
```

```

{
    private ApplicationDbContext db;
    private ApplicationSignInManager _signInManager;
    private ApplicationUserManager _userManager;

    public AccountController()
    {
        db = new ApplicationDbContext();
    }

    public AccountController(ApplicationUserManager userManager, ApplicationSignInManager signInManager )
    {
        UserManager = userManager;
        SignInManager = signInManager;
    }

    public ApplicationSignInManager SignInManager
    {
        get
        {
            return _signInManager ?? HttpContext.GetOwinContext().Get<ApplicationSignInManager>();
        }
        private set
        {
            _signInManager = value;
        }
    }

    public ApplicationUserManager UserManager
    {
        get
        {
            return _userManager ?? HttpContext.GetOwinContext().GetUserManager<ApplicationUserManager>();
        }
        private set
        {
            _userManager = value;
        }
    }

    //
    // GET: /Account/Login
    [AllowAnonymous]
    public ActionResult Login(string returnUrl)
    {
        ViewBag.ReturnUrl = returnUrl;
        return View();
    }

    //
    // POST: /Account/Login
    [HttpPost]
    [AllowAnonymous]
    [ValidateAntiForgeryToken]

```

```

public async Task<ActionResult> Login(LoginViewModel model, string returnUrl)
{
    if (!ModelState.IsValid)
    {
        return View(model);
    }

    // This doesn't count login failures towards account lockout
    // To enable password failures to trigger account lockout, change to shouldLockout: true
    var result = await SignInManager.PasswordSignInAsync(model.Email, model.Password, model.RememberMe,
shouldLockout: false);
    switch (result)
    {
        case SignInStatus.Success:
            var user = await UserManager.FindAsync(model.Email, model.Password);
            if (UserManager.IsInRole(user.Id, "Admin"))
            {
                return RedirectToAction("Index", "Admin");
            }
            else if (UserManager.IsInRole(user.Id, "Doctor"))
            {
                return RedirectToAction("Index", "Doctor");
            }
            else if (UserManager.IsInRole(user.Id, "Patient"))
            {
                var patient = db.Patients.Single(c => c.ApplicationUserId == user.Id);
                if (patient.BloodGroup == null || patient.Contact == null || patient.Gender == null)
                {
                    return RedirectToAction("UpdateProfile", "Patient", new {id = user.Id});
                }
                else
                {
                    return RedirectToAction("Index", "Patient");
                }
            }
            else
            {
                return RedirectToLocal(returnUrl);
            }
        case SignInStatus.LockedOut:
            return View("Lockout");
        case SignInStatus.RequiresVerification:
            return RedirectToAction("SendCode", new { returnUrl = returnUrl, RememberMe = model.RememberMe
});
        case SignInStatus.Failure:
        default:
            ModelState.AddModelError("", "Invalid login attempt.");
            return View(model);
    }
}

//
// GET: /Account/VerifyCode
[AllowAnonymous]

```

```

public async Task<ActionResult> VerifyCode(string provider, string returnUrl, bool rememberMe)
{
    // Require that the user has already logged in via username/password or external login
    if (!await SignInManager.HasBeenVerifiedAsync())
    {
        return View("Error");
    }
    return View(new VerifyCodeViewModel { Provider = provider, ReturnUrl = returnUrl, RememberMe =
rememberMe });
}

//
// POST: /Account/VerifyCode
[HttpPost]
[AllowAnonymous]
[ValidateAntiForgeryToken]
public async Task<ActionResult> VerifyCode(VerifyCodeViewModel model)
{
    if (!ModelState.IsValid)
    {
        return View(model);
    }

    // The following code protects for brute force attacks against the two factor codes.
    // If a user enters incorrect codes for a specified amount of time then the user account
    // will be locked out for a specified amount of time.
    // You can configure the account lockout settings in IdentityConfig
    var result = await SignInManager.TwoFactorSignInAsync(model.Provider, model.Code, isPersistent:
model.RememberMe, rememberBrowser: model.RememberBrowser);
    switch (result)
    {
        case SignInStatus.Success:
            return RedirectToLocal(model.ReturnUrl);
        case SignInStatus.LockedOut:
            return View("Lockout");
        case SignInStatus.Failure:
        default:
            ModelState.AddModelError("", "Invalid code.");
            return View(model);
    }
}

//
// GET: /Account/Register
[AllowAnonymous]
public ActionResult Register()
{
    return View();
}

//
// POST: /Account/Register
[HttpPost]
[AllowAnonymous]

```

```

[ValidateAntiForgeryToken]
public async Task<ActionResult> Register(RegisterViewModel model)
{
    if (ModelState.IsValid)
    {
        var user = new ApplicationUser { UserName = model.UserName, Email = model.Email, UserRole = "Patient",
RegisteredDate = DateTime.Now.Date};
        var result = await UserManager.CreateAsync(user, model.Password);
        if (result.Succeeded)
        {
            //await SignInManager.SignInAsync(user, isPersistent:false, rememberBrowser:false);

            // For more information on how to enable account confirmation and password reset please visit
http://go.microsoft.com/fwlink/?LinkID=320771
            // Send an email with this link
            // string code = await UserManager.GenerateEmailConfirmationTokenAsync(user.Id);
            // var callbackUrl = Url.Action("ConfirmEmail", "Account", new { userId = user.Id, code = code },
protocol: Request.Url.Scheme);
            // await UserManager.SendEmailAsync(user.Id, "Confirm your account", "Please confirm your account by
clicking <a href='\"\" + callbackUrl + \"\">here</a>");
            var patient = new Patient { FirstName = model.FirstName, LastName = model.LastName, EmailAddress =
model.Email, ApplicationUserId = user.Id };
            db.Patients.Add(patient);
            db.SaveChanges();
            await UserManager.AddToRoleAsync(user.Id, "Patient");
            return RedirectToAction("Login", "Account");
        }
        AddErrors(result);
    }

    // If we got this far, something failed, redisplay form

    return View(model);
}

//
// GET: /Account/ConfirmEmail
[AllowAnonymous]
public async Task<ActionResult> ConfirmEmail(string userId, string code)
{
    if (userId == null || code == null)
    {
        return View("Error");
    }
    var result = await UserManager.ConfirmEmailAsync(userId, code);
    return View(result.Succeeded ? "ConfirmEmail" : "Error");
}

//
// GET: /Account/ForgotPassword
[AllowAnonymous]
public ActionResult ForgotPassword()
{
    return View();
}

```

```

}

//
// POST: /Account/ForgotPassword
[HttpPost]
[AllowAnonymous]
[ValidateAntiForgeryToken]
public async Task<ActionResult> ForgotPassword(ForgotPasswordViewModel model)
{
    if (ModelState.IsValid)
    {
        var user = await UserManager.FindByNameAsync(model.Email);
        if (user == null || !(await UserManager.IsEmailConfirmedAsync(user.Id)))
        {
            // Don't reveal that the user does not exist or is not confirmed
            return View("ForgotPasswordConfirmation");
        }

        // For more information on how to enable account confirmation and password reset please visit
        http://go.microsoft.com/fwlink/?LinkID=320771
        // Send an email with this link
        // string code = await UserManager.GeneratePasswordResetTokenAsync(user.Id);
        // var callbackUrl = Url.Action("ResetPassword", "Account", new { userId = user.Id, code = code }, protocol:
Request.Url.Scheme);
        // await UserManager.SendEmailAsync(user.Id, "Reset Password", "Please reset your password by clicking <a
href=\"\" + callbackUrl + \"\">here</a>");
        // return RedirectToAction("ForgotPasswordConfirmation", "Account");
    }

    // If we got this far, something failed, redisplay form
    return View(model);
}

//
// GET: /Account/ForgotPasswordConfirmation
[AllowAnonymous]
public ActionResult ForgotPasswordConfirmation()
{
    return View();
}

//
// GET: /Account/ResetPassword
[AllowAnonymous]
public ActionResult ResetPassword(string code)
{
    return code == null ? View("Error") : View();
}

//
// POST: /Account/ResetPassword
[HttpPost]
[AllowAnonymous]
[ValidateAntiForgeryToken]

```



```

public async Task<ActionResult> ResetPassword(ResetPasswordViewModel model)
{
    if (!ModelState.IsValid)
    {
        return View(model);
    }
    var user = await UserManager.FindByNameAsync(model.Email);
    if (user == null)
    {
        // Don't reveal that the user does not exist
        return RedirectToAction("ResetPasswordConfirmation", "Account");
    }
    var result = await UserManager.ResetPasswordAsync(user.Id, model.Code, model.Password);
    if (result.Succeeded)
    {
        return RedirectToAction("ResetPasswordConfirmation", "Account");
    }
    AddErrors(result);
    return View();
}

//
// GET: /Account/ResetPasswordConfirmation
[AllowAnonymous]
public ActionResult ResetPasswordConfirmation()
{
    return View();
}

//
// POST: /Account/ExternalLogin
[HttpPost]
[AllowAnonymous]
[ValidateAntiForgeryToken]
public ActionResult ExternalLogin(string provider, string returnUrl)
{
    // Request a redirect to the external login provider
    return new ChallengeResult(provider, Url.Action("ExternalLoginCallback", "Account", new { ReturnUrl = returnUrl }));
}

//
// GET: /Account/SendCode
[AllowAnonymous]
public async Task<ActionResult> SendCode(string returnUrl, bool rememberMe)
{
    var userId = await SignInManager.GetVerifiedUserIdAsync();
    if (userId == null)
    {
        return View("Error");
    }
    var userFactors = await UserManager.GetValidTwoFactorProvidersAsync(userId);
    var factorOptions = userFactors.Select(purpose => new SelectListItem { Text = purpose, Value = purpose
}).ToList();

```

```

        return View(new SendCodeViewModel { Providers = factorOptions, returnUrl = returnUrl, RememberMe =
rememberMe });
    }

    //
    // POST: /Account/SendCode
    [HttpPost]
    [AllowAnonymous]
    [ValidateAntiForgeryToken]
    public async Task<ActionResult> SendCode(SendCodeViewModel model)
    {
        if (!ModelState.IsValid)
        {
            return View();
        }

        // Generate the token and send it
        if (!await SignInManager.SendTwoFactorCodeAsync(model.SelectedProvider))
        {
            return View("Error");
        }
        return RedirectToAction("VerifyCode", new { Provider = model.SelectedProvider, returnUrl =
model.returnUrl, RememberMe = model.RememberMe });
    }

    //
    // GET: /Account/ExternalLoginCallback
    [AllowAnonymous]
    public async Task<ActionResult> ExternalLoginCallback(string returnUrl)
    {
        var loginInfo = await AuthenticationManager.GetExternalLoginInfoAsync();
        if (loginInfo == null)
        {
            return RedirectToAction("Login");
        }

        // Sign in the user with this external login provider if the user already has a login
        var result = await SignInManager.ExternalSignInAsync(loginInfo, isPersistent: false);
        switch (result)
        {
            case SignInStatus.Success:
                return RedirectToLocal(returnUrl);
            case SignInStatus.LockedOut:
                return View("Lockout");
            case SignInStatus.RequiresVerification:
                return RedirectToAction("SendCode", new { returnUrl = returnUrl, RememberMe = false });
            case SignInStatus.Failure:
            default:
                // If the user does not have an account, then prompt the user to create an account
                ViewBag.ReturnUrl = returnUrl;
                ViewBag.LoginProvider = loginInfo.Login.LoginProvider;
                return View("ExternalLoginConfirmation", new ExternalLoginConfirmationViewModel { Email =
loginInfo.Email });
        }
    }

```

```

    }

    //
    // POST: /Account/ExternalLoginConfirmation
    [HttpPost]
    [AllowAnonymous]
    [ValidateAntiForgeryToken]
    public async Task<ActionResult> ExternalLoginConfirmation(ExternalLoginConfirmationViewModel model,
string returnUrl)
    {
        if (User.Identity.IsAuthenticated)
        {
            return RedirectToAction("Index", "Manage");
        }

        if (ModelState.IsValid)
        {
            // Get the information about the user from the external login provider
            var info = await AuthenticationManager.GetExternalLoginInfoAsync();
            if (info == null)
            {
                return View("ExternalLoginFailure");
            }
            var user = new ApplicationUser { UserName = model.Email, Email = model.Email };
            var result = await UserManager.CreateAsync(user);
            if (result.Succeeded)
            {
                result = await UserManager.AddLoginAsync(user.Id, info.Login);
                if (result.Succeeded)
                {
                    await SignInManager.SignInAsync(user, isPersistent: false, rememberBrowser: false);
                    return RedirectToLocal(returnUrl);
                }
            }
            AddErrors(result);
        }

        ViewBag.ReturnUrl = returnUrl;
        return View(model);
    }

    //
    // POST: /Account/LogOff
    [HttpPost]
    [ValidateAntiForgeryToken]
    public ActionResult LogOff()
    {
        AuthenticationManager.SignOut(DefaultAuthenticationTypes.ApplicationCookie);
        return RedirectToAction("Login", "Account");
    }

    //
    // GET: /Account/ExternalLoginFailure
    [AllowAnonymous]

```

```

public ActionResult ExternalLoginFailure()
{
    return View();
}

protected override void Dispose(bool disposing)
{
    if (disposing)
    {
        if (_userManager != null)
        {
            _userManager.Dispose();
            _userManager = null;
        }

        if (_signInManager != null)
        {
            _signInManager.Dispose();
            _signInManager = null;
        }
    }

    base.Dispose(disposing);
    db.Dispose();
}

#region Helpers
// Used for XSRF protection when adding external logins
private const string XsrfKey = "XsrfId";

private IAuthenticationManager AuthenticationManager
{
    get
    {
        return HttpContext.GetOwinContext().Authentication;
    }
}

private void AddErrors(IdentityResult result)
{
    foreach (var error in result.Errors)
    {
        ModelState.AddModelError("", error);
    }
}

private ActionResult RedirectToLocal(string returnUrl)
{
    if (Url.IsLocalUrl(returnUrl))
    {
        return Redirect(returnUrl);
    }
    return RedirectToAction("Index", "Home");
}

```

