SOFTWARE ENGINEERING

Project

Group Members:

Roshan-e- Asif	F2022065051
Malaika Shaukat	F2022065259
Sohaib	F2022065048

Instructor:

Ms. Halima Jamil

MODELING REQUIREMENTS ENGINEERING FOR HOSPITAL MANAGEMENT SYETEM

Functional Specification

All Versions **Version 3.1**

1.1 2.1 3.1
1.2 2.2
1.3

Software Requirements Specification

Introduction

This document completely specifies the committed software requirements for HOSPITAL MANAGEMENT SYSTEM, an online registration system. The aim of this project is to study and analyze the system which is going to run in a hospital. On the basis of the analysis performed our goal is to develop a requirements specification document that supports all the functional and non-functional requirements with improvements suggested for the current deficiencies.

Purpose

The purpose of the software requirements specification document is to specify all requirements for the current registration system as well as those requirements that are suggested as improvements for the current system. . The document explains the information that will be supplied as input to the system, its transformations and the required outputs. It also addresses the interactions between the desired system and its users. This document will also act as an aide for the upcoming object oriented analysis and design of the system. This will help the software designers in developing this system in accordance with the requirements given in this specification. This specification describes all functional and non-functional requirements, constraints, and other factors necessary to provide a complete and comprehensive description of the requirements necessary to design and develop the corresponding software systems

The software developers will use the document for the necessary understanding of the system when implementing and designing. The other concerned person is the client who would be able to understand the attributes and functions of the system being developed.

Scope

The scope of this document is to completely and correctly specify software requirements for the current registration system and other requirements that have come up as improvements suggested to be incorporated in the existing system. The following areas are comprehensively covered in the document

- Functional Requirements
- Non-Functional Requirements
- Actors
- Use Case Description
- Use Case Diagram
- Prototype
- Sequence diagram
- Data Flow Diagram (DFD)
- Activity diagram
- Code

Table of Contents

Introduction 3

Purpose 3

Scope 3

Functional Requirements 7

FR01: Patient portal Creation 7

FR02: Doctor Directory 7

FR03: Billing and Invoicing: 7

FR04: Electronic Health Records (EHR): 7

FR05: Staff Management: 8

FR06: Emergency Management: 8

FR07: Admission and Discharge: 8

FR9: Inventory Management: 9

FR10: Pharmacy Management: 9

Non-Functional Requirements 9

NFR01: Performance: 9

NFR02: Security: 10

NFR03: Reliability: 10

NFR04: Scalability: 10

NFR05: Usability: 11

4-Actors 11

4.1 Admin 11

4.2 Doctor 11

4.3 Patient 11

4.4 Manager 11

4.4.1 Inventory manager 11

4.4.2 Pharmacy manager 11

4.4.3 Laboratory manager 12
5-Use-Cases 12
5.1 UC01: Patient Registration 12
5.2 UC01: Communication with Hospital Faculty 12
5.3 UC01: An Option to Search 12
5.4 UC01: Schedule Appointment 12
5.5 UC01: View Scheduled Appointment 12
5.6 UC01: View Test Reports 13
5.6 UC01: Re-scheduling Appointments 13
5.7 UC01: Cancel Appointments 13
5.8 UC01: Pharmacy Record 13
5.9 UC01: Update Pharmacy Record 13
5.9 UC01: Bill information 13
5.10 UC01: Lab Test Reports 14
5.11 UC01: Update Lab Test Reports 14
5.12 UC01: Staff Assignment 14
5.13 UC01: Track employee 14
5.13 UC01: Database 14
5.14 UC01: Admission and Discharge 14
5.15 UC01: Allocate Rooms 15
5.16 UC01: Emergency Alerts 15
5.17 UC01: Emergency Alerts 15
5.18 UC01: Notify Appointments 15
5.19 UC01: Notify Reschedule 15
5.20 UC01: Low stock Alerts 15
5.21 UC01: Remove Inventory 16
5.22 UC01: Medicine Record 16

5.21 UC01: Remove Inventory 16

USE CASE DIAGRAM 17

Prototype 18

Sequence Diagram 21

Data Flow Diagram (DFD) 22

Code 24

Functional Requirements

FR01: Patient portal Creation

Req. No.	Functional Requirements
FR01-01	The system shall allow patient to schedule appointments by seding requeset to doctors via prtal.
FR01-02	The system shall allow patient to communicate with healthcare providers via email or phone.
FR01-03	The system shall be able to store comprehensive patient information, including personal details collect information and medical history.
FR01-04	The system shall generate a unique ID for each patient.

FR02: Doctor Directory

Req. No.	Functional Requirements
FR02-01	The system should be able to provide a feature to browse and search for doctors based on specialties, availability, and other relevant criteria and patient can discuss his/her problem via patient portal
FR02-02	The system should allow staff to schedule appointments for patients and automatically allocate rooms.
FR02-03	The system should provide a calendar view for easy scheduling and rescheduling of appointments.
FR02-04	The system should send reminders to patients and doctors about upcoming appointments
FR02-05	The system should allow doctors to reschedule appointments and notify the patient.

FR03: Billing and Invoicing:

Req. No.	Functional Requirements
FR03-01	The system should generate bills for services provided, including consultations, procedures, and medications.

FR04: Electronic Health Records (EHR):

Req. No.	Functional Requirements

FR04-01	The system should maintain electronic health records for each patient.
FR04-02	The system should record and update patient diagnoses, treatments, medications, and test results

FR05: Staff Management:

Req. No.	Functional Requirements
FR05-01	The system should maintain a comprehensive database of staff members, including doctors, nurses, and administrative staff.
FR05-02	The system should manage staff schedules and assignments
FR05-03	The system should track employee attendance and performance.

FR06: Emergency Management:

Req. No.	Functional Requirements
FR06-01	The system should support emergency response by providing quick access to critical patient information
FR06-02	The system should integrate with emergency services and systems

FR07: Admission and Discharge:

Req. No.	Functional Requirements
FR07-01	The system should efficiently manage the admission and discharge processes
FR07-02	The system should assign and track patient rooms and beds

FR08: Laboratory Reporting:

Req. No.	Functional Requirements
FR08-01	The system generate standard reports on patient statistics, revenue, and operational efficiency

FR08-01	The system should interface with laboratory and imaging systems to request and store test results
FR08-02	The system should associate test results with patient records for easy access by healthcare providers
FR08-03	The system should share laboratory reports online with patients

FR9: Inventory Management:

Req. No.	**Functional Requirements**
FR9-01	The system manage the inventory of medical supplies and equipment
FR9-02	The system should generate alerts for low stock levels and facilitate reordering
FR9-03	The system should track the usage of medical supplies

FR10: Pharmacy Management:

Req. No.	**Functional Requirements**
FR10-01	The system should manage pharmacy inventory and prescriptions
FR10-02	The system should track medication dispensing and maintain records of patient medications
FR10-03	The system should Integrate with the billing system for accurate billing of medications

Non-Functional Requirements

NFR01: Performance:

Req. No.	**Functional Requirements**
NFR01-01	The system should Ensure a response time of less than 5 seconds for page loading.

NfR01-02	The system support a specified number of concurrent users and transactions per second during peak times
NFR01-03	Average load time of the starting page of the system must be less than 2 second.

NFR02: Security:

Req. No.	**Functional Requirements**
NFR02-01	The system should encrypt all sensitive patient data during transmission and storage.
NfR02-02	The system be able to implement role-based access control to restrict access to specific functionalities and data
NFR02-03	No user can view data of any other user through any report or views provided by the system.