```

internal class ChallengeResult : HttpUnauthorizedResult
{
    public ChallengeResult(string provider, string redirectUri)
        : this(provider, redirectUri, null)
    {
    }

    public ChallengeResult(string provider, string redirectUri, string userId)
    {
        LoginProvider = provider;
        RedirectUri = redirectUri;
        UserId = userId;
    }

    public string LoginProvider { get; set; }
    public string RedirectUri { get; set; }
    public string UserId { get; set; }

    public override void ExecuteResult(ControllerContext context)
    {
        var properties = new AuthenticationProperties { RedirectUri = RedirectUri };
        if (UserId != null)
        {
            properties.Dictionary[XsrfKey] = UserId;
        }
        context.HttpContext.GetOwinContext().Authentication.Challenge(properties, LoginProvider);
    }
}
#endregion
}
}

```

## **Admin Controller :**

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using System.Data.Entity;
using System.Net;
using System.Threading.Tasks;
using Microsoft.AspNet.Identity;
using Microsoft.AspNet.Identity.Owin;
using Microsoft.Owin.Security;
using Hospital_Management_System.CollectionViewModels;
using Hospital_Management_System.Models;

namespace Hospital_Management_System.Controllers
{
    public class AdminController : Controller
    {

```

```

private ApplicationDbContext db;

private ApplicationUserManager _userManager;

//Constructor
public AdminController()
{
    db = new ApplicationDbContext();
}

//Destructor
protected override void Dispose(bool disposing)
{
    db.Dispose();
}

public ApplicationUserManager UserManager
{
    get { return _userManager ?? HttpContext.GetOwinContext().GetUserManager<ApplicationUserManager>(); }
    private set { _userManager = value; }
}

// GET: Admin
[Authorize(Roles = "Admin")]
public ActionResult Index(string message)
{
    var date = DateTime.Now.Date;
    ViewBag.Messege = message;
    var model = new CollectionOfAll
    {
        Ambulances = db.Ambulances.ToList(),
        Departments = db.Department.ToList(),
        Doctors = db.Doctors.ToList(),
        Patients = db.Patients.ToList(),
        Medicines = db.Medicines.ToList(),
        ActiveAppointments =
            db.Appointments.Where(c => c.Status).Where(c => c.AppointmentDate >= date).ToList(),
        PendingAppointments = db.Appointments.Where(c => c.Status == false)
            .Where(c => c.AppointmentDate >= date).ToList(),
        AmbulanceDrivers = db.AmbulanceDrivers.ToList()
    };
    return View(model);
}

//Department Section

//Department List
[Authorize(Roles = "Admin")]
public ActionResult DepartmentList()
{
    var model = db.Department.ToList();
    return View(model);
}

```

```

//Add Department
[Authorize(Roles = "Admin")]
public ActionResult AddDepartment()
{
    return View();
}

[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult AddDepartment(Department model)
{
    if (db.Department.Any(c => c.Name == model.Name))
    {
        ModelState.AddModelError("Name", "Name already present!");
        return View(model);
    }

    db.Department.Add(model);
    db.SaveChanges();
    return RedirectToAction("DepartmentList");
}

//Edit Department
[Authorize(Roles = "Admin")]
public ActionResult EditDepartment(int id)
{
    var model = db.Department.SingleOrDefault(c => c.Id == id);
    return View(model);
}

[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult EditDepartment(int id, Department model)
{
    var department = db.Department.Single(c => c.Id == id);
    department.Name = model.Name;
    department.Description = model.Description;
    department.Status = model.Status;
    db.SaveChanges();
    return RedirectToAction("DepartmentList");
}

[Authorize(Roles = "Admin")]
public ActionResult DeleteDepartment(int? id)
{
    var department = db.Department.Single(c => c.Id == id);
    return View(department);
}

[HttpPost, ActionName("DeleteDepartment")]
[ValidateAntiForgeryToken]
public ActionResult DeleteDepartment(int id)
{
    var department = db.Department.SingleOrDefault(c => c.Id == id);

```

```

    db.Department.Remove(department);
    db.SaveChanges();
    return RedirectToAction("DepartmentList");
}

//End Department Section

//Start Ambulance Section
//Ambulance Driver Section

//Add Ambulance Driver
[Authorize(Roles = "Admin")]
public ActionResult AddAmbulanceDriver()
{
    return View();
}

[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult AddAmbulanceDriver(AmbulanceDriver model)
{
    db.AmbulanceDrivers.Add(model);
    db.SaveChanges();
    return RedirectToAction("ListOfAmbulanceDrivers");
}

//Edit Ambulance Driver
[Authorize(Roles = "Admin")]
public ActionResult EditAmbulanceDriver(int id)
{
    var viewmodel = db.AmbulanceDrivers.SingleOrDefault(c => c.Id == id);
    return View(viewmodel);
}

[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult EditAmbulanceDriver(int id, AmbulanceDriver model)
{
    if (!ModelState.IsValid)
    {
        return View(model);
    }

    var driver = db.AmbulanceDrivers.Single(c => c.Id == id);
    driver.Name = model.Name;
    driver.Contact = model.Contact;
    driver.Address = model.Address;
    driver.Cnic = model.Cnic;
    db.SaveChanges();
    return RedirectToAction("ListOfAmbulanceDrivers");
}

//List of Ambulance Drivers
[Authorize(Roles = "Admin")]

```



```

public ActionResult ListOfAmbulanceDrivers()
{
    var model = db.AmbulanceDrivers.ToList();
    return View(model);
}

//Ambulance Section
//Add Ambulance
[Authorize(Roles = "Admin")]
public ActionResult AddAmbulance()
{
    var model = new AmbulanceCollection
    {
        Ambulance = new Ambulance(),
        AmbulanceDrivers = db.AmbulanceDrivers.ToList()
    };
    return View(model);
}

[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult AddAmbulance(AmbulanceCollection model)
{
    if (!ModelState.IsValid)
    {
        var viewmodel = new AmbulanceCollection
        {
            Ambulance = model.Ambulance,
            AmbulanceDrivers = db.AmbulanceDrivers.ToList()
        };
        return View(viewmodel);
    }

    db.Ambulances.Add(model.Ambulance);
    db.SaveChanges();
    return RedirectToAction("ListOfAmbulances");
}

//Edit Ambulance
[Authorize(Roles = "Admin")]
public ActionResult EditAmbulance(int id)
{
    var viewmodel = new AmbulanceCollection
    {
        Ambulance = db.Ambulances.Single(c => c.Id == id),
        AmbulanceDrivers = db.AmbulanceDrivers.ToList()
    };
    return View(viewmodel);
}

[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult EditAmbulance(int id, AmbulanceCollection model)
{

```

```

if (!ModelState.IsValid)
{
    var viewmodel = new AmbulanceCollection
    {
        Ambulance = model.Ambulance,
        AmbulanceDrivers = db.AmbulanceDrivers.ToList()
    };
    return View(viewmodel);
}
else
{
    var ambulance = db.Ambulances.Single(c => c.Id == id);
    ambulance.Name = model.Ambulance.Name;
    ambulance.AmbulanceId = model.Ambulance.AmbulanceId;
    ambulance.AmbulanceStatus = model.Ambulance.AmbulanceStatus;
    ambulance.AmbulanceDriverId = model.Ambulance.AmbulanceDriverId;
}

db.SaveChanges();
return RedirectToAction("ListOfAmbulances");
}

//List of Ambulances
[Authorize(Roles = "Admin")]
public ActionResult ListOfAmbulances()
{
    var model = db.Ambulances.Include(c => c.AmbulanceDriver).ToList();
    return View(model);
}

//Delete Ambulance
[Authorize(Roles = "Admin")]
public ActionResult DeleteAmbulance(int? id)
{
    var ambulance = db.Ambulances.Single(c => c.Id == id);
    return View(ambulance);
}

[HttpPost, ActionName("DeleteAmbulance")]
[ValidateAntiForgeryToken]
public ActionResult DeleteAmbulance(int id)
{
    var ambulance = db.Ambulances.SingleOrDefault(c => c.Id == id);
    var driver = db.AmbulanceDrivers.Single(c => c.Id == ambulance.AmbulanceDriverId);
    db.Ambulances.Remove(ambulance);
    db.AmbulanceDrivers.Remove(driver);
    db.SaveChanges();
    return RedirectToAction("ListOfAmbulances");
}

//End Ambulance Section

//Start Medicine Section

```

```

//Add Medicine
[Authorize(Roles = "Admin")]
public ActionResult AddMedicine()
{
    return View();
}

[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult AddMedicine(Medicine model)
{
    if (!ModelState.IsValid)
    {
        return View(model);
    }

    db.Medicines.Add(model);
    db.SaveChanges();
    return RedirectToAction("ListOfMedicine");
}

//List of Medicines
[Authorize(Roles = "Admin")]
public ActionResult ListOfMedicine()
{
    var medicine = db.Medicines.ToList();
    return View(medicine);
}

//Edit Medicine
[Authorize(Roles = "Admin")]
public ActionResult EditMedicine(int id)
{
    var medicine = db.Medicines.Single(c => c.Id == id);
    return View(medicine);
}

[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult EditMedicine(int id, Medicine model)
{
    if (!ModelState.IsValid)
    {
        return View(model);
    }

    var medicine = db.Medicines.Single(c => c.Id == id);
    medicine.Name = model.Name;
    medicine.Description = model.Description;
    medicine.Price = model.Price;
    medicine.Quantity = model.Quantity;

    db.SaveChanges();
    return RedirectToAction("ListOfMedicine");
}

```

```

}

//Delete Medicine
[Authorize(Roles = "Admin")]
public ActionResult DeleteMedicine(int? id)
{
    return View();
}

[HttpPost, ActionName("DeleteMedicine")]
[ValidateAntiForgeryToken]
public ActionResult DeleteMedicine(int id)
{
    var medicine = db.Medicines.Single(c => c.Id == id);
    db.Medicines.Remove(medicine);
    db.SaveChanges();
    return RedirectToAction("ListOfMedicine");
}

//End Medicine Section

//Start Doctor Section

//Add Doctor
[Authorize(Roles = "Admin")]
public ActionResult AddDoctor()
{
    var collection = new DoctorCollection
    {
        ApplicationUser = new RegisterViewModel(),
        Doctor = new Doctor(),
        Departments = db.Department.ToList()
    };
    return View(collection);
}

[HttpPost]
[ValidateAntiForgeryToken]
public async Task<ActionResult> AddDoctor(DoctorCollection model)
{
    var user = new ApplicationUser
    {
        UserName = model.ApplicationUser.UserName,
        Email = model.ApplicationUser.Email,
        UserRole = "Doctor",
        RegisteredDate = DateTime.Now.Date
    };
    var result = await UserManager.CreateAsync(user, model.ApplicationUser.Password);
    if (result.Succeeded)
    {
        await UserManager.AddToRoleAsync(user.Id, "Doctor");
        var doctor = new Doctor
        {
            FirstName = model.Doctor.FirstName,

```

```

        LastName = model.Doctor.LastName,
        FullName = "Dr. " + model.Doctor.FirstName + " " + model.Doctor.LastName,
        EmailAddress = model.ApplicationUser.Email,
        ContactNo = model.Doctor.ContactNo,
        PhoneNo = model.Doctor.PhoneNo,
        Designation = model.Doctor.Designation,
        Education = model.Doctor.Education,
        DepartmentId = model.Doctor.DepartmentId,
        Specialization = model.Doctor.Specialization,
        Gender = model.Doctor.Gender,
        BloodGroup = model.Doctor.BloodGroup,
        ApplicationUserId = user.Id,
        DateOfBirth = model.Doctor.DateOfBirth,
        Address = model.Doctor.Address,
        Status = model.Doctor.Status
    };
    db.Doctors.Add(doctor);
    db.SaveChanges();
    return RedirectToAction("ListOfDoctors");
}

return HttpNotFound();
}

//List Of Doctors
[Authorize(Roles = "Admin")]
public ActionResult ListOfDoctors()
{
    var doctor = db.Doctors.Include(c => c.Department).ToList();
    return View(doctor);
}

//Detail of Doctor
[Authorize(Roles = "Admin")]
public ActionResult DoctorDetail(int id)
{
    var doctor = db.Doctors.Include(c => c.Department).SingleOrDefault(c => c.Id == id);
    return View(doctor);
}

//Edit Doctors
[Authorize(Roles = "Admin")]
public ActionResult EditDoctors(int id)
{
    var collection = new DoctorCollection
    {
        Departments = db.Department.ToList(),
        Doctor = db.Doctors.Single(c => c.Id == id)
    };
    return View(collection);
}

[HttpPost]

```

```

[ValidateAntiForgeryToken]
public ActionResult EditDoctors(int id, DoctorCollection model)
{
    var doctor = db.Doctors.Single(c => c.Id == id);
    doctor.FirstName = model.Doctor.FirstName;
    doctor.LastName = model.Doctor.LastName;
    doctor.FullName = "Dr. " + model.Doctor.FirstName + " " + model.Doctor.LastName;
    doctor.ContactNo = model.Doctor.ContactNo;
    doctor.PhoneNo = model.Doctor.PhoneNo;
    doctor.Designation = model.Doctor.Designation;
    doctor.Education = model.Doctor.Education;
    doctor.DepartmentId = model.Doctor.DepartmentId;
    doctor.Specialization = model.Doctor.Specialization;
    doctor.Gender = model.Doctor.Gender;
    doctor.BloodGroup = model.Doctor.BloodGroup;
    doctor.DateOfBirth = model.Doctor.DateOfBirth;
    doctor.Address = model.Doctor.Address;
    doctor.Status = model.Doctor.Status;
    db.SaveChanges();

    return RedirectToAction("ListOfDoctors");
}

//Delete Doctor
[Authorize(Roles = "Admin")]
public ActionResult DeleteDoctor(string id)
{
    var UserId = db.Doctors.Single(c => c.ApplicationUserId == id);
    return View(UserId);
}

[HttpPost, ActionName("DeleteDoctor")]
[ValidateAntiForgeryToken]
public ActionResult DeleteDoctor(string id, Doctor model)
{
    var doctor = db.Doctors.Single(c => c.ApplicationUserId == id);
    var user = db.Users.Single(c => c.Id == id);
    if (db.Schedules.Where(c => c.DoctorId == doctor.Id).Equals(null))
    {
        var schedule = db.Schedules.Single(c => c.DoctorId == doctor.Id);
        db.Schedules.Remove(schedule);
    }

    db.Users.Remove(user);
    db.Doctors.Remove(doctor);
    db.SaveChanges();
    return RedirectToAction("ListOfDoctors");
}

//End Doctor Section

//Start Schedule Section
//Add Schedule
[Authorize(Roles = "Admin")]