NFR03: Reliability:

Req. No.	**Functional Requirements**
NFR03-01	Ensure 24/7 system availability with planned downtimes communicated in advance
NFR03-02	Design the system to handle hardware failures or issues without complete breakdown (fault tolerance).

NFR04: Scalability:

Req. No.	**Functional Requirements**
NFR04-01	The system should be able to handle a scalable number of users and data as the hospital expands
NFR04-02	The system should allow for easy scaling of components to accommodate growing data volumes and increased concurrent user activity

NFR05: Usability:

Req. No.	Functional Requirements
NFR05-01	The system should be able to Maintain user interface consistency with standardized design principles and navigation patterns.
NFR05-02	The system should Ensure accessibility compliance (e.g., WCAG) for users with disabilities.

4-Actors

Following is a list of the actors and their responsibilities involved in the registration process.

4.1 Admin

Admin is monitoring system setups, controlling user roles and permissions, and guaranteeing regulatory standards are being followed. Administrators are in charge of managing user access and system configurations as well as the overall management of the healthcare system.

4.2 Doctor

Doctors are registering new patients, corresponding with existing patients via many channels, making and changing appointment schedules, accessing test results and booking visits, and maintaining patient information. In addition to being heavily involved in patient treatment, doctors are also crucial to the administration of healthcare.

4.3 Patient

Patients are registering on the website, corresponding with medical professionals, looking up doctor schedules and profiles, making and cancelling appointments, seeing test results and planned appointments, and getting appointment alerts. In order to schedule their medical appointments and obtain pertinent information, patients actively interact with the system.

4.4 Manager

4.4.1 Inventory manager

Inventory manager is keeping an eye on and controlling the drug inventory, making sure there are enough supplies on hand, sending out low stock alerts, and getting rid of outdated goods. The maintenance of an effective and well-stocked pharmacy is mostly dependent on the inventory management.

4.4.2 Pharmacy manager

Pharmacy manager is keeping an eye on pharmacy records, monitoring medications for patients, updating pharmacy records, and making sure patient profiles are accurate. The general administration and effectiveness of pharmacy operations fall within the purview of the pharmacy manager.

4.4.3 Laboratory manager

Laboratory manager is keeping track of lab test results for patients, updating lab test reports, and organizing laboratory results data. Critical medical test data is accessible and accurate thanks to the laboratory manager's oversight.

5-Use-Cases

5.1 UC01: Patient Registration

5.1.1 UC01-1: Register New Patient

5.1.1.1 Brief Description

This use case's goal is to register patients. In order for hospital administration to retain a new patient's data, which will be beneficial in a number of ways, the patient must first log in and provide information when they visit the website and request a check-up.

5.2 UC01: Communication with Hospital Faculty

5.2.1 UC01-1: Communicate

5.2.1.1 Brief Description

The purpose of this use case is to provide a communication service so that the patient can ask any questions he may have of an expert. This service allows patients to see a doctor and receive assistance with lab and medical results, among other things. The patient can contact you by email or phone.

5.3 UC01: An Option to Search

5.3.1 UC01-1: Search

5.3.1.1 Brief Description

This use case aims to assist users in searching doctor profiles and their schedules. It will make it easier for the user to schedule a doctor's appointment.

5.4 UC01: Schedule Appointment

5.4.1 UC01-1: Scheduling appointment from doctors

5.4.1.1 Brief Description

This use case decides if the doctor is available based on the doctor's schedule and any predefined parameters (such as the location, the duration of the appointment, and the specialization). It takes both urgent cases and periodic check-ups into account, balancing patient needs with healthcare providers' availability.

5.5 UC01: View Scheduled Appointment

5.5.1 UC01-1: Doctor can view scheduled appointment of patient

5.5.1.1 Brief Description

This use case allows doctors to monitor and schedule their patients' upcoming appointments. This use case helps doctors stay organized and prepared for scheduled consultations, examinations, or treatments by increasing workflow efficiency.

5.6 UC01: View Test Reports

5.6.1 UC01-1: Doctor can view his patients test reports

5.6.1.1 Brief Description

In this use case, Doctors will have easy access to their patients' test reports. This feature increases the efficiency of medical consultations by providing doctors with access to comprehensive and up-to-date information.

5.6 UC01: Re-scheduling Appointments

5.6.1 UC01-1: Doctor can re-schedule of his patients' appointments

5.6.1.1 Brief Description

In this use case, doctors are given the flexibility to organize their schedules more effectively. By simply rescheduling patient visits, this tool enables clinicians to accommodate unanticipated events, crises, or personal commitments.

5.7 UC01: Cancel Appointments

5.7.1 UC01-1: Doctor can cancel his patients' appointments

5.7.1.1 Brief Description

Sometimes a doctor has to cancel a patient's appointment due to unanticipated circumstances, an unexpected illness, or an urgent medical procedure that needs immediate attention. This is being done to protect patient safety, address unforeseen occurrences, and give priority to life-threatening medical requirements. To avoid disruption and preserve service continuity, impacted patients should have open channels of communication and be quickly informed and rescheduled.

5.8 UC01: Pharmacy Record

5.8.1 UC01-1: Track pharmacy record

5.8.1.1 Brief Description

It is necessary to oversee and manage the administration of medications in a pharmacy. This process involves recording a variety of information, such as patient data, prescription details, and dates of medication administration.

5.9 UC01: Update Pharmacy Record

5.9.1 UC01-1: Update pharmacy records

5.9.1.1 Brief Description

In this use case, a pharmacy database's patient and prescription data are routinely updated. Pharmacists verify the accuracy of patient profiles, prescription data, and medication dispensing records. This important update helps to increase patient safety.

5.9 UC01: Bill information

5.9.1 UC01-1: Save bill information

5.9.1.1 Brief Description

In this use case, information about financial transactions or bills can be stored and arranged. We can enter and store details like bill amounts, due dates, and payment options using this for later use.

5.10 UC01: Lab Test Reports

5.10.1 UC01-1: Keep information of patients' lab reports

5.10.1.1 Brief Description

This use case requires that laboratory reports generated during experiments or examinations be maintained in an ordered fashion. This technique helps to keep important patient data in an orderly manner.

5.11 UC01: Update Lab Test Reports

5.11.1 UC01-1: Update patients' lab reports

5.11.1.1 Brief Description

In this use case, fresh test results for medical conditions are promptly and accurately added to each patient's record. By using this procedure, healthcare professionals can be sure they have access to the most up-to-date and relevant data when making decisions on patient care.

5.12 UC01: Staff Assignment

5.12.1 UC01-1: Track data of staff assignments

5.12.1.1 Brief Description

This use case involves the implementation of an entire system for monitoring and managing how hospital staff members are assigned tasks. This system would ensure proper work distribution, optimum resource efficiency, and enhance overall operational performance by gathering and storing data on staff assignments.

5.13 UC01: Track employee

5.13.1 UC01-1: Track data of employee's

5.13.1.1 Brief Description

This use case involves maintaining track of and documenting a worker's career path in a number of ways. Personal data, work responsibilities, performance indicators, attendance records, and education records are all included in this. Monitoring this information is done to help with effective HR management, evaluate worker productivity, and make sure that company policies are followed.

5.13 UC01: Database

5.13.1 UC01-1: Track data of employee's

5.13.1.1 Brief Description

Medical records, operational data, and patient information are all arranged and comprehensively stored in a hospital database. By making appointment scheduling, medical history, and treatment plans easier to manage, it improves patient care in general.