```

```

public ActionResult AddSchedule()
{
    var collection = new ScheduleCollection
    {
        Schedule = new Schedule(),
        Doctors = db.Doctors.ToList()
    };
    return View(collection);
}

[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult AddSchedule(ScheduleCollection model)
{
    if (!ModelState.IsValid)
    {
        var collection = new ScheduleCollection
        {
            Schedule = model.Schedule,
            Doctors = db.Doctors.ToList()
        };
        return View(collection);
    }

    db.Schedules.Add(model.Schedule);
    db.SaveChanges();
    return RedirectToAction("ListOfSchedules");
}

//List Of Schedules
[Authorize(Roles = "Admin")]
public ActionResult ListOfSchedules()
{
    var schedule = db.Schedules.Include(c => c.Doctor).ToList();
    return View(schedule);
}

//Edit Schedule
[Authorize(Roles = "Admin")]
public ActionResult EditSchedule(int id)
{
    var collection = new ScheduleCollection
    {
        Schedule = db.Schedules.Single(c => c.Id == id),
        Doctors = db.Doctors.ToList()
    };
    return View(collection);
}

[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult EditSchedule(int id, ScheduleCollection model)
{
    if (!ModelState.IsValid)

```

```

{
    return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
}

var schedule = db.Schedules.Single(c => c.Id == id);
schedule.DoctorId = model.Schedule.DoctorId;
schedule.AvailableEndDay = model.Schedule.AvailableEndDay;
schedule.AvailableEndTime = model.Schedule.AvailableEndTime;
schedule.AvailableStartDay = model.Schedule.AvailableStartDay;
schedule.AvailableStartTime = model.Schedule.AvailableStartTime;
schedule.Status = model.Schedule.Status;
schedule.TimePerPatient = model.Schedule.TimePerPatient;
db.SaveChanges();
return RedirectToAction("ListOfSchedules");
}

//Delete Schedule
[Authorize(Roles = "Admin")]
public ActionResult DeleteSchedule(int? id)
{
    return View();
}

[HttpPost, ActionName("DeleteSchedule")]
[ValidateAntiForgeryToken]
public ActionResult DeleteSchedule(int id)
{
    var schedule = db.Schedules.Single(c => c.Id == id);
    db.Schedules.Remove(schedule);
    return RedirectToAction("ListOfSchedules");
}

//End Schedule Section

//Start Patient Section

//List of Patients
[Authorize(Roles = "Admin")]
public ActionResult ListOfPatients()
{
    var patients = db.Patients.ToList();
    return View(patients);
}

[Authorize(Roles = "Admin")]
public ActionResult EditPatient(int id)
{
    var patient = db.Patients.Single(c => c.Id == id);
    return View(patient);
}

[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult EditPatient(int id, Patient model)

```



```

{
    if (!ModelState.IsValid)
    {
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
    }

    var patient = db.Patients.Single(c => c.Id == id);
    patient.FirstName = model.FirstName;
    patient.LastName = model.LastName;
    patient.FullName = model.FirstName + " " + model.LastName;
    patient.Address = model.Address;
    patient.BloodGroup = model.BloodGroup;
    patient.Contact = model.Contact;
    patient.DateOfBirth = model.DateOfBirth;
    patient.EmailAddress = model.EmailAddress;
    patient.Gender = model.Gender;
    patient.PhoneNo = model.PhoneNo;
    db.SaveChanges();
    return RedirectToAction("ListOfPatients");
}

//Delete Patient
[Authorize(Roles = "Admin")]
public ActionResult DeletePatient()
{
    return View();
}

[HttpPost, ActionName("DeletePatient")]
[ValidateAntiForgeryToken]
public ActionResult DeletePatient(string id)
{
    var patient = db.Patients.Single(c => c.ApplicationUserId == id);
    var user = db.Users.Single(c => c.Id == id);
    db.Users.Remove(user);
    db.Patients.Remove(patient);
    db.SaveChanges();
    return RedirectToAction("ListOfPatients");
}

//End Patient Section

//Start Appointment Section

//Add Appointment
[Authorize(Roles = "Admin")]
public ActionResult AddAppointment()
{
    var collection = new AppointmentCollection
    {
        Appointment = new Appointment(),
        Patients = db.Patients.ToList(),
        Doctors = db.Doctors.ToList()
    };
};

```

```

    return View(collection);
}

[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult AddAppointment(AppointmentCollection model)
{
    var collection = new AppointmentCollection
    {
        Appointment = model.Appointment,
        Patients = db.Patients.ToList(),
        Doctors = db.Doctors.ToList()
    };
    if (model.Appointment.AppointmentDate >= DateTime.Now.Date)
    {
        var appointment = new Appointment();
        appointment.PatientId = model.Appointment.PatientId;
        appointment.DoctorId = model.Appointment.DoctorId;
        appointment.AppointmentDate = model.Appointment.AppointmentDate;
        appointment.Problem = model.Appointment.Problem;
        appointment.Status = model.Appointment.Status;
        db.Appointments.Add(appointment);
        db.SaveChanges();

        if (model.Appointment.Status == true)
        {
            return RedirectToAction("ListOfAppointments");
        }
        else
        {
            return RedirectToAction("PendingAppointments");
        }
    }

    ViewBag.Messege = "Please Enter the Date greater than today or equal!!";
    return View(collection);
}

//List of Active Appointment
[Authorize(Roles = "Admin")]
public ActionResult ListOfAppointments()
{
    var date = DateTime.Now.Date;
    var appointment = db.Appointments.Include(c => c.Doctor).Include(c => c.Patient)
        .Where(c => c.Status == true).Where(c => c.AppointmentDate >= date).ToList();
    return View(appointment);
}

//List of pending Appointments
[Authorize(Roles = "Admin")]
public ActionResult PendingAppointments()
{
    var date = DateTime.Now.Date;

```

```

    var appointment = db.Appointments.Include(c => c.Doctor).Include(c => c.Patient)
        .Where(c => c.Status == false).Where(c => c.AppointmentDate >= date).ToList();
    return View(appointment);
}

```

**//Edit Appointment**

**[Authorize(Roles = "Admin")]**

**public ActionResult EditAppointment(int id)**

```

{
    var collection = new AppointmentCollection
    {
        Appointment = db.Appointments.Single(c => c.Id == id),
        Patients = db.Patients.ToList(),
        Doctors = db.Doctors.ToList()
    };
    return View(collection);
}

```

**[HttpPost]**

**[ValidateAntiForgeryToken]**

**public ActionResult EditAppointment(int id, AppointmentCollection model)**

```

{
    var collection = new AppointmentCollection
    {
        Appointment = model.Appointment,
        Patients = db.Patients.ToList(),
        Doctors = db.Doctors.ToList()
    };
    if (model.Appointment.AppointmentDate >= DateTime.Now.Date)
    {
        var appointment = db.Appointments.Single(c => c.Id == id);
        appointment.PatientId = model.Appointment.PatientId;
        appointment.DoctorId = model.Appointment.DoctorId;
        appointment.AppointmentDate = model.Appointment.AppointmentDate;
        appointment.Problem = model.Appointment.Problem;
        appointment.Status = model.Appointment.Status;
        db.SaveChanges();
        if (model.Appointment.Status == true)
        {
            return RedirectToAction("ListOfAppointments");
        }
        else
        {
            return RedirectToAction("PendingAppointments");
        }
    }
    ViewBag.Messege = "Please Enter the Date greater than today or equal!!";

    return View(collection);
}

```

**//Detail of Appointment**

**[Authorize(Roles = "Admin")]**

**public ActionResult DetailOfAppointment(int id)**

```

{
    var appointment = db.Appointments.Include(c => c.Doctor).Include(c => c.Patient).Single(c => c.Id == id);
    return View(appointment);
}

//Delete Appointment
[Authorize(Roles = "Admin")]
public ActionResult DeleteAppointment(int? id)
{
    var appointment = db.Appointments.Single(c => c.Id == id);
    return View(appointment);
}

[HttpPost, ActionName("DeleteAppointment")]
[ValidateAntiForgeryToken]
public ActionResult DeleteAppointment(int id)
{
    var appointment = db.Appointments.Single(c => c.Id == id);
    db.Appointments.Remove(appointment);
    db.SaveChanges();
    if (appointment.Status)
    {
        return RedirectToAction("ListOfAppointments");
    }
    else
    {
        return RedirectToAction("PendingAppointments");
    }
}

//End Appointment Section

//Start Announcement Section

//Add Announcement
[Authorize(Roles = "Admin")]
public ActionResult AddAnnouncement()
{
    return View();
}

[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult AddAnnouncement(Announcement model)
{
    if (model.End >= DateTime.Now.Date)
    {
        db.Announcements.Add(model);
        db.SaveChanges();
        return RedirectToAction("ListOfAnnouncement");
    }
    else
    {
        ViewBag.Messege = "Please Enter the Date greater than today!!";
    }
}

```

```

    }

    return View(model);
}

//List of Announcement
[Authorize(Roles = "Admin")]
public ActionResult ListOfAnnouncement()
{
    var list = db.Announcements.ToList();
    return View(list);
}

//Edit Announcement
[Authorize(Roles = "Admin")]
public ActionResult EditAnnouncement(int id)
{
    var announcement = db.Announcements.Single(c => c.Id == id);
    return View(announcement);
}

[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult EditAnnouncement(int id, Announcement model)
{
    var announcement = db.Announcements.Single(c => c.Id == id);
    if (model.End >= DateTime.Now.Date)
    {
        announcement.Announcements = model.Announcements;
        announcement.End = model.End;
        announcement.AnnouncementFor = model.AnnouncementFor;
        db.SaveChanges();
        return RedirectToAction("ListOfAnnouncement");
    }
    else
    {
        ViewBag.Messege = "Please Enter the Date greater than today!!";
    }

    return View(announcement);
}

//Delete Announcement
[Authorize(Roles = "Admin")]
public ActionResult DeleteAnnouncement(int? id)
{
    return View();
}

[HttpPost, ActionName("DeleteAnnouncement")]
[ValidateAntiForgeryToken]
public ActionResult DeleteAnnouncement(int id)
{
    var announcement = db.Announcements.Single(c => c.Id == id);

```

```

        db.Announcements.Remove(announcement);
        db.SaveChanges();
        return RedirectToAction("ListOfAnnouncement");
    }

//Start Complaint Section

//List of Complaints
[Authorize(Roles = "Admin")]
public ActionResult ListOfComplains()
{
    var complain = db.Complaints.ToList();
    return View(complain);
}

//Edit Complaint
[Authorize(Roles = "Admin")]
public ActionResult EditComplain(int id)
{
    var complain = db.Complaints.Single(c => c.Id == id);
    return View(complain);
}

[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult EditComplain(int id, Complaint model)
{
    var complain = db.Complaints.Single(c => c.Id == id);
    complain.Complain = model.Complain;
    complain.Reply = model.Reply;
    db.SaveChanges();
    return RedirectToAction("ListOfComplains");
}

//Delete Complaint
[Authorize(Roles = "Admin")]
public ActionResult DeleteComplain()
{
    return View();
}

[HttpPost, ActionName("DeleteComplain")]
[ValidateAntiForgeryToken]
public ActionResult DeleteComplain(int id)
{
    var complain = db.Complaints.Single(c => c.Id == id);
    db.Complaints.Remove(complain);
    db.SaveChanges();
    return RedirectToAction("ListOfComplains");
}
}
}

```

## Doctor Controller :

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Security.Policy;
using System.Web;
using System.Web.Mvc;
using Hospital_Management_System.CollectionViewModels;
using Hospital_Management_System.Models;
using Microsoft.AspNet.Identity;
using System.Data.Entity;
using Microsoft.AspNet.Identity.EntityFramework;

namespace Hospital_Management_System.Controllers
{
    public class DoctorController : Controller
    {
        private ApplicationDbContext db;
        //Constructor
        public DoctorController()
        {
            db = new ApplicationDbContext();
        }

        //Destructor
        protected override void Dispose(bool disposing)
        {
            db.Dispose();
        }

        // GET: Doctor
        [Authorize(Roles = "Doctor")]
        public ActionResult Index(string message)
        {
            var date = DateTime.Now.Date;
            ViewBag.Messege = message;
            var user = User.Identity.GetUserId();
            var doctor = db.Doctors.Single(c => c.ApplicationUserId == user);
            var model = new CollectionOfAll
            {
                Ambulances = db.Ambulances.ToList(),
                Departments = db.Department.ToList(),
                Doctors = db.Doctors.ToList(),
                Patients = db.Patients.ToList(),
                Medicines = db.Medicines.ToList(),
                ActiveAppointments = db.Appointments.Where(c => c.DoctorId == doctor.Id).Where(c => c.Status).Where(c
=> c.AppointmentDate >= date).ToList(),
                PendingAppointments = db.Appointments.Where(c => c.DoctorId == doctor.Id).Where(c => c.Status ==
false).Where(c => c.AppointmentDate >= date).ToList(),
                AmbulanceDrivers = db.AmbulanceDrivers.ToList(),
                Announcements = db.Announcements.Where(c => c.AnnouncementFor == "Doctor").ToList()
            };
        }
    }
}
```