5.14 UC01: Admission and Discharge

5.14.1 UC01-1: keep information of admission and discharge of patients

5.14.1.1 Brief Description

This use case involves maintaining a complete record system that monitors patient admission and discharge information within a medical facility. In order to provide effective patient care management and continuity of medical records, this involves capturing crucial information such as patient demographics, admission dates, medical

histories, therapies administered, and discharge summaries.

5.15 UC01: Allocate Rooms

5.15.1 UC01-1: Allocate rooms to patients

5.15.1.1 Brief Description

In this use case, suitable rooms are systematically assigned to patients. Through examination of factors such as patient demands, medical requirements, and availability of rooms, this approach ensures efficient use of resources and ultimately enhances the administration of healthcare services overall.

5.16 UC01: Emergency Alerts

5.16.1 UC01-1: Sends emergency alert

5.16.1.1 Brief Description

In this use case, the system is made to quickly distribute important alerts during emergencies, guaranteeing that pertinent people or groups receive key information in a timely manner. This function alerts users to impending threats or dangerous circumstances quickly, which is vital for improving public safety and crisis response.

5.17 UC01: Emergency Alerts

5.17.1 UC01-1: Sends emergency alert

5.17.1.1 Brief Description

In this use case, the system is made to quickly distribute important alerts during emergencies, guaranteeing that relevant people or groups receive key information in a timely manner. This function alerts users to coming threats or dangerous circumstances quickly, which is vital for improving public safety and crisis response.

5.18 UC01: Notify Appointments

5.18.1 UC01-1: Notify patients of their appointments

5.18.1.1 Brief Description

This use case involves that system will notify patients about their scheduled appointments, ensuring timely communication and minimizing inconvenience.

5.19 UC01: Notify Reschedule

5.19.1 UC01-1: Notify patients of re-scheduling of appointments

5.19.1.1 Brief Description

This use case involves that system will notify patients about any changes in their scheduled appointments, ensuring timely communication and minimizing inconvenience. Through proactive notifications, the system enhances patient experience and helps in efficient healthcare management.

5.20 UC01: Low stock Alerts

5.20.1 UC01-1: Notify low stock alerts of medicines

5.20.1.1 Brief Description

The purpose of this use case is to automatically generate notifications when the supply of medication is running low, ensuring that new supplies arrive quickly to prevent shortages. It increases the effectiveness of healthcare management and helps to ensure a consistent supply of essential medications by regularly checking inventory levels.

5.21 UC01: Remove Inventory

5.21.1 UC01-1: Remove expired inventory

5.21.1.1 Brief Description

In this use case, an organization's inventory management system's expired stock identification and refreshment procedure are automated. This keeps waste to a minimum, guarantees correct stock levels, and upholds product expiration date compliance.

5.22 UC01: Medicine Record

5.22.1 UC01-1: keep medicine record

5.22.1.1 Brief Description

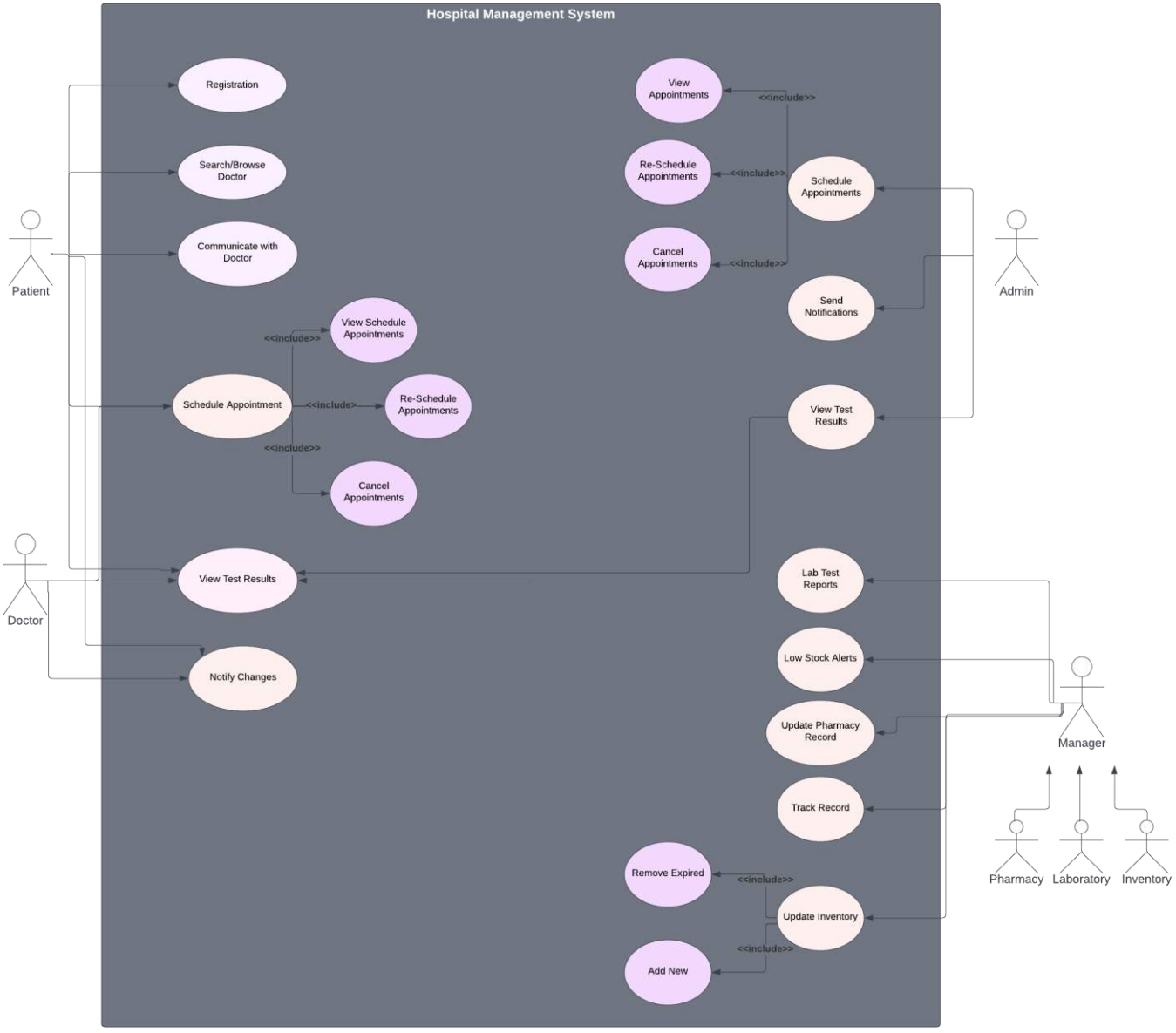
5.21 UC01: Remove Inventory

5.21.1 UC01-1: Remove expired inventory

5.21.1.1 Brief Description

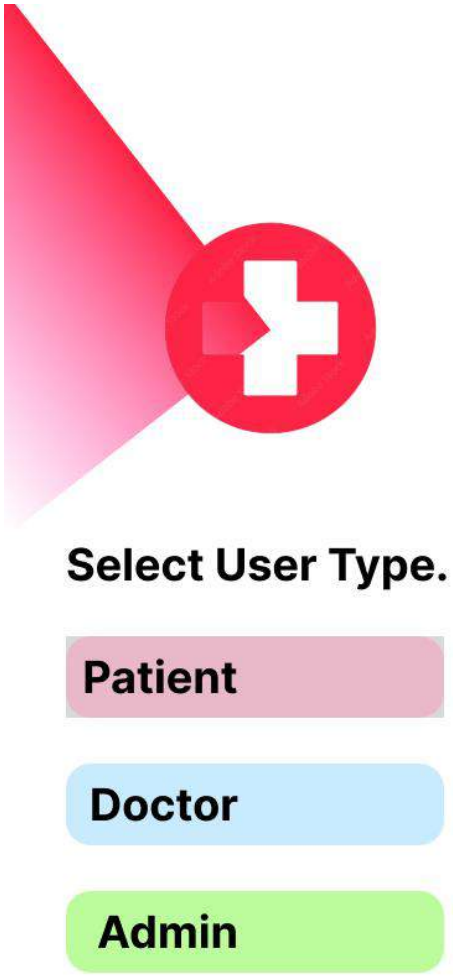
In this use case, an organization's inventory management system's expired stock identification and refreshment procedure are automated. This keeps waste to a minimum, guarantees correct stock levels, and upholds product expiration date compliance.

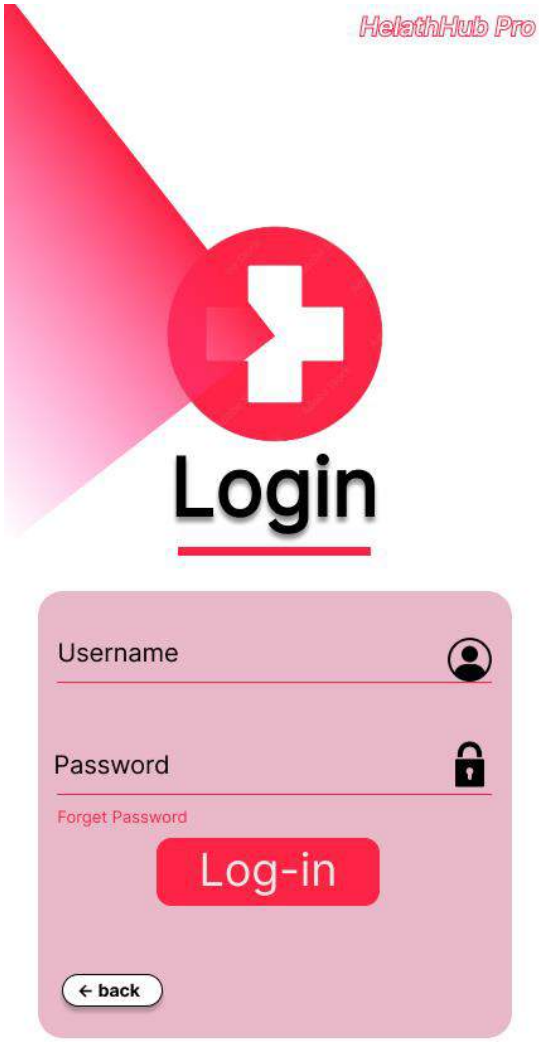
USE CASE DIAGRAM



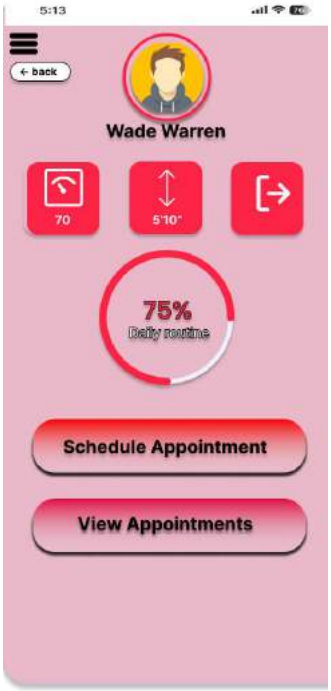
Prototype

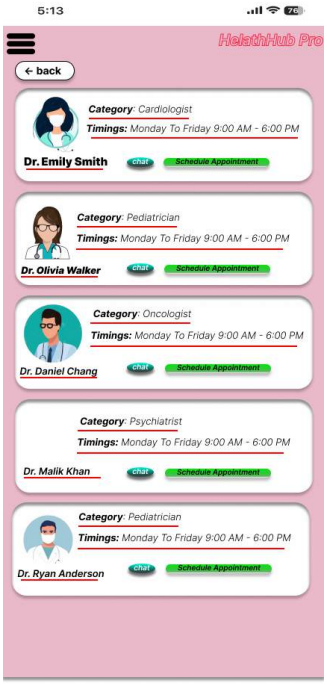
User will select the use type and will login





User can navigate throught his profile.
User can book appointments view appointments can send appointments Requests.