```

    return View(model);
}

//Add Prescription
[Authorize(Roles = "Doctor")]
public ActionResult AddPrescription()
{
    var collection = new PrescriptionCollection
    {
        Prescription = new Prescription(),
        Patients = db.Patients.ToList()
    };
    return View(collection);
}

[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult AddPrescription(PrescriptionCollection model)
{
    string user = User.Identity.GetUserId();
    var patient = db.Patients.Single(c => c.Id == model.Prescription.PatientId);
    var doctor = db.Doctors.Single(c => c.ApplicationUserId == user);
    var schedule = db.Schedules.Single(c => c.DoctorId == doctor.Id);
    var patientuser = db.Users.Single(c => c.Id == patient.ApplicationUserId);
    var prescription = new Prescription
    {
        PatientId = model.Prescription.PatientId,
        DoctorId = doctor.Id,
        DoctorName = doctor.FullName,
        DoctorSpecialization = doctor.Specialization,
        PatientName = patient.FullName,
        PatientGender = patient.Gender,
        UserName = patientuser.UserName,
        MedicalTest1 = model.Prescription.MedicalTest1,
        MedicalTest2 = model.Prescription.MedicalTest2,
        MedicalTest3 = model.Prescription.MedicalTest3,
        MedicalTest4 = model.Prescription.MedicalTest4,
        Medicine1 = model.Prescription.Medicine1,
        Medicine2 = model.Prescription.Medicine2,
        Medicine3 = model.Prescription.Medicine3,
        Medicine4 = model.Prescription.Medicine4,
        Medicine5 = model.Prescription.Medicine5,
        Medicine6 = model.Prescription.Medicine6,
        Medicine7 = model.Prescription.Medicine7,
        Morning1 = model.Prescription.Morning1,
        Morning2 = model.Prescription.Morning2,
        Morning3 = model.Prescription.Morning3,
        Morning4 = model.Prescription.Morning4,
        Morning5 = model.Prescription.Morning5,
        Morning6 = model.Prescription.Morning6,
        Morning7 = model.Prescription.Morning7,
        Afternoon1 = model.Prescription.Afternoon1,
        Afternoon2 = model.Prescription.Afternoon2,
        Afternoon3 = model.Prescription.Afternoon3,
    };
}

```



```

        Afternoon4 = model.Prescription.Afternoon4,
        Afternoon5 = model.Prescription.Afternoon5,
        Afternoon6 = model.Prescription.Afternoon6,
        Afternoon7 = model.Prescription.Afternoon7,
        Evening1 = model.Prescription.Evening1,
        Evening2 = model.Prescription.Evening2,
        Evening3 = model.Prescription.Evening3,
        Evening4 = model.Prescription.Evening4,
        Evening5 = model.Prescription.Evening5,
        Evening6 = model.Prescription.Evening6,
        Evening7 = model.Prescription.Evening7,
        CheckUpAfterDays = model.Prescription.CheckUpAfterDays,
        PrescriptionAddDate = DateTime.Now.Date,
        DoctorTiming = "From " + schedule.AvailableStartTime.ToShortTimeString() + " to " +
schedule.AvailableEndTime.ToShortTimeString()
    };

    db.Prescription.Add(prescription);
    db.SaveChanges();
    return RedirectToAction("ListOfPrescription");
}

//List of Prescription
[Authorize(Roles = "Doctor")]
public ActionResult ListOfPrescription()
{
    var user = User.Identity.GetUserId();
    var doctor = db.Doctors.Single(c => c.ApplicationUserId == user);
    var prescription = db.Prescription.Where(c => c.DoctorId == doctor.Id).ToList();
    return View(prescription);
}

//View Of Prescription
[Authorize(Roles = "Doctor")]
public ActionResult ViewPrescription(int id)
{
    var prescription = db.Prescription.Single(c => c.Id == id);
    return View(prescription);
}

//Delete Prescription
[Authorize(Roles = "Doctor")]
public ActionResult DeletePrescription(int? id)
{
    return View();
}

[HttpPost, ActionName("DeletePrescription")]
[ValidateAntiForgeryToken]
public ActionResult DeletePrescription(int id)
{
    var prescription = db.Prescription.Single(c => c.Id == id);
    db.Prescription.Remove(prescription);
    db.SaveChanges();
}

```

```

    return RedirectToAction("ListOfPrescription");
}

//Edit Prescription
[Authorize(Roles = "Doctor")]
public ActionResult EditPrescription(int id)
{
    var prescription = db.Prescription.Single(c => c.Id == id);
    return View(prescription);
}

[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult EditPrescription(int id, Prescription model)
{
    var prescription = db.Prescription.Single(c => c.Id == id);
    prescription.MedicalTest1 = model.MedicalTest1;
    prescription.MedicalTest2 = model.MedicalTest2;
    prescription.MedicalTest3 = model.MedicalTest3;
    prescription.MedicalTest4 = model.MedicalTest4;
    prescription.Medicine1 = model.Medicine1;
    prescription.Medicine2 = model.Medicine2;
    prescription.Medicine3 = model.Medicine3;
    prescription.Medicine4 = model.Medicine4;
    prescription.Medicine5 = model.Medicine5;
    prescription.Medicine6 = model.Medicine6;
    prescription.Medicine7 = model.Medicine7;
    prescription.Morning1 = model.Morning1;
    prescription.Morning2 = model.Morning2;
    prescription.Morning3 = model.Morning3;
    prescription.Morning4 = model.Morning4;
    prescription.Morning5 = model.Morning5;
    prescription.Morning6 = model.Morning6;
    prescription.Morning7 = model.Morning7;
    prescription.Afternoon1 = model.Afternoon1;
    prescription.Afternoon2 = model.Afternoon2;
    prescription.Afternoon3 = model.Afternoon3;
    prescription.Afternoon4 = model.Afternoon4;
    prescription.Afternoon5 = model.Afternoon5;
    prescription.Afternoon6 = model.Afternoon6;
    prescription.Afternoon7 = model.Afternoon7;
    prescription.Evening1 = model.Evening1;
    prescription.Evening2 = model.Evening2;
    prescription.Evening3 = model.Evening3;
    prescription.Evening4 = model.Evening4;
    prescription.Evening5 = model.Evening5;
    prescription.Evening6 = model.Evening6;
    prescription.Evening7 = model.Evening7;
    prescription.CheckUpAfterDays = model.CheckUpAfterDays;
    db.SaveChanges();
    return RedirectToAction("ListOfPrescription");
}

//End Prescription Section

```

**//Start Schedule Section**

**//Check his Schedule**

**[Authorize(Roles = "Doctor")]**

**public ActionResult ScheduleDetail()**

```
{
    string user = User.Identity.GetUserId();
    var doctor = db.Doctors.Single(c => c.ApplicationUserId == user);
    var schedule = db.Schedules.Where(c => c.DoctorId == doctor.Id).FirstOrDefault();
    if (schedule==null)
    {
        schedule = new Schedule();
    }
    return View(schedule);
}
```

**//Edit Schedule**

**[Authorize(Roles = "Doctor")]**

**public ActionResult EditSchedule(int id)**

```
{
    var schedule = db.Schedules.Single(c => c.Id == id);
    return View(schedule);
}
```

**[HttpPost]**

**[ValidateAntiForgeryToken]**

**public ActionResult EditSchedule(int id, Schedule model)**

```
{
    var schedule = db.Schedules.Single(c => c.Id == id);
    schedule.AvailableEndDay = model.AvailableEndDay;
    schedule.AvailableEndTime = model.AvailableEndTime;
    schedule.AvailableStartDay = model.AvailableStartDay;
    schedule.AvailableStartTime = model.AvailableStartTime;
    schedule.Status = model.Status;
    schedule.TimePerPatient = model.TimePerPatient;
    db.SaveChanges();
    return RedirectToAction("ScheduleDetail");
}
```

**//End schedule Section**

**//Start Appointment Section**

**[Authorize(Roles = "Doctor")]**

**public ActionResult AddAppointment()**

```
{
    var collection = new AppointmentCollection
    {
        Appointment = new Appointment(),
        Patients = db.Patients.ToList()
    };
    return View(collection);
}
```

```

[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult AddAppointment(AppointmentCollection model)
{
    string user = User.Identity.GetUserId();
    var collection = new AppointmentCollection
    {
        Appointment = model.Appointment,
        Patients = db.Patients.ToList()
    };
    if (model.Appointment.AppointmentDate >= DateTime.Now.Date)
    {
        var doctor = db.Doctors.Single(c => c.ApplicationUserId == user);
        var appointment = new Appointment();
        appointment.PatientId = model.Appointment.PatientId;
        appointment.DoctorId = doctor.Id;
        appointment.AppointmentDate = model.Appointment.AppointmentDate;
        appointment.Problem = model.Appointment.Problem;
        appointment.Status = model.Appointment.Status;

        db.Appointments.Add(appointment);
        db.SaveChanges();

        if (model.Appointment.Status == true)
        {
            return RedirectToAction("ActiveAppointments");
        }
        else
        {
            return RedirectToAction("PendingAppointments");
        }
    }
    ViewBag.Messege = "Please Enter the Date greater than today or equal!!!";

    return View(collection);
}

//List of Active Appointments
[Authorize(Roles = "Doctor")]
public ActionResult ActiveAppointments()
{
    var user = User.Identity.GetUserId();
    var doctor = db.Doctors.Single(c => c.ApplicationUserId == user);
    var date = DateTime.Now.Date;
    var appointment = db.Appointments.Include(c => c.Doctor).Include(c => c.Patient).Where(c => c.DoctorId ==
doctor.Id).Where(c => c.Status == true).Where(c => c.AppointmentDate >= date).ToList();
    return View(appointment);
}

//List of Pending Appointments
public ActionResult PendingAppointments()
{
    var user = User.Identity.GetUserId();
    var doctor = db.Doctors.Single(c => c.ApplicationUserId == user);

```

```

        var date = DateTime.Now.Date;
        var appointment = db.Appointments.Include(c => c.Doctor).Include(c => c.Patient).Where(c => c.DoctorId ==
doctor.Id).Where(c => c.Status == false).Where(c => c.AppointmentDate >= date).ToList();
        return View(appointment);
    }

//Edit Appointment
[Authorize(Roles = "Doctor")]
public ActionResult EditAppointment(int id)
{
    var collection = new AppointmentCollection
    {
        Appointment = db.Appointments.Single(c => c.Id == id),
        Patients = db.Patients.ToList()
    };
    return View(collection);
}

[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult EditAppointment(int id, AppointmentCollection model)
{
    var collection = new AppointmentCollection
    {
        Appointment = model.Appointment,
        Patients = db.Patients.ToList()
    };
    if (model.Appointment.AppointmentDate >= DateTime.Now.Date)
    {
        var appointment = db.Appointments.Single(c => c.Id == id);
        appointment.PatientId = model.Appointment.PatientId;
        appointment.AppointmentDate = model.Appointment.AppointmentDate;
        appointment.Problem = model.Appointment.Problem;
        appointment.Status = model.Appointment.Status;
        db.SaveChanges();
        if (model.Appointment.Status == true)
        {
            return RedirectToAction("ActiveAppointments");
        }
        else
        {
            return RedirectToAction("PendingAppointments");
        }
    }
    ViewBag.Messege = "Please Enter the Date greater than today or equal!!";

    return View(collection);
}

//Detail of appointment
[Authorize(Roles = "Doctor")]
public ActionResult DetailOfAppointment(int id)
{
    var appointment = db.Appointments.Include(c => c.Doctor).Include(c => c.Patient).Single(c => c.Id == id);

```

```

        return View(appointment);
    }

    //Delete Appointment
    [Authorize(Roles = "Doctor")]
    public ActionResult DeleteAppointment(int? id)
    {
        var appointment = db.Appointments.Single(c => c.Id == id);
        return View(appointment);
    }

    [HttpPost, ActionName("DeleteAppointment")]
    [ValidateAntiForgeryToken]
    public ActionResult DeleteAppointment(int id)
    {
        var appointment = db.Appointments.Single(c => c.Id == id);
        db.Appointments.Remove(appointment);
        db.SaveChanges();
        if (appointment.Status)
        {
            return RedirectToAction("ActiveAppointments");
        }
        else
        {
            return RedirectToAction("PendingAppointments");
        }
    }
}
}

```

## **Home Controller :**

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using Hospital_Management_System.Models;

namespace Hospital_Management_System.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }

        //[HttpPost]
        //[ValidateAntiForgeryToken]
        //public ActionResult Index(Reviews model)
        //{

        //}

        public ActionResult About()
        {
            ViewBag.Message = "Your application description page.";

```

```

        return View();
    }

    public ActionResult Contact()
    {
        ViewBag.Message = "Your contact page.";

        return View();
    }
}

```

## Manage Controller :

```

using System;
using System.Linq;
using System.Threading.Tasks;
using System.Web;
using System.Web.Mvc;
using Microsoft.AspNet.Identity;
using Microsoft.AspNet.Identity.Owin;
using Microsoft.Owin.Security;
using Hospital_Management_System.Models;

```

```

namespace Hospital_Management_System.Controllers
{

```

```

    [Authorize]

```

```

    public class ManageController : Controller
    {

```

```

        private ApplicationSignInManager _signInManager;
        private ApplicationUserManager _userManager;

```

```

        public ManageController()
        {
        }

```

```

        public ManageController(ApplicationUserManager userManager, ApplicationSignInManager signInManager)
        {
            UserManager = userManager;
            SignInManager = signInManager;
        }