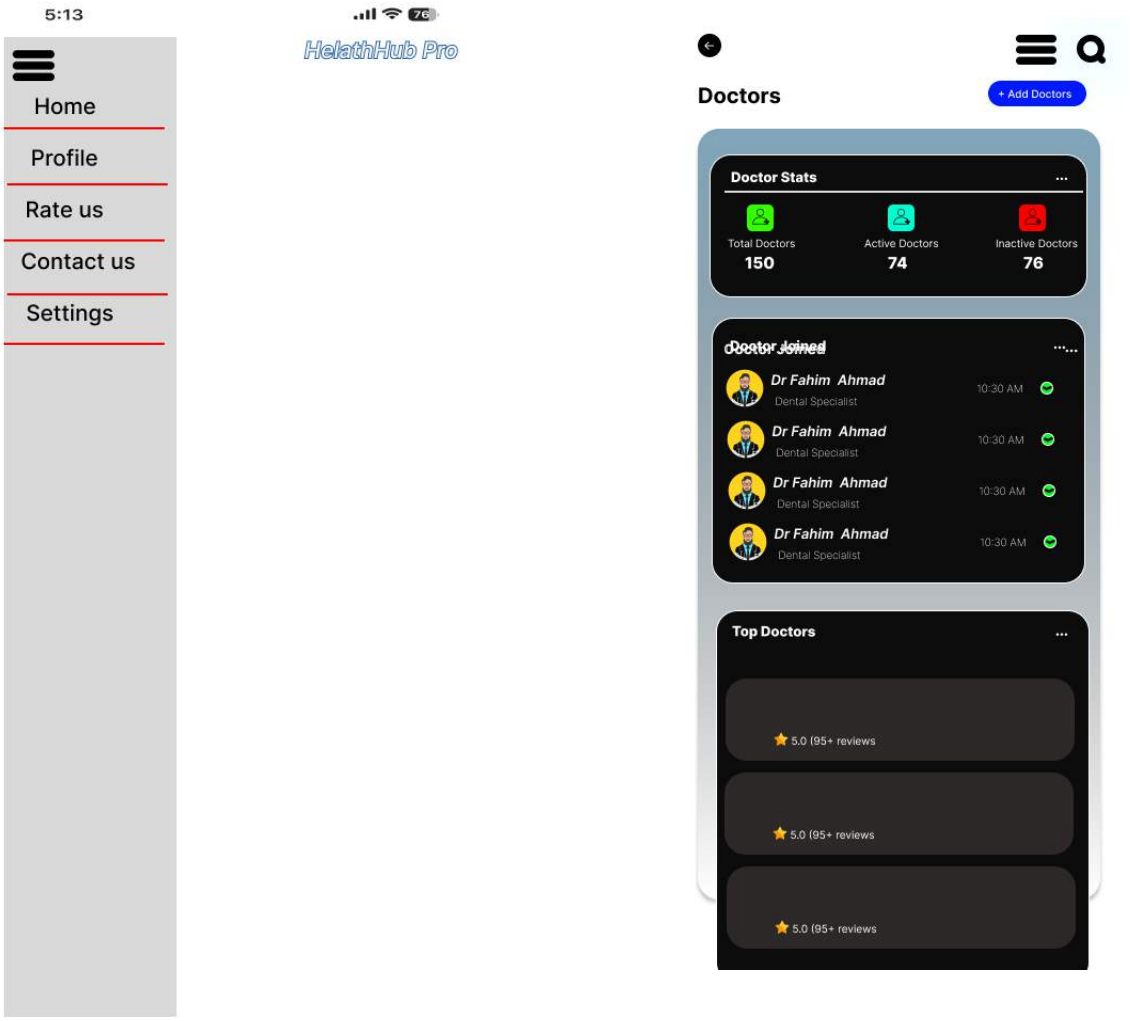




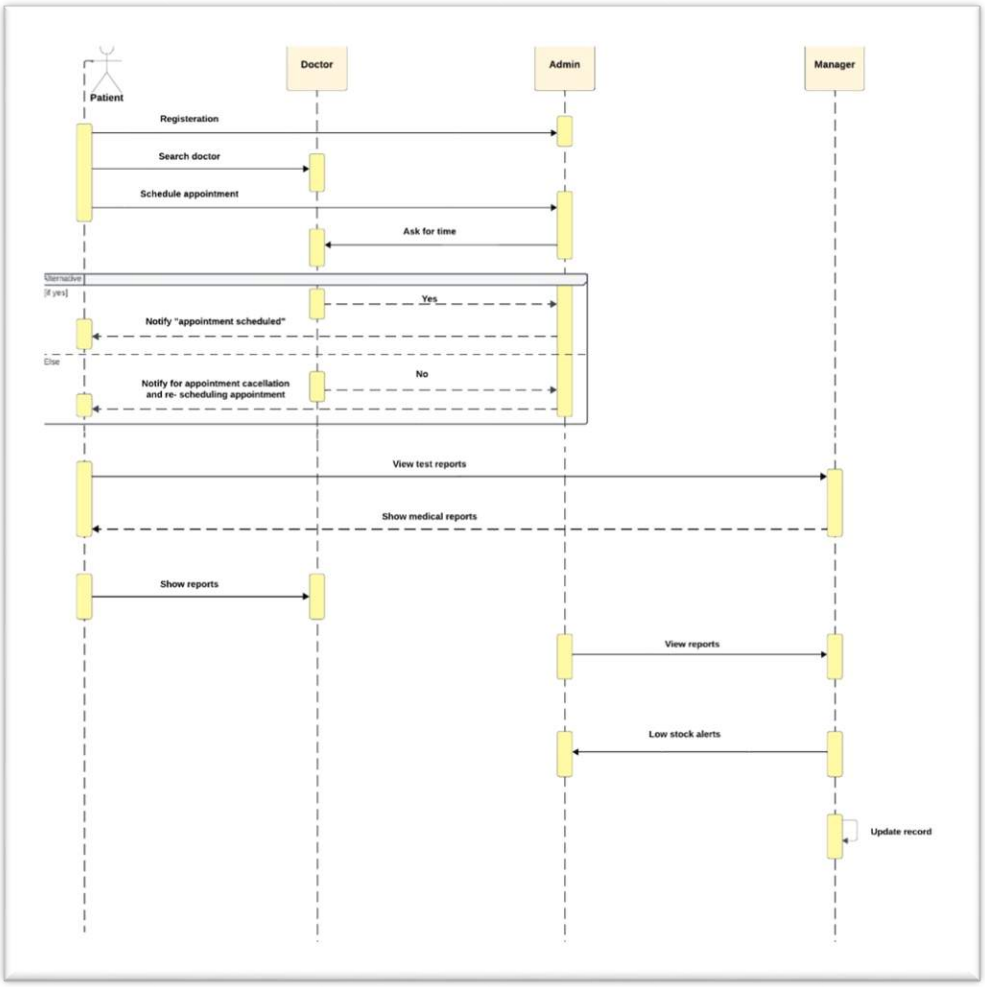
If user is doctor he can navigate throught profile can check appointments.
Doctor can accept or decline patients requests and can chat

If user is admin he can add doctors allocate rooms or manage the directory.

User can also go back to Home or can rate or contact us by using 3 bar icon.

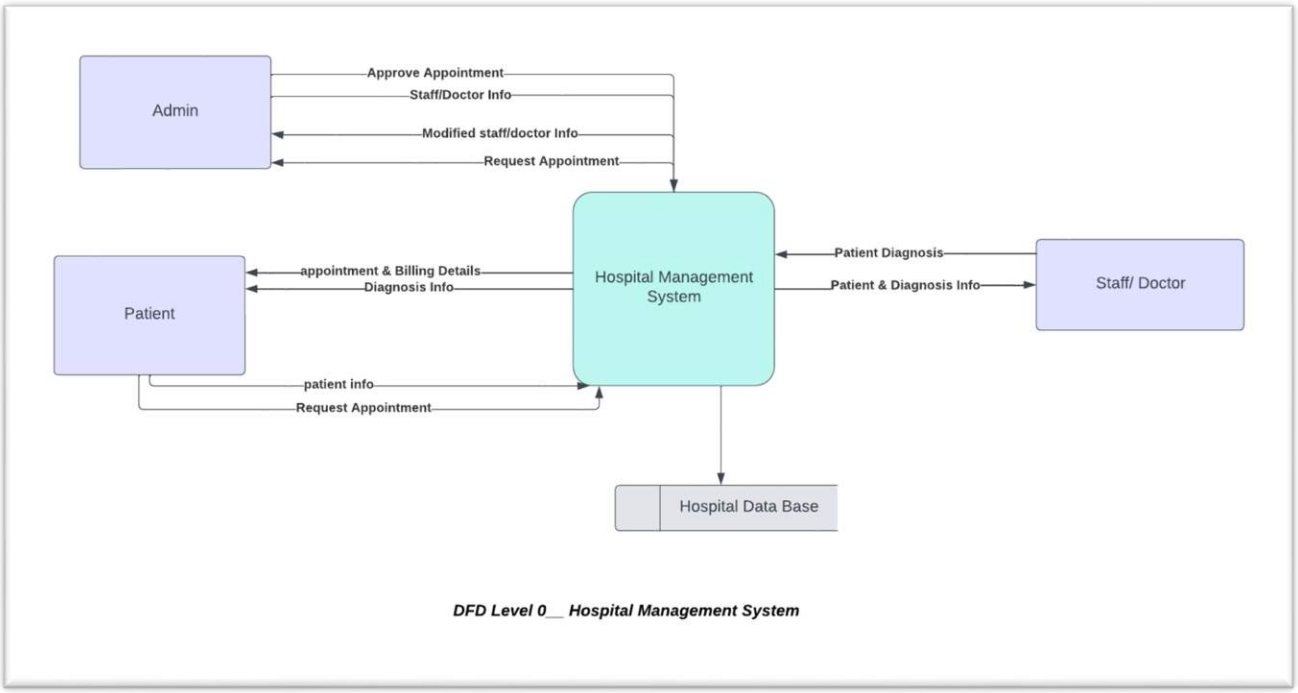


Sequence Diagram



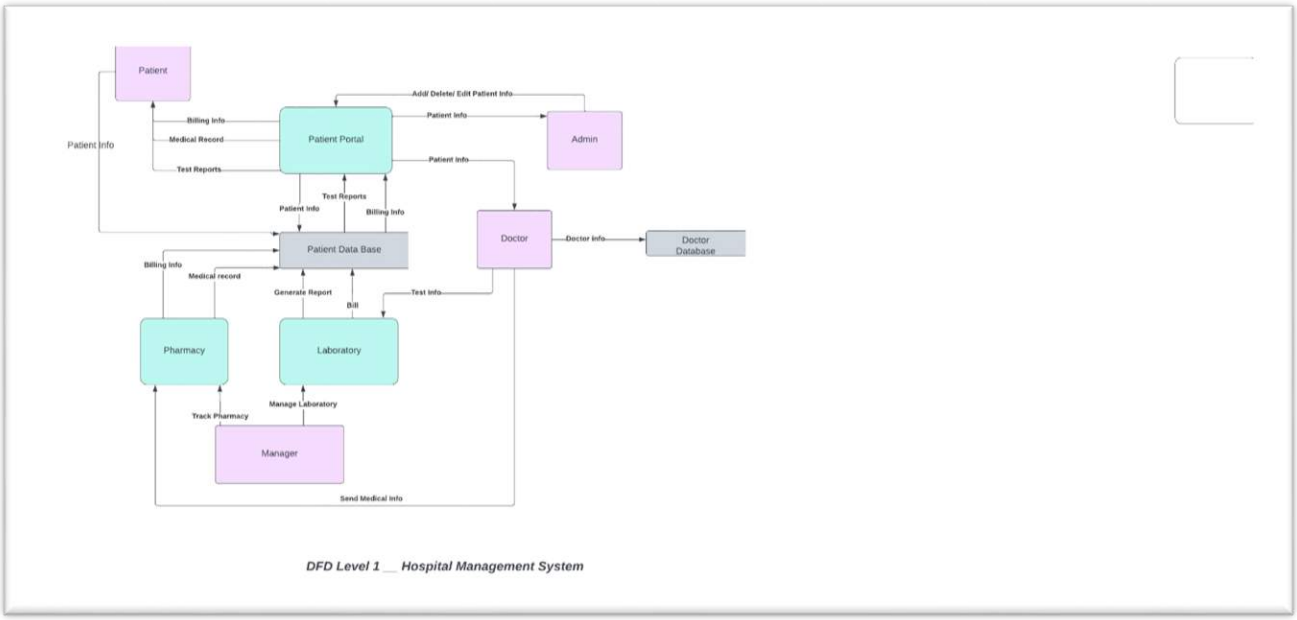
Data Flow Diagram (DFD)

Level 0



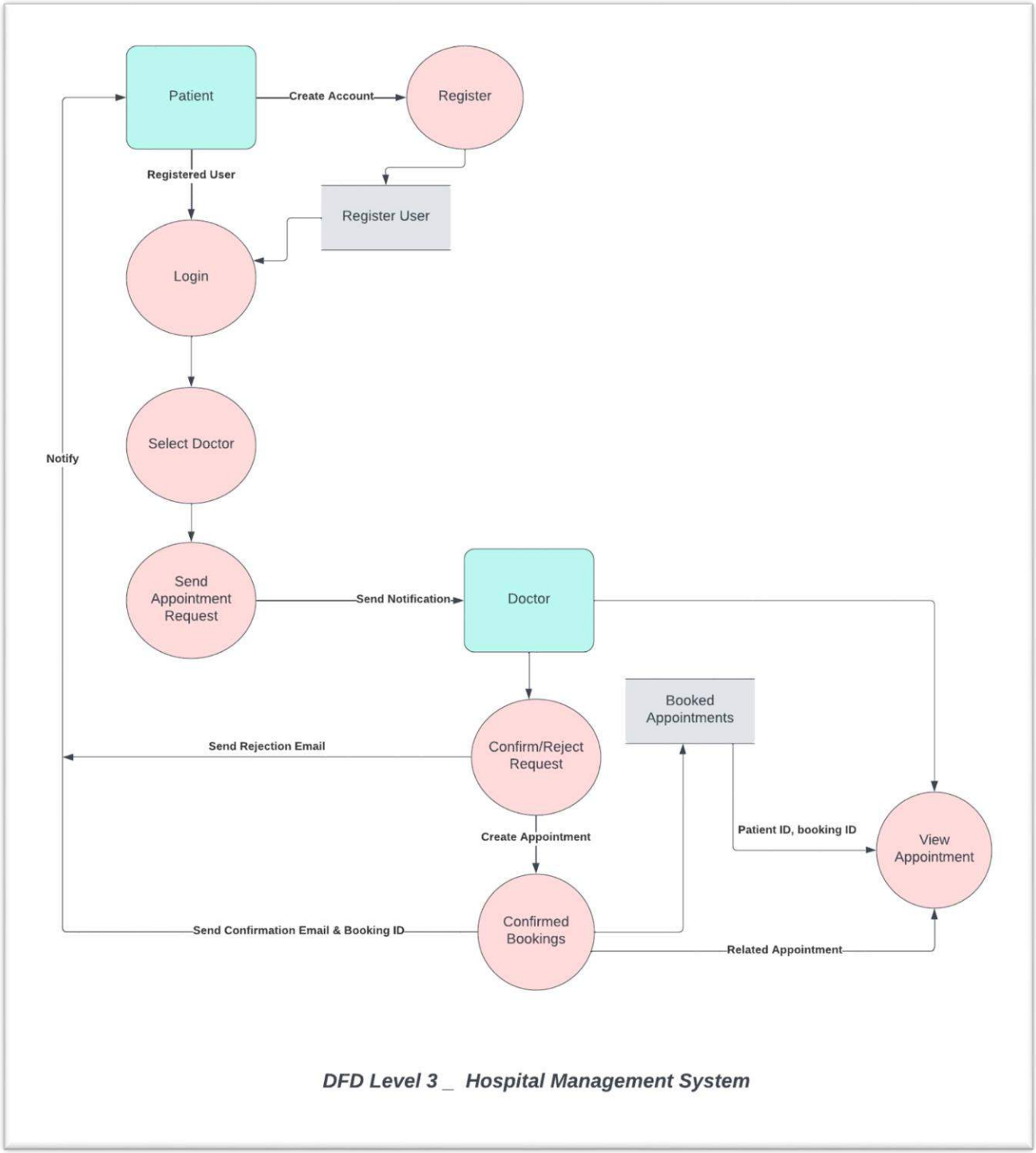
DFD Level 0__ Hospital Management System