```

```

        public ApplicationSignInManager SignInManager

```

```

{
    get
    {
        return _signInManager ?? HttpContext.GetOwinContext().Get<ApplicationSignInManager>();
    }
    private set
    {
        _signInManager = value;
    }
}

public ApplicationUserManager UserManager
{
    get
    {
        return _userManager ?? HttpContext.GetOwinContext().GetUserManager<ApplicationUserManager>();
    }
    private set
    {
        _userManager = value;
    }
}

//
// GET: /Manage/Index
public async Task<ActionResult> Index(ManageMessageId? message)
{
    ViewBag.StatusMessage =
        message == ManageMessageId.ChangePasswordSuccess ? "Your password has been changed."
        : message == ManageMessageId.SetPasswordSuccess ? "Your password has been set."
        : message == ManageMessageId.SetTwoFactorSuccess ? "Your two-factor authentication provider has been
set."
        : message == ManageMessageId.Error ? "An error has occurred."
        : message == ManageMessageId.AddPhoneSuccess ? "Your phone number was added."
        : message == ManageMessageId.RemovePhoneSuccess ? "Your phone number was removed."
        : "";

    var userId = User.Identity.GetUserId();
    var model = new IndexViewModel
    {
        HasPassword = HasPassword(),
        PhoneNumber = await UserManager.GetPhoneNumberAsync(userId),
        TwoFactor = await UserManager.GetTwoFactorEnabledAsync(userId),
        Logins = await UserManager.GetLoginsAsync(userId),
        BrowserRemembered = await AuthenticationManager.TwoFactorBrowserRememberedAsync(userId)
    };
    return View(model);
}

//
// POST: /Manage/RemoveLogin
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<ActionResult> RemoveLogin(string loginProvider, string providerKey)

```



```

{
    ManageMessageId? message;
    var result = await UserManager.RemoveLoginAsync(User.Identity.GetUserId(), new
UserLoginInfo(loginProvider, providerKey));
    if (result.Succeeded)
    {
        var user = await UserManager.FindByIdAsync(User.Identity.GetUserId());
        if (user != null)
        {
            await SignInManager.SignInAsync(user, isPersistent: false, rememberBrowser: false);
        }
        message = ManageMessageId.RemoveLoginSuccess;
    }
    else
    {
        message = ManageMessageId.Error;
    }
    return RedirectToAction("ManageLogins", new { Message = message });
}

//
// GET: /Manage/AddPhoneNumber
public ActionResult AddPhoneNumber()
{
    return View();
}

//
// POST: /Manage/AddPhoneNumber
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<ActionResult> AddPhoneNumber(AddPhoneNumberViewModel model)
{
    if (!ModelState.IsValid)
    {
        return View(model);
    }
    // Generate the token and send it
    var code = await UserManager.GenerateChangePhoneNumberTokenAsync(User.Identity.GetUserId(),
model.Number);
    if (UserManager.SmsService != null)
    {
        var message = new IdentityMessage
        {
            Destination = model.Number,
            Body = "Your security code is: " + code
        };
        await UserManager.SmsService.SendAsync(message);
    }
    return RedirectToAction("VerifyPhoneNumber", new { PhoneNumber = model.Number });
}

//
// POST: /Manage/EnableTwoFactorAuthentication

```

```

[HttpPost]
[ValidateAntiForgeryToken]
public async Task<ActionResult> EnableTwoFactorAuthentication()
{
    await UserManager.SetTwoFactorEnabledAsync(User.Identity.GetUserId(), true);
    var user = await UserManager.FindByIdAsync(User.Identity.GetUserId());
    if (user != null)
    {
        await SignInManager.SignInAsync(user, isPersistent: false, rememberBrowser: false);
    }
    return RedirectToAction("Index", "Manage");
}

//
// POST: /Manage/DisableTwoFactorAuthentication
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<ActionResult> DisableTwoFactorAuthentication()
{
    await UserManager.SetTwoFactorEnabledAsync(User.Identity.GetUserId(), false);
    var user = await UserManager.FindByIdAsync(User.Identity.GetUserId());
    if (user != null)
    {
        await SignInManager.SignInAsync(user, isPersistent: false, rememberBrowser: false);
    }
    return RedirectToAction("Index", "Manage");
}

//
// GET: /Manage/VerifyPhoneNumber
public async Task<ActionResult> VerifyPhoneNumber(string phoneNumber)
{
    var code = await UserManager.GenerateChangePhoneNumberTokenAsync(User.Identity.GetUserId(),
phoneNumber);
    // Send an SMS through the SMS provider to verify the phone number
    return phoneNumber == null ? View("Error") : View(new VerifyPhoneNumberViewModel { PhoneNumber =
phoneNumber });
}

//
// POST: /Manage/VerifyPhoneNumber
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<ActionResult> VerifyPhoneNumber(VerifyPhoneNumberViewModel model)
{
    if (!ModelState.IsValid)
    {
        return View(model);
    }
    var result = await UserManager.ChangePhoneNumberAsync(User.Identity.GetUserId(), model.PhoneNumber,
model.Code);
    if (result.Succeeded)
    {
        var user = await UserManager.FindByIdAsync(User.Identity.GetUserId());

```

```

        if (user != null)
        {
            await SignInManager.SignInAsync(user, isPersistent: false, rememberBrowser: false);
        }
        return RedirectToAction("Index", new { Message = ManageMessageId.AddPhoneSuccess });
    }
    // If we got this far, something failed, redisplay form
    ModelState.AddModelError("", "Failed to verify phone");
    return View(model);
}

//
// POST: /Manage/RemovePhoneNumber
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<ActionResult> RemovePhoneNumber()
{
    var result = await UserManager.SetPhoneNumberAsync(User.Identity.GetUserId(), null);
    if (!result.Succeeded)
    {
        return RedirectToAction("Index", new { Message = ManageMessageId.Error });
    }
    var user = await UserManager.FindByIdAsync(User.Identity.GetUserId());
    if (user != null)
    {
        await SignInManager.SignInAsync(user, isPersistent: false, rememberBrowser: false);
    }
    return RedirectToAction("Index", new { Message = ManageMessageId.RemovePhoneSuccess });
}

//
// GET: /Manage/ChangePassword
public ActionResult ChangePassword()
{
    return View();
}

//
// POST: /Manage/ChangePassword
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<ActionResult> ChangePassword(ChangePasswordViewModel model)
{
    if (!ModelState.IsValid)
    {
        return View(model);
    }
    var result = await UserManager.ChangePasswordAsync(User.Identity.GetUserId(), model.OldPassword,
model.NewPassword);
    if (result.Succeeded)
    {
        var user = await UserManager.FindByIdAsync(User.Identity.GetUserId());
        //if (user != null)
        //{

```

```

// await SignInManager.SignInAsync(user, isPersistent: false, rememberBrowser: false);
//}

if (UserManager.IsInRole(user.Id, UserRoles.Admin))
{
    return RedirectToAction("Index", "Admin", new { Message = "Password has been Reset" });
}
else if (UserManager.IsInRole(user.Id, UserRoles.Doctor))
{
    return RedirectToAction("Index", "Doctor", new { Message = "Password has been Reset" });
}
return RedirectToAction("Index", "Patient", new { Message = "Password has been Reset" });
}
AddErrors(result);
ViewBag.Messege = "Password or Something went Wrong";
return View(model);
}

//
// GET: /Manage/SetPassword
public ActionResult SetPassword()
{
    return View();
}

//
// POST: /Manage/SetPassword
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<ActionResult> SetPassword(SetPasswordViewModel model)
{
    if (ModelState.IsValid)
    {
        var result = await UserManager.AddPasswordAsync(User.Identity.GetUserId(), model.NewPassword);
        if (result.Succeeded)
        {
            var user = await UserManager.FindByIdAsync(User.Identity.GetUserId());
            if (user != null)
            {
                await SignInManager.SignInAsync(user, isPersistent: false, rememberBrowser: false);
            }
            return RedirectToAction("Index", new { Message = ManageMessageId.SetPasswordSuccess });
        }
        AddErrors(result);
    }

    // If we got this far, something failed, redisplay form
    return View(model);
}

//
// GET: /Manage/ManageLogins
public async Task<ActionResult> ManageLogins(ManageMessageId? message)
{

```

```

ViewBag.StatusMessage =
    message == ManageMessageId.RemoveLoginSuccess ? "The external login was removed."
    : message == ManageMessageId.Error ? "An error has occurred."
    : "";
var user = await UserManager.FindByIdAsync(User.Identity.GetUserId());
if (user == null)
{
    return View("Error");
}
var userLogins = await UserManager.GetLoginsAsync(User.Identity.GetUserId());
var otherLogins = AuthenticationManager.GetExternalAuthenticationTypes().Where(auth => userLogins.All(ul
=> auth.AuthenticationType != ul.LoginProvider)).ToList();
ViewBag.ShowRemoveButton = user.PasswordHash != null || userLogins.Count > 1;
return View(new ManageLoginsViewModel
{
    CurrentLogins = userLogins,
    OtherLogins = otherLogins
});
}

//
// POST: /Manage/LinkLogin
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult LinkLogin(string provider)
{
    // Request a redirect to the external login provider to link a login for the current user
    return new AccountController.ChallengeResult(provider, Url.Action("LinkLoginCallback", "Manage"),
User.Identity.GetUserId());
}

//
// GET: /Manage/LinkLoginCallback
public async Task<ActionResult> LinkLoginCallback()
{
    var loginInfo = await AuthenticationManager.GetExternalLoginInfoAsync(XsrfKey, User.Identity.GetUserId());
    if (loginInfo == null)
    {
        return RedirectToAction("ManageLogins", new { Message = ManageMessageId.Error });
    }
    var result = await UserManager.AddLoginAsync(User.Identity.GetUserId(), loginInfo.Login);
    return result.Succeeded ? RedirectToAction("ManageLogins") : RedirectToAction("ManageLogins", new {
Message = ManageMessageId.Error });
}

protected override void Dispose(bool disposing)
{
    if (disposing && _userManager != null)
    {
        _userManager.Dispose();
        _userManager = null;
    }

    base.Dispose(disposing);
}

```

```
}
```

#### **#region Helpers**

```
// Used for XSRF protection when adding external logins
```

```
private const string XsrfKey = "XsrfId";
```

```
private IAuthenticationManager AuthenticationManager
```

```
{
```

```
    get
```

```
    {
```

```
        return HttpContext.GetOwinContext().Authentication;
```

```
    }
```

```
}
```

```
private void AddErrors(IdentityResult result)
```

```
{
```

```
    foreach (var error in result.Errors)
```

```
    {
```

```
        ModelState.AddModelError("", error);
```

```
    }
```

```
}
```

```
private bool HasPassword()
```

```
{
```

```
    var user = UserManager.FindById(User.Identity.GetUserId());
```

```
    if (user != null)
```

```
    {
```

```
        return user.PasswordHash != null;
```

```
    }
```

```
    return false;
```

```
}
```

```
private bool HasPhoneNumber()
```

```
{
```

```
    var user = UserManager.FindById(User.Identity.GetUserId());
```

```
    if (user != null)
```

```
    {
```

```
        return user.PhoneNumber != null;
```

```
    }
```

```
    return false;
```

```
}
```

```
public enum ManageMessageId
```

```
{
```

```
    AddPhoneSuccess,
```

```
    ChangePasswordSuccess,
```

```
    SetTwoFactorSuccess,
```

```
    SetPasswordSuccess,
```

```
    RemoveLoginSuccess,
```

```
    RemovePhoneSuccess,
```

```
    Error
```

```
}
```

#### **#endregion**

```
}  
}
```

## **Patient Controller :**

```
using System;  
using System.Collections.Generic;  
using System.Data.Entity;  
using System.Linq;  
using System.Web;  
using System.Web.Mvc;  
using Hospital_Management_System.CollectionViewModels;  
using Hospital_Management_System.Models;  
using Microsoft.AspNet.Identity;  
  
namespace Hospital_Management_System.Controllers  
{  
    public class PatientController : Controller  
    {  
        private ApplicationDbContext db;  
  
        //Constructor  
        public PatientController()  
        {  
            db = new ApplicationDbContext();  
        }  
  
        //Destructor  
        protected override void Dispose(bool disposing)  
        {  
            db.Dispose();  
        }  
  
        [Authorize(Roles = "Patient")]  
        public ActionResult Index(string message)  
        {  
            ViewBag.Messege = message;  
            string user = User.Identity.GetUserId();  
            var patient = db.Patients.Single(c => c.ApplicationUserId == user);  
            var date = DateTime.Now.Date;  
            var model = new CollectionOfAll  
            {  
                Ambulances = db.Ambulances.ToList(),  
                Departments = db.Department.ToList(),  
                Doctors = db.Doctors.ToList(),  
                Patients = db.Patients.ToList(),  
                Medicines = db.Medicines.ToList(),  
                ActiveAppointments = db.Appointments.Where(c => c.Status).Where(c => c.PatientId == patient.Id).Where(c  
=> c.AppointmentDate >= date).ToList(),  
                PendingAppointments = db.Appointments.Where(c => c.Status == false).Where(c => c.PatientId ==
```

```

patient.Id).Where(c => c.AppointmentDate >= date).ToList(),
    AmbulanceDrivers = db.AmbulanceDrivers.ToList(),
    Announcements = db.Announcements.Where(c => c.AnnouncementFor == "Patient").ToList()
};
return View(model);
}

```

**//Update Patient profile**

**[Authorize(Roles = "Patient")]**

**public ActionResult UpdateProfile(string id)**

```

{
    var patient = db.Patients.Single(c => c.ApplicationUserId == id);
    return View(patient);
}

```

**[HttpPost]**

**[ValidateAntiForgeryToken]**

**public ActionResult UpdateProfile(string id, Patient model)**

```

{
    var patient = db.Patients.Single(c => c.ApplicationUserId == id);
    patient.FirstName = model.FirstName;
    patient.LastName = model.LastName;
    patient.FullName = model.FirstName + " " + model.LastName;
    patient.Contact = model.Contact;
    patient.Address = model.Address;
    patient.BloodGroup = model.BloodGroup;
    patient.DateOfBirth = model.DateOfBirth;
    patient.Gender = model.Gender;
    patient.PhoneNo = model.PhoneNo;
    db.SaveChanges();
    return View();
}

```

**//Start Appointment Section**

**//Add Appointment**

**[Authorize(Roles = "Patient")]**

**public ActionResult AddAppointment()**

```

{
    var collection = new AppointmentCollection
    {
        Appointment = new Appointment(),
        Doctors = db.Doctors.ToList()
    };
    return View(collection);
}

```

**[HttpPost]**

**[ValidateAntiForgeryToken]**

**public ActionResult AddAppointment(AppointmentCollection model)**

```

{
    var collection = new AppointmentCollection
    {

```



```

        Appointment = model.Appointment,
        Doctors = db.Doctors.ToList()
    };
    if (model.Appointment.AppointmentDate >= DateTime.Now.Date)
    {
        string user = User.Identity.GetUserId();
        var patient = db.Patients.Single(c => c.ApplicationUserId == user);
        var appointment = new Appointment();
        appointment.PatientId = patient.Id;
        appointment.DoctorId = model.Appointment.DoctorId;
        appointment.AppointmentDate = model.Appointment.AppointmentDate;
        appointment.Problem = model.Appointment.Problem;
        appointment.Status = false;

        db.Appointments.Add(appointment);
        db.SaveChanges();
        return RedirectToAction("ListOfAppointments");
    }
    ViewBag.Messege = "Please Enter the Date greater than today or equal!!";

    return View(collection);
}

//List of Appointments
[Authorize(Roles = "Patient")]
public ActionResult ListOfAppointments()
{
    string user = User.Identity.GetUserId();
    var patient = db.Patients.Single(c => c.ApplicationUserId == user);
    var appointment = db.Appointments.Include(c => c.Doctor).Where(c => c.PatientId == patient.Id).ToList();
    return View(appointment);
}

//Edit Appointment
[Authorize(Roles = "Patient")]
public ActionResult EditAppointment(int id)
{
    var collection = new AppointmentCollection
    {
        Appointment = db.Appointments.Single(c => c.Id == id),
        Doctors = db.Doctors.ToList()
    };
    return View(collection);
}

[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult EditAppointment(int id, AppointmentCollection model)
{
    var collection = new AppointmentCollection
    {
        Appointment = model.Appointment,
        Doctors = db.Doctors.ToList()
    };

```

```

    };
    if (model.Appointment.AppointmentDate >= DateTime.Now.Date)
    {
        var appointment = db.Appointments.Single(c => c.Id == id);
        appointment.DoctorId = model.Appointment.DoctorId;
        appointment.AppointmentDate = model.Appointment.AppointmentDate;
        appointment.Problem = model.Appointment.Problem;
        db.SaveChanges();
        return RedirectToAction("ListOfAppointments");
    }
    ViewBag.Messege = "Please Enter the Date greater than today or equal!!";

    return View(collection);
}

//Delete Appointment
[Authorize(Roles = "Patient")]
public ActionResult DeleteAppointment(int? id)
{
    var appointment = db.Appointments.Single(c => c.Id == id);
    return View(appointment);
}

[HttpPost, ActionName("DeleteAppointment")]
[ValidateAntiForgeryToken]
public ActionResult DeleteAppointment(int id)
{
    var appointment = db.Appointments.Single(c => c.Id == id);
    db.Appointments.Remove(appointment);
    db.SaveChanges();
    return RedirectToAction("ListOfAppointments");
}

//End Appointment Section

//Start Doctor Section

//List of Available Doctors
[Authorize(Roles = "Patient")]
public ActionResult AvailableDoctors()
{
    var doctor = db.Doctors.Include(c => c.Department).Where(c => c.Status == "Active").ToList();
    return View(doctor);
}

//Show Doctor Schedule
[Authorize(Roles = "Patient")]
public ActionResult DoctorSchedule(int id)
{
    var schedule = db.Schedules.Include(c => c.Doctor).Single(c => c.DoctorId == id);
    if (schedule != null)
    {
    }
}

```

```

    return View(schedule);
}

//Doctor Detail
[Authorize(Roles = "Patient")]
public ActionResult DoctorDetail(int id)
{
    var doctor = db.Doctors.Include(c => c.Department).Single(c => c.Id == id);
    return View(doctor);
}

//End Doctor Section

//Start Complaint Section

[Authorize(Roles = "Patient")]
public ActionResult AddComplain()
{
    return View();
}

[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult AddComplain(Complaint model)
{
    var complain = new Complaint();
    complain.Complain = model.Complain;
    complain.ComplainDate = DateTime.Now.Date;
    db.Complaints.Add(complain);
    db.SaveChanges();
    return RedirectToAction("ListOfComplains");
}

[Authorize(Roles = "Patient")]
public ActionResult ListOfComplains()
{
    var complain = db.Complaints.ToList();
    return View(complain);
}

[Authorize(Roles = "Patient")]
public ActionResult EditComplain(int id)
{
    var complain = db.Complaints.Single(c => c.Id == id);
    return View(complain);
}

[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult EditComplain(int id, Complaint model)
{
    var complain = db.Complaints.Single(c => c.Id == id);
    complain.Complain = model.Complain;
    db.SaveChanges();
}

```

```

        return RedirectToAction("ListOfComplains");
    }

    [Authorize(Roles = "Patient")]
    public ActionResult DeleteComplain()
    {
        return View();
    }

    [HttpPost, ActionName("DeleteComplain")]
    [ValidateAntiForgeryToken]
    public ActionResult DeleteComplain(int id)
    {
        var complain = db.Complaints.Single(c => c.Id == id);
        db.Complaints.Remove(complain);
        db.SaveChanges();
        return RedirectToAction("ListOfComplains");
    }

    //End Complain Section

    //Start Prescription Section

    //List of Prescription
    [Authorize(Roles = "Patient")]
    public ActionResult ListOfPrescription()
    {
        string user = User.Identity.GetUserId();
        var patient = db.Patients.Single(c => c.ApplicationUserId == user);
        var prescription = db.Prescription.Include(c => c.Doctor).Where(c => c.PatientId == patient.Id).ToList();
        return View(prescription);
    }

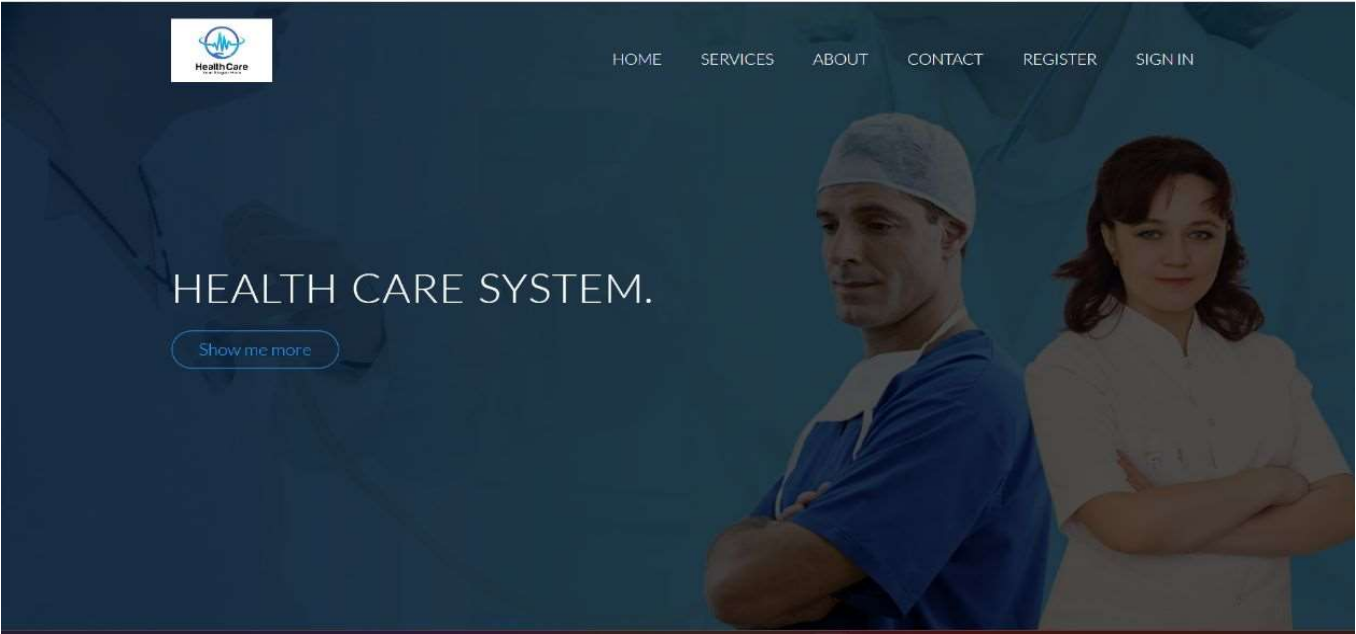
    //Prescription View
    public ActionResult PrescriptionView(int id)
    {
        var prescription = db.Prescription.Single(c => c.Id == id);
        return View(prescription);
    }

    //End Prescription Section
}
}

```

## SAMPLE SCREENSHOTS

**Screens:**  
**Home Page:**



This screen shows the basic view of the application home page and the list of modules.

**Admin Login Page:**

## LOG IN.

A login form with a light gray background and rounded corners. It contains two input fields: 'User Name\*' with 'UserName' as placeholder text, and 'Password\*' with 'Password' as placeholder text. Below the password field is a red error message: 'The Password field is required.' There is a checkbox labeled 'Remember me?'. Below the checkbox are two links: 'Forgot your password?' and 'Register as a new user'. At the bottom is a blue 'Log in' button.

User Name\*

UserName

Password\*

Password

The Password field is required.

☐ Remember me?

[Forgot your password?](#)

[Register as a new user](#)

Log in

[Back to Home](#)

In this page, admin can log in by giving username and password.

## Admin Dashboard:

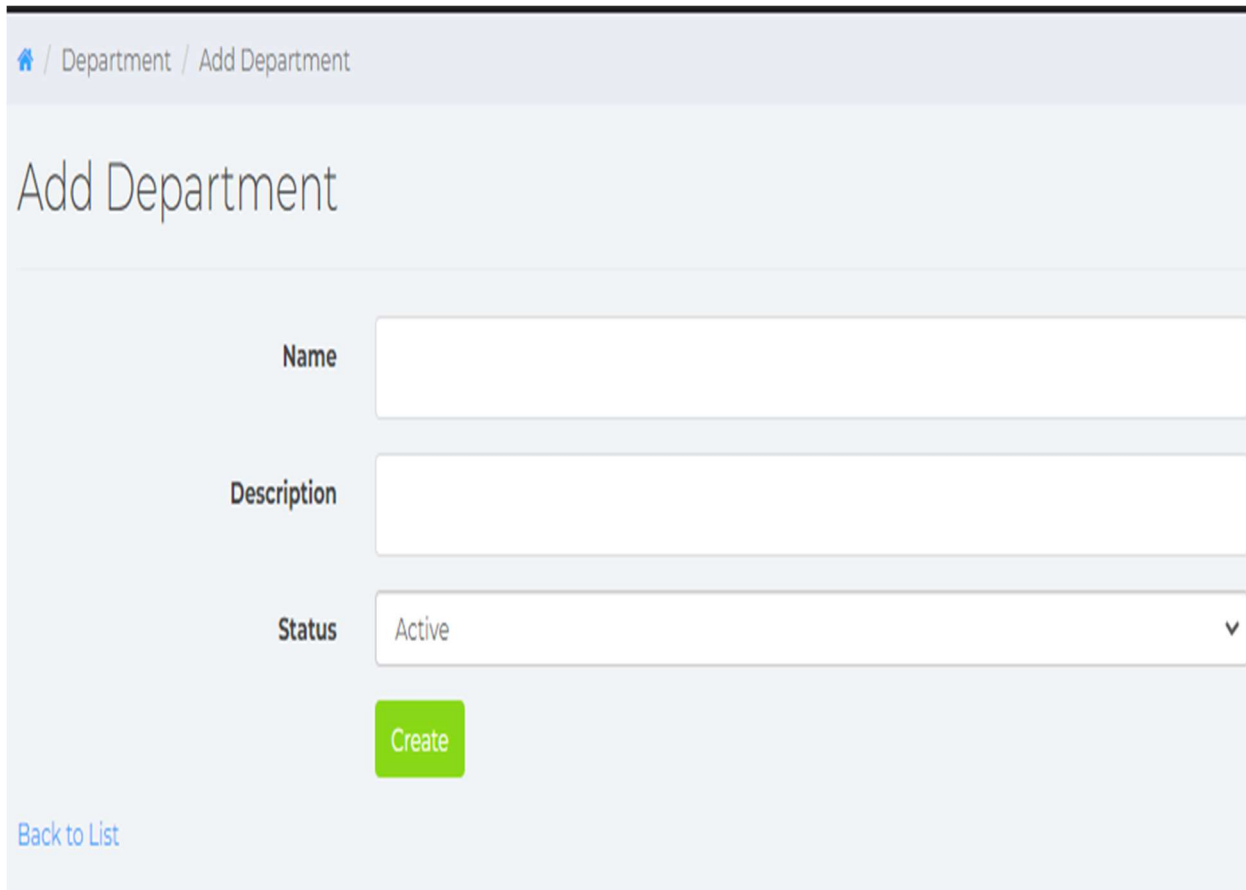
The Admin Dashboard is shown with a dark header bar labeled 'DASHBOARDADMIN'. On the left is a sidebar with a user profile for 'Usman123' (ADMIN) and a list of menu items: Dashboard, Department, Doctor, Patient, Schedule, Announcement, Appointment, Complaint, Medicine, Ambulance, and Manage. A 'Logout' button is at the bottom of the sidebar. The main content area is titled 'Dashboard' and displays eight statistics in a grid:

Icon	Value	Category
Building	10	DEPARTMENTS
Person	2	DOCTORS
Person	6	PATIENTS
Ambulance	2	AMBULANCES
Calendar	0	ACTIVE APPOINTMENTS
Checkmark	1	PENDING APPOINTMENTS
Medicine bottle	1	MEDICINES
Car	8	AMBULANCE DRIVERS

After successful login, the application shows the admin dashboard page in which the basic functionalities are shown.