Level 1



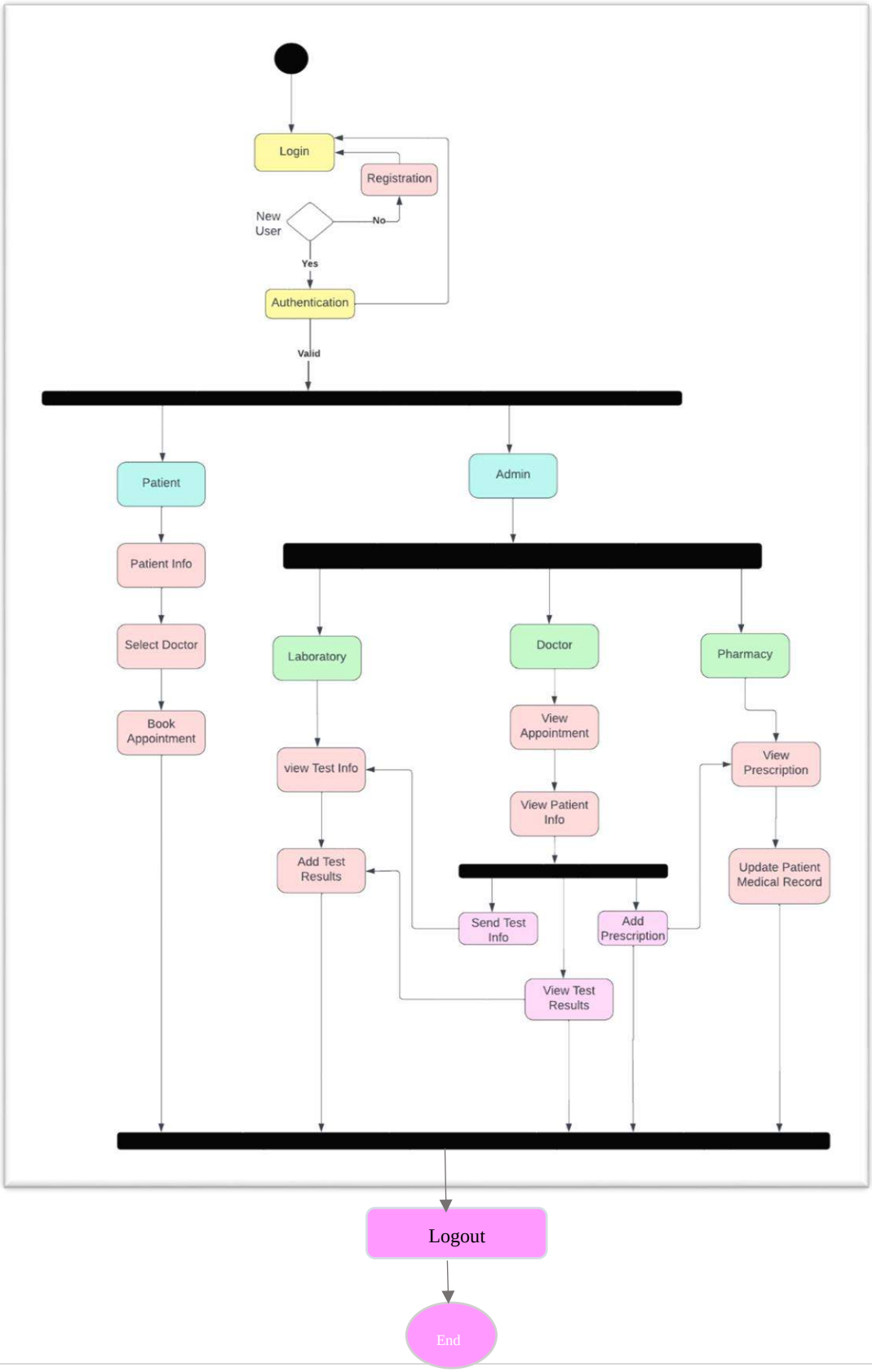
DFD Level 1 __ Hospital Management System