**Department :**

- A department is one of the sections in an organization such as a government, business, or Hospital
- Admin can add / edit / view / delete the Department
- The department will show to their respective employee like doctor.



The screenshot shows a web application interface for adding a new department. At the top, there is a breadcrumb navigation bar with a home icon, followed by "/ Department / Add Department". Below this, the main heading "Add Department" is displayed. The form consists of three input fields: "Name", "Description", and "Status". The "Status" field is a dropdown menu currently showing "Active". Below the form fields is a green "Create" button. In the bottom left corner, there is a blue link labeled "Back to List".

Home / Department / Add Department

## Add Department

Name

Description

Status

Create

[Back to List](#)

**Doctor:**

- A person who has been trained in medicine and who treats people who are ill.
- Admin can add / edit / delete the Doctor
- The doctor will show to their respective patient.

### Add Doctor

User Name	<input type="text"/>
First Name	<input type="text"/>
Last Name	<input type="text"/>
Email	<input type="text"/>
Phone No	<input type="text"/>
Mobile No	<input type="text"/>
Blood Group	<input type="text" value="Select Blood Group"/>
Gender	<input type="text" value="Select Gender"/>
Date of Birth	<input type="text" value="dd-mm-yyyy"/>
Designation	<input type="text"/>
Department	<input type="text" value="Select Department"/>
Specialization	<input type="text"/>
Education/Degree	<input type="text"/>
Status	<input type="text" value="Select Status"/>
Password	<input type="password"/>
Confirm password	<input type="password"/>

[Create](#)

### Patient:

- Admin can only view the list of patient.
- Admin can edit / delete the information of patient

### List Of Patients

Show  entries

Search:

Name	Contact	Blood Group	Gender	Action
				<a href="#">Edit</a> <a href="#">Delete</a>
				<a href="#">Edit</a> <a href="#">Delete</a>
Khushboo Soni	7379000000	B+	Female	<a href="#">Edit</a> <a href="#">Delete</a>
Sajid Nabeel	343434343	A+	Male	<a href="#">Edit</a> <a href="#">Delete</a>
Smith John	3076449398	B+	Male	<a href="#">Edit</a> <a href="#">Delete</a>
Usman Khan	3076449398	B+	Male	<a href="#">Edit</a> <a href="#">Delete</a>

Showing 1 to 6 of 6 entries


Previous [1](#) Next

### Schedule:

- Admin can add / edit / view / delete the schedule.
- Admin can schedule the timing for the doctor, he will assign the start and end time.



- The schedule will show to their respective employee like doctor.

 / [Schedule](#) / [Add Schedule](#)

## Add Schedule

**Doctor Name**

**Start Day**

**End Day**

**Start Time**

**End Time**

**Per Patient Time**

**Status**

Create

[Back to List](#)

### Announcement:

- Announcement will apply to their staff and employee, like notice
- Admin can add / edit / view / delete the announcement
- Announcement will notify that the announcement goes to whom.

## Add Announcement

Announcement

Announcement For

Select ▼

End Date

dd-mm-yyyy 📅

Create

[Back to List](#)

### View Doctor List:

DASHBOARDADMIN

Usman123  
ADMIN

[DashBoard](#)

[Department](#)

[Doctor](#)

[Add Doctor](#)

[List of Doctor](#)

[Patient](#)

[Schedule](#)

[Announcement](#)

[Appointment](#)

[Complaint](#)

[Medicine](#)

[Ambulance](#)

[Manage](#)

localhost:61243/Admin/ListOfDoctors

[Doctor](#) / [List of Doctors](#)

### List Of Doctors

[Add Doctor](#)

Show 10 entries

Search:

Name	Department	Designation	Mobile No	Gender	Education/Degree	Status	Action
Dr. Adeel Safdar	Cardiology	19 Scale	03656665655	Male	MBBS(K.E)	Active	<a href="#">Edit</a> <a href="#">Delete</a>
Dr. Umar Farooq	Bone	19 Scale	03476156525	Male	MBBS(K.E)	Active	<a href="#">Edit</a> <a href="#">Delete</a>

Showing 1 to 2 of 2 entries

[Previous](#) [1](#) [Next](#)

In this page, admin can view the list of registered doctors and admin can edit/delete the doctor.

### View Patient List:

In this page, admin can view the list of registered patients and take edit/delete functionality.

DASHBOARDADMIN

Usman123

ADMIN

Dashboard

Department

Doctor

Patient

Schedule

Announcement

Appointment

Complaint

Medicine

Ambulance

Manage

Logout

Patient / List of Patients

List Of Patients

Show 10 entries

Search:

Name	Contact	Blood Group	Gender	Action
Khushboo Sori	73790000000	B+	Femiale	<a href="#">Edit</a> <a href="#">Delete</a>
Sajid Nabeel	343434343	A+	Male	<a href="#">Edit</a> <a href="#">Delete</a>
Smith John	3076449396	B+	Male	<a href="#">Edit</a> <a href="#">Delete</a>
Usman Khan	3076449396	B+	Male	<a href="#">Edit</a> <a href="#">Delete</a>

Showing 1 to 6 of 6 entries

Previous 1 Next

## Add Appointment Page:

DASHBOARDADMIN

Usman123

ADMIN

Dashboard

Department

Doctor

Patient

Schedule

Announcement

Appointment

Complaint

Medicine

Ambulance

Manage

Logout

Appointment / Add Appointment

Add Appointment

Patient Name

Select Patient

Doctor Name

Select Doctor

Appointment Date

dd-mm-yyyy

Problem

Status

☐

Create

Back to List

In this page admin can add the appointment of patient with corresponding doctor.

## Active Appointment Page:

In this page show the active appointment list.

DASHBOARDADMIN

Usman123

ADMIN

Dashboard

Department

Doctor

Patient

Schedule

Announcement

Appointment

Complaint

Medicine

Ambulance

Manage

Logout

Appointment / Active Appointments

Active Appointments

Show 10 entries

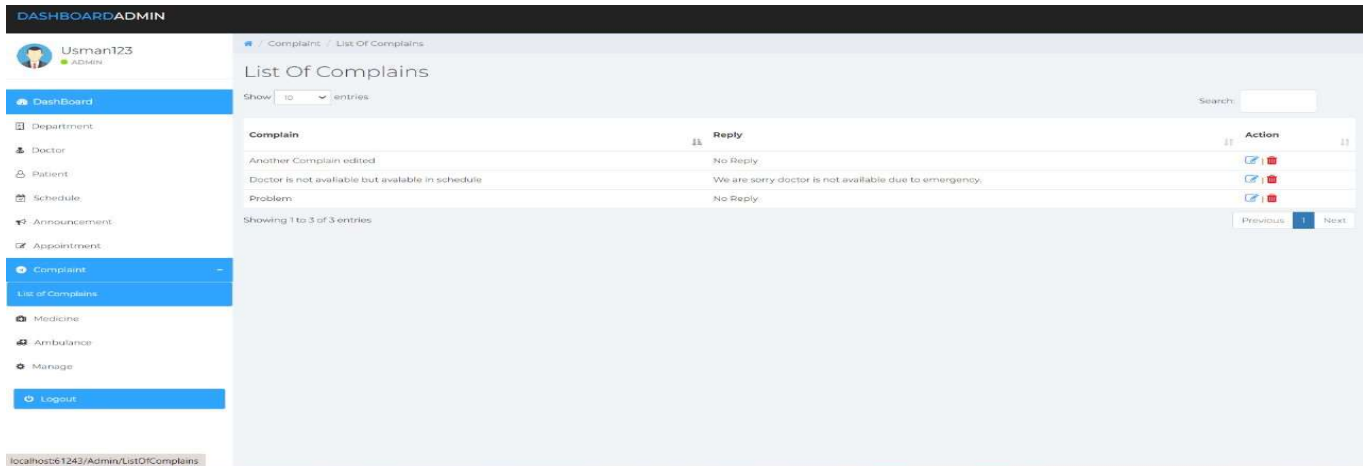
Search:

Doctor Name	Patient Name	Appointment Date	Problem	Status	Action
No data available in table					

Showing 0 to 0 of 0 entries

Previous Next

## List Of Complains:



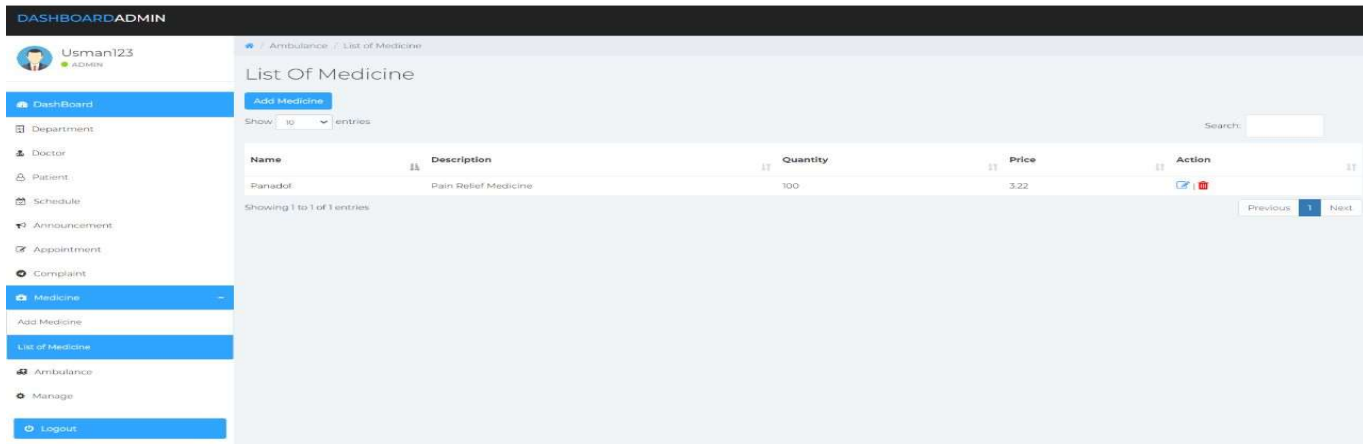
In this page admin can show list of complaints and complain edit and delete functionality.

## Add Medicine:



In this page admin can add medicine name, quantity and price of related medicine.

## List Of Medicine:



In this page admin can show the medicine list and it can be edit/deleted by the admin.

## Manage Password :

# CHANGE PASSWORD.

Old Password\*

Old Password

New Password\*

New Password

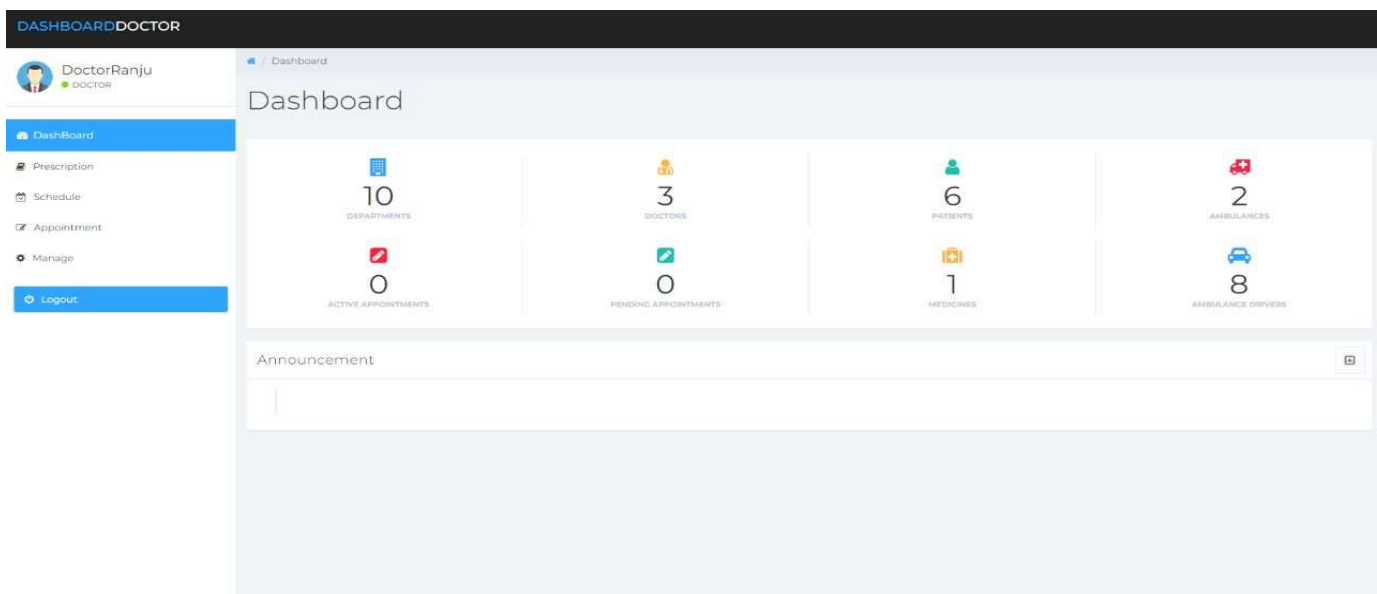
Confirm Password\*

Confirm password

Change Password

- Manage is just like the Setting of password
- Admin can their password.
- Manage (Setting) page is same for –
  1. Patient
  2. Doctor
  3. Admin

## Doctor Dashboard:



In this page after the doctor login show the doctor dashboard and show the related data on its dashboard.

## Add Prescription by doctor:

In this page doctor can add prescription.

- Doctor will create the Prescription (status of patient's), like –
  1. how many test
  2. how much medicine will take patient

3. needs to be checked up after number of days

- Doctor can add / edit / view / delete the Prescription
- Prescription will show into patient dashboard.

The screenshot shows the 'Add Prescription' form in the DoctorRanju dashboard. The form includes a 'Patient Name' dropdown menu, a 'Medical Tests' section with four input fields, and a 'Medicine' section with eight rows. Each row contains a 'Medicine' input field and a frequency dropdown menu with options: 'Morning', 'Afternoon', and 'Evening'. At the bottom of the form, there is a 'Frequency After Days' input field and a 'Save' button.

## List Of Prescription:

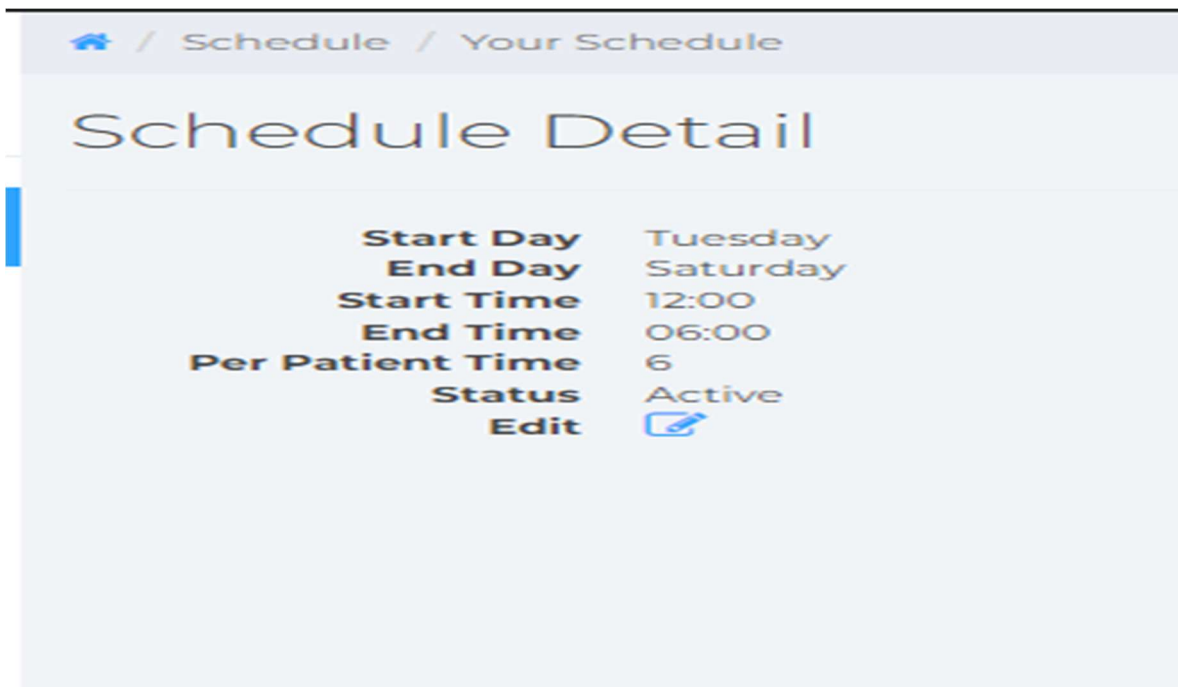
The screenshot shows the 'List Of Prescription' table in the DoctorRanju dashboard. The table has four columns: 'Patient Username', 'Patient Name', 'Date', and 'Action'. The table is currently empty, displaying 'Showing 0 to 0 of 0 entries'. There are 'Previous' and 'Next' buttons at the bottom right of the table.

Patient Username	Patient Name	Date	Action
Showing 0 to 0 of 0 entries			

In this page doctor can show the prescription list and it can be taken edit/delete functionality.

## Schedule :

- The Schedule details created by Admin to doctor
- Doctor can edit / view his own Schedule details
- The Schedule will decide from which date - to date doctor will work for patient



### Add Appointment by Doctor:

- Doctor can add on which date the appointment will be with which patient
- Doctor can add / edit / view / delete the appointment
- Doctor can see that which appointment is pending and who to work for.

DASHBOARDDOCTOR

DoctorRanju

doctor

DashBoard

Prescription

Schedule

Appointment

Add Appointment

Active Appointments

Pending Appointments

Manage

Logout

AppointmentAdd Appointment

Add Appointment

Patient NameSelect Patient

Appointment Datedd-mm-yyyy

Problem

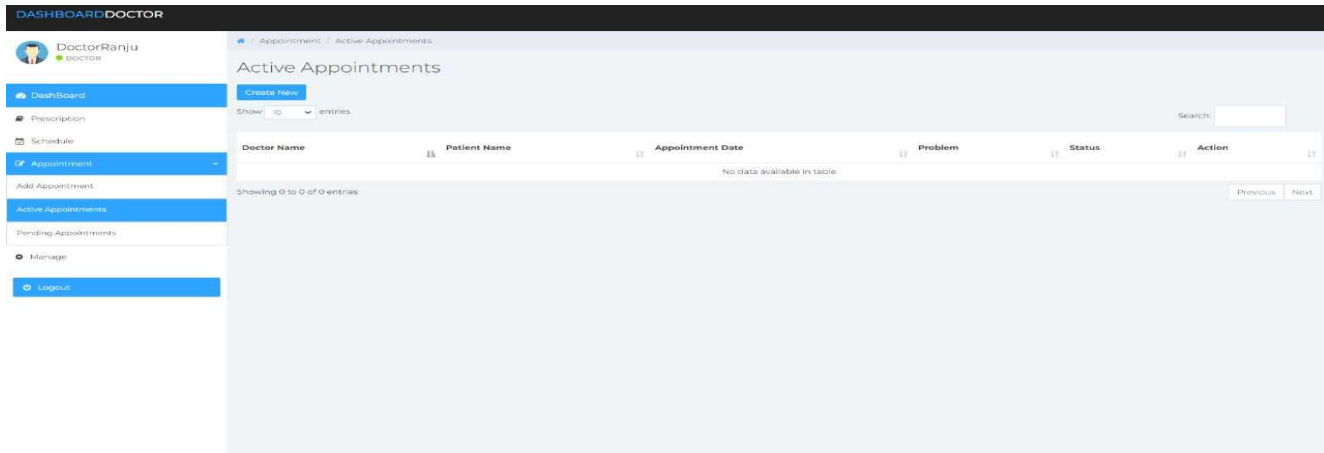
Status☐

Create

Back to List

In this page doctor can add the Appointment to the patient.

### Active Appointment of Doctor:



In this page show the active appointment of this doctor.

### Patient self-register page:

REGISTER.

**User Name\***  
User Name

**First Name\***  
First Name

**Last Name\***  
Last Name

**Email\***  
Your Email

**Password\***  
Password

**Password\***  
Confirm Password

[Already a user?](#)

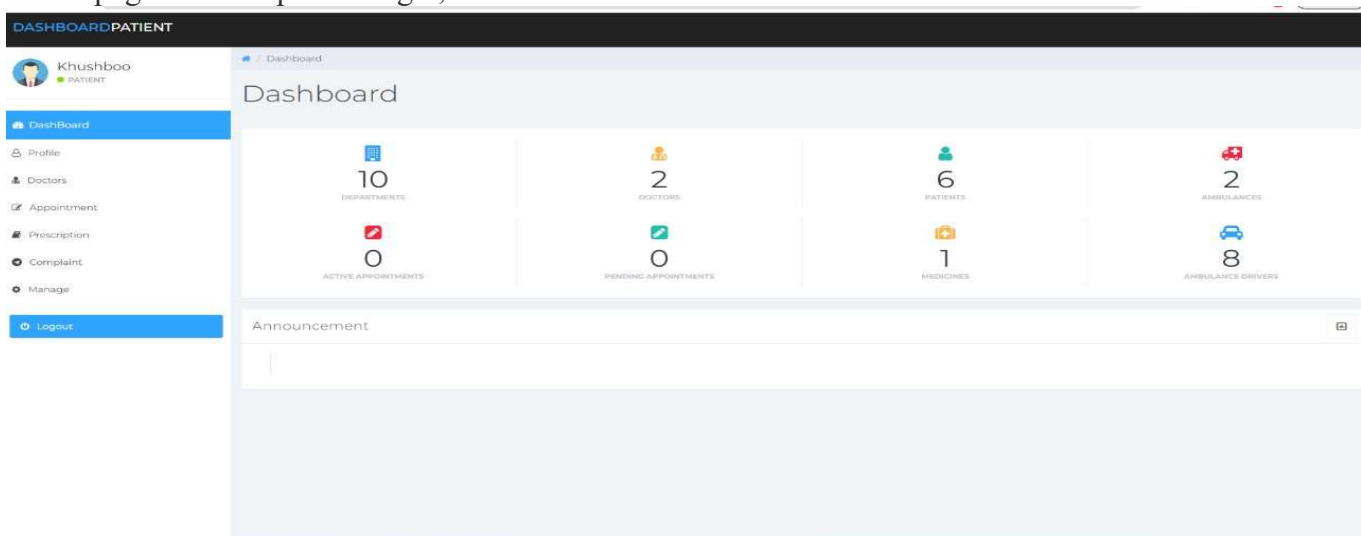
[Register](#)

[Back to Home](#)

In this page the patient can self-register and create with own username and password then after patient can login through login page.

### Patient Dashboard:

In this page after the patient login, he can show the own dashboard.



### Patient Update Profile:

- Patient can update his own profile on the Dashboard.
- Patient can edit / view his details.



**DASHBOARD PATIENT**

Khushboo PATIENT

Profile / Update Profile

### Update Profile

First Name:

Last Name:

Phone No:

Contact:

Blood Group:

Gender:

Date of Birth:

Address:

[Update Profile](#)

localhost61243/Patient/UpdateProfile/900c2ea8-ee05-459c-aba4-8a8c984492c

In this page patient can update your profile details.

### Appointment:

- Patient can book his appointment to specified doctors.
- Patient can see the requested appointment  
The appointment will not until doctor will not approved the status.
- Patient can add/ edit / view / delete the created appointment.

Appointment / Add Appointment

## Add Appointment

Doctor Name:

Appointment Date:

Problem:

[Request Appointment](#)

[Back to List](#)

### Available Doctor:

- Patient will see the doctor details like – Schedule time, doctor info
- After confirmation the details patient can book his appointment to specified doctors.
- When Doctor will approve the appointment the patient can do the checkup.

DASHBOARDPATIENT

Khushboo

PATIENT

Dashboard

Profile

Doctors

Available Doctors

Appointment

Prescription

Complaint

Manage

Logout

Doctor / Available Doctor

Available Doctors

Show 10 entries

Search:

Name	Department	Designation	Mobile No	Gender	Education/Degree	Status	Action
Dr. Adeel Safdar	Cardiology	19 Scale	03656556655	Male	MBBS(K.E)	Active	
Dr. Umar Farooq	Bone	19 Scale	03476156525	Male	MBBS(K.E)	Active	

Showing 1 to 2 of 2 entries

Previous

1

Next

In this page patient can show the available doctor list which can take appointment.

### List of Prescription:

- Doctor will the Prescription for patient.
- Patient can see the Prescription(Total checkup and medicine Information) given by doctor.

Prescription / List of Prescriptions

List Of Prescriptions

Show 10 entries

Search:

Doctor Name	Date	Action
No data available in table		

Showing 0 to 0 of 0 entries

Previous

Next

### List of Appointments:

DASHBOARD PATIENT

Khushboo

PATIENT

Dashboard

Profile

Doctors

Appointment

Request Appointment

Your Appointments

Prescription

Complaint

Manage

Logout

Appointment / List of Appointment

List Of Appointments

Description:

1. You Have to wait until your Appointment is Approved.  
2. If Status is Checked then your Appointment is Done with Doctor.  
3. If Appointment is not Approved then Again Request for Appointment.

Create New

Show 10 entries

Search:

Doctor Name	Appointment Date	Problem	Appointment Id	Status	Action
Dr. Umar Farooq	2025-03-31	teeth problem	Not Approved		<a href="#">Edit</a> <a href="#">Delete</a>

Showing 1 to 1 of 1 entries

Previous

1

Next

In this page patient show the appointment list which have taken from more than one doctor.

### Add Complain by Patient:

DASHBOARD PATIENT

Khushboo

PATIENT

Dashboard

Profile

Doctors

Appointment

Prescription

Complaint

Add Complaint

List of Complaints

Manage

Logout

Complaint / Add Complain

Add Complain

Complain

Submit Complain

Back to List

In this page patient can submit own complain to the doctor and doctor give the response of this complain.

# RESULTS

## CONCLUSION

The conclusion of a healthcare system can vary depending on the specific aspects being discussed. However, here are a few general points that can be considered when drawing a conclusion about healthcare systems:

1. **Vital Importance:** Healthcare systems are of paramount importance in any society as they play a crucial role in promoting and maintaining the health and well-being of individuals. Access to quality healthcare services is essential for the overall development and prosperity of a nation.
2. **Complex and Interconnected:** Healthcare systems are intricate and involve a multitude of components, including healthcare providers, facilities, medical technology, insurance systems, and government regulations. These components are interconnected and rely on each other to ensure the delivery of effective and efficient healthcare.
3. **Challenges and Opportunities:** Healthcare systems face numerous challenges such as rising costs, unequal access to care, shortage of healthcare professionals, and evolving healthcare needs. However, these challenges also present opportunities for innovation, research, and advancements in medical technology to improve healthcare delivery and outcomes.
4. **Patient-Centred Approach:** A patient-centred approach is increasingly emphasised in healthcare systems. The focus is shifting towards personalised care, shared decision-making, and improving patient experience and satisfaction. Ensuring patient empowerment and involvement in their own healthcare is becoming a priority.
5. **Integration of Technology:** Healthcare systems are incorporating technology at various levels to enhance efficiency, accuracy, and accessibility. Electronic health records, telemedicine, wearable devices, and artificial intelligence are revolutionising healthcare delivery, diagnostics, and monitoring.
6. **Collaborative Efforts:** Collaboration among healthcare providers, researchers, policymakers, and other stakeholders is vital for the success of a healthcare system. Cooperation and coordination are essential to address complex healthcare challenges, share best practices, and ensure the equitable distribution of resources.
7. **Ongoing Evolution:** Healthcare systems are continuously evolving to adapt to changing demographics, medical advancements, and societal needs. Flexibility and adaptability are crucial to meet the evolving healthcare demands and provide comprehensive, high-quality care to individuals across different populations.