Level 0



DFD Level 3 _ Hospital Management System

Activity Dlagram



Code

```
#include <string>
#include <queue>
#include <stack>
#include <list>
#include <Windows.h>
#include <cstdlib>
#include <iostream>
using namespace std;

// Forward declarations
struct node
{
   int prior;
   int data; // Assuming data represents patient ID
   string name;
   string contact;
   string doctorName;
   string medicine;
   string tests;
   string diagnosis;
   bool admitted; // New field to track admission status
   struct node* link;

   // Constructor to initialize member variables
   node() : prior(0), data(0), medicine("--"), tests("--"), diagnosis("--"), admitted(false), link(nullptr) {}
};

class Priority_Queue
{
private:
   node* front;
   stack<node*> oldPatientsStack;
   queue<string> complaintsQueue; // Queue to store complaints
   list<node*> admittedPatientsList; // List to store admitted patients
public:
   Priority_Queue()
   {
      front = nullptr;
   }

   void BookAppointments()
   {
      node* tmp, * q;
      tmp = new node;

      // Initialize member variables in the BookAppointments function
      tmp->data = 0;
      tmp->prior = 0;
      tmp->link = nullptr;
```

```
// Take input from the user
cout << "Enter patient ID: ";
cin >> tmp->data;

cout << "Enter patient name: ";
cin.ignore(); // Ignore the newline character in the buffer
getline(cin, tmp->name);

cout << "Enter patient contact: ";
cin >> tmp->contact;

cout << "Enter doctor name: ";
cin.ignore(); // Ignore the newline character in the buffer
getline(cin, tmp->doctorName);

cout << "Enter priority: ";
cin >> tmp->prior;

EnterPatientType: // goto statement
cout << "Enter Few Detail:-\n1> Follow Up\n2> New Patient\nEnter Here: ";
int followup_new; cin >> followup_new;
if (followup_new == 1)
{
   cout << "Enter Medicine Prescribed: ";
   cin.ignore();
   getline(cin, tmp->medicine);

   cout << "Enter Tests Recommended: ";
   getline(cin, tmp->tests);

   cout << "Enter Previous Diagnosis: ";
   getline(cin, tmp->diagnosis);
}
else if (followup_new == 2)
{
   cout << endl;
}
else
{
   cout << "\nInvalid Input!\n";
   goto EnterPatientType; // goto statement
}

// Insert the new appointment into the priority queue
if (front == nullptr || tmp->prior < front->prior)
{
   tmp->link = front;
   front = tmp;
}
else
{
   q = front;
```

```
        while (q->link != nullptr && q->link->prior <= tmp->prior)
            q = q->link;
        tmp->link = q->link;
        q->link = tmp;
    }

    cout << "Appointment booked successfully.\n";
}

void CancelAppointments(int patientID)
{
    node* tmp, * prev;
    tmp = front;
    prev = nullptr;

    // Search for the appointment to cancel based on patient ID
    while (tmp != nullptr && tmp->data != patientID)
    {
        prev = tmp;
        tmp = tmp->link;
    }

    if (tmp == nullptr)
    {
        cout << "Appointment with Patient ID " << patientID << " not found.\n";
    }
    else
    {
        // Appointment found, perform deletion
        if (prev == nullptr)
        {
            // If the appointment to be canceled is the first in the queue
            front = front->link;
        }
        else
        {
            // If the appointment to be canceled is not the first in the queue
            prev->link = tmp->link;
        }

        cout << "Appointment with Patient ID " << patientID << " canceled successfully.\n";
        delete tmp;
    }
}

void SearchAppointments(int patientID)
{
    node* tmp = front;

    // Search for the appointment based on patient ID
    while (tmp != nullptr && tmp->data != patientID)
    {
        tmp = tmp->link;
```

```
    }

    if (tmp == nullptr)
    {
        cout << "Appointment with Patient ID " << patientID << " not found.\n";
    }
    else
    {
        // Appointment found, display details
        cout << "Appointment details for Patient ID " << patientID << ":\n";
        cout << "Patient Name: " << tmp->name << "\n";
        cout << "Patient Contact: " << tmp->contact << "\n";
        cout << "Doctor Name: " << tmp->doctorName << "\n";
        cout << "Priority: " << tmp->prior << "\n";
    }
}

void displayAllAppointments()
{
    node* tmp = front;

    if (tmp == nullptr)
    {
        cout << "No appointments to display.\n";
        return;
    }

    cout << "All Appointments (sorted by priority):\n";
    while (tmp != nullptr)
    {
        cout << "Patient ID: " << tmp->data << "\n";
        cout << "Patient Name: " << tmp->name << "\n";
        cout << "Doctor Name: " << tmp->doctorName << "\n";
        cout << "Medicine: " << tmp->medicine << "\n";
        cout << "Tests: " << tmp->tests << "\n";
        cout << "Diagnosis: " << tmp->diagnosis << "\n";
        cout << "Priority: " << tmp->prior << "\n";
        cout << "-----------------------\n";

        tmp = tmp->link;
    }
}

void ModifyAppointment(int patientID)
{
    node* tmp = front;
    node* prev = nullptr;

    // Search for the appointment based on patient ID
    while (tmp != nullptr && tmp->data != patientID)
    {
        prev = tmp;
        tmp = tmp->link;
```

```
    }

    if (tmp == nullptr)
    {
        cout << "Appointment with Patient ID " << patientID << " not found.\n";
    }
    else
    {
        // Dequeue the patient from the priority queue
        if (prev == nullptr)
        {
            // If the appointment to be modified is the first in the queue
            front = front->link;
        }
        else
        {
            // If the appointment to be modified is not the first in the queue
            prev->link = tmp->link;
        }

        // Allow modification
        cout << "Modify details for Appointment with Patient ID " << patientID << ":\n";
        cout << "Enter updated medicine: ";
        cin.ignore();
        getline(cin, tmp->medicine);

        cout << "Enter updated tests: ";
        getline(cin, tmp->tests);

        cout << "Enter updated diagnosis: ";
        getline(cin, tmp->diagnosis);

        // Option to change admission status
        cout << "Need To Admitted Patient?(1 for YES :: 0 for NO)\nEnter Here:: ";
        cin >> tmp->admitted;

        // Push the old patient data to the stack
        oldPatientsStack.push(tmp);

        // Add to admitted patients list if admitted
        if (tmp->admitted)
        {
            admittedPatientsList.push_back(tmp);
        }

        cout << "Appointment details updated successfully.\n";
    }
}

// Function to admit a patient
void AdmitPatient(int patientID)
{
    node* tmp = front;
```

```
    // Search for the patient based on patient ID
    while (tmp != nullptr && tmp->data != patientID)
    {
        tmp = tmp->link;
    }

    if (tmp == nullptr)
    {
        cout << "Patient with ID " << patientID << " not found.\n";
    }
    else
    {
        // Update admission status
        tmp->admitted = true;

        // Move the patient to the admitted patients list
        admittedPatientsList.push_back(tmp);

        cout << "Patient with ID " << patientID << " admitted successfully.\n";
    }
}

void DisplayOldPatients()
{
    if (oldPatientsStack.empty())
    {
        cout << "No old patient records available.\n";
        return;
    }

    // Create a temporary stack to preserve the original data
    stack<node*> tempStack = oldPatientsStack;

    cout << "Old Patient Records:\n";
    while (!tempStack.empty())
    {
        node* oldPatient = tempStack.top();
        cout << "Patient ID: " << oldPatient->data << "\n";
        cout << "Patient Name: " << oldPatient->name << "\n";
        cout << "Doctor Name: " << oldPatient->doctorName << "\n";
        cout << "Medicine: " << oldPatient->medicine << "\n";
        cout << "Tests: " << oldPatient->tests << "\n";
        cout << "Diagnosis: " << oldPatient->diagnosis << "\n";
        cout << "-----------------------\n";

        tempStack.pop();
    }
}

void SearchOldPatient(int patientID)
{
    if (oldPatientsStack.empty())
```

```
    {
        cout << "No old patient records available.\n";
        return;
    }

    // Create a temporary stack to preserve the original data
    stack<node*> tempStack = oldPatientsStack;
    bool found = false;

    while (!tempStack.empty())
    {
        node* oldPatient = tempStack.top();

        if (oldPatient->data == patientID)
        {
            found = true;
            cout << "Old Patient found with ID " << patientID << ":\n";
            cout << "Patient Name: " << oldPatient->name << "\n";
            cout << "Doctor Name: " << oldPatient->doctorName << "\n";
            cout << "Medicine: " << oldPatient->medicine << "\n";
            cout << "Tests: " << oldPatient->tests << "\n";
            cout << "Diagnosis: " << oldPatient->diagnosis << "\n";
            cout << "------------------------\n";
            break;
        }

        tempStack.pop();
    }

    if (!found)
    {
        cout << "Old Patient with ID " << patientID << " not found.\n";
    }
}


void DisplayAdmittedPatients()
{
    if (admittedPatientsList.empty())
    {
        cout << "No admitted patients to display.\n";
        return;
    }

    cout << "Admitted Patients:\n";
    for (const auto& patient : admittedPatientsList)
    {
        cout << "Patient ID: " << patient->data << "\n";
        cout << "Patient Name: " << patient->name << "\n";
        cout << "Doctor Name: " << patient->doctorName << "\n";
        cout << "Medicine: " << patient->medicine << "\n";
        cout << "Tests: " << patient->tests << "\n";
        cout << "Diagnosis: " << patient->diagnosis << "\n";
```

```
        cout << "------------------------\n";
    }
}

// Function to modify details of an admitted patient
void ModifyAdmittedPatient(int patientID)
{
    for (auto& patient : admittedPatientsList)
    {
        if (patient->data == patientID)
        {
            // Allow modification
            cout << "Modify details for Admitted Patient with ID " << patientID << ":\n";
            cout << "Enter updated medicine: ";
            cin.ignore();
            getline(cin, patient->medicine);

            cout << "Enter updated tests: ";
            getline(cin, patient->tests);

            cout << "Enter updated diagnosis: ";
            getline(cin, patient->diagnosis);

            cout << "Admitted Patient details updated successfully.\n";
            return;
        }
    }

    cout << "Admitted Patient with ID " << patientID << " not found.\n";
}

void SearchAdmittedPatient(int patientID)
{
    bool found = false;

    for (const auto& patient : admittedPatientsList)
    {
        if (patient->data == patientID)
        {
            found = true;
            cout << "Admitted Patient found with ID " << patientID << ":\n";
            cout << "Patient Name: " << patient->name << "\n";
            cout << "Doctor Name: " << patient->doctorName << "\n";
            cout << "Medicine: " << patient->medicine << "\n";
            cout << "Tests: " << patient->tests << "\n";
            cout << "Diagnosis: " << patient->diagnosis << "\n";
            cout << "Contact: " << patient->contact << "\n";
            break;
        }
    }

    if (!found)
    {
```

```
            cout << "Admitted Patient with ID " << patientID << " not found.\n";
        }
    }

    void RegisterComplaint(const string& complaint)
    {
        complaintsQueue.push(complaint);
        cout << "Complaint registered successfully.\n";
    }

    // Function to display complaints
    void DisplayComplaints()
    {
        if (complaintsQueue.empty())
        {
            cout << "No complaints registered.\n";
            return;
        }

        cout << "Registered Complaints:\n";
        int count = 1;
        while (!complaintsQueue.empty())
        {
            cout << count << ". " << complaintsQueue.front() << "\n";
            complaintsQueue.pop();
            count++;
        }
    }

    void LoadingFunction()
    {
        cout << "\nLoading";
        for (int i = 0; i < 5; i++)
        {
            cout << ". ";
            Sleep(500);
        }
        cout << endl;
    }
};
```